

# Projeto e Análise de Algoritmos

## Redução entre problemas

Lehilton Pedrosa

Primeiro Semestre de 2020

Cotas para um problema

# Formalizando problema

Como formalizar um problema **genericamente**?

## Problema computacional

Um problema computacional é uma **relação**  $P \subseteq \mathcal{I} \times \mathcal{S}$ , onde

- $\mathcal{I}$  é o conjunto de entradas e
- $\mathcal{S}$  é o conjunto de saídas.

# Formalizando problema

Como formalizar um problema **genericamente**?

## Problema computacional

Um problema computacional é uma **relação**  $P \subseteq \mathcal{I} \times \mathcal{S}$ , onde

- ▶  $\mathcal{I}$  é o conjunto de entradas e
- ▶  $\mathcal{S}$  é o conjunto de saídas.

# Formalizando problema

Como formalizar um problema **genericamente**?

## Problema computacional

Um problema computacional é uma **relação**  $P \subseteq \mathcal{I} \times \mathcal{S}$ , onde

- ▶  $\mathcal{I}$  é o conjunto de entradas e
- ▶  $\mathcal{S}$  é o conjunto de saídas.

# Formalizando problema

Como formalizar um problema **genericamente**?

## Problema computacional

Um problema computacional é uma **relação**  $P \subseteq \mathcal{I} \times \mathcal{S}$ , onde

- ▶  $\mathcal{I}$  é o conjunto de entradas e
- ▶  $\mathcal{S}$  é o conjunto de saídas.

# Algoritmo para um problema

## Definição

Dizemos que um algoritmo  $\text{ALG}$  **resolve** um problema  $P = (\mathcal{I}, \mathcal{P})$  se para toda entrada  $I \in \mathcal{I}$ , ele devolve uma saída  $S \in \mathcal{S}$  tal que  $(I, S) \in P$ .

- ▶ escrevemos  $I \in P$  para representar uma entrada
- ▶ escrevemos  $A(I)$  para representar a saída do algoritmo
- ▶ denotamos por  $n$  o “tamanho” de  $I$
- ▶ normalmente  $n$  é o números de bits de  $I$

# Algoritmo para um problema

## Definição

Dizemos que um algoritmo  $\text{ALG}$  **resolve** um problema  $P = (\mathcal{I}, \mathcal{P})$  se para toda entrada  $I \in \mathcal{I}$ , ele devolve uma saída  $S \in \mathcal{S}$  tal que  $(I, S) \in \mathcal{P}$ .

- ▶ escrevemos  $I \in P$  para representar uma entrada
- ▶ escrevemos  $A(I)$  para representar a saída do algoritmo
- ▶ denotamos por  $n$  o “tamanho” de  $I$
- ▶ normalmente  $n$  é o números de bits de  $I$

# Algoritmo para um problema

## Definição

Dizemos que um algoritmo  $\text{ALG}$  **resolve** um problema  $P = (\mathcal{I}, \mathcal{P})$  se para toda entrada  $I \in \mathcal{I}$ , ele devolve uma saída  $S \in \mathcal{S}$  tal que  $(I, S) \in P$ .

- ▶ escrevemos  $I \in P$  para representar uma entrada
- ▶ escrevemos  $A(I)$  para representar a saída do algoritmo
- ▶ denotamos por  $n$  o “tamanho” de  $I$
- ▶ normalmente  $n$  é o números de bits de  $I$

# Algoritmo para um problema

## Definição

Dizemos que um algoritmo  $\text{ALG}$  **resolve** um problema  $P = (\mathcal{I}, \mathcal{P})$  se para toda entrada  $I \in \mathcal{I}$ , ele devolve uma saída  $S \in \mathcal{S}$  tal que  $(I, S) \in P$ .

- ▶ escrevemos  $I \in P$  para representar uma entrada
- ▶ escrevemos  $A(I)$  para representar a saída do algoritmo
- ▶ denotamos por  $n$  o “tamanho” de  $I$
- ▶ normalmente  $n$  é o números de bits de  $I$

# Algoritmo para um problema

## Definição

Dizemos que um algoritmo  $\text{ALG}$  **resolve** um problema  $P = (\mathcal{I}, \mathcal{P})$  se para toda entrada  $I \in \mathcal{I}$ , ele devolve uma saída  $S \in \mathcal{S}$  tal que  $(I, S) \in P$ .

- ▶ escrevemos  $I \in P$  para representar uma entrada
- ▶ escrevemos  $A(I)$  para representar a saída do algoritmo
- ▶ denotamos por  $n$  o “tamanho” de  $I$
- ▶ normalmente  $n$  é o números de bits de  $I$

# Revisitando a complexidade de um algoritmo

Seja  $\text{ALG}$  um algoritmo para um problema  $P$  e  $n$  um parâmetro.

Notação  $O$ :

- ▶ Se o algoritmo leva tempo **no máximo**  $f(n)$  para toda entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $O(f(n))$ .

Notação  $\Omega$ :

- ▶ Se o algoritmo leva tempo **pelo menos**  $g(n)$  para alguma entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $\Omega(g(n))$ .

# Revisitando a complexidade de um algoritmo

Seja  $\text{ALG}$  um algoritmo para um problema  $P$  e  $n$  um parâmetro.

**Notação  $O$ :**

- ▶ Se o algoritmo leva tempo **no máximo**  $f(n)$  para toda entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $O(f(n))$ .

**Notação  $\Omega$ :**

- ▶ Se o algoritmo leva tempo **pelo menos**  $g(n)$  para alguma entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $\Omega(g(n))$ .

# Revisitando a complexidade de um algoritmo

Seja  $\text{ALG}$  um algoritmo para um problema  $P$  e  $n$  um parâmetro.

Notação  $O$ :

- ▶ Se o algoritmo leva tempo **no máximo**  $f(n)$  para toda entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $O(f(n))$ .

Notação  $\Omega$ :

- ▶ Se o algoritmo leva tempo **pelo menos**  $g(n)$  para alguma entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $\Omega(g(n))$ .

# Revisitando a complexidade de um algoritmo

Seja  $\text{ALG}$  um algoritmo para um problema  $P$  e  $n$  um parâmetro.

Notação  $O$ :

- ▶ Se o algoritmo leva tempo **no máximo**  $f(n)$  para toda entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $O(f(n))$ .

Notação  $\Omega$ :

- ▶ Se o algoritmo leva tempo **pelo menos**  $g(n)$  para alguma entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $\Omega(g(n))$ .

# Revisitando a complexidade de um algoritmo

Seja  $\text{ALG}$  um algoritmo para um problema  $P$  e  $n$  um parâmetro.

Notação  $O$ :

- ▶ Se o algoritmo leva tempo **no máximo**  $f(n)$  para toda entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $O(f(n))$ .

Notação  $\Omega$ :

- ▶ Se o algoritmo leva tempo **pelo menos**  $g(n)$  para alguma entrada de tamanho  $n$ , então dizemos que  $\text{ALG}$  executa em tempo  $\Omega(g(n))$ .

# Cotas superior e inferior de um problema

Seja  $P$  um problema e seja  $n$  um parâmetro:

## Cota superior

Uma função  $f(n)$  é chamada de cota superior para  $P$  se  
existe algum algoritmo que resolve  $P$  em tempo  $O(f(n))$ .

## Cota inferior

Uma função  $g(n)$  é chamada de cota inferior para  $P$  se  
todo algoritmo que resolve  $P$  leva tempo  $\Omega(f(n))$ .

# Cotas superior e inferior de um problema

Seja  $P$  um problema e seja  $n$  um parâmetro:

## Cota superior

Uma função  $f(n)$  é chamada de cota superior para  $P$  se  
**existe algum algoritmo** que resolve  $P$  em tempo  $O(f(n))$ .

## Cota inferior

Uma função  $g(n)$  é chamada de cota inferior para  $P$  se  
**todo algoritmo** que resolve  $P$  leva tempo  $\Omega(g(n))$ .

# Cotas superior e inferior de um problema

Seja  $P$  um problema e seja  $n$  um parâmetro:

## Cota superior

Uma função  $f(n)$  é chamada de cota superior para  $P$  se  
**existe algum algoritmo** que resolve  $P$  em tempo  $O(f(n))$ .

## Cota inferior

Uma função  $g(n)$  é chamada de cota inferior para  $P$  se  
**todo algoritmo** que resolve  $P$  leva tempo  $\Omega(g(n))$ .

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ HEAP-SORT e MERGE-SORT são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $O(n \lg n)$
- ▶ Busca-BINARIA é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $O(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ HEAP-SORT e MERGE-SORT são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $O(n \lg n)$
- ▶ Busca-BINARIA é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $O(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ HEAP-SORT e MERGE-SORT são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $O(n \lg n)$
- ▶ Busca-BINARIA é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $O(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ **HEAP-SORT** e **MERGE-SORT** são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $\Omega(n \lg n)$
- ▶ **Busca-BINARIA** é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $\Omega(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ **HEAP-SORT** e **MERGE-SORT** são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $\Omega(n \lg n)$
- ▶ **Busca-BINARIA** é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $\Omega(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ **HEAP-SORT** e **MERGE-SORT** são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $\Omega(n \lg n)$
- ▶ **Busca-BINARIA** é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $\Omega(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ **HEAP-SORT** e **MERGE-SORT** são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $\Omega(n \lg n)$
- ▶ **Busca-BINARIA** é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $\Omega(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ **HEAP-SORT** e **MERGE-SORT** são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $\Omega(n \lg n)$
- ▶ **Busca-BINARIA** é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $\Omega(\lg n)$

# Algoritmo ótimo

Um algoritmo  $\text{ALG}$  é **ótimo** para um problema  $P$  se:

1.  $\text{ALG}$  resolve  $P$  em tempo  $O(f(n))$  e
2.  $f(n)$  é uma cota inferior de  $P$ .

- ▶ **HEAP-SORT** e **MERGE-SORT** são ótimos para ordenação:
  - ▶ eles têm complexidade  $O(n \lg n)$
  - ▶ ordenação tem cota inferior  $\Omega(n \lg n)$
- ▶ **Busca-BINARIA** é ótimo para busca em vetor ordenado:
  - ▶ tem complexidade  $O(\lg n)$
  - ▶ qualquer algoritmo leva tempo  $\Omega(\lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então A é mais “fácil” do que B
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então A é mais “fácil” do que B
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então A é mais “fácil” do que B
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então A é mais “fácil” do que B
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então **A** é mais “fácil” do que **B**
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então **A** é mais “fácil” do que **B**
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então **A** é mais “fácil” do que **B**
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então **A** é mais “fácil” do que **B**
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

# Comparando problemas

Como comparamos dois algoritmos para **um único problema**?

- ▶ comparamos a complexidade de cada algoritmo
- ▶ descobrimos se um algoritmo é mais “rápido” que outro

E se quisermos comparar **dois problemas A e B**?

- ▶ queremos descobrir se então **A** é mais “fácil” do que **B**
- ▶ podemos comparar as cotas de cada algoritmo

**Exemplo:** Achar o máximo é mais **fácil** que ordenar um vetor!

- ▶ máximo tem cota superior  $O(n)$
- ▶ ordenação tem cota inferior  $\Omega(n \lg n)$

## Reduções

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de inglês para português.
- ▶ Levin, responsável por traduzir de russo para inglês.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de russo para português?

- ▶ Em geral, lidamos com problemas bem conhecidos
- ▶ Mas eventualmente, topamos com problemas novos

**Pergunta:** como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de inglês para português.
- ▶ Levin, responsável por traduzir de russo para inglês.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de russo para português?

- ▶ Em geral, lidamos com problemas bem conhecidos
- ▶ Mas eventualmente, topamos com problemas novos

Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
- ▶ Mas eventualmente, topamos com problemas novos

Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
- ▶ Mas eventualmente, topamos com problemas novos

Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
  - ▶ Mas eventualmente, topamos com problemas novos
- Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
  - ▶ Mas eventualmente, topamos com problemas novos
- Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
  - ▶ Mas eventualmente, topamos com problemas novos
- Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
- ▶ Mas eventualmente, topamos com problemas novos

Pergunta: como relacionar esses problemas?

# Combinando problemas

## Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- ▶ Cook, responsável por traduzir de **inglês para português**.
- ▶ Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo; como então traduzir de **russo para português**?

- ▶ Em geral, lidamos com problemas bem conhecidos
- ▶ Mas eventualmente, topamos com problemas novos

**Pergunta:** como relacionar esses problemas?

# Redução

## Problema A:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

## Problema B:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema A ao problema B é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de A em uma instância  $I_B$  de B
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

**Problema A:**

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

**Problema B:**

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema A ao problema B é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de A em uma instância  $I_B$  de B
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema A:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema B:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema A ao problema B é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de A em uma instância  $I_B$  de B
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema *A*:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema *B*:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema *A* ao problema *B* é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de *A* em uma instância  $I_B$  de *B*.
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema *A*:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema *B*:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema *A* ao problema *B* é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de *A* em uma instância  $I_B$  de *B*.
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema *A*:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema *B*:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema *A* ao problema *B* é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de *A* em uma instância  $I_B$  de *B*.
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema *A*:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema *B*:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema *A* ao problema *B* é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de *A* em uma instância  $I_B$  de *B*
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema *A*:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema *B*:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema *A* ao problema *B* é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de *A* em uma instância  $I_B$  de *B*
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução

Problema *A*:

- ▶ Instância:  $I_A$
- ▶ Solução:  $S_A$

Problema *B*:

- ▶ Instância:  $I_B$
- ▶ Solução:  $S_B$

## Definição

Uma **redução** do problema *A* ao problema *B* é um par de sub-rotinas  $\tau_I$  e  $\tau_S$  tais que:

- ▶  $\tau_I$  transforma instância  $I_A$  de *A* em uma instância  $I_B$  de *B*
- ▶  $\tau_S$  transforma solução  $S_B$  de  $I_B$  em uma solução  $S_A$  de  $I_A$ .

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Redução como um algoritmo

Como podemos resolver o problema  $A$ ?

1. Suponha que existe um algoritmo  $\text{ALG}_B$  para o problema  $B$ .
2. Podemos usar  $\text{ALG}_B$  como uma **caixa-preta**

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Em outras palavras:

- ▶ se sei resolver  $B$ , então também sei resolver  $A$ !
- ▶  $A$  é mais “fácil” que  $B$
- ▶ denotamos  $A \leq B$

# Um problema de origem

## Problema da Alocação de Centros (AC)

Entrada:

- ▶ um grafo bipartido conexo  $G = ((X \cup Y), E)$  e
- ▶ uma função de pesos nas arestas  $\omega : E \rightarrow \mathbb{R}_+$

Saída:

- ▶ alocar cada vértice  $v$  em  $X$  a um vértice  $\phi[v]$  em  $Y$  tal que o peso  $\omega(v, \phi[v])$  seja **mínimo**

# Um problema de origem

## Problema da Alocação de Centros (AC)

Entrada:

- ▶ um grafo bipartido conexo  $G = ((X \cup Y), E)$  e
- ▶ uma função de pesos nas arestas  $\omega : E \rightarrow \mathbb{R}_+$

Saída:

- ▶ alocar cada vértice  $v$  em  $X$  a um vértice  $\phi[v]$  em  $Y$  tal que o peso  $\omega(v, \phi[v])$  seja **mínimo**

# Um problema de origem

## Problema da Alocação de Centros (AC)

Entrada:

- ▶ um grafo bipartido conexo  $G = ((X \cup Y), E)$  e
- ▶ uma função de pesos nas arestas  $\omega : E \rightarrow \mathbb{R}_+$

Saída:

- ▶ alocar cada vértice  $v$  em  $X$  a um vértice  $\phi[v]$  em  $Y$  tal que o peso  $\omega(v, \phi[v])$  seja **mínimo**

# Um problema de origem

## Problema da Alocação de Centros (AC)

Entrada:

- ▶ um grafo bipartido conexo  $G = ((X \cup Y), E)$  e
- ▶ uma função de pesos nas arestas  $\omega : E \rightarrow \mathbb{R}_+$

Saída:

- ▶ alocar cada vértice  $v$  em  $X$  a um vértice  $\phi[v]$  em  $Y$  tal que o peso  $\omega(v, \phi[v])$  seja **mínimo**

# Um problema de origem

## Problema da Alocação de Centros (AC)

Entrada:

- ▶ um grafo bipartido conexo  $G = ((X \cup Y), E)$  e
- ▶ uma função de pesos nas arestas  $\omega : E \rightarrow \mathbb{R}_+$

Saída:

- ▶ alocar cada vértice  $v$  em  $X$  a um vértice  $\phi[v]$  em  $Y$  tal que o peso  $\omega(v, \phi[v])$  seja **mínimo**

# Um problema de origem

## Problema da Alocação de Centros (AC)

Entrada:

- ▶ um grafo bipartido conexo  $G = ((X \cup Y), E)$  e
- ▶ uma função de pesos nas arestas  $\omega : E \rightarrow \mathbb{R}_+$

Saída:

- ▶ alocar cada vértice  $v$  em  $X$  a um vértice  $\phi[v]$  em  $Y$  tal que o peso  $\omega(v, \phi[v])$  seja **mínimo**

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

# Um problema de destino

## Problema do Caminho Mínimo (CM)

Entrada:

- ▶ grafo direcionado acíclico  $G = (V, E)$
- ▶ função de peso  $\omega : E \rightarrow \mathbb{R}_+$  nas arestas
- ▶ origem  $s$ .

Saída:

- ▶ vetor  $d$  com  $d[v] = \text{dist}(s, v)$  para  $v \in V$
- ▶ vetor  $\pi$  definindo uma árvore de caminhos mínimos

## Reduzindo: transformação da entrada

Recebemos uma **entrada** do problema de origem AC:

$\tau_I(G, \omega)$

- 1     $G' \leftarrow G, \omega' \leftarrow \omega$
- 2    Adicione um novo vértice  $s$  a  $G'$ .
- 3    para cada  $v \in Y$  faça
- 4       Adicione aresta  $(s, v)$  a  $G'$ .
- 5        $\omega'(s, v) \leftarrow 0$
- 6    devolva  $(G', \omega', s)$

Tempo:  $O(Y)$

## Reduzindo: transformação da entrada

Recebemos uma **entrada** do problema de origem AC:

$\tau_I(G, \omega)$

- 1     $G' \leftarrow G, \omega' \leftarrow \omega$
- 2    Adicione um novo vértice  $s$  a  $G'$ .
- 3    para cada  $v \in Y$  faça
- 4       Adicione aresta  $(s, v)$  a  $G'$ .
- 5        $\omega'(s, v) \leftarrow 0$
- 6    devolva  $(G', \omega', s)$

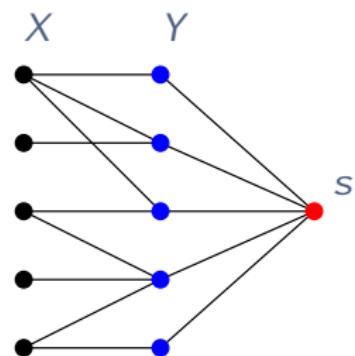
Tempo:  $O(Y)$

# Reduzindo: transformação da entrada

Recebemos uma **entrada** do problema de origem AC:

$\tau_I(G, \omega)$

- 1  $G' \leftarrow G, \omega' \leftarrow \omega$
- 2 Adicione um novo vértice  $s$  a  $G'$ .
- 3 para cada  $v \in Y$  faça
- 4 Adicione aresta  $(s, v)$  a  $G'$ .
- 5  $\omega'(s, v) \leftarrow 0$
- 6 devolva  $(G', \omega', s)$



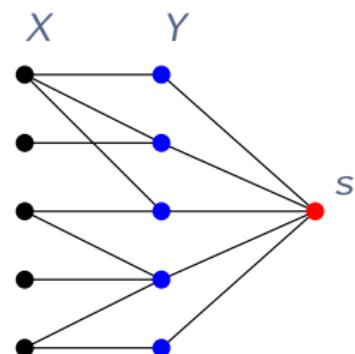
Tempo:  $O(Y)$

## Reduzindo: transformação da entrada

Recebemos uma **entrada** do problema de origem AC:

$\tau_I(G, \omega)$

- 1  $G' \leftarrow G, \omega' \leftarrow \omega$
- 2 Adicione um novo vértice  $s$  a  $G'$ .
- 3 para cada  $v \in Y$  faça
- 4 Adicione aresta  $(s, v)$  a  $G'$ .
- 5  $\omega'(s, v) \leftarrow 0$
- 6 devolva  $(G', \omega', s)$



Tempo:  $O(Y)$

## Reduzindo: transformação da saída

Também recebemos uma **solução** do problema de destino CM:

$\tau_S(G, \omega, d, \pi)$

- 1 para cada  $v \in X$  faça
- 2       $\phi[v] \leftarrow \pi[v]$
- 3 devolva  $\phi$

Tempo:  $O(X)$

## Reduzindo: transformação da saída

Também recebemos uma **solução** do problema de destino CM:

$$\tau_S(G, \omega, d, \pi)$$

- 1 para cada  $v \in X$  faça
- 2       $\phi[v] \leftarrow \pi[v]$
- 3 devolva  $\phi$

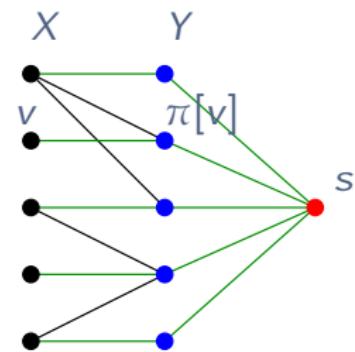
Tempo:  $O(X)$

## Reduzindo: transformação da saída

Também recebemos uma **solução** do problema de destino CM:

$\tau_S(G, \omega, d, \pi)$

- 1 para cada  $v \in X$  faça
- 2       $\phi[v] \leftarrow \pi[v]$
- 3 devolva  $\phi$



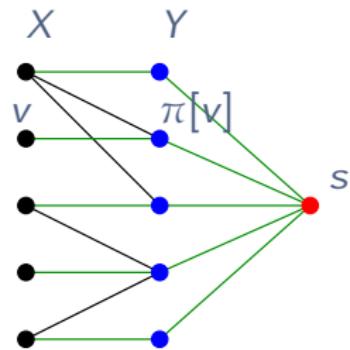
Tempo:  $O(X)$

## Reduzindo: transformação da saída

Também recebemos uma **solução** do problema de destino CM:

$\tau_S(G, \omega, d, \pi)$

- 1 para cada  $v \in X$  faça
- 2       $\phi[v] \leftarrow \pi[v]$
- 3 devolva  $\phi$



Tempo:  $O(X)$

## Reduzindo: $AC \leq CM$

Seja  $ALG_{CM}$  um algoritmo para Caminho Mínimo.

- ▶  $ALG_{CM}$  poderia ser DIJKSTRA, BELLMAN-FORD...
- ▶ Pode ser que **não conheçamos** um algoritmo para o problema de destino!

Tempo total: [tempo da redução] + [tempo de  $ALG_{CM}$ ]

## Reduzindo: $AC \leq CM$

Seja  $ALG_{CM}$  um algoritmo para Caminho Mínimo.

- ▶  $ALG_{CM}$  poderia ser DIJKSTRA, BELLMAN-FORD...
- ▶ Pode ser que não conheçamos um algoritmo para o problema de destino!

Tempo total: [tempo da redução] + [tempo de  $ALG_{CM}$ ]

## Reduzindo: $AC \leq CM$

Seja  $ALG_{CM}$  um algoritmo para Caminho Mínimo.

- ▶  $ALG_{CM}$  poderia ser DIJKSTRA, BELLMAN-FORD...
- ▶ Pode ser que **não conheçamos** um algoritmo para o problema de destino!

Tempo total: [tempo da redução] + [tempo de  $ALG_{CM}$ ]

# Reduzindo: $\text{AC} \leq \text{CM}$

Seja  $\text{ALG}_{\text{CM}}$  um algoritmo para Caminho Mínimo.

- ▶  $\text{ALG}_{\text{CM}}$  poderia ser DIJKSTRA, BELLMAN-FORD...
- ▶ Pode ser que **não conheçamos** um algoritmo para o problema de destino!

**REDUÇÃO-AC-CM( $G, \omega$ )**

- 1  $(G', \omega', s) \leftarrow \tau_I(G, \omega)$
- 2  $(d, \pi) \leftarrow \text{ALG}_{\text{CM}}(G', \omega', s)$
- 3  $\phi \leftarrow \tau_S(G, \omega, d, \pi)$
- 4 devolva  $\phi$

Tempo total: [tempo da redução] + [tempo de  $\text{ALG}_{\text{CM}}$ ]

# Reduzindo: AC $\leq$ CM

Seja  $\text{ALG}_{\text{CM}}$  um algoritmo para Caminho Mínimo.

- ▶  $\text{ALG}_{\text{CM}}$  poderia ser DIJKSTRA, BELLMAN-FORD...
- ▶ Pode ser que **não conheçamos** um algoritmo para o problema de destino!

**REDUÇÃO-AC-CM( $G, \omega$ )**

- 1  $(G', \omega', s) \leftarrow \tau_I(G, \omega)$
- 2  $(d, \pi) \leftarrow \text{ALG}_{\text{CM}}(G', \omega', s)$
- 3  $\phi \leftarrow \tau_S(G, \omega, d, \pi)$
- 4 devolva  $\phi$

Tempo total: [tempo da redução] + [tempo de  $\text{ALG}_{\text{CM}}$ ]

# Tempo da redução

Quanto tempo gastamos só com a redução?

- ▶ não contamos o tempo do algoritmo para o problema  $B$
- ▶ a complexidade de uma redução  $f(n)$  é a soma dos tempos das transformações  $\tau_I$  e  $\tau_S$
- ▶ escrevemos  $A \leq_{f(n)} B$

Nesse caso:  $AC \leq_{|X|+|Y|} CM$

# Tempo da redução

Quanto tempo gastamos só com a redução?

- ▶ não contamos o tempo do algoritmo para o problema  $B$
- ▶ a complexidade de uma redução  $f(n)$  é a soma dos tempos das transformações  $\tau_I$  e  $\tau_S$
- ▶ escrevemos  $A \leq_{f(n)} B$

Nesse caso:  $AC \leq_{|X|+|Y|} CM$

# Tempo da redução

Quanto tempo gastamos só com a redução?

- ▶ não contamos o tempo do algoritmo para o problema  $B$
- ▶ a **complexidade de uma redução**  $f(n)$  é a soma dos tempos das transformações  $\tau_I$  e  $\tau_S$
- ▶ escrevemos  $A \leq_{f(n)} B$

Nesse caso:  $AC \leq_{|x|+|y|} CM$

# Tempo da redução

Quanto tempo gastamos só com a redução?

- ▶ não contamos o tempo do algoritmo para o problema  $B$
- ▶ a **complexidade de uma redução**  $f(n)$  é a soma dos tempos das transformações  $\tau_I$  e  $\tau_S$
- ▶ escrevemos  $A \leq_{f(n)} B$

Nesse caso:  $AC \leq_{|x|+|y|} CM$

# Tempo da redução

Quanto tempo gastamos só com a redução?

- ▶ não contamos o tempo do algoritmo para o problema  $B$
- ▶ a **complexidade de uma redução**  $f(n)$  é a soma dos tempos das transformações  $\tau_I$  e  $\tau_S$
- ▶ escrevemos  $A \leq_{f(n)} B$

**REDUÇÃO-AC-CM**( $G, \omega$ )

- 1  $(G', \omega', s) \leftarrow \tau_I(G, \omega)$
- 2  $(d, \pi) \leftarrow \text{ALG}_{\text{CM}}(G', \omega', s)$
- 3  $\phi \leftarrow \tau_S(G, \omega, d, \pi)$
- 4 devolva  $\phi$

Nesse caso:  $AC \leq_{|X|+|Y|} CM$

# Tempo da redução

Quanto tempo gastamos só com a redução?

- ▶ não contamos o tempo do algoritmo para o problema  $B$
- ▶ a **complexidade de uma redução**  $f(n)$  é a soma dos tempos das transformações  $\tau_I$  e  $\tau_S$
- ▶ escrevemos  $A \leq_{f(n)} B$

**REDUÇÃO-AC-CM**( $G, \omega$ )

- 1  $(G', \omega', s) \leftarrow \tau_I(G, \omega)$
- 2  $(d, \pi) \leftarrow \text{ALG}_{\text{CM}}(G', \omega', s)$
- 3  $\phi \leftarrow \tau_S(G, \omega, d, \pi)$
- 4 devolva  $\phi$

Nesse caso:  $AC \leq_{|X|+|Y|} CM$

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em tempo polinomial
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  não tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  não tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

# Parênteses: reduções polinomiais

Queremos construir algoritmos rápidos. Mas o que é “rápido”?

- ▶ normalmente, dizemos que um algoritmo é rápido se ele executa em **tempo polinomial**
- ▶ daí, queremos reduções de tempo polinomial
- ▶ nesse caso, escrevemos  $A \leq_{\text{poli}} B$

Qual a consequência de  $A \leq_{\text{poli}} B$ ?

1. se  $B$  tem algoritmo de tempo polinomial, então  $A$  também
2. se  $A$  **não** tem algoritmo de tempo polinomial, tampouco  $B$

- ▶ Isso é útil para distinguir problemas fáceis de difíceis!
- ▶ Mas é assunto para depois...

## Exemplos de reduções

## Exemplos de reduções

- ▶ Sistema Linear (LS) para Sistema Linear Simétrico (SLS)

# Problema de origem

## Sistema Linear (LS)

Entrada:

- ▶ matriz  $M$  de dimensões  $n \times n$  com determinante **não** nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de origem

## Sistema Linear (LS)

Entrada:

- ▶ matriz  $M$  de dimensões  $n \times n$  com determinante **não** nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de origem

## Sistema Linear (LS)

Entrada:

- ▶ matriz  $M$  de dimensões  $n \times n$  com determinante **não** nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de origem

## Sistema Linear (LS)

Entrada:

- ▶ matriz  $M$  de dimensões  $n \times n$  com determinante **não** nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de origem

## Sistema Linear (LS)

Entrada:

- ▶ matriz  $M$  de dimensões  $n \times n$  com determinante **não** nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de origem

## Sistema Linear (LS)

Entrada:

- ▶ matriz  $M$  de dimensões  $n \times n$  com determinante **não** nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de destino

## Sistema Linear Simétrico (SLS)

Entrada:

- ▶ matriz  $M$  simétrica de dimensões  $n \times n$  com determinante não nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de destino

## Sistema Linear Simétrico (SLS)

Entrada:

- ▶ matriz  $M$  simétrica de dimensões  $n \times n$  com determinante não nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de destino

## Sistema Linear Simétrico (SLS)

Entrada:

- ▶ matriz  $M$  simétrica de dimensões  $n \times n$  com determinante não nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de destino

## Sistema Linear Simétrico (SLS)

Entrada:

- ▶ matriz  $M$  simétrica de dimensões  $n \times n$  com determinante não nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de destino

## Sistema Linear Simétrico (SLS)

Entrada:

- ▶ matriz  $M$  simétrica de dimensões  $n \times n$  com determinante não nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Problema de destino

## Sistema Linear Simétrico (SLS)

Entrada:

- ▶ matriz  $M$  simétrica de dimensões  $n \times n$  com determinante não nulo
- ▶ vetor  $b$  de dimensão  $n$

Saída:

- ▶ vetor  $x$  de dimensão  $n$  que satisfaz o seguinte sistema linear:

$$Mx = b$$

# Perguntas

SLS é um caso particular de LS

- ▶ então trivialmente  $SLS \leq LS$
- ▶ será que LS é estritamente mais difícil?

A resposta é **não!**

- ▶ Iremos reduzir LS para SLS.
- ▶ Isso é,  $LS \leq SLS$ .

# Perguntas

SLS é um caso particular de LS

- ▶ então trivialmente  $SLS \leq LS$
- ▶ será que LS é estritamente mais difícil?

A resposta é **não!**

- ▶ Iremos reduzir LS para SLS.
- ▶ Isso é,  $LS \leq SLS$ .

# Perguntas

SLS é um caso particular de LS

- ▶ então trivialmente  $SLS \leq LS$
- ▶ será que LS é estritamente mais difícil?

A resposta é **não!**

- ▶ Iremos reduzir LS para SLS.
- ▶ Isso é,  $LS \leq SLS$ .

# Perguntas

SLS é um caso particular de LS

- ▶ então trivialmente  $SLS \leq LS$
- ▶ será que LS é estritamente mais difícil?

A resposta é **não!**

- ▶ Iremos reduzir LS para SLS.
- ▶ Isso é,  $LS \leq SLS$ .

# Perguntas

SLS é um caso particular de LS

- ▶ então trivialmente  $SLS \leq LS$
- ▶ será que LS é estritamente mais difícil?

A resposta é **não!**

- ▶ Iremos reduzir LS para SLS.
- ▶ Isso é,  $LS \leq SLS$ .

# Perguntas

SLS é um caso particular de LS

- ▶ então trivialmente  $SLS \leq LS$
- ▶ será que LS é estritamente mais difícil?

A resposta é **não!**

- ▶ Iremos reduzir LS para SLS.
- ▶ Isso é,  $LS \leq SLS$ .

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

Demonstração:

( $\Rightarrow$ ) ➤ multiplicamos  $Mx = b$  por  $M^T$   
➤ obtemos  $M^T Mx = M^T b$

( $\Leftarrow$ ) ➤  $M^T$  tem determinante não nulo  
➤ então  $M^T$  tem inversa  $Z$   
➤ multiplicando  $M^T Mx = M^T b$  por  $Z$   
➤ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

( $\Rightarrow$ )  $\Rightarrow$  multiplicamos  $Mx = b$  por  $M^T$   
 $\Rightarrow$  obtemos  $M^T Mx = M^T b$

( $\Leftarrow$ )  $\Rightarrow$   $M^T$  tem determinante não nulo  
 $\Rightarrow$  então  $M^T$  tem inversa  $Z$   
 $\Rightarrow$  multiplicando  $M^T Mx = M^T b$  por  $Z$   
 $\Rightarrow$  obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

# Um fato simples

## Lema

Um vetor  $x$  é solução de  $Mx = b$  se, e só se,  $x$  é solução de  $M^T Mx = M^T b$ .

### Demonstração:

- ( $\Rightarrow$ ) ▶ multiplicamos  $Mx = b$  por  $M^T$   
▶ obtemos  $M^T Mx = M^T b$

- ( $\Leftarrow$ ) ▶  $M^T$  tem determinante não nulo  
▶ então  $M^T$  tem inversa  $Z$   
▶ multiplicando  $M^T Mx = M^T b$  por  $Z$   
▶ obtemos  $Mx = b$

Observe que  $M' = M^T M$  é uma matriz simétrica!

$\text{LS} \leq \text{SLS}$

**REDUÇÃO-LS-SLS**( $M, b$ )

- 1     $M' \leftarrow M^T M$
- 2     $b' \leftarrow M^T b$
- 3     $x \leftarrow \text{ALG}_{\text{SLS}}(M', b')$
- 4    **devolva**  $x$

Concluímos que de fato  $\text{LS} \leq \text{SLS}$ .

$\text{LS} \leq \text{SLS}$

**REDUÇÃO-LS-SLS**( $M, b$ )

- 1     $M' \leftarrow M^T M$
- 2     $b' \leftarrow M^T b$
- 3     $x \leftarrow \text{ALG}_{\text{SLS}}(M', b')$
- 4    devolva  $x$

Concluímos que de fato  $\text{LS} \leq \text{SLS}$ .

## Exemplos de reduções

- ▶ Casamento Cíclico de Strings para Casamento de Strings

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um **deslocamento cíclico** de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um **deslocamento cíclico** de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

**Exemplo:**

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de origem

## Problema do casamento cíclico de strings (CSM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{n-1}$  com  $n$  símbolos

Saída:

- ▶ decidir se  $B$  é um deslocamento cíclico de  $A$
- ▶ se sim, então o número  $k$  de letras deslocadas

Exemplo:

- ▶ Entrada:  $A = acgtact$  e  $B = gtactac$
- ▶ Saída: TRUE,  $k = 2$

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

Exemplo:

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

**Exemplo:**

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

**Exemplo:**

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

**Exemplo:**

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: TRUE,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

# Problema de destino

## Problema do casamento de strings (SM)

Entrada:

- ▶ alfabeto  $\Sigma$
- ▶ cadeia  $A = a_0a_1\dots a_{n-1}$  com  $n$  símbolos
- ▶ cadeia  $B = b_0b_1\dots b_{m-1}$  com  $m$  símbolos

Saída:

- ▶ decidir se  $B$  é **subcadeia** de  $A$
- ▶ se sim, qual índice  $k$  da primeira ocorrência de  $B$  em  $A$

**Exemplo:**

- ▶ Entrada:  $A = acgttaccgtacccg$  e  $B = tac$
- ▶ Saída: **TRUE**,  $k = 4$

**Observação:** o problema SM pode ser resolvido em tempo  $O(n + m)$  pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

## REDUÇÃO-CSM-SM( $A, B, n$ )

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva  $\text{ALG}_{\text{SM}}(A', n', B', m')$

- ▶ Tempo da redução:  $O(n)$
- ▶ Correção: basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

Exemplo:

- ▶  $I_{\text{CSM}} = (\text{acgtact}, \text{gtactac}, 7)$
- ▶  $I_{\text{SM}} = (\text{acgtactacgtact}, 14, \text{gtactac}, 7)$
- ▶  $S_{\text{SM}} = S_{\text{CSM}} = (\text{TRUE}, 2)$

**REDUÇÃO-CSM-SM( $A, B, n$ )**

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva  $\text{ALG}_{\text{SM}}(A', n', B', m')$

- ▶ **Tempo da redução:**  $O(n)$
- ▶ **Correção:** basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

Exemplo:

- ▶  $I_{\text{CSM}} = (\text{acgtact}, \text{gtactac}, 7)$
- ▶  $I_{\text{SM}} = (\text{acgtactacgtact}, 14, \text{gtactac}, 7)$
- ▶  $S_{\text{SM}} = S_{\text{CSM}} = (\text{TRUE}, 2)$

## REDUÇÃO-CSM-SM( $A, B, n$ )

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva  $\text{ALG}_{\text{SM}}(A', n', B', m')$

- ▶ **Tempo da redução:**  $O(n)$
- ▶ **Correção:** basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

Exemplo:

- ▶  $I_{\text{CSM}} = (\text{acgtact}, \text{gtactac}, 7)$
- ▶  $I_{\text{SM}} = (\text{acgtactacgtact}, 14, \text{gtactac}, 7)$
- ▶  $S_{\text{SM}} = S_{\text{CSM}} = (\text{TRUE}, 2)$

## REDUÇÃO-CSM-SM( $A, B, n$ )

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva ALG<sub>SM</sub>( $A', n', B', m'$ )

- ▶ **Tempo da redução:**  $O(n)$
- ▶ **Correção:** basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

### Exemplo:

- ▶  $I_{CSM} = (acgtact, gtactac, 7)$
- ▶  $I_{SM} = (acgtactacgtact, 14, gtactac, 7)$
- ▶  $S_{SM} = S_{CSM} = (\text{TRUE}, 2)$

**REDUÇÃO-CSM-SM( $A, B, n$ )**

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva  $\text{ALG}_{\text{SM}}(A', n', B', m')$

- ▶ **Tempo da redução:**  $O(n)$
- ▶ **Correção:** basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

**Exemplo:**

- ▶  $I_{\text{CSM}} = (\text{acgtact}, \text{gtactac}, 7)$
- ▶  $I_{\text{SM}} = (\text{acgtactacgtact}, 14, \text{gtactac}, 7)$
- ▶  $S_{\text{SM}} = S_{\text{CSM}} = (\text{TRUE}, 2)$

**REDUÇÃO-CSM-SM( $A, B, n$ )**

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva  $\text{ALG}_{\text{SM}}(A', n', B', m')$

- ▶ **Tempo da redução:**  $O(n)$
- ▶ **Correção:** basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

**Exemplo:**

- ▶  $I_{\text{CSM}} = (\text{acgtact}, \text{gtactac}, 7)$
- ▶  $I_{\text{SM}} = (\text{acgtactacgtact}, 14, \text{gtactac}, 7)$
- ▶  $S_{\text{SM}} = S_{\text{CSM}} = (\text{TRUE}, 2)$

**REDUÇÃO-CSM-SM( $A, B, n$ )**

- 1  $A' \leftarrow AA$        $\triangleright$  concatena duas cópias de  $A$
- 2  $B' \leftarrow B$
- 3  $n' \leftarrow 2n$
- 4  $m' \leftarrow n$
- 5 devolva  $\text{ALG}_{\text{SM}}(A', n', B', m')$

- ▶ **Tempo da redução:**  $O(n)$
- ▶ **Correção:** basta mostrar que se  $k$  é a solução de SM, então  $k$  também é solução de CSM.

**Exemplo:**

- ▶  $I_{\text{CSM}} = (\text{acgtact}, \text{gtactac}, 7)$
- ▶  $I_{\text{SM}} = (\text{acgtactacgtact}, 14, \text{gtactac}, 7)$
- ▶  $S_{\text{SM}} = S_{\text{CSM}} = (\text{TRUE}, 2)$

## Exemplos de reduções

- ▶ Existência de Triângulo para Multiplicação de Matrizes Quadradas

# Problema de origem

## Problema da existência de triângulo (PET)

Entrada:

- ▶ grafo conexo  $G = (V, E)$  sem laços com  $n = |V|$  e  $m = |E|$

Saída:

- ▶ decidir se  $G$  contém um triângulo

# Problema de origem

## Problema da existência de triângulo (PET)

Entrada:

- ▶ grafo conexo  $G = (V, E)$  sem laços com  $n = |V|$  e  $m = |E|$

Saída:

- ▶ decidir se  $G$  contém um triângulo

# Problema de origem

## Problema da existência de triângulo (PET)

Entrada:

- ▶ grafo conexo  $G = (V, E)$  sem laços com  $n = |V|$  e  $m = |E|$

Saída:

- ▶ decidir se  $G$  contém um triângulo

# Problema de origem

## Problema da existência de triângulo (PET)

Entrada:

- ▶ grafo conexo  $G = (V, E)$  sem laços com  $n = |V|$  e  $m = |E|$

Saída:

- ▶ decidir se  $G$  contém um triângulo

# Problema de origem

## Problema da existência de triângulo (PET)

Entrada:

- ▶ grafo conexo  $G = (V, E)$  sem laços com  $n = |V|$  e  $m = |E|$

Saída:

- ▶ decidir se  $G$  contém um triângulo

Exemplo:

# Problema de origem

## Problema da existência de triângulo (PET)

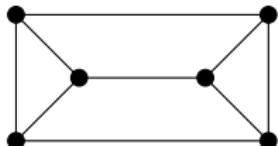
Entrada:

- ▶ grafo conexo  $G = (V, E)$  sem laços com  $n = |V|$  e  $m = |E|$

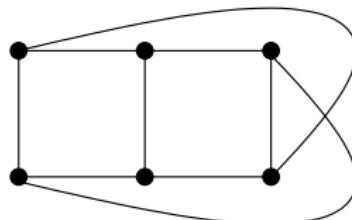
Saída:

- ▶ decidir se  $G$  contém um triângulo

Exemplo:



SIM



NÃO

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é esparsa

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é esparsa

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é esparsa

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é **esparso**

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é **esparsos**

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é **esparso**

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Observações sobre o PET

Alguns algoritmos conhecidos:

- ▶ um algoritmo trivial de tempo  $O(n^3)$ :
  - ▶ verificar todas as triplas de vértices
- ▶ um algoritmo  $O(mn)$  que
  - ▶ é muito bom se o grafo é **esparsos**

Vamos supor que o grafo é denso:

- ▶  $G$  será representado por uma matriz de adjacência  $A$

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

Demonstração:

( $\Rightarrow$ ) 

- » se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
- » segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
- » ou seja, há arestas  $(i,k)$  e  $(k,j)$

( $\Leftarrow$ ) 

- » seja um caminho  $\{i,k,j\}$  de  $i$  até  $j$
- » então  $a_{ik} = 1$  e  $a_{kj} = 1$
- » daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

Demonstração:

- ( $\Rightarrow$ )
- se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- seja um caminho  $\{i, k, j\}$  de  $i$  até  $j$
  - então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

( $\Rightarrow$ )

- se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
- segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
- ou seja, há arestas  $(i,k)$  e  $(k,j)$

( $\Leftarrow$ )

- seja um caminho  $(i,k,j)$  de  $i$  até  $j$
- então  $a_{ik} = 1$  e  $a_{kj} = 1$
- daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Um lema útil

## Lema

Seja  $A^2 = A \times A$ , ou seja,  $a_{ij}^2 = \sum_{k=1}^n a_{ik} a_{kj}$ .

Então  $a_{ij}^2 > 0$  se e somente se existe caminho de tamanho dois saindo de  $i$  e chegando em  $j$ .

### Demonstração:

- ( $\Rightarrow$ )
- ▶ se  $a_{ij}^2 > 0$ , então algum termo  $a_{ik} a_{kj}$  é positivo
  - ▶ segue que  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ ou seja, há arestas  $(i, k)$  e  $(k, j)$

- ( $\Leftarrow$ )
- ▶ seja um caminho  $(i, k, j)$  de  $i$  até  $j$
  - ▶ então  $a_{ik} = 1$  e  $a_{kj} = 1$
  - ▶ daí  $a_{ik} a_{kj} > 0$  e, portanto,  $a_{ij}^2 > 0$

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

### Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.37})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728539})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2,807})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2,3728639})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2,807})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2,3728639})$  pelo de François Le Gall (2014)

# Problema destino

## Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada:

- ▶ matriz quadrada  $A$  de ordem  $n$ .
- ▶ matriz quadrada  $B$  de ordem  $n$ .

Saída:

- ▶ produto  $P = A \times B$ .

Observações:

- ▶ há um algoritmo óbvio de complexidade  $O(n^3)$
- ▶ MMQ pode ser resolvida mais rapidamente
  - ▶ em tempo  $O(n^{2.807})$  pelo algoritmo de Strassen (1969)
  - ▶ em tempo  $O(n^{2.3728639})$  pelo de François Le Gall (2014)

# PET $\leq$ MMQ

Observe que só existe triângulo com aresta  $(i, j)$  se

1. existir caminho de tamanho 2 de  $i$  a  $j$
2. existir aresta  $(i, j)$

- ▶ Tempo da redução:  $O(n^2)$
- ▶ Tempo total:  $O(n^{2,3728639})$

Observe que só existe triângulo com aresta  $(i, j)$  se

1. existir caminho de tamanho 2 de  $i$  a  $j$
2. existir aresta  $(i, j)$

- ▶ Tempo da redução:  $O(n^2)$
- ▶ Tempo total:  $O(n^{2,3728639})$

Observe que só existe triângulo com aresta  $(i, j)$  se

1. existir caminho de tamanho 2 de  $i$  a  $j$
2. existir aresta  $(i, j)$

- ▶ Tempo da redução:  $O(n^2)$
- ▶ Tempo total:  $O(n^{2,3728639})$

# PET $\leq$ MMQ

Observe que só existe triângulo com aresta  $(i, j)$  se

1. existir caminho de tamanho 2 de  $i$  a  $j$
2. existir aresta  $(i, j)$

**REDUÇÃO-PET-MMQ** $(A, n)$

- 1     $A^2 \leftarrow \text{ALG}_{\text{MMQ}}(A, n)$
- 2    para  $i = 1$  até  $n$  faça
- 3       para  $j = 1$  até  $n$  faça
- 4           se  $a_{ij}^2 > 0$  e  $a_{ij} = 1$  então
- 5               devolva SIM
- 6       devolva NÃO

- ▶ Tempo da redução:  $O(n^2)$
- ▶ Tempo total:  $O(n^{2,3728639})$

# PET $\leq$ MMQ

Observe que só existe triângulo com aresta  $(i, j)$  se

1. existir caminho de tamanho 2 de  $i$  a  $j$
2. existir aresta  $(i, j)$

**REDUÇÃO-PET-MMQ** $(A, n)$

- 1     $A^2 \leftarrow \text{ALG}_{\text{MMQ}}(A, n)$
- 2    para  $i = 1$  até  $n$  faça
- 3       para  $j = 1$  até  $n$  faça
- 4           se  $a_{ij}^2 > 0$  e  $a_{ij} = 1$  então
- 5               devolva SIM
- 6       devolva NÃO

► **Tempo da redução:**  $O(n^2)$

► **Tempo total:**  $O(n^{2,3728639})$

# PET $\leq$ MMQ

Observe que só existe triângulo com aresta  $(i, j)$  se

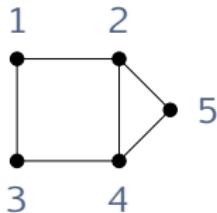
1. existir caminho de tamanho 2 de  $i$  a  $j$
2. existir aresta  $(i, j)$

**REDUÇÃO-PET-MMQ** $(A, n)$

- 1     $A^2 \leftarrow \text{ALG}_{\text{MMQ}}(A, n)$
- 2    para  $i = 1$  até  $n$  faça
- 3       para  $j = 1$  até  $n$  faça
- 4           se  $a_{ij}^2 > 0$  e  $a_{ij} = 1$  então
- 5               devolva SIM
- 6       devolva NÃO

- ▶ **Tempo da redução:**  $O(n^2)$
- ▶ **Tempo total:**  $O(n^{2,3728639})$

# PET $\leq$ MMQ



$$A(G)$$

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

$$A^2 = A(G) \times A(G)$$

	1	2	3	4	5
1	2	0	0	2	1
2	0	3	2	1	1
3	0	2	2	0	1
4	2	1	0	3	1
5	1	1	1	1	2

## Exemplos de reduções

- ▶ Multiplicação de Matrizes para Multiplicação de Matrizes Simétricas

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que MMS  $\leqslant_{m^2}$  MMQ.

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que  $\text{MMS} \leqslant_{m^2} \text{MMQ}$ .

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que  $\text{MMS} \leqslant_{m^2} \text{MMQ}$ .

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# Considerando o caso particular

Considere um caso particular de MMQ:

## Multiplicação de Matrizes Simétricas (MMS)

Entrada:

- ▶ matriz simétrica quadrada  $A$  de ordem  $m$ .
- ▶ matriz simétrica quadrada  $B$  de ordem  $m$ .

Saída:

- ▶ produto  $P = A \times B$ .

Claro que  $\text{MMS} \leqslant_{m^2} \text{MMQ}$ .

- ▶ Portanto, MMQ é pelo menos tão difícil quanto MMS
- ▶ Será que MMS também é pelo menos tão difícil quanto MMQ?

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

Tempo da redução:  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo MMQ  $\leqslant_{n^2}$  MMS:

1. Considere uma instância de MMQ,  $I_{MMQ} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{MMS} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

Tempo da redução:  $O(n^2)$ .

- ▶ construir  $I_{MMQ}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

Tempo da redução:  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

Tempo da redução:  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

Tempo da redução:  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

**Tempo da redução:**  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

**Tempo da redução:**  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# MMQ $\leqslant$ MMS

Reduzindo  $\text{MMQ} \leqslant_{n^2} \text{MMS}$ :

1. Considere uma instância de MMQ,  $I_{\text{MMQ}} = (A, B, n)$ .
2. Construa uma instância de MMS,  $I_{\text{MMS}} = (A', B', 2n)$ , em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$

3. A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$

4. Devolva o primeiro bloco da matriz  $P'$ .

Tempo da redução:  $O(n^2)$ .

- ▶ construir  $I_{\text{MMQ}}$  leva tempo  $O(n^2)$
- ▶ copiar o bloco e  $P'$  leva tempo  $O(n^2)$

# Interpretando os fatos

- ▶ suponha que exista um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $\tilde{O}(m^{2.372859})$ ?

Para responder isso, usamos a redução MMQ  $\leq_{n^2}$  MMS:

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que exista um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $\tilde{O}(m^{2.372859})$ ?

Para responder isso, usamos a redução MMQ  $\leq_{n^2}$  MMS:

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que exista um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução MMQ  $\leq_{n^2}$  MMS:

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução MMQ  $\leq_{n^2}$  MMS:

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução MMQ  $\leq_{n^2}$  MMS:

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução MMQ  $\leq_{n^2}$  MMS:

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução  $\text{MMQ} \leq_{n^2} \text{MMS}$ :

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

## Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução  $\text{MMQ} \leq_{n^2} \text{MMS}$ :

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução  $\text{MMQ} \leq_{n^2} \text{MMS}$ :

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

# Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução  $\text{MMQ} \leq_{n^2} \text{MMS}$ :

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

## Interpretando os fatos

- ▶ suponha que existe um algoritmo para MMS de tempo  $O(T(m))$  para algum polinômio  $T(m)$ 
  - ▶ relembrre que  $m$  é a ordem das matrizes  $A'$  e  $B'$
- ▶ quão pequeno pode ser  $T(m)$ ?
  - ▶ é claro que  $T(m) = \Omega(m^2)$ , pois é preciso ler a entrada
  - ▶ será que pode ser mais rápido que  $o(m^{2,3728639})$ ?

Para responder isso, usamos a redução  $\text{MMQ} \leq_{n^2} \text{MMS}$ :

- ▶ ela implica em um algoritmo de tempo total  $O(T(m) + n^2)$
- ▶ como  $m = 2n$ , tempo é  $O(T(2n) + n^2) = O(T(n) + n^2)$
- ▶ como  $T(n)$  domina  $n^2$ , o tempo é simplesmente  $O(T(n))$

Ou seja:

⇒ algoritmo  $T(m)$  para MMS implica algoritmo  $T(n)$  para MMQ!

Reduções para obtenção de cota inferior

# Uma redução $A \leqslant_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

$\text{REDUCAO}$  é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

$\text{REDUCAO}$  é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leqslant_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

$\text{REDUCAO}$  é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leqslant_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leqslant_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leqslant_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Uma redução $A \leqslant_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

Uma redução  $A \leq_{f(n)} B$

```
REDUÇÃO( $I_A$ )
1    $I_B \leftarrow \tau_I(I_A)$ 
2    $S_B \leftarrow \text{ALG}_B(I_B)$ 
3    $S_A \leftarrow \tau_S(I_A, S_B)$ 
4   devolva  $S_A$ 
```

Suponha que

- ▶  $A$  tem cota inferior  $h(n)$
- ▶  $\text{ALG}_B$  resolve  $B$  em tempo  $g(n)$
- ▶ a redução gasta tempo  $f(n) \leq \frac{h(n)}{2}$

**REDUCAO** é um algoritmo para  $A$  de tempo

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

Conclusão:  $g(n) \geq \Omega(h(n))$

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

### Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

## Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

### Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

## Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

## Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

## Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

# Transferindo cotas inferiores

## Teorema

Considere dois problemas  $A$  e  $B$  e suponha que

1.  $h(n)$  é cota inferior para  $A$  e
2.  $A \leq_{f(n)} B$ ,
3.  $f(n) = o(h(n))$ .

Então  $h(n)$  é cota inferior para  $B$ .

### Observação:

- ▶ a cota inferior depende do **modelo de computação**
- ▶ supomos o mesmo modelo para ambos problemas

## Reduções para obtenção de cota inferior

- ▶ Ordenação para Envoltória Convexa

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada oráculo)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de origem

## Problema da Ordenação (ORD)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ permutação  $X$  cujos elementos estejam ordenados

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$

# Problema de destino

## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos

Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos

Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos

Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos

Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos

Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

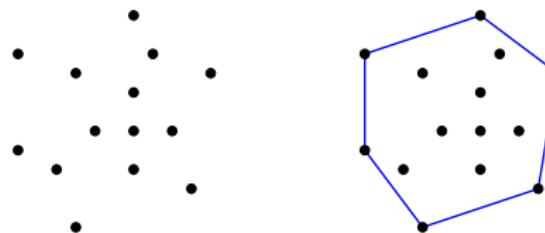
## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos



Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

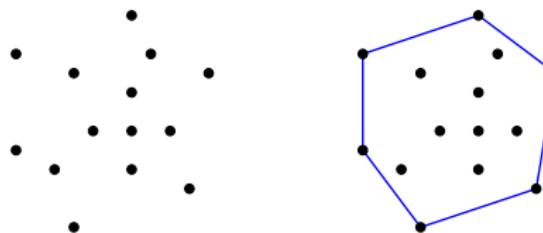
## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos



Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

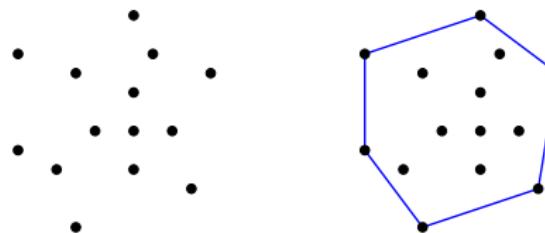
## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos



Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de Geometria Computacional
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

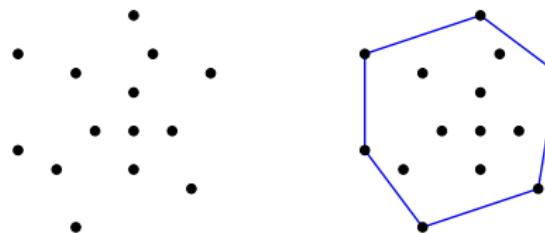
## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos



Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de **Geometria Computacional**
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

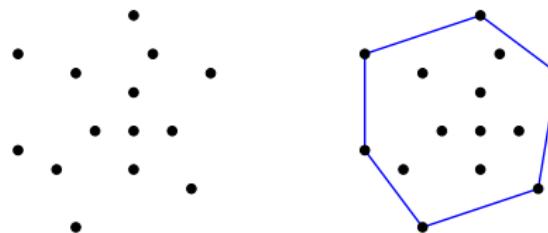
## Problema da Envoltória Convexa (EC)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ menor polígono convexo que contém os  $n$  pontos



Observações:

- ▶ os vértices são representados em ordem anti-horária
- ▶ problema clássico de **Geometria Computacional**
- ▶ pode ser resolvido em tempo  $O(n \lg n)$

ORD  $\leq_n$  EC

Reduzindo ORD  $\leq_n$  EC:

1. Considere uma instância de  $I_{ORD} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \dots, (x_n, x_n^2)\}.$$

ORD  $\leq_n$  EC

Reduzindo ORD  $\leq_n$  EC:

1. Considere uma instância de  $I_{ORD} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \dots, (x_n, x_n^2)\}.$$

ORD  $\leq_n$  EC

Reduzindo ORD  $\leq_n$  EC:

1. Considere uma instância de  $I_{ORD} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

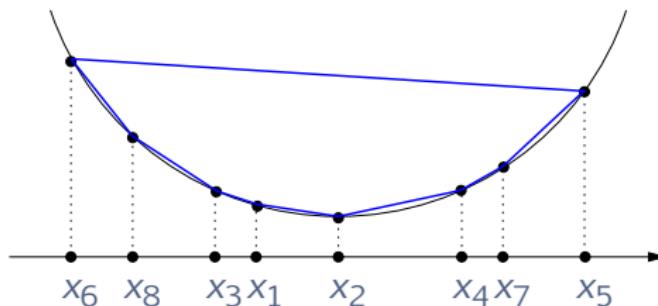
$$I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \dots, (x_n, x_n^2)\}.$$

# ORD $\leq_n$ EC

Reduzindo ORD  $\leq_n$  EC:

1. Considere uma instância de  $I_{ORD} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \dots, (x_n, x_n^2)\}.$$

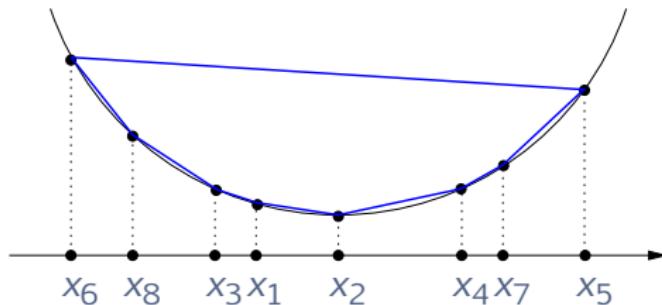


## ORD $\leq_n$ EC (cont)

3. Resolva  $I_{EC}$  e obtenha solução  $S_{EC}$ , que é uma lista **cíclica** dos vértices do polígono.
  4. Determine índice  $i$  de  $S_{EC}$  do ponto com menor abcissa.
  5. Liste os todos os índices a partir de  $i$ .
- ▶ O tempo da redução é  $O(n)$ .
  - ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para EC.

## ORD $\leq_n$ EC (cont)

3. Resolva  $I_{EC}$  e obtenha solução  $S_{EC}$ , que é uma lista **cíclica** dos vértices do polígono.

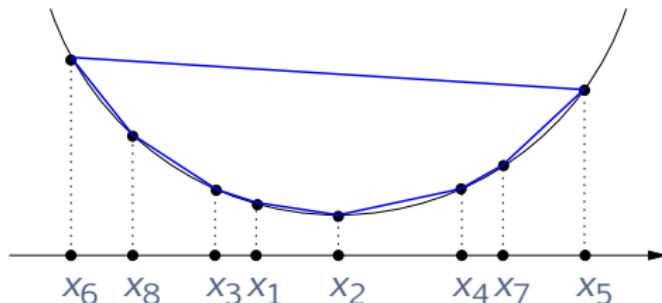


4. Determine índice  $i$  de  $S_{EC}$  do ponto com menor abscissa.  
5. Liste os todos os índices a partir de  $i$ .

- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para EC.

## ORD $\leq_n$ EC (cont)

3. Resolva  $I_{EC}$  e obtenha solução  $S_{EC}$ , que é uma lista **cíclica** dos vértices do polígono.

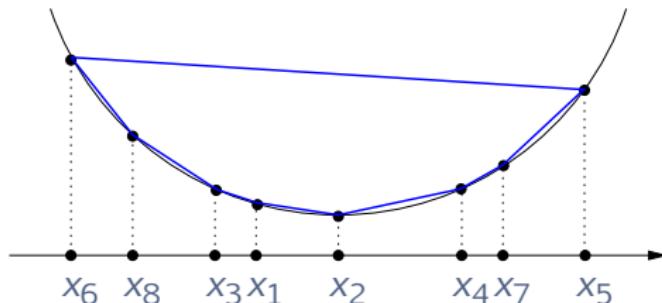


4. Determine índice  $i$  de  $S_{EC}$  do ponto com menor abscissa.  
5. Liste os todos os índices a partir de  $i$ .

- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para EC.

## ORD $\leq_n$ EC (cont)

3. Resolva  $I_{EC}$  e obtenha solução  $S_{EC}$ , que é uma lista **cíclica** dos vértices do polígono.

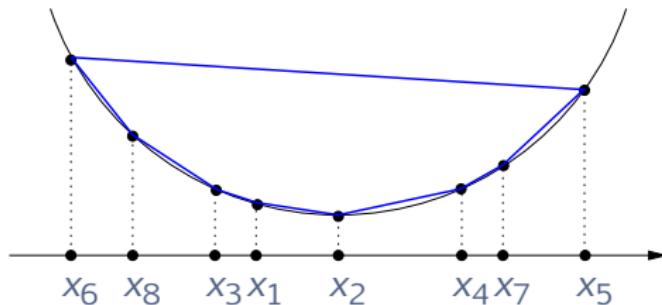


4. Determine índice  $i$  de  $S_{EC}$  do ponto com menor abcissa.
5. Liste os todos os índices a partir de  $i$ .

- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para EC.

## ORD $\leq_n$ EC (cont)

3. Resolva  $I_{EC}$  e obtenha solução  $S_{EC}$ , que é uma lista **cíclica** dos vértices do polígono.

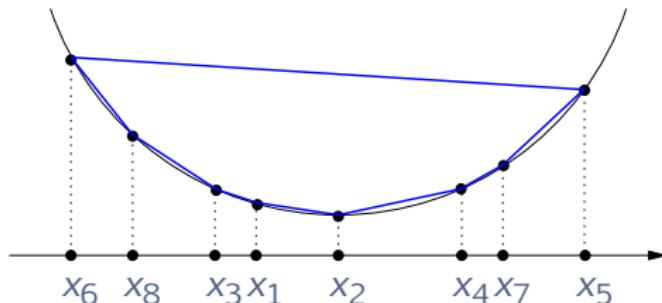


4. Determine índice  $i$  de  $S_{EC}$  do ponto com menor abcissa.
5. Liste os todos os índices a partir de  $i$ .

- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para EC.

## ORD $\leq_n$ EC (cont)

3. Resolva  $I_{EC}$  e obtenha solução  $S_{EC}$ , que é uma lista **cíclica** dos vértices do polígono.



4. Determine índice  $i$  de  $S_{EC}$  do ponto com menor abscissa.
5. Liste os todos os índices a partir de  $i$ .

- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para EC.

## Reduções para obtenção de cota inferior

- ▶ Unicidade de Elementos para Ponto Mais Próximo em 2D

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de origem

## Problema da Unicidade de Elementos (UE)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  elementos comparáveis

Saída:

- ▶ decidir se os elementos são **todos** distintos

Observações:

- ▶ só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **oráculo**)
- ▶ esse problema tem cota inferior  $\Omega(n \lg n)$
- ▶ o problema pode ser resolvido em tempo  $O(n \lg n)$ . (Como?)

# Problema de destino

## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.

Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.

Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.

Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.

Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.

Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

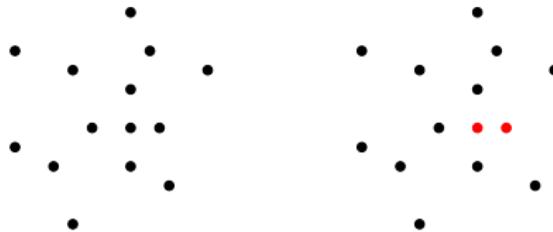
## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.



Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

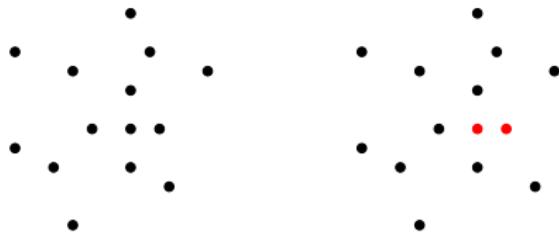
## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.



Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# Problema de destino

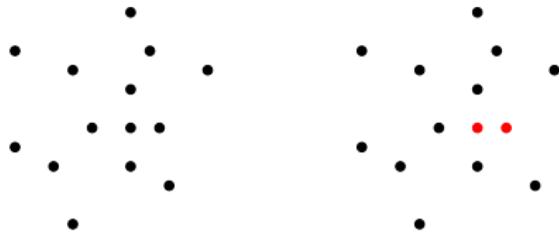
## Problema do Par Mais Próximo (PMP)

Entrada:

- ▶ coleção  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ par de pontos  $i$  e  $j$  que estejam a menor distância.



Observação:

- ▶ pode ser resolvido em tempo  $O(n \lg n)$

# $\text{UE} \leq_n \text{PMP}$

Reduzindo  $\text{UE} \leq_n \text{PMP}$ :

1. Considere instância  $I_{\text{UE}} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{\text{PMP}} = \{(x_1, 0), (x_2, 0), \dots, (x_n, 0)\}.$$

$\text{UE} \leq_n \text{PMP}$

Reduzindo  $\text{UE} \leq_n \text{PMP}$ :

1. Considere instância  $I_{\text{UE}} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{\text{PMP}} = \{(x_1, 0), (x_2, 0), \dots, (x_n, 0)\}.$$

$$\text{UE} \leq_n \text{PMP}$$

Reduzindo  $\text{UE} \leq_n \text{PMP}$ :

1. Considere instância  $I_{UE} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

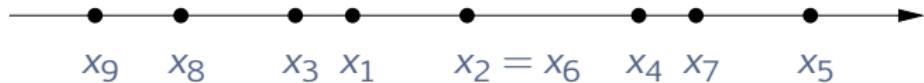
$$I_{PMP} = \{(x_1, 0), (x_2, 0), \dots, (x_n, 0)\}.$$

$$\text{UE} \leq_n \text{PMP}$$

Reduzindo  $\text{UE} \leq_n \text{PMP}$ :

1. Considere instância  $I_{UE} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{PMP} = \{(x_1, 0), (x_2, 0), \dots, (x_n, 0)\}.$$



## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
  4. Calcule a **distância**  $d$  entre os pontos:
    - (a) Se  $d = 0$ , então devolva NÃO.
    - (b) Se  $d > 0$ , devolva SIM.
- 
- ▶ O tempo da redução é  $O(n)$ .
  - ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
  4. Calcule a **distância**  $d$  entre os pontos:
    - (a) Se  $d = 0$ , então devolva NÃO.
    - (b) Se  $d > 0$ , devolva SIM.
- 
- ▶ O tempo da redução é  $O(n)$ .
  - ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
  4. Calcule a **distância**  $d$  entre os pontos:
    - (a) Se  $d = 0$ , então devolva NÃO.
    - (b) Se  $d > 0$ , devolva SIM.
- 
- ▶ O tempo da redução é  $O(n)$ .
  - ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
  4. Calcule a **distância**  $d$  entre os pontos:
    - (a) Se  $d = 0$ , então devolva NÃO.
    - (b) Se  $d > 0$ , devolva SIM.
- 
- ▶ O tempo da redução é  $O(n)$ .
  - ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
4. Calcule a **distância**  $d$  entre os pontos:
  - (a) Se  $d = 0$ , então devolva NÃO.
  - (b) Se  $d > 0$ , devolva SIM.



- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
4. Calcule a **distância**  $d$  entre os pontos:
  - (a) Se  $d = 0$ , então devolva NÃO.
  - (b) Se  $d > 0$ , devolva SIM.



- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## UE $\leq_n$ PMP (cont)

3. Resolva  $I_{PMP}$  e obtenha par de pontos  $(x_i, 0), (x_j, 0)$ .
4. Calcule a **distância**  $d$  entre os pontos:
  - (a) Se  $d = 0$ , então devolva NÃO.
  - (b) Se  $d > 0$ , devolva SIM.



- ▶ O tempo da redução é  $O(n)$ .
- ▶ Portanto  $\Omega(n \lg n)$  também é **cota inferior** para PMP.

## Reduções para obtenção de cota inferior

- ▶ 3-Soma para Colinearidade

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (4, -6, 1, 8, 7, -5)$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (4, -6, 1, 8, 7, -5)$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (4, -6, 1, 8, 7, -5)$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (4, -6, 1, 8, 7, -5)$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- instância  $X = (4, -6, 1, 8, 7, -5)$
- solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- pode ser resolvido em  $O(n^2)$  (Como?)
- acreditava-se que  $\Omega(n^2)$  era cota inferior
- pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

**Exemplo:**

- instância  $X = (4, -6, 1, 8, 7, -5)$
- solução  $i = 1, j = 3$  e  $k = 6$

**Observações:**

- pode ser resolvido em  $O(n^2)$  (Como?)
- acreditava-se que  $\Omega(n^2)$  era cota inferior
- pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

**Exemplo:**

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

**Observações:**

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

**Exemplo:**

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de origem

## Problema da 3-Soma (3SUM)

Entrada:

- ▶ sequência  $X = (x_1, x_2, \dots, x_n)$  de  $n$  reais

Saída:

- ▶ determinar se existem índices distintos  $i, j$  e  $k$  tais que:

$$x_i + x_j + x_k = 0$$

Exemplo:

- ▶ instância  $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- ▶ solução  $i = 1, j = 3$  e  $k = 6$

Observações:

- ▶ pode ser resolvido em  $O(n^2)$  (Como?)
- ▶ acreditava-se que  $\Omega(n^2)$  era cota inferior
- ▶ pode ser resolvido em  $o(n^2)$  (Grønlund e Pettie, 2014)
- ▶ ainda se acredita que não dá pra fazer melhor que  $n^{2-\Omega(1)}$

# Problema de destino

## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma **reta não horizontal**

# Problema de destino

## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma **reta não horizontal**

# Problema de destino

## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma reta não horizontal

# Problema de destino

## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma reta não horizontal

# Problema de destino

## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma reta não horizontal

# Problema de destino

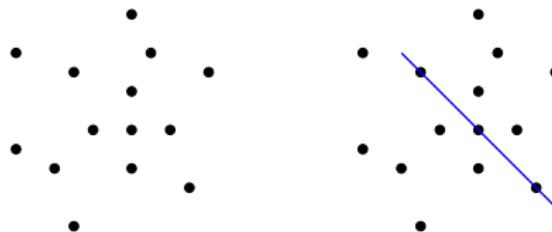
## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma **reta não horizontal**



# Problema de destino

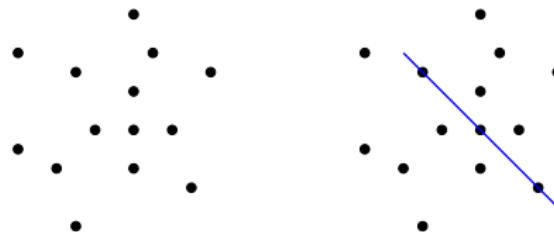
## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma **reta não horizontal**



Observações:

# Problema de destino

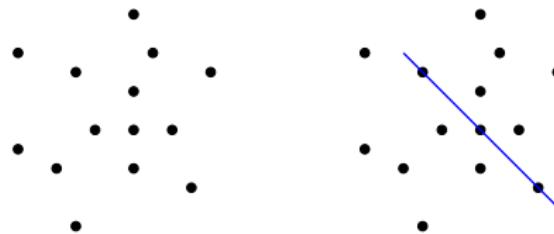
## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma **reta não horizontal**



Observações:

- ▶ pode ser resolvido em tempo  $O(n^2)$

# Problema de destino

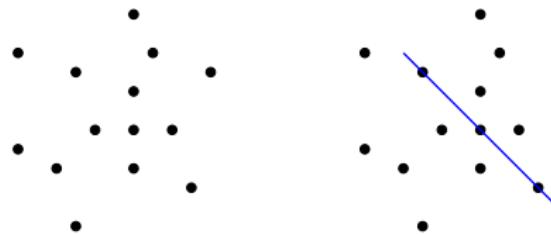
## Problema da Colinearidade Não Horizontal (COL)

Entrada:

- ▶ conjunto  $\{(x_1, y_1), \dots, (x_n, y_n)\}$  de  $n$  pontos no plano

Saída:

- ▶ determinar se três pontos estão em alguma **reta não horizontal**



Observações:

- ▶ pode ser resolvido em tempo  $O(n^2)$
- ▶ acredita-se que  $\Omega(n^2)$  é **cota inferior**

# $3\text{SUM} \leq_n \text{COL}$

Reduzindo  $3\text{SUM} \leq_n \text{COL}$ :

1. Considere instância  $I_{3\text{SUM}} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{\text{COL}} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$

3. Resolva  $I_{\text{COL}}$  e obtenha  $S_{\text{COL}}$ .
4. Se a resposta  $S_{\text{COL}}$  for SIM, responda SIM;  
mas se a resposta  $S_{\text{COL}}$  for NÃO, responda NÃO.

# $3\text{SUM} \leq_n \text{COL}$

Reduzindo  $3\text{SUM} \leq_n \text{COL}$ :

1. Considere instância  $I_{3\text{SUM}} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{\text{COL}} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$

3. Resolva  $I_{\text{COL}}$  e obtenha  $S_{\text{COL}}$ .
4. Se a resposta  $S_{\text{COL}}$  for SIM, responda SIM;  
mas se a resposta  $S_{\text{COL}}$  for NÃO, responda NÃO.

# $3\text{SUM} \leq_n \text{COL}$

Reduzindo  $3\text{SUM} \leq_n \text{COL}$ :

1. Considere instância  $I_{3\text{SUM}} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{\text{COL}} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$

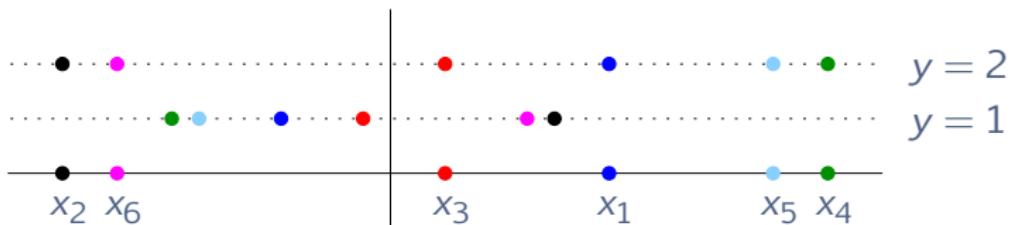
3. Resolva  $I_{\text{COL}}$  e obtenha  $S_{\text{COL}}$ .
4. Se a resposta  $S_{\text{COL}}$  for SIM, responda SIM;  
mas se a resposta  $S_{\text{COL}}$  for NÃO, responda NÃO.

# 3SUM $\leq_n$ COL

Reduzindo 3SUM  $\leq_n$  COL:

1. Considere instância  $I_{3SUM} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$



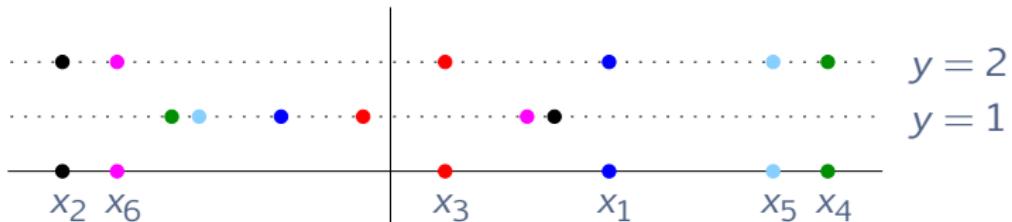
3. Resolva  $I_{COL}$  e obtenha  $S_{COL}$ .
4. Se a resposta  $S_{COL}$  for SIM, responda SIM;  
mas se a resposta  $S_{COL}$  for NÃO, responda NÃO.

# 3SUM $\leq_n$ COL

Reduzindo 3SUM  $\leq_n$  COL:

1. Considere instância  $I_{3SUM} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$



**Exemplo:**  $X = (4, -6, 1, 8, 7, -5)$

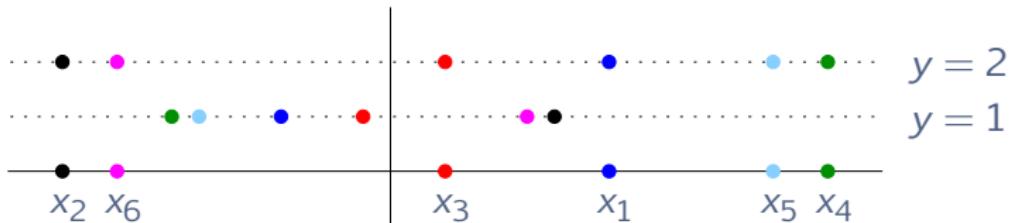
3. Resolva  $I_{COL}$  e obtenha  $S_{COL}$ .
4. Se a resposta  $S_{COL}$  for SIM, responda SIM; mas se a resposta  $S_{COL}$  for NÃO, responda NÃO.

# 3SUM $\leq_n$ COL

Reduzindo 3SUM  $\leq_n$  COL:

1. Considere instância  $I_{3SUM} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$



**Exemplo:**  $X = (4, -6, 1, 8, 7, -5)$

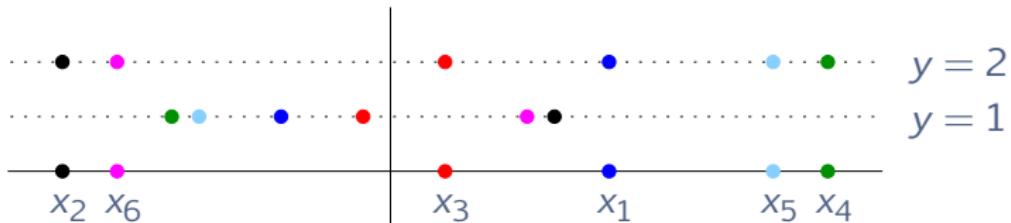
3. Resolva  $I_{COL}$  e obtenha  $S_{COL}$ .
4. Se a resposta  $S_{COL}$  for SIM, responda SIM;  
mas se a resposta  $S_{COL}$  for NÃO, responda NÃO.

# 3SUM $\leq_n$ COL

Reduzindo 3SUM  $\leq_n$  COL:

1. Considere instância  $I_{3SUM} = (x_1, x_2, \dots, x_n)$ .
2. Construa instância

$$I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \dots, n\}.$$



Exemplo:  $X = (4, -6, 1, 8, 7, -5)$

3. Resolva  $I_{COL}$  e obtenha  $S_{COL}$ .
4. Se a resposta  $S_{COL}$  for SIM, responda SIM;  
mas se a resposta  $S_{COL}$  for NÃO, responda NÃO.

## 3SUM $\leq_n$ COL (cont)

- ▶ A solução de  $I_{COL}$  (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

Portanto,  $x_i + x_j + x_k = 0$ .

- ▶ De modo análogo, se  $x_i + x_j + x_k = 0$ , então os pontos citados são colineares.

Exemplo:  $X = (4, -6, 1, 8, 7, -5)$

## 3SUM $\leq_n$ COL (cont)

- ▶ A solução de  $I_{COL}$  (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

Portanto,  $x_i + x_j + x_k = 0$ .

- ▶ De modo análogo, se  $x_i + x_j + x_k = 0$ , então os pontos citados são colineares.

Exemplo:  $X = (4, -6, 1, 8, 7, -5)$

## 3SUM $\leq_n$ COL (cont)

- ▶ A solução de  $I_{COL}$  (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

Portanto,  $x_i + x_j + x_k = 0$ .

- ▶ De modo análogo, se  $x_i + x_j + x_k = 0$ , então os pontos citados são colineares.

Exemplo:  $X = (4, -6, 1, 8, 7, -5)$

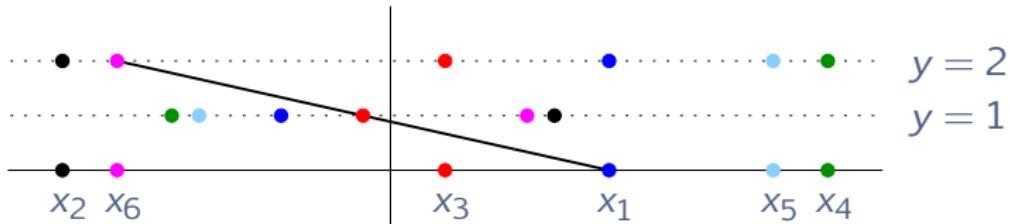
## 3SUM $\leq_n$ COL (cont)

- ▶ A solução de  $I_{COL}$  (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

Portanto,  $x_i + x_j + x_k = 0$ .

- ▶ De modo análogo, se  $x_i + x_j + x_k = 0$ , então os pontos citados são colineares.



Exemplo:  $X = (4, -6, 1, 8, 7, -5)$

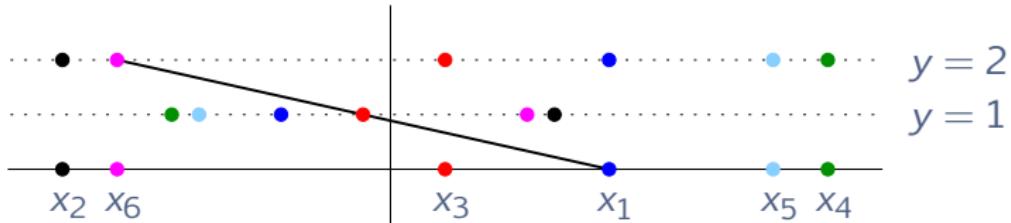
## 3SUM $\leq_n$ COL (cont)

- ▶ A solução de  $I_{COL}$  (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

Portanto,  $x_i + x_j + x_k = 0$ .

- ▶ De modo análogo, se  $x_i + x_j + x_k = 0$ , então os pontos citados são colineares.



Exemplo:  $X = (4, -6, 1, 8, 7, -5)$

## $3\text{SUM} \leq_n \text{COL}$

- ▶ É claro que  $\Omega(n)$  é uma cota inferior para 3SUM.
- ▶ E se houver cota inferior  $\Omega(h(n))$  maior para 3SUM?
  - ▶ a redução gasta tempo  $f(n) = O(n)$
  - ▶ nesse caso,  $f(m) = o(h(m))$
  - ▶ então  $\Omega(h(n))$  seria cota inferior para COL
- ▶ Mas só conhecemos a cota trivial  $\Omega(n)$  para 3SUM.

## $3\text{SUM} \leq_n \text{COL}$

- ▶ É claro que  $\Omega(n)$  é uma cota inferior para 3SUM.
- ▶ E se houver cota inferior  $\Omega(h(n))$  maior para 3SUM?
  - ▶ a redução gasta tempo  $f(n) = O(n)$
  - ▶ nesse caso,  $f(n) = o(h(n))$
  - ▶ então  $\Omega(h(n))$  seria cota inferior para COL
- ▶ Mas só conhecemos a cota trivial  $\Omega(n)$  para 3SUM.

## $3\text{SUM} \leq_n \text{COL}$

- ▶ É claro que  $\Omega(n)$  é uma cota inferior para 3SUM.
- ▶ E se houver cota inferior  $\Omega(h(n))$  maior para 3SUM?
  - ▶ a redução gasta tempo  $f(n) = O(n)$
  - ▶ nesse caso,  $f(n) = o(h(n))$
  - ▶ então  $\Omega(h(n))$  seria cota inferior para COL
- ▶ Mas só conhecemos a cota trivial  $\Omega(n)$  para 3SUM.

## $3\text{SUM} \leq_n \text{COL}$

- ▶ É claro que  $\Omega(n)$  é uma cota inferior para 3SUM.
- ▶ E se houver cota inferior  $\Omega(h(n))$  maior para 3SUM?
  - ▶ a redução gasta tempo  $f(n) = O(n)$
  - ▶ nesse caso,  $f(n) = o(h(n))$
  - ▶ então  $\Omega(h(n))$  seria cota inferior para COL
- ▶ Mas só conhecemos a cota trivial  $\Omega(n)$  para 3SUM.

## $3\text{SUM} \leq_n \text{COL}$

- ▶ É claro que  $\Omega(n)$  é uma cota inferior para  $3\text{SUM}$ .
- ▶ E se houver cota inferior  $\Omega(h(n))$  maior para  $3\text{SUM}$ ?
  - ▶ a redução gasta tempo  $f(n) = O(n)$
  - ▶ nesse caso,  $f(n) = o(h(n))$
  - ▶ então  $\Omega(h(n))$  seria cota inferior para  $\text{COL}$
- ▶ Mas só conhecemos a cota trivial  $\Omega(n)$  para  $3\text{SUM}$ .

## $3\text{SUM} \leq_n \text{COL}$

- ▶ É claro que  $\Omega(n)$  é uma cota inferior para  $3\text{SUM}$ .
- ▶ E se houver cota inferior  $\Omega(h(n))$  maior para  $3\text{SUM}$ ?
  - ▶ a redução gasta tempo  $f(n) = O(n)$
  - ▶ nesse caso,  $f(n) = o(h(n))$
  - ▶ então  $\Omega(h(n))$  seria cota inferior para  $\text{COL}$
- ▶ Mas só conhecemos a cota trivial  $\Omega(n)$  para  $3\text{SUM}$ .

# Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leqslant_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leqslant_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1.9})$  para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1.5})$  para 3SUMplus.
  - (ii) Não existe algoritmo  $O(n^{1.5})$  para 3SUM.
  - (iii) Existe um algoritmo  $O(n^{1.9})$  para 3SUMplus.
  - (iv) Existe um algoritmo  $O(n^{1.9})$  para 3SUM.

# Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1.9})$  para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1.5})$  para 3SUMplus.
  - (ii) Não existe algoritmo  $O(n^{1.5})$  para 3SUM.
  - (iii) Existe um algoritmo  $O(n^{1.9})$  para 3SUMplus.
  - (iv) Existe um algoritmo  $O(n^{1.9})$  para 3SUM.

# Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1.9})$  para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1.5})$  para 3SUMplus.
  - (ii) Não existe algoritmo  $O(n^{1.5})$  para 3SUM.
  - (iii) Existe um algoritmo  $O(n^{1.9})$  para 3SUMplus.
  - (iv) Existe um algoritmo  $O(n^{1.9})$  para 3SUM.

## Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1,9})$  para  $\text{3SUMplus}$ . Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUMplus}$ .
  - (ii) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUM}$ .
  - (iii) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUMplus}$ .
  - (iv) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUM}$ .

## Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1,9})$  para  $\text{3SUMplus}$ . Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUMplus}$ .
  - (ii) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUM}$ .
  - (iii) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUMplus}$ .
  - (iv) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUM}$ .

## Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1,9})$  para  $\text{3SUMplus}$ . Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUMplus}$ .
  - (ii) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUM}$ .
  - (iii) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUMplus}$ .
  - (iv) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUM}$ .

## Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1,9})$  para  $\text{3SUMplus}$ . Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUMplus}$ .
  - (ii) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUM}$ .
  - (iii) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUMplus}$ .
  - (iv) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUM}$ .

## Exercício

O problema 3SUMplus consiste em: dados uma sequência  $X = (x_1, x_2, \dots, x_n)$  de reais e um valor real  $b$ , determinar se existem três índices distintos  $i, j$  e  $k$  tais que  $x_i + x_j + x_k = b$ .

1. Mostre que  $\text{3SUM} \leq_n \text{3SUMplus}$ .
2. Mostre que  $\text{3SUMplus} \leq_n \text{3SUM}$ .
3. Suponha que o Professor Sabit Udo descobriu uma **cota inferior** de  $\Omega(n^{1,9})$  para  $\text{3SUMplus}$ . Nesse caso, quais das afirmações abaixo são verdadeiras?
  - (i) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUMplus}$ .
  - (ii) Não existe algoritmo  $O(n^{1,5})$  para  $\text{3SUM}$ .
  - (iii) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUMplus}$ .
  - (iv) Existe um algoritmo  $O(n^{1,9})$  para  $\text{3SUM}$ .

## Outros exemplos de reduções

## Outros exemplos de reduções

- ▶ Sistema de Representantes Distintos (SRD) para Emparelhamento Máximo (EM)

# Problema de origem

## Sistema de Representantes Distintos (SRD)

Entrada:

- ▶ uma coleção de conjuntos  $S_1, \dots, S_k$

Saída:

- ▶ conjunto  $R = \{r_1, \dots, r_k\}$  tal que  $r_i \in S_i$  para  $i = 1, \dots, k$
- ▶  $R$  é chamado de sistema de representantes distintos

# Problema de origem

## Sistema de Representantes Distintos (SRD)

Entrada:

- ▶ uma coleção de conjuntos  $S_1, \dots, S_k$

Saída:

- ▶ conjunto  $R = \{r_1, \dots, r_k\}$  tal que  $r_i \in S_i$  para  $i = 1, \dots, k$
- ▶  $R$  é chamado de sistema de representantes distintos

# Problema de origem

## Sistema de Representantes Distintos (SRD)

Entrada:

- ▶ uma coleção de conjuntos  $S_1, \dots, S_k$

Saída:

- ▶ conjunto  $R = \{r_1, \dots, r_k\}$  tal que  $r_i \in S_i$  para  $i = 1, \dots, k$
- ▶  $R$  é chamado de sistema de representantes distintos

# Problema de origem

## Sistema de Representantes Distintos (SRD)

Entrada:

- ▶ uma coleção de conjuntos  $S_1, \dots, S_k$

Saída:

- ▶ conjunto  $R = \{r_1, \dots, r_k\}$  tal que  $r_i \in S_i$  para  $i = 1, \dots, k$
- ▶  $R$  é chamado de sistema de representantes distintos

# Problema de origem

## Sistema de Representantes Distintos (SRD)

Entrada:

- ▶ uma coleção de conjuntos  $S_1, \dots, S_k$

Saída:

- ▶ conjunto  $R = \{r_1, \dots, r_k\}$  tal que  $r_i \in S_i$  para  $i = 1, \dots, k$
- ▶  $R$  é chamado de sistema de representantes distintos

# Problema de origem

## Sistema de Representantes Distintos (SRD)

Entrada:

- ▶ uma coleção de conjuntos  $S_1, \dots, S_k$

Saída:

- ▶ conjunto  $R = \{r_1, \dots, r_k\}$  tal que  $r_i \in S_i$  para  $i = 1, \dots, k$
- ▶  $R$  é chamado de sistema de representantes distintos

# Exemplos

## Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos: Ana, Alberto
  - Ruralistas: João, Alberto
  - Feministas: Ana, Maria
- ▶ então {Ana, João, Maria} é um SRD

## Um não exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos

Ecológicos: Ana, Alberto

Ruralistas: João, Alberto

Feministas: Ana, Maria

- ▶ então {Ana, João, Maria} é um SRD

Um não exemplo:

- ▶ considere o sistema de conjuntos

$$S_1 = \{1, 2\}$$

$$S_2 = \{3, 4\}$$

$$S_3 = \{3, 4\}$$

$$S_4 = \{1, 2, 4\}$$

$$S_5 = \{2, 4\}$$

- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos

Ecológicos: Ana, Alberto

Ruralistas: João, Alberto

Feministas: Ana, Maria

- ▶ então {Ana, João, Maria} é um SRD

Um não exemplo:

- ▶ considere o sistema de conjuntos

$$S_1 = \{1, 2\}$$

$$S_2 = \{3, 4\}$$

$$S_3 = \{3, 4\}$$

$$S_4 = \{1, 2, 4\}$$

$$S_5 = \{2, 4\}$$

- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos

Ecológicos: Ana, Alberto

Ruralistas: João, Alberto

Feministas: Ana, Maria

- ▶ então {Ana, João, Maria} é um SRD

Um não exemplo:

- ▶ considere o sistema de conjuntos

$$S_1 = \{1, 2\}$$

$$S_2 = \{3, 4\}$$

$$S_3 = \{3, 4\}$$

$$S_4 = \{1, 2, 4\}$$

$$S_5 = \{2, 4\}$$

- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então {Ana, João, Maria} é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então {Ana, João, Maria} é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos

$$S_1 = \{1, 2\}$$

$$S_2 = \{3, 4\}$$

$$S_3 = \{3, 4\}$$

$$S_4 = \{1, 2, 4\}$$

$$S_5 = \{2, 4\}$$

- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos

$$S_1 = \{1, 2\}$$

$$S_2 = \{3, 4\}$$

$$S_3 = \{3, 4\}$$

$$S_4 = \{1, 2, 4\}$$

$$S_5 = \{2, 4\}$$

- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos

$$S_1 = \{1, 2\}$$

$$S_2 = \{3, 4\}$$

$$S_3 = \{3, 4\}$$

$$S_4 = \{1, 2, 4\}$$

$$S_5 = \{2, 4\}$$

- ▶ não existe SRD para esse sistema

# Exemplos

Um exemplo:

- ▶ considere o sistema de conjuntos
  - Ecológicos:** Ana, Alberto
  - Ruralistas:** João, Alberto
  - Feministas:** Ana, Maria
- ▶ então  $\{Ana, João, Maria\}$  é um SRD

Um **não** exemplo:

- ▶ considere o sistema de conjuntos
  - $S_1 = \{1, 2\}$
  - $S_2 = \{3, 4\}$
  - $S_3 = \{3, 4\}$
  - $S_4 = \{1, 2, 4\}$
  - $S_5 = \{2, 4\}$
- ▶ não existe SRD para esse sistema

# Um fato útil

## Teorema de Hall

Uma coleção  $S_1, \dots, S_k$  tem um SRD se e somente se

$$\left| \{S_{i_1} \cup \dots \cup S_{i_m}\} \right| \geq m$$

para **toda** coleção  $\{i_1, \dots, i_m\} \subseteq \{1, 2, 3, \dots, k\}$ .

- ▶ i.e., cada subcoleção de  $m$  conjuntos tem  $m$  elementos
- ▶ isso sugere que basta testar todas possíveis subcoleções
- ▶ mas o número de subcoleções é  $2^k$

# Um fato útil

## Teorema de Hall

Uma coleção  $S_1, \dots, S_k$  tem um SRD se e somente se

$$\left| \{S_{i_1} \cup \dots \cup S_{i_m}\} \right| \geq m$$

para **toda** coleção  $\{i_1, \dots, i_m\} \subseteq \{1, 2, 3, \dots, k\}$ .

- ▶ i.e., cada subcoleção de  $m$  conjuntos tem  $m$  elementos
- ▶ isso sugere que basta testar todas possíveis subcoleções
- ▶ mas o número de subcoleções é  $2^k$

# Um fato útil

## Teorema de Hall

Uma coleção  $S_1, \dots, S_k$  tem um SRD se e somente se

$$\left| \{S_{i_1} \cup \dots \cup S_{i_m}\} \right| \geq m$$

para **toda** coleção  $\{i_1, \dots, i_m\} \subseteq \{1, 2, 3, \dots, k\}$ .

- ▶ i.e., cada subcoleção de  $m$  conjuntos tem  $m$  elementos
- ▶ isso sugere que basta testar todas possíveis subcoleções
- ▶ mas o número de subcoleções é  $2^k$

# Um fato útil

## Teorema de Hall

Uma coleção  $S_1, \dots, S_k$  tem um SRD se e somente se

$$\left| \{S_{i_1} \cup \dots \cup S_{i_m}\} \right| \geq m$$

para **toda** coleção  $\{i_1, \dots, i_m\} \subseteq \{1, 2, 3, \dots, k\}$ .

- ▶ i.e., cada subcoleção de  $m$  conjuntos tem  $m$  elementos
- ▶ isso sugere que basta testar todas possíveis subcoleções
- ▶ mas o número de subcoleções é  $2^k$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# Problema de destino

## Problema do Emparelhamento Máximo (EM)

Entrada:

- ▶ grafo bipartido  $G = (X \cup Y, E)$  com bipartição  $X$  e  $Y$

Solução:

- ▶ subconjunto de arestas  $M$  que não compartilham vértices

Objetivo:

- ▶ maximizar  $|M|$

# SRD $\leq$ EM

## Redução:

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_j \cup \{\text{US}_j\}$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j, 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

## Teorema

### SRD $\leq$ EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

## Redução:

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_j \cup \{S_j\}$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j, 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

## Teorema

SRD  $\leq$  EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

## Redução:

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

## Teorema

SRD  $\leq$  EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

**Redução:**

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

**Teorema**SRd  $\leq$  EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

## Redução:

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

## Teorema

SRD  $\leq$  EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

# SRD $\leq$ EM

## Redução:

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

## Teorema

### SRD $\leq$ EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

**Redução:**

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

**Teorema**

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

## Redução:

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

## Teorema

### SRD $\leq$ EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

**Redução:**

1. Tome uma instância de  $S_1, \dots, S_k$  de SRD sobre  $A$
2. Construa um grafo bipartido  $G = (X \cup Y, E)$  com
  - ▶  $X = S_1 \cup \dots \cup S_k$
  - ▶  $Y = \{1, 2, \dots, k\}$
  - ▶  $E = \{(a, j) : a \in S_j \text{ e } 1 \leq j \leq k\}$
3. Encontre um emparelhamento máximo de  $G$

**Teorema**SRd  $\leq$  EM

- ▶ note que existe um SRD se e somente se  $G$  tem emparelhamento com  $k$  arestas

## Outros exemplos de reduções

- ▶ Edição de String para Problema do Caminho Mínimo

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ **inserção** de um caractere
- ▶ **remoção** de um caractere
- ▶ **troca** de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ **minimizar** número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

Objetivo:

- ▶ minimizar número de operações

# Problema de origem

Considere as seguintes operações sobre strings:

- ▶ inserção de um caractere
- ▶ remoção de um caractere
- ▶ troca de um caractere por outro

## Edição de String

Entrada:

- ▶ duas strings  $A$  e  $B$

Solução:

- ▶ sequência de operações para transformar  $A$  em  $B$

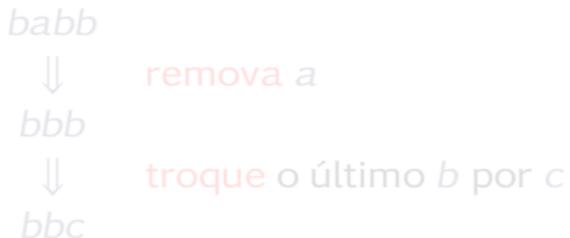
Objetivo:

- ▶ minimizar número de operações

# Exemplo

Considere strings  $A = babb$  e  $B = bbc$

- ▶ podemos transformar  $A$  em  $B$



- ▶ realizamos duas operações

Observação:

- ▶ pode ser resolvido por programação dinâmica (exercicio)

# Exemplo

Considere strings  $A = babb$  e  $B = bbc$

- ▶ podemos transformar  $A$  em  $B$

$babb$   
↓      **remova a**  
 $bbb$   
↓      **troque o último b por c**  
 $bbc$

- ▶ realizamos duas operações

Observação:

- ▶ pode ser resolvido por programação dinâmica (exercicio)

# Exemplo

Considere strings  $A = babb$  e  $B = bbc$

- ▶ podemos transformar  $A$  em  $B$

$babb$   
↓      **remova a**  
 $bbb$   
↓      **troque** o último *b* por *c*  
 $bbc$

- ▶ realizamos duas operações

Observação:

- ▶ pode ser resolvido por programação dinâmica (exercicio)

# Exemplo

Considere strings  $A = babb$  e  $B = bbc$

- ▶ podemos transformar  $A$  em  $B$

$babb$   
↓      **remova a**  
 $bbb$   
↓      **troque** o último *b* por *c*  
 $bbc$

- ▶ realizamos duas operações

Observação:

- ▶ pode ser resolvido por programação dinâmica (exercicio)

## Exemplo

Considere strings  $A = babb$  e  $B = bbc$

- ▶ podemos transformar  $A$  em  $B$

$babb$   
↓      remova a  
 $bbb$   
↓      troque o último  $b$  por  $c$   
 $bbc$

- ▶ realizamos duas operações

Observação:

- ▶ pode ser resolvido por programação dinâmica (exercicio)

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho

# Problema de destino

## Problema do Caminho Mínimo

Entrada:

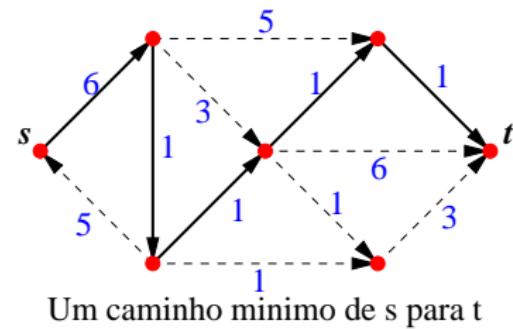
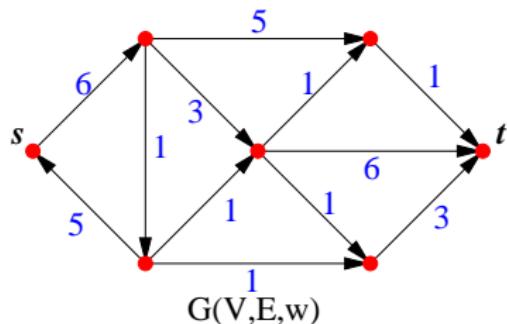
- ▶ grafo direcionado  $G(V, E)$
- ▶ peso  $c_{ij} \geq 0$  para cada aresta  $(i, j) \in E$
- ▶ vértices  $s$  e  $t$

Solução:

- ▶ caminho de  $s$  a  $t$  em  $G$

Objetivo:

- ▶ minimizar o comprimento do caminho



# Redução

## Redução:

# Redução

Redução:

1. Tome duas strings  $A = a_1 a_2 \dots a_n$  e  $B = b_1 b_2 \dots b_m$

2. Construa um grafo da seguinte forma:

3. Encontre um caminho mínimo de  $I$  até  $F$

# Redução

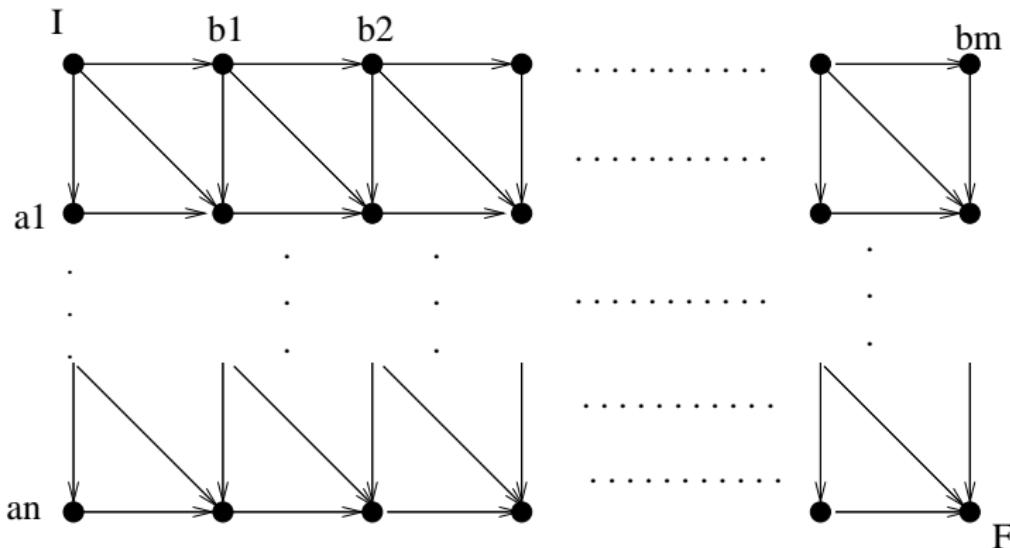
## Redução:

1. Tome duas strings  $A = a_1a_2\dots a_n$  e  $B = b_1b_2\dots b_m$
  2. Construa um grafo da seguinte forma:
  3. Encontre um caminho mínimo de  $I$  até  $F$

# Redução

Redução:

1. Tome duas strings  $A = a_1 a_2 \dots a_n$  e  $B = b_1 b_2 \dots b_m$
2. Construa um grafo da seguinte forma:

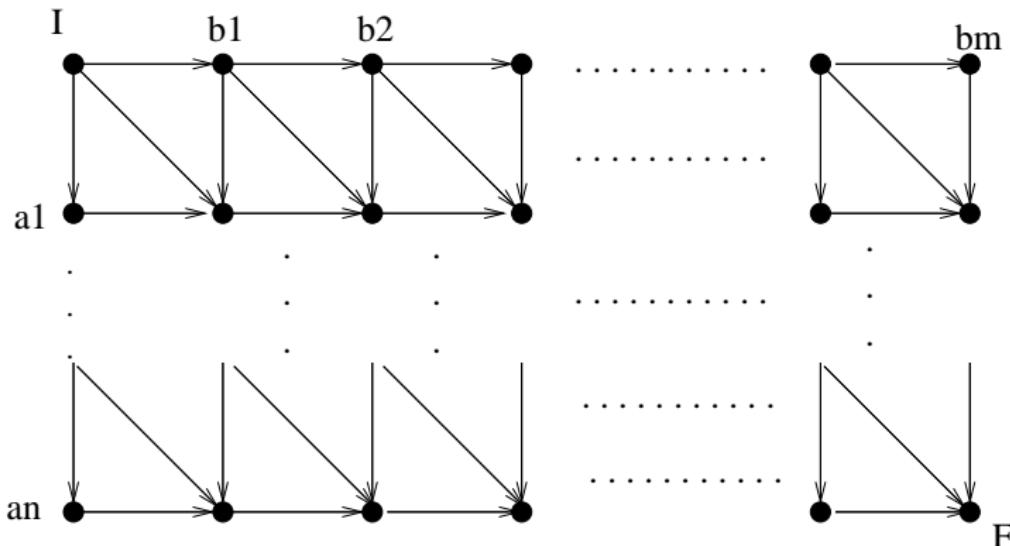


3. Encontre um caminho mínimo de  $I$  até  $F$

# Redução

Redução:

1. Tome duas strings  $A = a_1 a_2 \dots a_n$  e  $B = b_1 b_2 \dots b_m$
2. Construa um grafo da seguinte forma:



3. Encontre um caminho mínimo de  $I$  até  $F$

# Edição de String

4. Converta o caminho em operações de strings:
  - ▶ Arestras horizontais correspondem a inserção de um caractere e possuem custo 1.
  - ▶ Arestras verticais correspondem a remoção de um caractere e possuem custo 1.
  - ▶ Arestras diagonais correspondem a uma troca e tem custo 1 caso os caracteres sejam diferentes, e 0 caso sejam iguais.
5. Devolva a sequência de operações construída

# Edição de String

4. Converta o caminho em operações de strings:
  - ▶ Arestras horizontais correspondem a inserção de um caractere e possuem custo 1.
  - ▶ Arestras verticais correspondem a remoção de um caractere e possuem custo 1.
  - ▶ Arestras diagonais correspondem a uma troca e tem custo 1 caso os caracteres sejam diferentes, e 0 caso sejam iguais.
5. Devolva a sequência de operações construída

# Edição de String

4. Converta o caminho em operações de strings:
  - ▶ Arestras horizontais correspondem a inserção de um caractere e possuem custo 1.
  - ▶ Arestras verticais correspondem a remoção de um caractere e possuem custo 1.
  - ▶ Arestras diagonais correspondem a uma troca e tem custo 1 caso os caracteres sejam diferentes, e 0 caso sejam iguais.
5. Devolva a sequência de operações construída

# Edição de String

4. Converta o caminho em operações de strings:
  - ▶ Arestras horizontais correspondem a inserção de um caractere e possuem custo 1.
  - ▶ Arestras verticais correspondem a remoção de um caractere e possuem custo 1.
  - ▶ Arestras diagonais correspondem a uma troca e tem custo 1 caso os caracteres sejam diferentes, e 0 caso sejam iguais.
5. Devolva a sequência de operações construída

# Edição de String

4. Converta o caminho em operações de strings:
  - ▶ Arestras horizontais correspondem a inserção de um caractere e possuem custo 1.
  - ▶ Arestras verticais correspondem a remoção de um caractere e possuem custo 1.
  - ▶ Arestras diagonais correspondem a uma troca e tem custo 1 caso os caracteres sejam diferentes, e 0 caso sejam iguais.
5. Devolva a sequência de operações construída

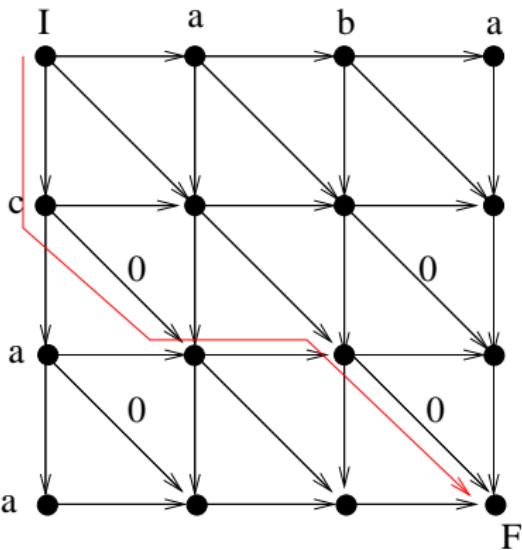
## Exemplo da redução

Considere strings  $A = caa$  e  $B = aba$  e o grafo  $G$

- ▶ há duas operações de custo não nulo
- ▶ o custo de edição corresponde a  $1 + 0 + 1 + 0 = 2$

# Exemplo da redução

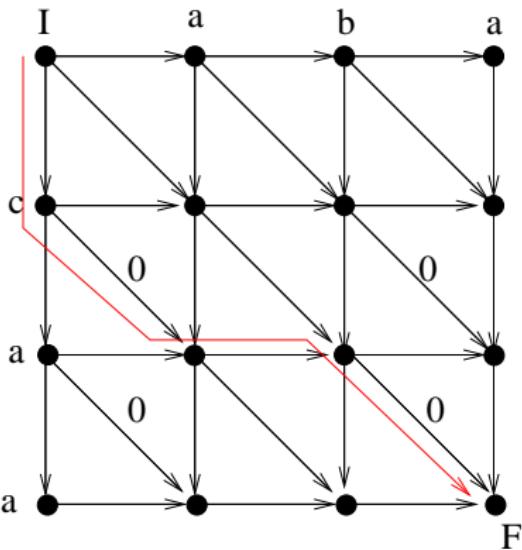
Considere strings  $A = caa$  e  $B = aba$  e o grafo  $G$



- ▶ há duas operações de custo não nulo
- ▶ o custo de edição corresponde a  $1 + 0 + 1 + 0 = 2$

# Exemplo da redução

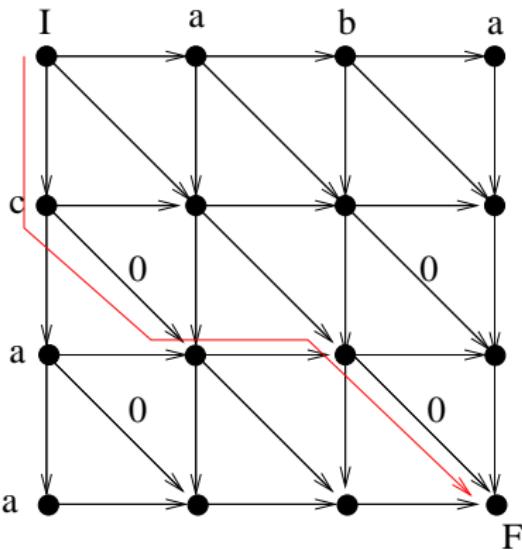
Considere strings  $A = caa$  e  $B = aba$  e o grafo  $G$



- ▶ há duas operações de custo não nulo
- ▶ o custo de edição corresponde a  $1 + 0 + 1 + 0 = 2$

# Exemplo da redução

Considere strings  $A = caa$  e  $B = aba$  e o grafo  $G$



- ▶ há duas operações de custo não nulo
- ▶ o custo de edição corresponde a  $1 + 0 + 1 + 0 = 2$

## Outros exemplos de reduções

- ▶ Ordenação para Codificação de Huffman

# Problema de origem

## Ordenação

Entrada:

- ▶ sequência de números naturais distintos  $x_1, x_2, \dots, x_n$

Saída:

- ▶ permutação dos números de entrada

# Problema de origem

## Ordenação

Entrada:

- ▶ sequência de números naturais distintos  $x_1, x_2, \dots, x_n$

Saída:

- ▶ permutação dos números de entrada

# Problema de origem

## Ordenação

Entrada:

- ▶ sequência de números naturais distintos  $x_1, x_2, \dots, x_n$

Saída:

- ▶ permutação dos números de entrada

# Problema de origem

## Ordenação

Entrada:

- ▶ sequência de números naturais distintos  $x_1, x_2, \dots, x_n$

Saída:

- ▶ permutação dos números de entrada

# Problema de origem

## Ordenação

Entrada:

- ▶ sequência de números naturais distintos  $x_1, x_2, \dots, x_n$

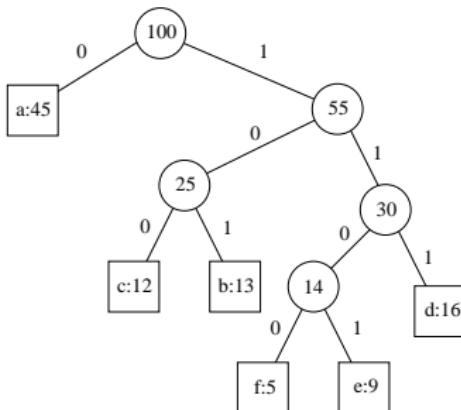
Saída:

- ▶ permutação dos números de entrada

# Árvore binária para codificação variável

Relembando: codificação de comprimento variável

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência	45	13	12	16	9	5
Código variável	0	101	100	111	1101	1100



# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ minimizar o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ **minimizar** o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ **minimizar** o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ **minimizar** o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Problema de destino

## Codificação de Huffman

- ▶ Entrada:
  - ▶ alfabeto  $C$
  - ▶ tabela de frequências  $f$
- ▶ Solução:
  - ▶ codificação de comprimento variável
- ▶ Objetivo:
  - ▶ **minimizar** o tamanho do texto codificado

## Observação

- ▶ já vimos um algoritmo  $O(n \log n)$  para o problema
- ▶ vamos utilizá-lo para resolver ordenação

# Lema útil

## Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

# Lema útil

## Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

# Lema útil

## Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

# Lema útil

## Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

## Lema útil

### Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

## Lema útil

### Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

## Lema útil

### Lema

Considere caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1, f_2, \dots, f_n$  tais que

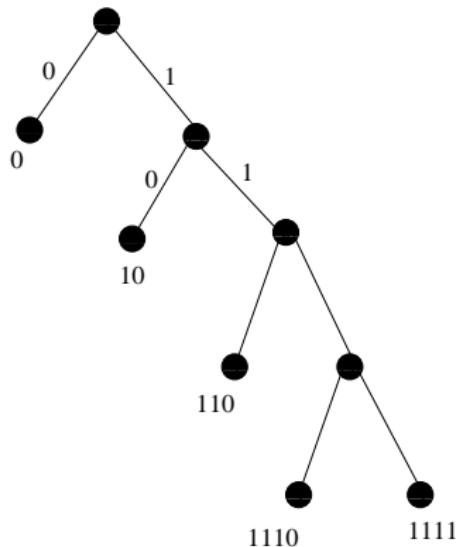
$$\sum_{i=1}^{j-1} f_i < f_j \text{ para cada } j = 2, \dots, n.$$

Então existe uma árvore de Huffman tal que

- ▶  $C_n$  é folha no nível 1,
- ▶  $C_{n-1}$  é folha no nível 2
- ▶ ...
- ▶  $C_2$  e  $C_1$  são folhas no nível  $n - 1$ .

## Exemplo de árvore

Instâncias baseadas no último teorema terão uma árvore de Huffman com a seguinte forma:



# Prova do lema

## Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_{C_n}$
  - ▶ e aumenta de no máximo  $\sum_{j=1}^{n-1} f_j$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_{C_n}$
  - ▶ e aumenta de no máximo  $\sum_{j=1}^{n-1} f_j$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_{C_n}$
  - ▶ e aumenta de no máximo  $\sum_{j=1}^{n-1} f_j$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{j=1}^{n-1} f_j$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Prova do lema

Demonstração:

- ▶ seja  $T^*$  uma árvore de Huffman para instância enunciada
- ▶ suponha por contradição que  $C_n$  não esteja no nível 1
- ▶ adicione uma nova raiz em  $T^*$ 
  - ▶ torne  $C_n$  filho esquerdo da nova raiz
  - ▶ e faça  $T^*$  filho direto dessa raiz
- ▶ o custo da árvore muda
  - ▶ ele diminuiu de pelo menos  $f_n$
  - ▶ e aumenta de no máximo  $\sum_{i=1}^{n-1} f_i$
- ▶ portanto o custo diminui
- ▶ isso é um absurdo porque  $T^*$  é codificação ótima
- ▶ podemos repetir o argumento para os demais caracteres

# Redução

1. Tome uma instância de ordenação  $x_1, x_2, \dots, x_n$ .
2. Crie caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1 = 2^{x_1}, f_2 = 2^{x_2}, \dots, f_n = 2^{x_n}$ .
3. Observe que para todo caractere  $j$  temos

$$\sum_{i=1}^{j-1} 2^i < 2^j$$

4. Encontre um codificação de huffman para a instância criada.
5. Percorra a árvore em ordem de níveis, listando as folhas percorridas.

# Redução

1. Tome uma instância de ordenação  $x_1, x_2, \dots, x_n$ .
2. Crie caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1 = 2^{x_1}, f_2 = 2^{x_2}, \dots, f_n = 2^{x_n}$ .
3. Observe que para todo caractere  $j$  temos

$$\sum_{i=1}^{j-1} 2^i < 2^j$$

4. Encontre um codificação de huffman para a instância criada.
5. Percorra a árvore em ordem de níveis, listando as folhas percorridas.

# Redução

1. Tome uma instância de ordenação  $x_1, x_2, \dots, x_n$ .
2. Crie caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1 = 2^{x_1}, f_2 = 2^{x_2}, \dots, f_n = 2^{x_n}$ .
3. Observe que para todo caractere  $j$  temos

$$\sum_{i=1}^{j-1} 2^i < 2^j$$

4. Encontre um codificação de huffman para a instância criada.
5. Percorra a árvore em ordem de níveis, listando as folhas percorridas.

# Redução

1. Tome uma instância de ordenação  $x_1, x_2, \dots, x_n$ .
2. Crie caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1 = 2^{x_1}, f_2 = 2^{x_2}, \dots, f_n = 2^{x_n}$ .
3. Observe que para todo caractere  $j$  temos

$$\sum_{i=1}^{j-1} 2^i < 2^j$$

4. Encontre um codificação de huffman para a instância criada.
5. Percorra a árvore em ordem de níveis, listando as folhas percorridas.

# Redução

1. Tome uma instância de ordenação  $x_1, x_2, \dots, x_n$ .
2. Crie caracteres  $C_1, C_2, \dots, C_n$  com frequências  $f_1 = 2^{x_1}, f_2 = 2^{x_2}, \dots, f_n = 2^{x_n}$ .
3. Observe que para todo caractere  $j$  temos

$$\sum_{i=1}^{j-1} 2^i < 2^j$$

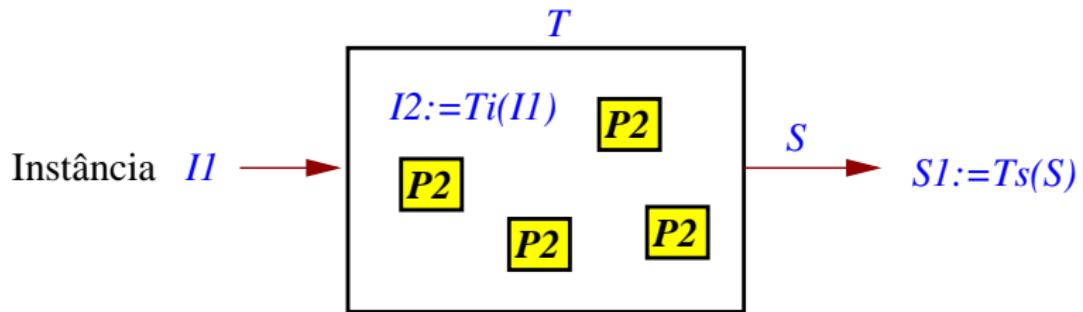
4. Encontre um codificação de huffman para a instância criada.
5. Percorra a árvore em ordem de níveis, listando as folhas percorridas.

## Outros exemplos de reduções

- ▶ Reduções de Turing

# Redução de Turing

Podemos reduzir  $P_1$  para  $P_2$  fazendo várias aplicações de  $P_2$ .



# Exemplo de redução de Turing

## Problema de Multiplicação de Inteiros

Dados inteiros  $a$  e  $b$ , calcular  $a \cdot b$ .

## Problema do Quadrado

Dados inteiro  $x$ , calcular  $x^2$ .

### Redução:

- ▶ Podemos reduzir Multiplicação de Inteiros para Quadrado.
- ▶ Fazemos apenas um número constante de somas, subtrações e divisão por dois

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

# Exemplo de redução de Turing

## Problema de Multiplicação de Inteiros

Dados inteiros  $a$  e  $b$ , calcular  $a \cdot b$ .

## Problema do Quadrado

Dados inteiro  $x$ , calcular  $x^2$ .

### Redução:

- ▶ Podemos reduzir Multiplicação de Inteiros para Quadrado.
- ▶ Fazemos apenas um número constante de somas, subtrações e divisão por dois

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

# Exemplo de redução de Turing

## Problema de Multiplicação de Inteiros

Dados inteiros  $a$  e  $b$ , calcular  $a \cdot b$ .

## Problema do Quadrado

Dados inteiro  $x$ , calcular  $x^2$ .

### Redução:

- ▶ Podemos reduzir Multiplicação de Inteiros para Quadrado.
- ▶ Fazemos apenas um número constante de somas, subtrações e divisão por dois

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

# Exemplo de redução de Turing

## Problema de Multiplicação de Inteiros

Dados inteiros  $a$  e  $b$ , calcular  $a \cdot b$ .

## Problema do Quadrado

Dados inteiro  $x$ , calcular  $x^2$ .

### Redução:

- ▶ Podemos reduzir Multiplicação de Inteiros para Quadrado.
- ▶ Fazemos apenas um número constante de somas, subtrações e divisão por dois

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

# Exemplo de redução de Turing

## Problema de Multiplicação de Inteiros

Dados inteiros  $a$  e  $b$ , calcular  $a \cdot b$ .

## Problema do Quadrado

Dados inteiro  $x$ , calcular  $x^2$ .

### Redução:

- ▶ Podemos reduzir Multiplicação de Inteiros para Quadrado.
- ▶ Fazemos apenas um número constante de somas, subtrações e divisão por dois

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

# Exemplo de redução de Turing

## Problema de Multiplicação de Inteiros

Dados inteiros  $a$  e  $b$ , calcular  $a \cdot b$ .

## Problema do Quadrado

Dados inteiro  $x$ , calcular  $x^2$ .

### Redução:

- ▶ Podemos reduzir Multiplicação de Inteiros para Quadrado.
- ▶ Fazemos apenas um número constante de somas, subtrações e divisão por dois

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$