Projeto e Análise de Algoritmos

Algoritmos gulosos

Lehilton Pedrosa

Primeiro Semestre de 2020

Algoritmos gulosos

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- algoritmo de programação dinâmica:
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- algoritmo guloso:
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- decompomos um problema em vários subproblemas
- de novo, um deles corresponde à subestrutura ótima
- mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- **algoritmo de programação dinâmica:**
  1. primeiro resolvemos todos os subproblemas
  2. depois decidimos o subproblema ótimo

- **algoritmo guloso:**
  1. primeiro escolhemos o subproblema ótimo
  2. depois resolvemos apenas esse subproblema

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Algoritmos gulosos

Ideia:

- não resolver todos os subproblemas
- fazer uma escolha gulosa por um deles
- resolver apenas o subproblema escolhido

Premissas dos algoritmos gulosos:

- existe um critério guloso para selecionar o subproblema corresponde à subestrutura ótima
- sabemos executar esse critério eficientemente

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas tem a seguinte estrutura

- existe um conjunto de elementos S
- uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução:
   (a) escolha um elemento e com algum critério guloso
   (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
   (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Algoritmos gulosos

- Seleção de atividades

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- podem ser palestras, reuniões em um sala etc.
- denote essas atividades por $S = \{a_1, \ldots, a_n\}$

Duração das atividades

- a atividade a_i começa no tempo s_i e termina no tempo f_i
- assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são compatíveis se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Problema de seleção de atividades

Problema:

- Entrada:
  - conjunto de atividades $S = \{a_1, \ldots, a_n\}$
  - tempos de início s e de termino f
- Solução:
  - subconjunto A de atividades compatíveis
- Objetivo:
  - maximizar $|A|$

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- as atividades estão em ordem de tempo de término
- iremos usar essa ordem em seguida

Exemplos de soluções

- $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \ldots \leq f_n$

- ou seja, as atividades estão em ordem de término
- se não estiverem, podemos ordenar

Para um par (i, j), defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- atividades que começam depois que a_i termina
- e além disso terminam antes que a_j inicie

Considere atividades dummies

- atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- assim S_{ij} está definido para todo $0 \leq i, j \leq n + 1$
- e o conjunto de todas atividades é $S = S_{0,n+1}$

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima

- suponha que a_k está nessa solução ótima
- as demais atividades da solução devem estar
  - antes do início de a_k
  - depois do término de a_k

Descobrimos uma subestrutura ótima

- queremos uma solução ótima para o conjunto S_{0k}
- além de uma solução ótima para o conjunto $S_{k,n+1}$

Mas não sabemos qual é a atividade a_k na solução ótima!

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Usando programação dinâmica

Definimos o seguinte subproblema

- considere um par (i, j) para $0 \leq i, j \leq n + 1$
- defina $c[i, j]$ o valor de uma solução ótima para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i,j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max\limits_{i<k<j : a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- o valor ótimo corresponde à entrada $c[0, n + 1]$
- é fácil calcular usando programação dinâmica (exercício)

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Escolha gulosa

Algumas observações

- na programação dinâmica, testamos todas atividades a_k
- em um algoritmo guloso, queremos escolher uma apenas

Escolha gulosa

Suponha que $S_{ij} \neq \emptyset$ e seja a_m uma atividade em S_{ij} com

$$f_m = \min\{f_k : a_k \in S_{ij}\}.$$

Então existe uma solução ótima para S_{ij} que contém a_m.

Algumas consequências

- como as atividades estão em ordem de término, $a_m = a_i$
- então o conjunto S_{im} é vazio
- e resta resolver apenas um subproblema para S_{mj}

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma solução ótima para S_{ij}

- se $a_m \in A$, então nada há a fazer
- suponha então que $a_m \notin A$

Vamos criar outra solução ótima A' contendo a_m

- seja $a_k \in A$ a atividade com menor f_k
- defina $A' = A \setminus \{a_k\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- é claro que $|A^*| = |A'|$
- então resta mostrar que A' é viável
- nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_k
- daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- assim as atividade de A' são mutualmente compatíveis

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- suponha que estamos tentando resolver S_{ij}
- escolha a atividade a_m com menor término em S_{ij}
- resolva o subproblema S_{mj} e junte a_m

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- suponha que estamos tentando resolver S_{ij}
- escolha a atividade a_m com menor término em S_{ij}
- resolva o subproblema S_{mj} e junte a_m

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- suponha que estamos tentando resolver S_{ij}
- escolha a atividade a_m com menor término em S_{ij}
- resolva o subproblema S_{mj} e junte a_m

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- suponha que estamos tentando resolver S_{ij}
- escolha a atividade a_m com menor término em S_{ij}
- resolva o subproblema S_{mj} e junte a_m

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1  m ← i + 1
2  enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3      m ← m + 1
4  se m ≤ j então
5      devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6  senão
7      devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1  m ← i + 1
2  enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3      m ← m + 1
4  se m ≤ j então
5      devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6  senão
7      devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1  m ← i + 1
2  enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3      m ← m + 1
4  se m ≤ j então
5      devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6  senão
7      devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1   m ← i + 1
2   enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3       m ← m + 1
4   se m ≤ j então
5       devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6   senão
7       devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1   m ← i + 1
2   enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3       m ← m + 1
4   se m ≤ j então
5       devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6   senão
7       devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1   m ← i + 1
2   enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3       m ← m + 1
4   se m ≤ j então
5       devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6   senão
7       devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo recursivo

Observações

- as atividades estão em ordem de tempo de término
- então $S_{ij} = \{a_{i+1}, a_{i+2}, \ldots, a_{j-1}\}$

```
SELEC-ATIV-GUL-REC(s, f, i, j)
1   m ← i + 1
2   enquanto m < j e s_m < f_i faça      ▷ acha primeira em S_ij
3       m ← m + 1
4   se m ≤ j então
5       devolva {a_m} ∪ SELEC-ATIV-GUL-REC(s, f, m, j)
6   senão
7       devolva ∅
```

Análise

- cada elemento é visto uma vez, daí o tempo total é $\Theta(n)$
- pode ser que precisemos ordenar as atividades

Algoritmo iterativo

Podemos reescrever de maneira iterativa

```
SELEC-ATIV-GUL-ITER(s, f, n)
1  A ← {a_1}
2  i ← 1
3  para m ← 2 até n faça
4      se s_m ≥ f_i então
5          A ← A ∪ {a_m}
6          i ← m
7  devolva A.
```

Algoritmo iterativo

Podemos reescrever de maneira iterativa

```
SELEC-ATIV-GUL-ITER(s, f, n)
1   A ← {a_1}
2   i ← 1
3   para m ← 2 até n faça
4       se s_m ≥ f_i então
5           A ← A ∪ {a_m}
6           i ← m
7   devolva A.
```

Algoritmos gulosos

- Codificação de Huffman

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um texto

- é uma sequência de caracteres de um alfabeto C
- cada caractere está associado a uma sequência de bits

Tipos de codificação

- comprimento fixo
  - cada sequência tem o mesmo número de bits
  - basta que elas sejam distintas
- codificação de comprimento variável
  - as sequências podem ter tamanhos diferentes
  - são livres de prefixo: uma sequência não é prefixo de outra

Tamanho do texto codificado

- é o número de bits usados para representar o texto
- codificações diferentes têm tamanhos diferentes

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- com a codificação de tamanho variável, usamos

$$(\underbrace{45 \cdot 1}_{a} + \underbrace{13 \cdot 3}_{b} + \underbrace{12 \cdot 3}_{c} + \underbrace{16 \cdot 3}_{d} + \underbrace{9 \cdot 4}_{e} + \underbrace{5 \cdot 4}_{f}) \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- com a codificação de tamanho variável, usamos

$$(\underbrace{45 \cdot 1}_{a} + \underbrace{13 \cdot 3}_{b} + \underbrace{12 \cdot 3}_{c} + \underbrace{16 \cdot 3}_{d} + \underbrace{9 \cdot 4}_{e} + \underbrace{5 \cdot 4}_{f}) \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- com a codificação de tamanho variável, usamos

$$(\underbrace{45 \cdot 1}_{a} + \underbrace{13 \cdot 3}_{b} + \underbrace{12 \cdot 3}_{c} + \underbrace{16 \cdot 3}_{d} + \underbrace{9 \cdot 4}_{e} + \underbrace{5 \cdot 4}_{f}) \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- com a codificação de tamanho variável, usamos

$$(\underbrace{45 \cdot 1}_{a} + \underbrace{13 \cdot 3}_{b} + \underbrace{12 \cdot 3}_{c} + \underbrace{16 \cdot 3}_{d} + \underbrace{9 \cdot 4}_{e} + \underbrace{5 \cdot 4}_{f}) \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- com a codificação de tamanho variável, usamos

$$(\underbrace{45 \cdot 1}_{a} + \underbrace{13 \cdot 3}_{b} + \underbrace{12 \cdot 3}_{c} + \underbrace{16 \cdot 3}_{d} + \underbrace{9 \cdot 4}_{e} + \underbrace{5 \cdot 4}_{f}) \cdot 1.000 = 224.000 \text{ bits}$$

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- problema para a compressão de dados
- dependendo da aplicação, reduz de 20 a 90%

Problema

- Entrada:
  - alfabeto C
  - tabela de frequências f
- Solução:
  - codificação de comprimento variável
- Objetivo:
  - minimizar o tamanho do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Representação de codificação

Uma codificação é representada por uma árvore binária

- o filho esquerdo está associado ao **bit** 0
- o filho direito está associado ao **bit** 1
- as folhas representam os caracteres do alfabeto

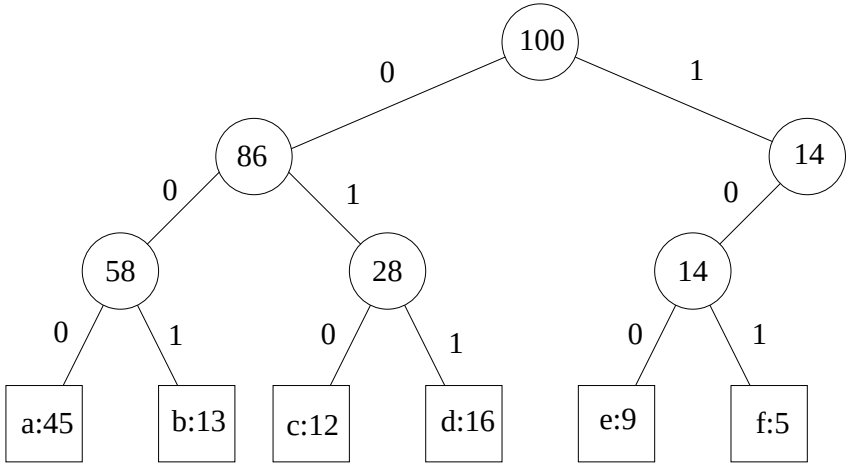
Vantagens dessa estrutura de dados

- representam codificações variáveis livres de contexto
- permite fácil decodificação do texto codificado

Arvore binária para codificação fixa

Codificação de comprimento fixo

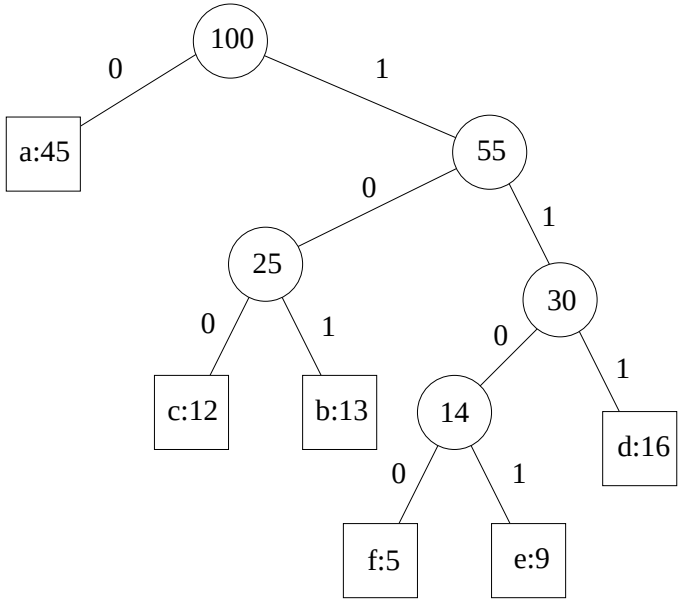
	a	*b*	*c*	*d*	*e*	*f*
Frequência	45	13	12	16	9	5
Código fixo	000	001	010	011	100	101



Arvore binária para codificação variável

Codificação de comprimento variável

	a	*b*	*c*	*d*	*e*	*f*
Frequência	45	13	12	16	9	5
Código variável	0	101	100	111	1101	1100



Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Detalhes da estrutura

Arvores binárias cheias

- são árvores em que cada nó interno tem dois filhos
- assim, há $|C|$ folhas e $|C| - 1$ nós internos
- sempre existe uma codificação ótima cheia (exercício)

Estrutura de um nó z

- o filho esquerdo é denotado por $z.esq$
- o filho direito é denotado por $z.dir$
- a frequência dos caracteres na subárvore é $z.freq$

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- caracteres infrequentes estão em folhas mais profundas
- construímos a árvore de maneira bottom-up

Ideia:

- começar com $|C|$ nós correspondendo aos caracteres
- juntar os dois nós menos frequentes
- repetir $|C| - 1$ vezes até sobrar apenas um nó
- o nó restante é a raiz da árvore devolvida

Algoritmo de Huffman

```
HUFFMAN(C)
1   n ← |C|
2   Q ← C
3   para i ← 1 até n − 1 faça
4       alocar novo registro z
5       z.esq ← x ← EXTRACT-MIN(Q)
6       z.dir ← y ← EXTRACT-MIN(Q)
7       z.freq ← x.freq + y.freq
8       INSERT(Q, z)
9   devolva EXTRACT-MIN(Q)
```

- Q é uma fila de prioridades ordenada pela frequência

Algoritmo de Huffman

```
HUFFMAN(C)
1   n ← |C|
2   Q ← C
3   para i ← 1 até n − 1 faça
4       alocar novo registro z
5       z.esq ← x ← EXTRACT-MIN(Q)
6       z.dir ← y ← EXTRACT-MIN(Q)
7       z.freq ← x.freq + y.freq
8       INSERT(Q, z)
9   devolva EXTRACT-MIN(Q)
```

- Q é uma fila de prioridades ordenada pela frequência

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- T é uma árvore binária representando uma codificação
- $d_T(c)$ é a profundidade do nó representado o caractere c
- $f(c)$ é a sua frequência do caractere c

Tamanho do texto codificado

- o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- dizemos que $B(T)$ é o custo de T

Escolha gulosa

Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa

Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa

Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa

Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa

Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa

Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa

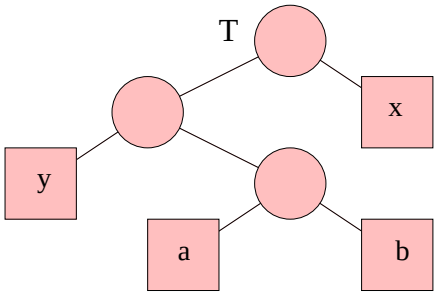
Escolha gulosa

Sejam x e y os dois caracteres em C com as menores frequências. Então, existe uma codificação ótima na qual os códigos de x e y diferem apenas no último bit.

Demonstração

- Considere uma árvore ótima T
- Sejam a e b duas folhas irmãs mais profundas
- Também, sejam x e y as duas folhas de menor frequência
- Construa uma árvore T' trocando a e x e depois uma árvore T'' trocando b por y
- Vamos mostrar que T'' também é uma árvore ótima

Escolha gulosa (cont)

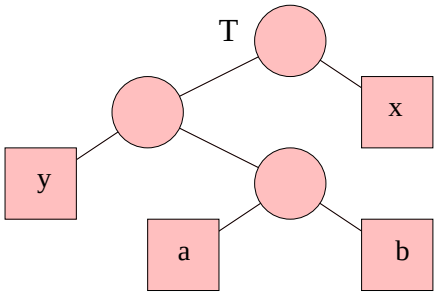


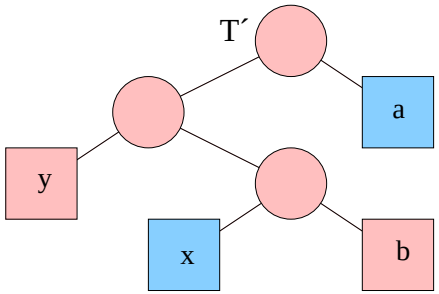
- sando a definição de B e cancelando termos idênticos,

$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a)$$
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x)$$
$$= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)



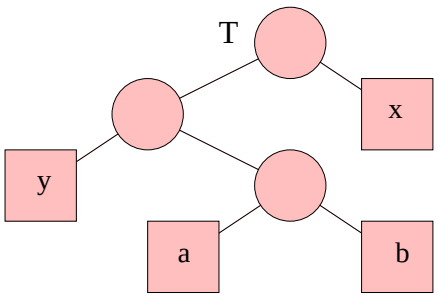


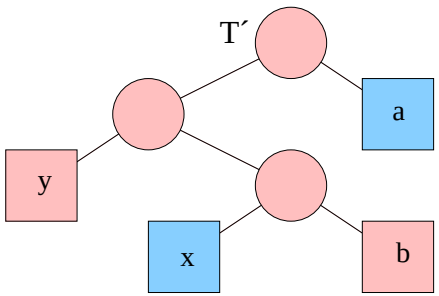
- sando a definição de B e cancelando termos idênticos,

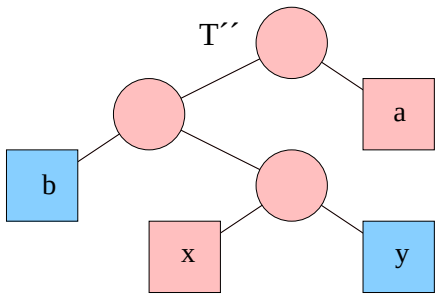
$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a)$$
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x)$$
$$= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)







- sando a definição de B e cancelando termos idênticos,

$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$

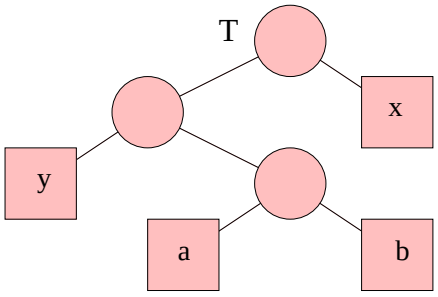
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a)$$

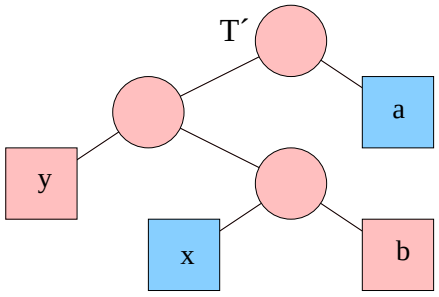
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x)$$

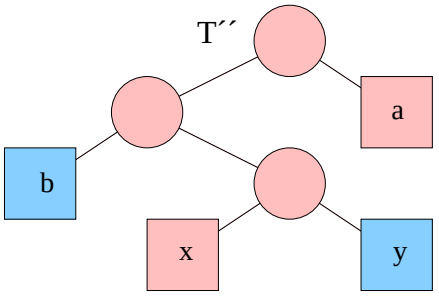
$$= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)





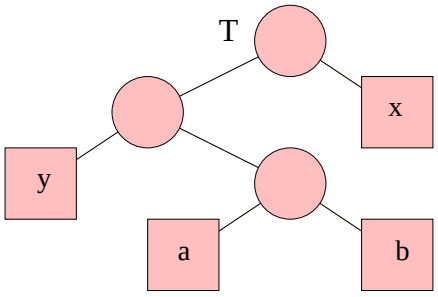


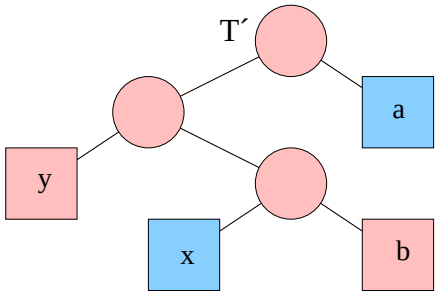
- sando a definição de B e cancelando termos idênticos,

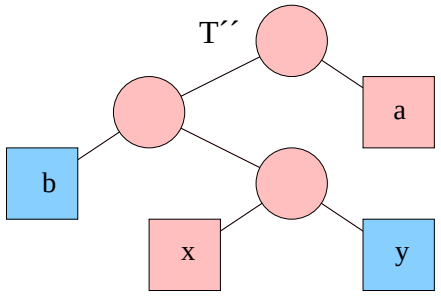
$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\
&= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\
&= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0
\end{aligned}
$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)







- sando a definição de B e cancelando termos idênticos,

$$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$

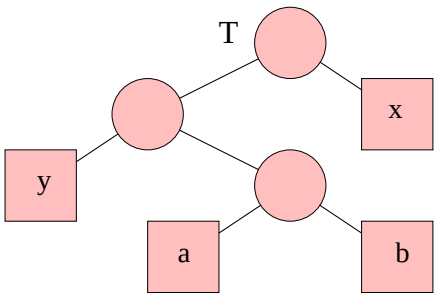
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a)$$

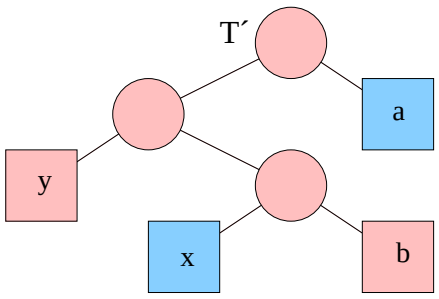
$$= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x)$$

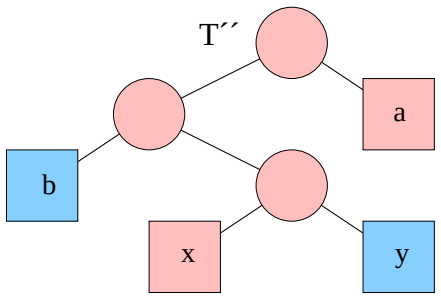
$$= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)





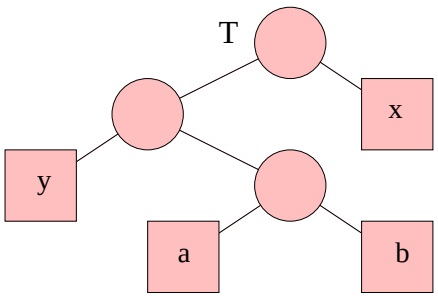


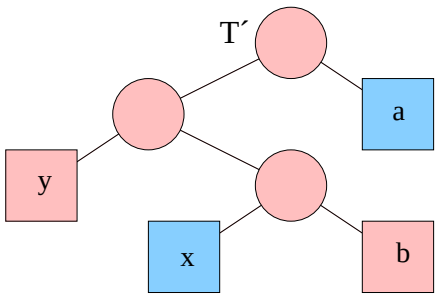
- sando a definição de B e cancelando termos idênticos,

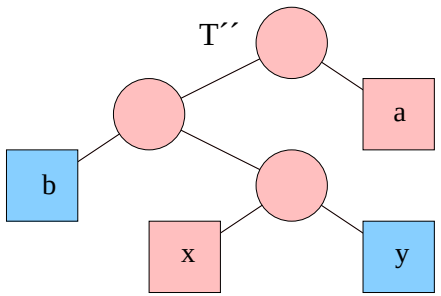
$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)





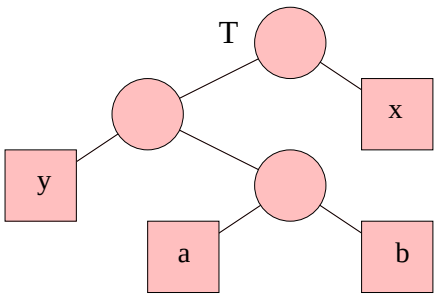


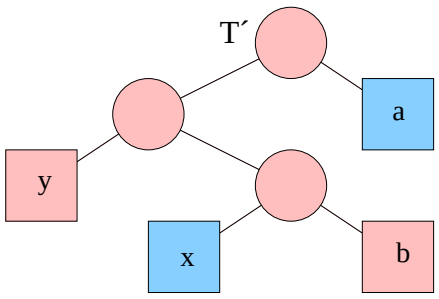
- sando a definição de B e cancelando termos idênticos,

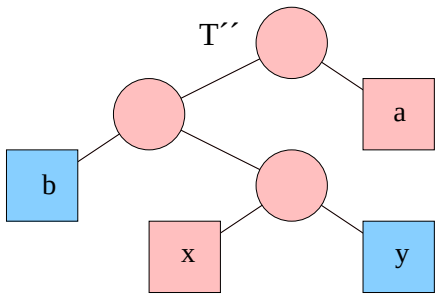
$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)





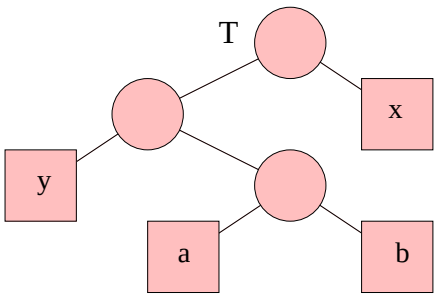


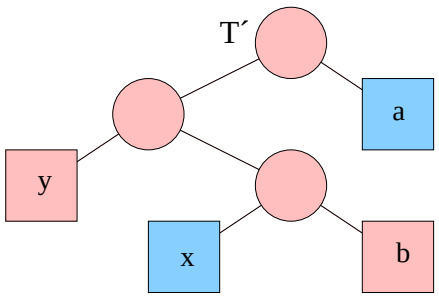
- sando a definição de B e cancelando termos idênticos,

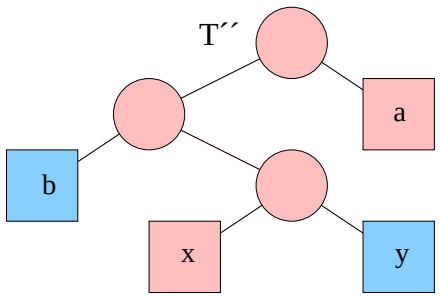
$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)





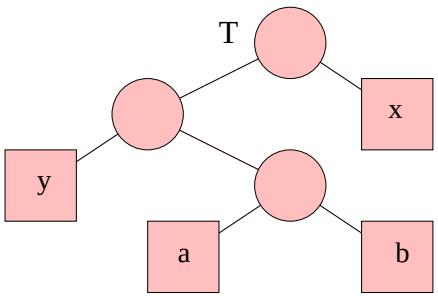


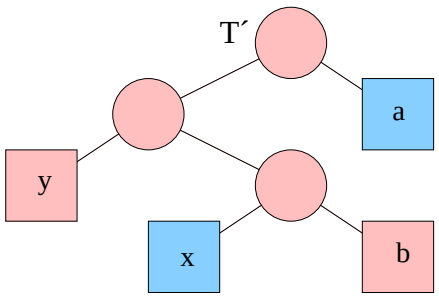
- sando a definição de B e cancelando termos idênticos,

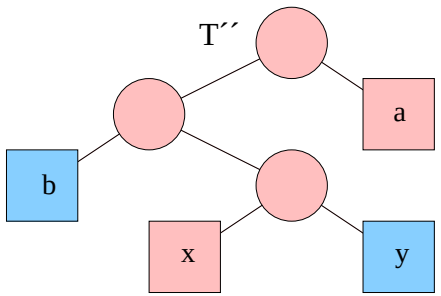
$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)







- sando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- como T é ótima, T'' também é ótima e o enunciado segue

Subestrutura ótima

Subestrutura ótima

Suponha que

- x e y são os dois caracteres com menores frequências
- $C' = C \cup \{z\} - \{x, y\}$ é alfabeto com $f(z) = f(x) + f(y)$
- T' é uma árvore ótima para o alfabeto C'
- T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C.

Subestrutura ótima

Subestrutura ótima

Suponha que

- x e y são os dois caracteres com menores frequências
- $C' = C \cup \{z\} - \{x, y\}$ é alfabeto com $f(z) = f(x) + f(y)$
- T' é uma árvore ótima para o alfabeto C'
- T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C.

Subestrutura ótima

Subestrutura ótima

Suponha que

- x e y são os dois caracteres com menores frequências
- $C' = C \cup \{z\} - \{x, y\}$ é alfabeto com $f(z) = f(x) + f(y)$
- T' é uma árvore ótima para o alfabeto C'
- T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C.

Subestrutura ótima

Subestrutura ótima

Suponha que

- x e y são os dois caracteres com menores frequências
- $C' = C \cup \{z\} - \{x, y\}$ é alfabeto com $f(z) = f(x) + f(y)$
- T' é uma árvore ótima para o alfabeto C'
- T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C.

Subestrutura ótima

Subestrutura ótima

Suponha que

- x e y são os dois caracteres com menores frequências
- $C' = C \cup \{z\} - \{x, y\}$ é alfabeto com $f(z) = f(x) + f(y)$
- T' é uma árvore ótima para o alfabeto C'
- T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C.

Subestrutura ótima

Subestrutura ótima

Suponha que

- x e y são os dois caracteres com menores frequências
- $C' = C \cup \{z\} - \{x, y\}$ é alfabeto com $f(z) = f(x) + f(y)$
- T' é uma árvore ótima para o alfabeto C'
- T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C.

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Subestrutura ótima (cont)

Comparando os custos de T e T'

- se $c \in C - \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- para x e y temos

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- portanto, $B(T') = B(T) - f(x) - f(y)$

Suponha que existe árvore ótima T'' para C com $B(T'') < B(T)$

- pelo lema anterior, suponha que x e y são folhas irmãs em T''
- seja T''' obtida de T'' trocando x e y por z com $f(z) = f(x) + f(y)$
- então o custo de T''' é tal que

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- isso contradiz o fato de que T' é uma árvore ótima para C'

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Segue imediatamente dos Lemas 1 e 2.

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Segue imediatamente dos Lemas 1 e 2.