

Algoritmos Parametrizados

Decomposição em Árvore

Lehilton Pedrosa

Segundo Semestre de 2016

Instituto de Computação – Unicamp

1. Decomposição em Árvore
2. Programação Dinâmica com largura de árvore definidas
3. Teorema de Courcelle
4. Caracterizações Alternativas
5. Obtendo uma decomposição em árvore
6. Shifting

Decomposição em Árvore

Pergunta: Quão parecido um grafo G é de uma árvore?

Pergunta: Quão parecido um grafo G é de uma árvore?

Possibilidades:

- Quantas círculos eu preciso quebrar para torná-lo acíclico?

Pergunta: Quão parecido um grafo G é de uma árvore?

Possibilidades:

- Quantos círculos eu preciso quebrar para torná-lo acíclico?
- Quantos vértices eu preciso remover?

Pergunta: Quão parecido um grafo G é de uma árvore?

Possibilidades:

- Quantas círculos eu preciso quebrar para torná-lo acíclico?
- Quantos vértices eu preciso remover?

Alternativa: como **decompor** o grafo em nós de uma árvore?

Motivação: vários problemas são mais fáceis em árvores

Motivação: vários problemas são mais fáceis em árvores

Ideia:

- Tentamos obter uma decomposição “pequena” do grafo

Motivação: vários problemas são mais fáceis em árvores

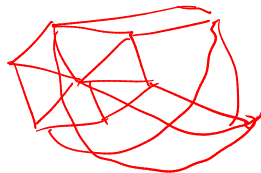
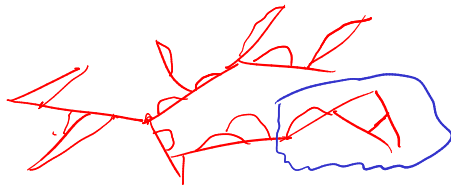
Ideia:

- Tentamos obter uma decomposição “pequena” do grafo
 - (a) Se conseguirmos, então usamos um algoritmo parecido com o de árvore

Motivação: vários problemas são mais fáceis em árvores

Ideia:

- Tentamos obter uma decomposição “pequena” do grafo
 - (a) Se conseguirmos, então usamos um algoritmo parecido com o de árvore
 - (b) Se não conseguirmos, então descobrimos uma estrutura forte sobre o grafo



Motivação: vários problemas são mais fáceis em árvores

Ideia:

- Tentamos obter uma decomposição “pequena” do grafo
 - (a) Se conseguirmos, então usamos um algoritmo parecido com o de árvore
 - (b) Se não conseguirmos, então descobrimos uma estrutura forte sobre o grafo

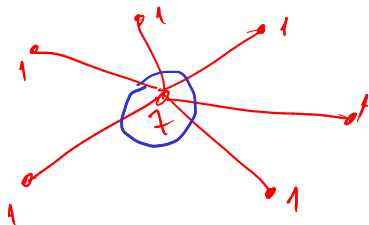
e.g.: instâncias sim de Cobertura por vértices e Conjuntos de retroalimentação têm decomposições pequenas

Conjunto independente máximo

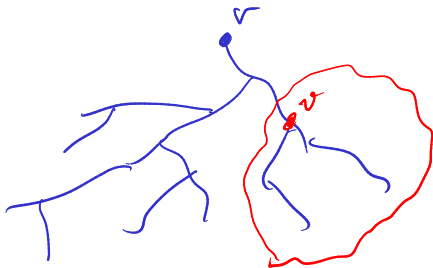
Problema do Cobertura Independente Máximo com pesos

Dado um grafo G e uma função com pesos nas arestas $w : V(G) \rightarrow \mathbb{R}_+$, queremos encontrar um conjunto independente $I \subseteq V(G)$ que maximize

$$\sum_{v \in I} w(v).$$



Ideia: o problema é muito fácil em árvore T



Programação Dinâmica

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

- Enraíze T em um vértice $r \in V(G)$ qualquer

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

- Enraíze T em um vértice $r \in V(G)$ qualquer
- Seja T_v a subárvore com raiz v

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

- Enraíze T em um vértice $r \in V(G)$ qualquer
- Seja T_v a subárvore com raiz v
- Consideramos recursivamente cada subproblema para T_v de maneira *bottom-up*

Programação Dinâmica

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

- Enraíze T em um vértice $r \in V(G)$ qualquer
- Seja T_v a subárvore com raiz v
- Consideramos recursivamente cada subproblema para T_v de maneira *bottom-up*

Tabela: seja v um vértice com $N(v) = \{v_1, \dots, v_q\}$:

Programação Dinâmica

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

- Enraíze T em um vértice $r \in V(G)$ qualquer
- Seja T_v a subárvore com raiz v
- Consideramos recursivamente cada subproblema para T_v de maneira *bottom-up*

Tabela: seja v um vértice com ~~$N(v)$~~ ^{Filhos} $= \{v_1, \dots, v_q\}$:

- $B[v] = \sum_{i=1}^q A[v_i]$

Programação Dinâmica

Ideia: o problema é muito fácil em árvore T

Programação Dinâmica:

- Enraíze T em um vértice $r \in V(G)$ qualquer
- Seja T_v a subárvore com raiz v
- Consideramos recursivamente cada subproblema para T_v de maneira *bottom-up*

Tabela: seja v um vértice com $N(v) = \{v_1, \dots, v_q\}$:

- $B[v] = \sum_{i=1}^q A[v_i]$
- $A[v] = \max\{B[v], w(v) + \sum_{i=1}^q B[v_i]\}$

$A[v] =$ Valor do *conjunto* ind. de peso máximo em T_v

$B[v] =$ Valor do *conjunto* ind. de peso máximo em $T_v \setminus \{v\}$

Paso base : v é folha

$$A[v] = w(v)$$

$$B[v] = 0$$

Paso geral : v tem filhos

Seja I uma sol de mala máximo em T_v .

a) Se $v \in I \Rightarrow$ então $v_1, \dots, v_g \notin I$ (estão em I)

$$\Rightarrow w(I) = w(v) + \sum_{i=1}^g \sum_{u \text{ filho de } v_i} A[u]$$

$$= w(v) + \sum_{i=1}^g B(v) \leq A[v].$$

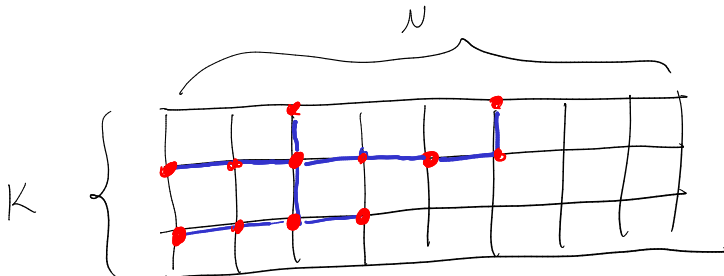
Tempo : Visitamos cada vértice uma vez.

• Em cada vértice v , fazemos $d(v)$ lookups

$$T_{OTAL} = \sum_{v \in V} (d(v) + 1) = |V| + |E| = \mathcal{O}(n).$$

Engrossando a árvore

Vamos considerar o problema restrito a **subgrafos** de uma grade $k \times N$:



Vamos considerar o problema restrito a **subgrafos** de uma grade $k \times N$:

Defina:

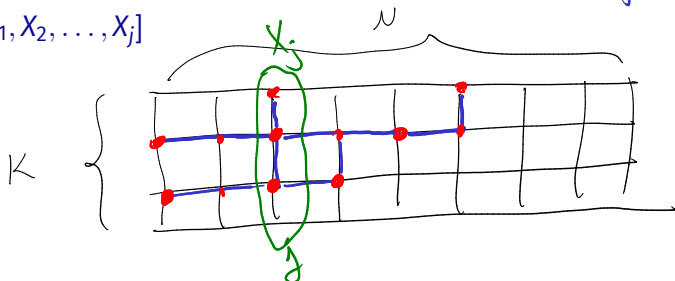
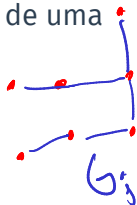
- X_j os vértices que estão na coluna j

Engrossando a árvore

Vamos considerar o problema restrito a **subgrafos** de uma grade $k \times N$:

Defina:

- X_j os vértices que estão na coluna j
- $G_j = G[X_1, X_2, \dots, X_j]$

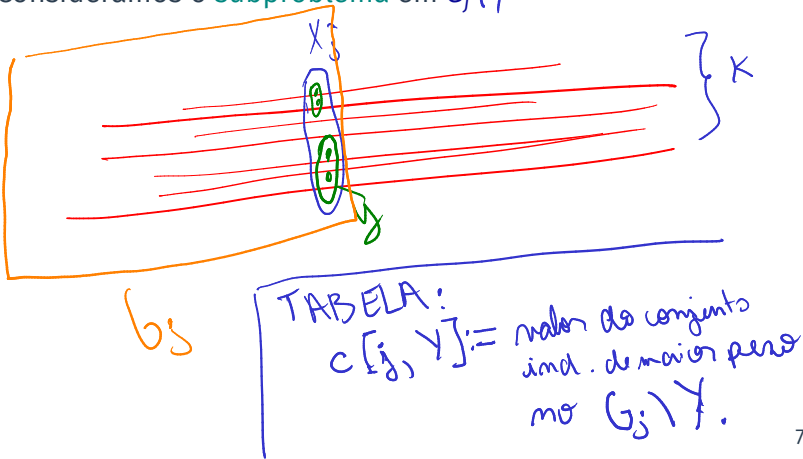


Programação Dinâmica na Grade

- Para cada j , enumeramos um conjunto que $Y \subseteq X_j$ que **não** está na solução

Programação Dinâmica na Grade

- Para cada j , enumeramos um conjunto que $Y \subseteq X_j$ que **não** está na solução
- Consideramos o subproblema em $G_j \setminus Y$



Calculando caso base

Caso: $c[1, Y]$

- Basta enumerar a solução $S \subseteq X_1 \setminus Y$

Fixa Y . Enumeramos $2^{|X_1| - |Y|}$ para Y .

Para todos Y . Quantos enumeramos

no total?

$$\sum_{Y \subseteq X_1} 2^{|X_1| - |Y|} = \sum_{d=0}^{|X_1|} \binom{|X_1|}{d} \cdot 2^{|X_1| - d} = 3^{|X_1|} = \Theta(3^k).$$

Calculando caso geral

Caso: $c[j, Y]$, com $j \geq 2$

- Enumeramos a parte da solução $S \subseteq X_j \setminus Y$ na coluna j

Calculando caso geral

Caso: $c[j, Y]$, com $j \geq 2$

- Enumeramos a parte da solução $S \subseteq X_j \setminus Y$ na coluna j
- Isso induz um subproblema em G_{j-1}



Fixe Y . Enumere $S \subseteq X_j \setminus Y$. Então, $Z = X_j \setminus (Y \cup S)$ faz parte da solução. Concluímos que $Y' = N(Z) \cap X_{j-1}$. $n̄n$ pode fazer parte da solução. Basta agora encontrar o c_{j-1} in a . ou valor máximo $G_{j-1} \setminus Y'$.

Algoritmo (j, Y)

0) $c(j, Y) \leftarrow 0$

1) Enumera $S \in X_j \setminus Y$.

2) $S, S \bar{m}$ for independente, tente próximo.

3) $c \leftarrow w(z) + c(j-1, Y')$

4) $S, c > c(j, Y)$

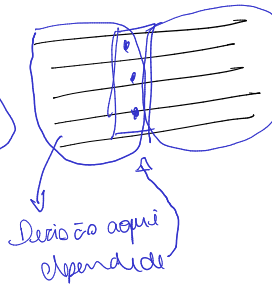
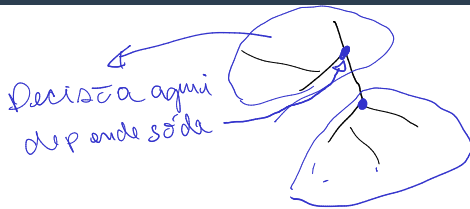
$c(j, Y) \leftarrow c$.

Tempo para coluna j : $k^{O(n)} 3^{|X_j|} \in 3^k k^{O(n)}$

Tempo total = $O\left(\sum_{j=1}^N 3^k\right) = O\left(3^k k^{O(n)}\right)$ D

Entendendo: separadores

Fato crucial



Fato crucial

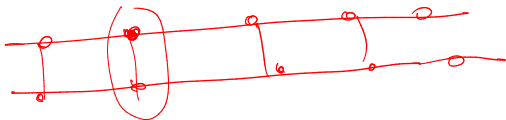
- X_{j-1} separa X_j do grafo G_{j-2}

Fato crucial

- X_{j-1} separa X_j do grafo G_{j-2}

Generalizando

Entendendo: separadores



Fato crucial

- X_{j-1} separa X_j do grafo G_{j-2}

Generalizando

- grades de altura k são apenas caminhas mais “gordos”

Fato crucial

- X_{j-1} separa X_j do grafo G_{j-2}

Generalizando

- grades de altura k são apenas caminhas mais “gordos”
- bastar obter uma sequência de separadores

Definição (Decomposição em Caminho)

Uma **decomposição em caminho** de um grafo⁶ é uma sequência $\mathcal{P} = (X_1, X_2, \dots, X_r)$ de **bags** (sacos), onde $X_i \subseteq V(G)$ para cada $i \in \{1, 2, \dots, r\}$, tal que vale:

Decomposição em caminho

Definição (Decomposição em Caminho)

Uma **decomposição em caminho** de um grafo é uma sequência $\mathcal{P} = (X_1, X_2, \dots, X_r)$ de **bags** (sacos), onde $X_i \subseteq V(G)$ para cada $i \in \{1, 2, \dots, r\}$, tal que vale:

$$(P1) \bigcup_{i=1}^r X_i = V(G);$$

Decomposição em caminho

Definição (Decomposição em Caminho)

Uma **decomposição em caminho** de um grafo é uma sequência $\mathcal{P} = (X_1, X_2, \dots, X_r)$ de **bags** (sacos), onde $X_i \subseteq V(G)$ para cada $i \in \{1, 2, \dots, r\}$, tal que vale:

$$(P1) \bigcup_{i=1}^r X_i = V(G);$$

(P2) Para cada $uv \in E(G)$, existe $\ell \in \{1, 2, \dots, r\}$ tal que X_ℓ contém u e v ;

Decomposição em caminho

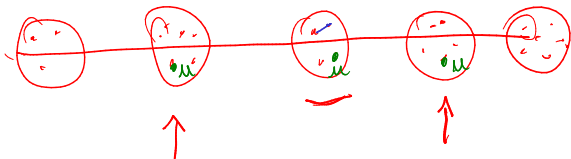
Definição (Decomposição em Caminho)

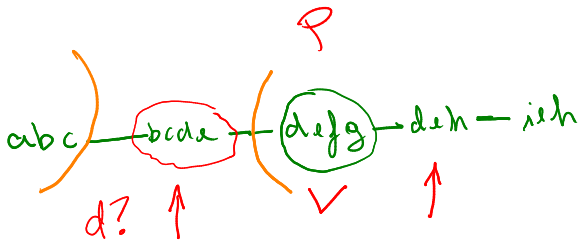
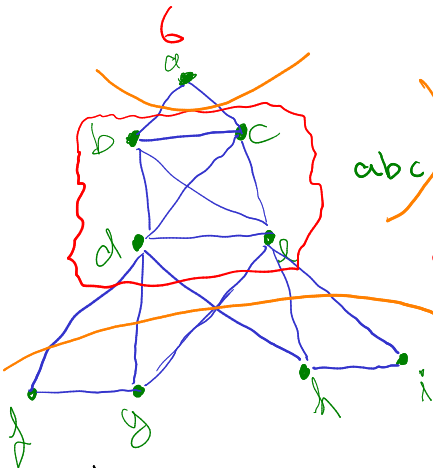
Uma **decomposição em caminho** de um grafo é uma sequência $\mathcal{P} = (X_1, X_2, \dots, X_r)$ de **bags** (sacos), onde $X_i \subseteq V(G)$ para cada $i \in \{1, 2, \dots, r\}$, tal que vale:

(P1) $\bigcup_{i=1}^r X_i = V(G)$;

(P2) Para cada $uv \in E(G)$, existe $\ell \in \{1, 2, \dots, r\}$ tal que X_ℓ contém u e v ;

(P3) Para cada $u \in V(G)$, se $u \in X_i \cap X_k$ com $i \leq k$, então $u \in X_j$ para cada j em $i \leq j \leq k$.





Tamanho da decomposição = (tam do maior bag - 1)



P

ab - bc - cd

Decomposição em caminho

Definição (Decomposição em Caminho)

Uma **decomposição em caminho** de um grafo é uma sequência $\mathcal{P} = (X_1, X_2, \dots, X_r)$ de **bags** (sacos), onde $X_i \subseteq V(G)$ para cada $i \in \{1, 2, \dots, r\}$, tal que vale:

$$(P1) \bigcup_{i=1}^r X_i = V(G);$$

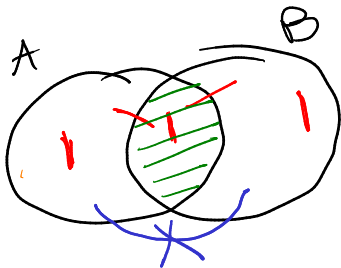
(P2) Para cada $uv \in E(G)$, existe $\ell \in \{1, 2, \dots, r\}$ tal que X_ℓ contém u e v ;

(P3) Para cada $u \in V(G)$, se $u \in X_i \cap X_k$ com $i \leq k$, então $u \in X_j$ para cada j em $i \leq j \leq k$.

Largura de caminho: a largura de caminho é o tamanho da maior bag menos 1.

Algumas definições

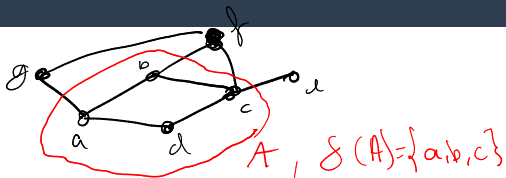
- **Separação:** dizemos que (A, B) é uma separação se $A \cup B = V(G)$ e não há arestas entre $A \setminus B$ e $B \setminus A$



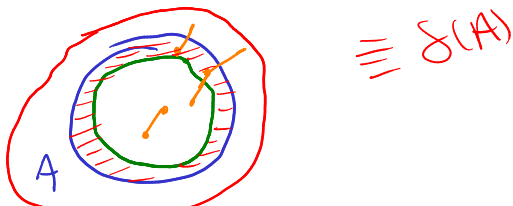
Algumas definições

- **Separação:** dizemos que (A, B) é uma separação se $A \cup B = V(G)$ e não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:** o separador de uma separação é $A \cap B$, com ordem $|A \cap B|$

Algumas definições



- **Separação:** dizemos que (A, B) é uma separação se $A \cup B = V(G)$ e não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:** o separador de uma separação é $A \cap B$, com ordem $|A \cap B|$
- **Borda:** a borda $\delta(A)$ de um conjunto A é o conjunto de vértices de A com vizinhos em $N(A) \setminus A$



Lema

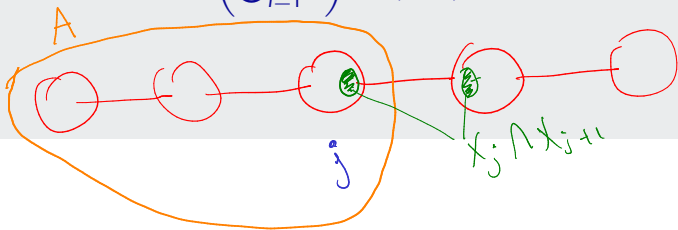
Seja (X_1, \dots, X_r) uma decomposição em caminhos de um grafo G . Então para cada $j \in \{1, \dots, r-1\}$, vale

Decomposição em caminho e separadores

Lema

Seja (X_1, \dots, X_r) uma decomposição em caminhos de um grafo G . Então para cada $j \in \{1, \dots, r-1\}$, vale

$$\delta \left(\bigcup_{i=1}^j X_i \right) \subseteq X_j \cap X_{j+1}.$$



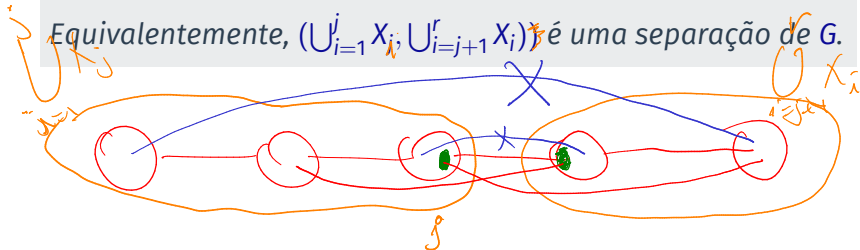
Decomposição em caminho e separadores

Lema

Seja (X_1, \dots, X_r) uma decomposição em caminhos de um grafo G . Então para cada $j \in \{1, \dots, r-1\}$, vale

$$\delta\left(\bigcup_{i=1}^j X_i\right) \subseteq X_j \cap X_{j+1}.$$

Equivalentemente, $(\bigcup_{i=1}^j X_i, \bigcup_{i=j+1}^r X_i)$ é uma separação de G .



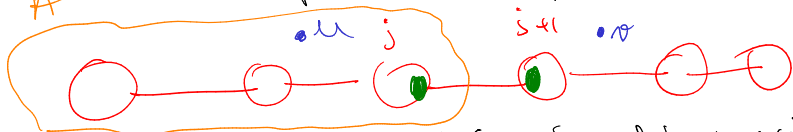
Prova. Seja j e seja $(A, B) = \left(\bigcup_{i=1}^j X_i; \bigcup_{i=j+1}^r X_i \right)$.

• Vamos mostrar que $\delta(A) \subseteq X_j \cap X_{j+1}$.

• Sup. por abs. que $\exists u \in \delta(A)$ e $u \notin X_j \cap X_{j+1}$.

• Então $\exists uv \in E(G)$ tq $u \in A; v \notin A;$ e $u \notin X_j \cap X_{j+1}$

• Seja i último índice tq $u \in X_i$ e k primeiro índice tq $v \in X_k$. • $X_j \cap X_{j+1}$



• Portanto $i \leq j$ e $k \geq j+1$. Seja l tq $u, v \in X_l$ (P2).

• Finalmente, $l \leq i \leq j < j+1 \leq k \leq l \nabla$

Decomposição em caminho padrão

Decomposição boa

Dizemos que uma decomposição em caminhos (X_1, X_2, \dots, X_r) é **boa** se:

Decomposição em caminho padrão

Decomposição boa

Dizemos que uma decomposição em caminhos (X_1, X_2, \dots, X_r) é **boa** se:

- $X_1 = X_r = \emptyset$ e

Decomposição em caminho padrão

Decomposição boa

Dizemos que uma decomposição em caminhos (X_1, X_2, \dots, X_r) é **boa** se:

- $X_1 = X_r = \emptyset$ e
- para cada $i \in \{1, 2, \dots, r-1\}$:

Decomposição em caminho padrão

Decomposição boa

Dizemos que uma decomposição em caminhos (X_1, X_2, \dots, X_r) é **boa** se:

- $X_1 = X_r = \emptyset$ e
- para cada $i \in \{1, 2, \dots, r-1\}$:
 1. ou existe um vértice $v \in X_i$ tal que $X_{i+1} = X_i \cup \{v\}$,

Decomposição em caminho padrão

Decomposição boa

Dizemos que uma decomposição em caminhos (X_1, X_2, \dots, X_r) é **boa** se:

- $X_1 = X_r = \emptyset$ e
- para cada $i \in \{1, 2, \dots, r-1\}$:
 1. ou existe um vértice $v \in X_i$ tal que $X_{i+1} = X_i \cup \{v\}$,
 2. ou existe um vértice $w \in X_i$ tal que $X_{i+1} = X_i \setminus \{w\}$



Decomposição em caminho padrão

Decomposição boa

Dizemos que uma decomposição em caminhos (X_1, X_2, \dots, X_r) é **boa** se:

- $X_1 = X_r = \emptyset$ e
- para cada $i \in \{1, 2, \dots, r-1\}$:
 1. ou existe um vértice $v \in X_i$ tal que $X_{i+1} = X_i \cup \{v\}$,
 2. ou existe um vértice $w \in X_i$ tal que $X_{i+1} = X_i \setminus \{w\}$
- No caso 1, dizemos que **introduzimos** o vértice v
- No caso 2, dizemos que **esquecemos** o vértice w

\Rightarrow podemos obter uma dec. boa a partir de uma dec. arbitrária.

Decomposição em árvore

Definição (Decomposição em Árvore)

Uma **decomposição em árvore** de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, onde T é uma árvore em que cada nó t é atribuído a um conjunto de vértices $X_t \subseteq V(G)$, chamado de **bag**, tal que vale:

Decomposição em árvore

Definição (Decomposição em Árvore)

Uma **decomposição em árvore** de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ onde T é uma árvore em que cada nó t é atribuído a um conjunto de vértices $X_t \subseteq V(G)$, chamado de **bag**, tal que vale:

$$(T1) \bigcup_{i=1}^r X_i = V(G);$$

Decomposição em árvore

Definição (Decomposição em Árvore)

Uma **decomposição em árvore** de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ onde T é uma árvore em que cada nó t é atribuído a um conjunto de vértices $X_t \subseteq V(G)$, chamado de **bag**, tal que vale:

$$(T1) \bigcup_{i=1}^r X_i = V(G);$$

(T2) Para cada $uv \in E(G)$, existe $\ell \in \{1, 2, \dots, r\}$ tal que X_ℓ contém u e v ;

Decomposição em árvore

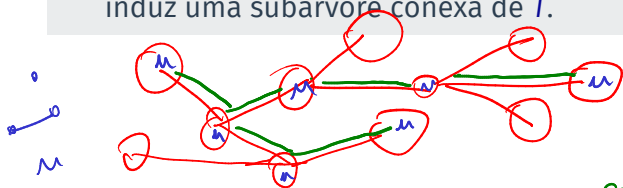
Definição (Decomposição em Árvore)

Uma **decomposição em árvore** de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ onde T é uma árvore em que cada nó t é atribuído a um conjunto de vértices $X_t \subseteq V(G)$, chamado de **bag**, tal que vale:

(T1) $\bigcup_{i=1}^r X_i = V(G)$;

(T2) Para cada $uv \in E(G)$, existe $\ell \in \{1, 2, \dots, r\}$ tal que X_ℓ contém u e v ;

(T3) Para cada $u \in V(G)$, o conjunto $T_u = \{t \in V(T) : u \in X_t\} \iff$ induz uma subárvore conexa de T .



$\forall x, x' \in G$
 $X_x \ni u \wedge X_{x'} \ni u$
todas os nós x e x'
conexos contém u . 15

Decomposição em árvore

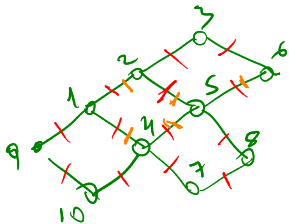
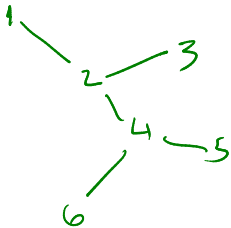
Definição (Decomposição em Árvore)

Uma **decomposição em árvore** de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ onde T é uma árvore em que cada nó t é atribuído a um conjunto de vértices $X_t \subseteq V(G)$, chamado de **bag**, tal que vale:

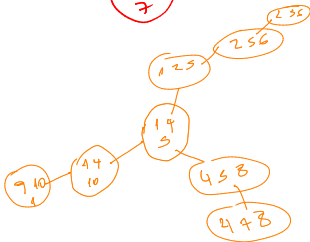
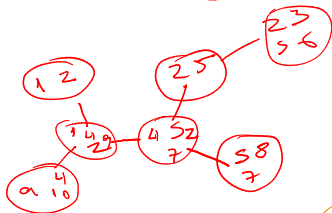
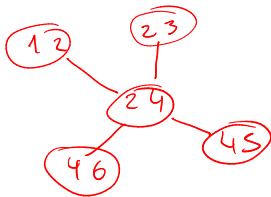
- (T1) $\bigcup_{i=1}^r X_i = V(G)$;
- (T2) Para cada $uv \in E(G)$, existe $\ell \in \{1, 2, \dots, r\}$ tal que X_ℓ contém u e v ;
- (T3) Para cada $u \in V(G)$, o conjunto $T_u = \{t \in V(T) : u \in X_t\}$ induz uma subárvore conexa de T .

Largura de árvore: a largura de árvore (ou largura arbórea) é o tamanho da maior bag menos 1.

Example
G



τ



Lema

Seja $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ uma decomposição em árvore de um grafo G e seja ab uma aresta de T . A floresta $T - ab$ obtida de T deletando-se a aresta ab consiste de duas componentes conexas T_a (com a) e T_b (com b).

Lema

Seja $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ uma decomposição em árvore de um grafo G e seja ab uma aresta de T . A floresta $T - ab$ obtida de T deletando-se a aresta ab consiste de duas componentes conexas T_a (com a) e T_b (com b).

Seja $A = \cup_{t \in V(T_a)} X_t$ e $B = \cup_{t \in V(T_b)} X_t$.

Lema

Seja $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ uma decomposição em árvore de um grafo G e seja ab uma aresta de T . A floresta $T - ab$ obtida de T deletando-se a aresta ab consiste de duas componentes conexas T_a (com a) e T_b (com b).

Seja $A = \cup_{t \in V(T_a)} X_t$ e $B = \cup_{t \in V(T_b)} X_t$. Então

$$\delta(A), \delta(B) \subseteq X_a \cap X_b.$$

Decomposição em árvore e separadores

Lema

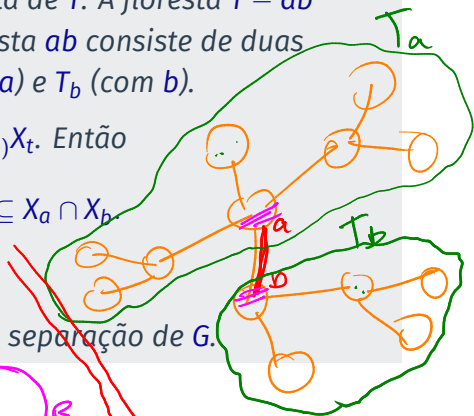
Seja $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ uma decomposição em árvore de um grafo G e seja ab uma aresta de T . A floresta $T - ab$ obtida de T deletando-se a aresta ab consiste de duas componentes conexas T_a (com a) e T_b (com b).

Seja $A = \cup_{t \in V(T_a)} X_t$ e $B = \cup_{t \in V(T_b)} X_t$. Então

$$\delta(A), \delta(B) \subseteq X_a \cap X_b$$

Equivalentemente, (A, B) é uma separação de G .

A



r, s estão na interseção.

Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t

Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

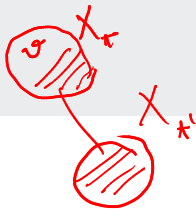
- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t
- cada nó interno de T é de um dos tipos:

Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t
- cada nó interno de T é de um dos tipos:
 1. **Introdução:** um nó t com exatamente um filho t' , tal que $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$;

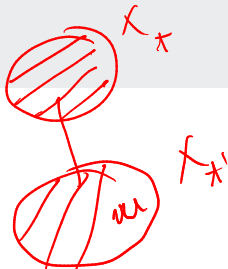


Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t
- cada nó interno de T é de um dos tipos:
 1. **Introdução:** um nó t com exatamente um filho t' , tal que $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$;
 2. **Esquecimento:** um nó t com exatamente um filho t' , tal que $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t+1}$;

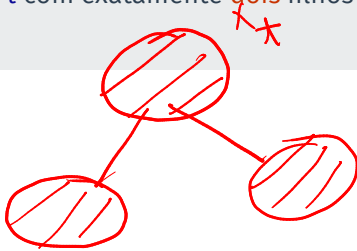


Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t
- cada nó interno de T é de um dos tipos:
 1. **Introdução:** um nó t com exatamente **um** filho t' , tal que $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$;
 2. **Esquecimento:** um nó t com exatamente **um** filho t' , tal que $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t+1}$;
 3. **Junção:** um nó t com exatamente **dois** filhos t_1 e t_2 , tais que $X_t = X_{t_1} = X_{t_2}$.



Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t
- cada nó interno de T é de um dos tipos:
 1. **Introdução:** um nó t com exatamente **um** filho t' , tal que $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$;
 2. **Esquecimento:** um nó t com exatamente **um** filho t' , tal que $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t+1}$;
 3. **Junção:** um nó t com exatamente **dois** filhos t_1 e t_2 , tais que $X_t = X_{t_1} = X_{t_2}$.

Dada uma decomposição em árvore com largura k :

- encontramos uma decomposição boa com $\mathcal{O}(k|V(G)|)$ nós;
com largura k

Decomposição em árvore padrão

Decomposição boa

Dizemos que uma decomposição em árvore associada com um r é **boa** se:

- $X_r = \emptyset$ e $X_t = \emptyset$ para toda folha t
- cada nó interno de T é de um dos tipos:
 1. **Introdução:** um nó t com exatamente **um** filho t' , tal que $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$;
 2. **Esquecimento:** um nó t com exatamente **um** filho t' , tal que $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t+1}$;
 3. **Junção:** um nó t com exatamente **dois** filhos t_1 e t_2 , tais que $X_t = X_{t_1} = X_{t_2}$.

Dada uma decomposição em árvore com largura k :

- encontramos uma decomposição boa com $\mathcal{O}(k|V(G)|)$ nós;
- o tempo de execução é $\mathcal{O}(k^2 \cdot \max\{|V(T)|, |V(G)|\})$.

Programação Dinâmica com largura de árvore ~~definidas~~ *fixas*

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:

Programação dinâmica: comparando

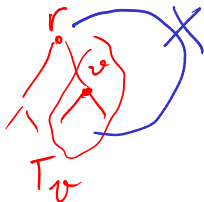
Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

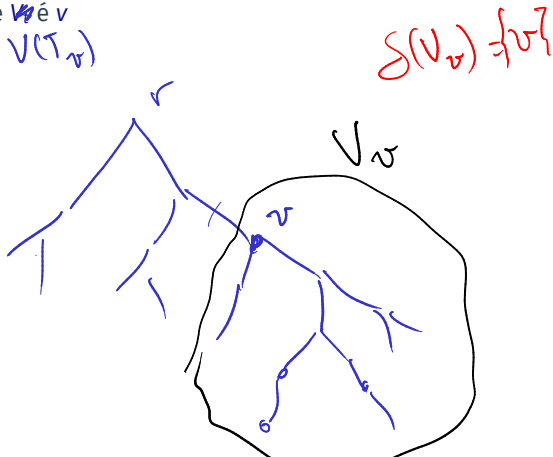
- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou



Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V é v



Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema** ;

Dado que eu escolhi/não escolhi v , qual o conj. ind. máximo em T_v ?

Programação dinâmica: comparando

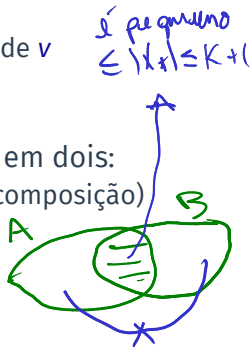
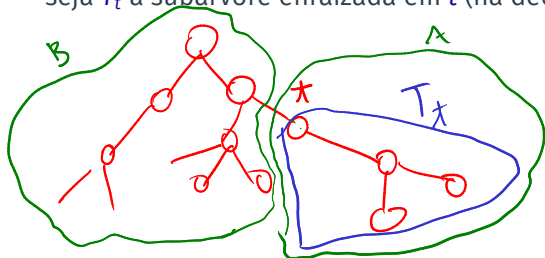
Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** um nó t divide o grafo em dois:

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** um nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)



Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** um nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)
 - seja V_t todos os vértices das bags de $V[T_t]$:

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** um nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)
 - seja V_t todos os vértices das bags de $V[T_t]$: $V_t = \cup_{r \in T_t} X_r$

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:

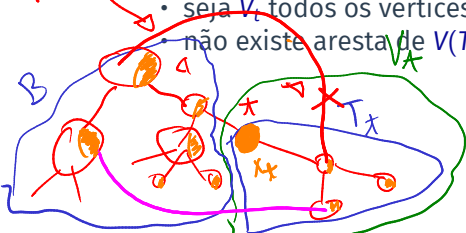
- seja T_v a subárvore enraizada em v
- não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
- uma decisão na subárvore T_v só depende de v
- enumeramos o destino de v
- criamos um **subproblema**

- **Na decomposição:** uma nó t divide o grafo em dois:

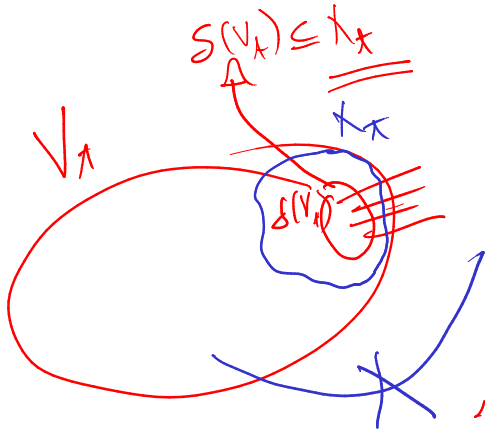
- seja T_t a subárvore enraizada em t (na decomposição)
- seja V_t todos os vértices das bags de $V[T_t]$: $V_t = \cup_{r \in T_t} X_r$
- não existe aresta de $V(T_v)$ a $X_s \setminus X_t$, para antecessor s de v

introdução!
com
 X_t

na aresta



∄ aresta entre
 V_t e $B \setminus X_t$



Vamos supor
 que já sabemos
 o destino de K_A :
 se cada vértice
 está ou não na
 solução.

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** um nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)
 - seja V_t todos os vértices das bags de $V[T_t]$: $V_t = \cup_{r \in T_t} X_r$
 - não existe aresta de $V(T_v)$ a $X_s \setminus X_t$, para antecessor s de v
→ a (borda de V_t) $\subseteq X_t$


Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** uma nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)
 - seja V_t todos os vértices das bags de $V[T_t]$: $V_t = \cup_{r \in T_t} X_r$
 - não existe aresta de $V(T_v)$ a $X_s \setminus X_t$, para antecessor s de v
→ a borda de $V_t \subseteq X_t$
 - uma decisão na subárvore T_t só depende de X_t

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** uma nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)
 - seja V_t todos os vértices das bags de $V[T_t]$: $V_t = \cup_{r \in T_t} X_r$
 - não existe aresta de $V(T_v)$ a $X_s \setminus X_t$, para antecessor s de v
→ a borda de $V_t \subseteq X_t$
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de 

Programação dinâmica: comparando

Estender a programação dinâmica da árvore:

- **Na árvore:** uma aresta (u, v) divide o grafo em dois:
 - seja T_v a subárvore enraizada em v
 - não existe aresta de $V(T_v)$ a antecessor de v , ou
→ a borda de V_t é v
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de v
 - criamos um **subproblema**
- **Na decomposição:** uma nó t divide o grafo em dois:
 - seja T_t a subárvore enraizada em t (na decomposição)
 - seja V_t todos os vértices das bags de $V[T_t]$: $V_t = \cup_{r \in T_t} X_r$
 - não existe aresta de $V(T_v)$ a $X_s \setminus X_t$, para antecessor s de v
→ a borda de $V_t \subseteq X_t$
 - uma decisão na subárvore T_v só depende de v
 - enumeramos o destino de X_t
 - criamos um **subproblema**: *gnt ind. não em V_t dado o destino de X_t .*

Programação dinâmica: subproblema

Precisamos listar e enumerar:

Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore

Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t

Programação dinâmica: subproblema

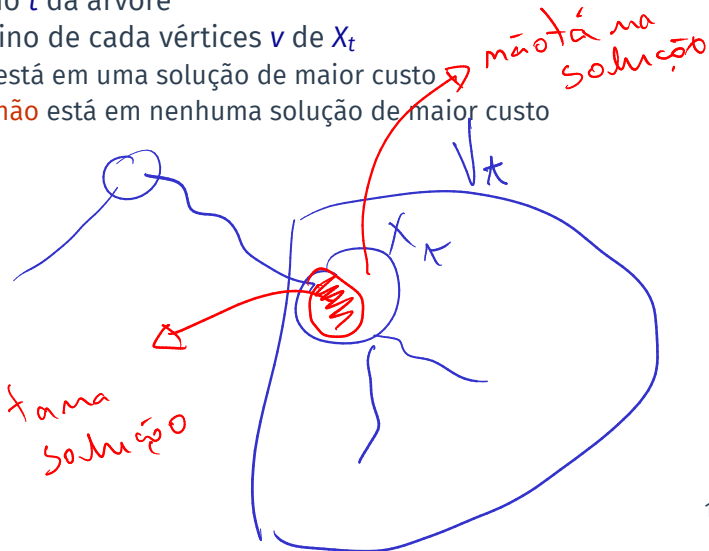
Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t
 - v está em uma solução de maior custo

Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t
 - v está em uma solução de maior custo
 - v não está em nenhuma solução de maior custo

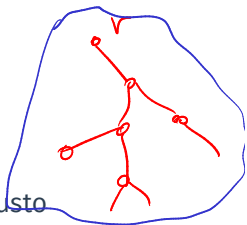


Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t
 - v está em uma solução de maior custo
 - v não está em nenhuma solução de maior custo

$$V = V_v$$



Subproblema

Consideramos o seguinte subproblema definido por:

$c[t, S] =$ máximo peso de um conjunto independente \hat{S} tal que
 $S \subseteq \hat{S} \subseteq V_t$, e \hat{S} é independente.

para cada $S \subseteq X_t$

Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t
 - v está em uma solução de maior custo
 - v não está em nenhuma solução de maior custo

Subproblema

Consideramos o seguinte subproblema definido por:

seja $S \subseteq N_t$!

$c[t, S] =$ máximo peso de um conjunto independente \hat{S} tal que
 $S \subseteq \hat{S} \subseteq V_t$, e \hat{S} é independente.

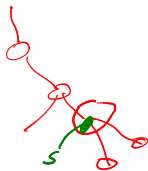
- se não há solução, marcamos $c[t, S] = -\infty$

no prob. original $V_t = V$, e não conhecemos nenhuma vértice de solução

Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t
 - v está em uma solução de maior custo
 - v não está em nenhuma solução de maior custo



Subproblema

Consideramos o seguinte subproblema definido por:

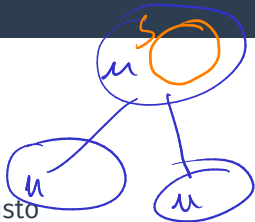
$c[t, S]$ = máximo peso de um conjunto independente \hat{S} tal que $S \subseteq \hat{S} \subseteq V_t$, e \hat{S} é independente.

- se não há solução, marcamos $c[t, S] = -\infty$
- o problema original é $c(r, \emptyset)$

Programação dinâmica: subproblema

Precisamos listar e enumerar:

- cada nó t da árvore
- o destino de cada vértices v de X_t
 - v está em uma solução de maior custo
 - v não está em nenhuma solução de maior custo



Subproblema

Consideramos o seguinte subproblema definido por:

$c[t, S] =$ máximo peso de um conjunto independente \hat{S} tal que $S \subseteq \hat{S} \subseteq V_t$, e \hat{S} é independente.

- se não há solução, marcamos $c[t, S] = -\infty$
- o problema original é $c[r, \emptyset]$

Algoritmo

Algoritmo

1. Obter uma decomposição boa \mathcal{T}

Algoritmo

1. Obter uma decomposição **boa** \mathcal{T}
2. Preencher tabela de maneira *bottom-up*, dependendo do tipo do nó: **folha**, **esquecimento**, **introdução**, *junção*

Preenchendo a tabela

$$S \subseteq X = \{ \}$$
$$\forall x = \emptyset$$

$$C[X, S]$$

Algoritmo

1. Obter uma decomposição **boa** \mathcal{T}
2. Preencher tabela de maneira *bottom-up*, dependendo do tipo do nó: **folha**, **esquecimento**, **introdução**

Folha t :

Preenchendo a tabela

Algoritmo

1. Obter uma decomposição **boa** \mathcal{T}
2. Preencher tabela de maneira *bottom-up*, dependendo do tipo do nó: **folha**, **esquecimento**, **introdução**

Folha t :

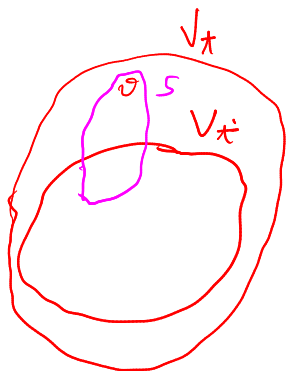
- Definimos $c[t, \emptyset] = 0$.

$$\text{para } \lambda_A = V_A = \emptyset.$$



Preenchendo a tabela: nó de introdução

Nó de introdução t :



Nó de introdução t :

- Seja t' o único filho de t

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente,

Existe $\hat{S} \supseteq S$

$$S \subseteq \hat{S}$$

$$\hat{S} \subseteq V$$

\hat{S} é independente? NÃO!

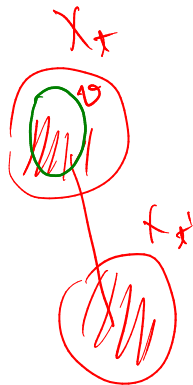
Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] = \left\{ \right.$$

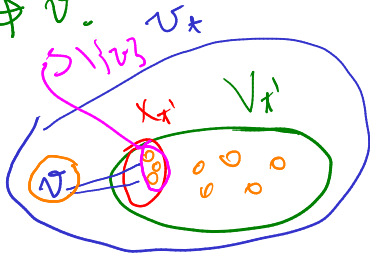


Como f é de introdução

Sobemos $X_x = X_{x'} \cup \{v\}$ e $X_{x'} \neq v$.

Portanto $V_{x'} \neq v$.

$$c(x, S \setminus \{v\}) + w(v) \\ \leq w(\hat{S}) = w(\hat{S} \setminus \{v\}) + w(v)$$



Como $X_{x'}$ é um separador, não existe aresta entre $V_{x'} \setminus X_{x'}$ e v .

Seja \hat{S} que obtém $c(x, S)$ tq

$X_x \cap \hat{S} = S$. Se $v \in \hat{S}$, então

$\hat{S} \setminus \{v\}$ é tal que $w(\hat{S} \setminus \{v\}) = c(x', S \setminus \{v\})$.

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] = \begin{cases} c[t', S] & \text{se } v \notin S, \end{cases}$$

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] = \begin{cases} c[t', S] & \text{se } v \notin S, \\ c[t', S \setminus \{v\}] + w(v) // & \text{se } v \in S. \end{cases}$$

Nó de introdução t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \cup \{v\}$ para algum $v \in X_t$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] = \begin{cases} c[t', S] & \text{se } v \notin S, \\ c[t', S \setminus \{v\}] + w(v) // & \text{se } v \in S. \end{cases}$$

Afirmção: o passo está bem definido

Preenchendo a tabela: nó de esquecimento

Nó de esquecimento t :

Preenchendo a tabela: nó de esquecimento

Nó de esquecimento t :

- Seja t' o único filho de t

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente,

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] =$$

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

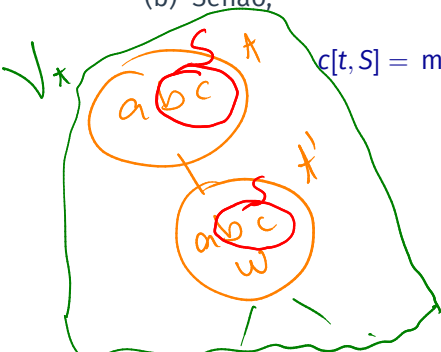
$$c[t, S] = \max \{ c[t', S],$$

Preenchendo a tabela: nó de esquecimento

Nó de esquecimento t :

- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] = \max \{ c[t', S], c[t', S \cup \{w\}] \}.$$



w não faz parte do set

w faz parte do set.

Nó de esquecimento t :

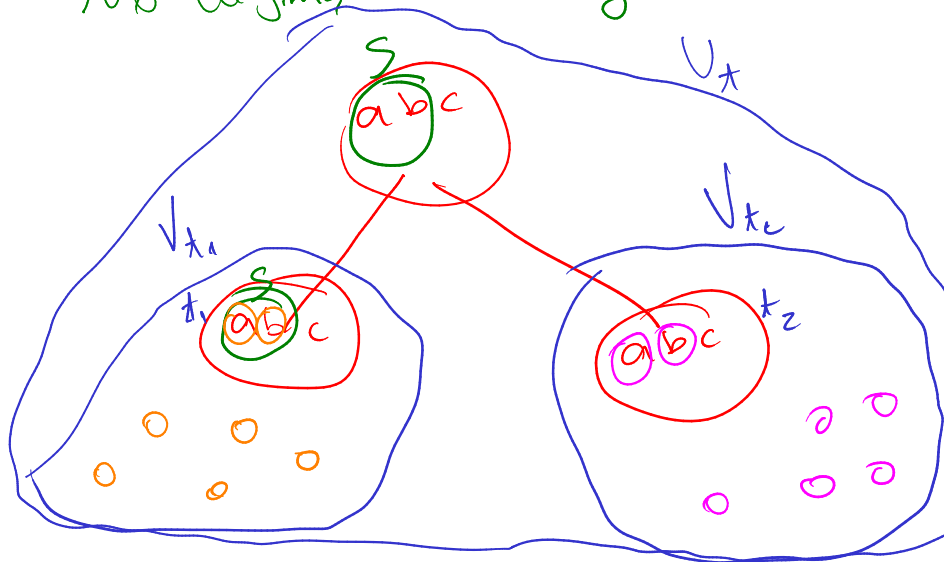
- Seja t' o único filho de t
- Temos $X_t = X_{t'} \setminus \{w\}$ para algum $w \in X_{t'}$
- Seja $S \subseteq X_t$:
 - (a) Se S não é independente, $c[t, S] = -\infty$
 - (b) Senão,

$$c[t, S] = \max \left\{ c[t', S], c[t', S \cup \{w\}] \right\}.$$

Afirmção: o passo está bem definido

$$c(t, S) = c(t_1, S) + dt_2(S) - w(S).$$

Nº de jmeço á análogo.



Seja $\hat{S} \subseteq V_X$ tq $c(t, S) = w(\hat{S})$.
 tq $S \subseteq \hat{S}$ e \hat{S} é ind.

Seja S_1 e S_2 tq $c(t_1, S) = w(S)$ e $c(t_2, S) = w(S)$

Sobemos que $(S_1 \setminus X_t) \cup (S_2 \setminus X_t)$ é ind,
 pois X_t é separador. Portanto, $(S_1 \setminus X_t) \cup (S_2 \setminus X_t) \cup S := A$

também é independente, pois $(S_i \setminus X_t) \cup S \subseteq S_i$.
 Daí, $c(t, S) \geq w(A) = c(t_1, S) - w(S) + c(t_2, S) - w(S) + w(S)$.



Tempo de execução

Teorema

Seja G um grafo com n vértices e pesos nos vértices. Dados G e uma decomposição em árvore de G com largura no máximo k , então o Problema do Conjunto Independente de Peso Máximo pode ser resolvido em tempo $2^k \cdot k^{O(k)} \cdot n$, i.e., em tempo linear para k fixo.

Qual o tamanho da tabela? $c[x, s]$

→ Quanto a x ? $\Theta(k|V|) = \Theta(k \cdot n)$

→ Quanto a s ? $\max_{t \in T} 2^{|\{x, t\}|} \leq 2^{k+1}$

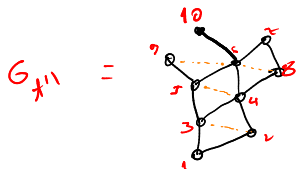
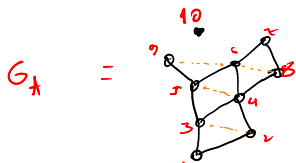
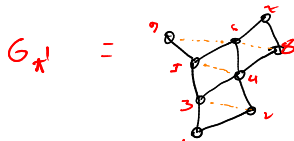
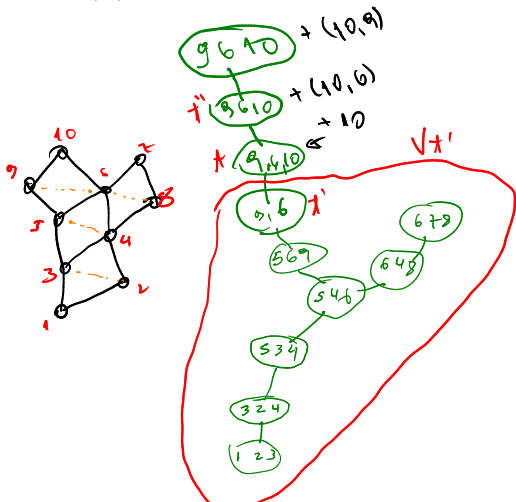
⇒ Temos $2^k k n$ conf.

⇒ Quanto tempo por configuração? Para o fazer em tempo $\Theta(k^{O(k)})$ se for processarmos adeq.

Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:



Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:
 - $u, v \in X_t$

Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:
 - $u, v \in X_t$
 - tem exatamente **um** filho t' tal $X_t = X_{t'}$.

Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:
 - $u, v \in X_t$
 - tem exatamente um filho t' tal $X_t = X_{t'}$.

Decomposição boa estendida

Uma decomposição em árvore boa que além dos tipos tradicionais, pode ter nós de **introdução de aresta** e:

Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:
 - $u, v \in X_t$
 - tem exatamente um filho t' tal $X_t = X_{t'}$.

Decomposição boa estendida

Uma decomposição em árvore boa que além dos tipos tradicionais, pode ter nós de **introdução de aresta** e:

- para cada $uv \in E(G)$, existe um único nó que introduz uv

Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:
 - $u, v \in X_t$
 - tem exatamente um filho t' tal $X_t = X_{t'}$.

Decomposição boa estendida

Uma decomposição em árvore boa que além dos tipos tradicionais, pode ter nós de **introdução de aresta** e:

- para cada $uv \in E(G)$, existe um único nó que introduz uv

Uma subárvore T_t da decomposição induz um subgrafo:

Decomposição boa estendida

Mais um tipo de nó:

- **Introdução de aresta:** um nó t , rotulado com uma aresta $E(G)$ tal que:
 - $u, v \in X_t$
 - tem exatamente um filho t' tal $X_t = X_{t'}$.

Decomposição boa estendida

Uma decomposição em árvore boa que além dos tipos tradicionais, pode ter nós de **introdução de aresta** e:

- para cada $uv \in E(G)$, existe um único nó que introduz uv

Uma subárvore T_t da decomposição induz um subgrafo:

$$G_t = (V_t, E_t = \{e : e \text{ é introduzida em um nó de } T_t\})$$

Problema de Steiner

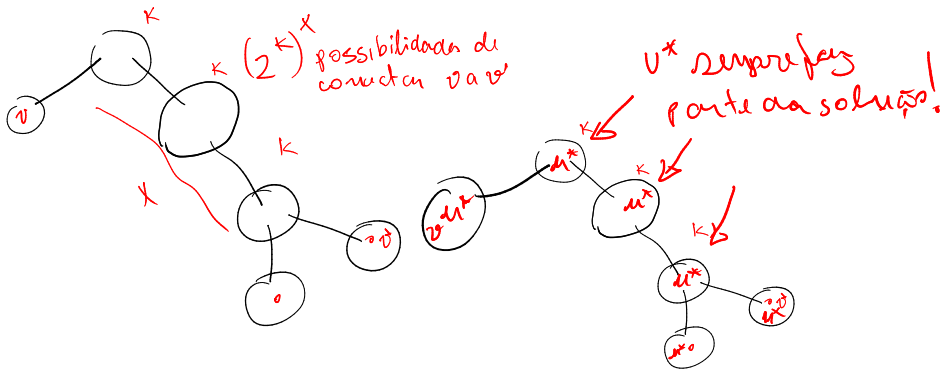
Árvore de Steiner mínima: Dados grafo G , conjunto de terminais $K \subseteq V(G)$, queremos encontrar uma árvore de Steiner H cujo número $|E(H)|$ seja mínimo.

Simplificando: evitando subproblemas vazios:

Quebrando o problema

Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag



Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag
- largura da decomposição aumenta somente em uma unidade

Quebrando o problema

Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag
- largura da decomposição aumenta somente em uma unidade

Dividindo: Como um separador X_t divide uma árvore de Steiner H ?

Quebrando o problema

Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag
- largura da decomposição aumenta somente em uma unidade

Dividindo: Como um separador X_t divide uma árvore de Steiner H ?

- somente um subconjunto $X \subseteq X_t$ está em H

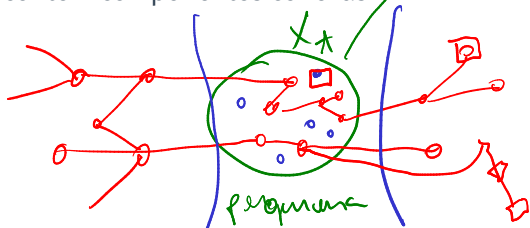
Quebrando o problema

Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag
- largura da decomposição aumenta somente em uma unidade

Dividindo: Como um separador X_t divide uma árvore de Steiner H ?

- somente um subconjunto $X \subseteq X_t$ está em H
- $G[X]$ contém componentes conexas



• no máximo
 $K+1$ comp. conexas (1)
contendo $K \subseteq X_t$

• Partições de X em l partes

• $K \cap X_t \subseteq X$

Quebrando o problema

Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag
- largura da decomposição aumenta somente em uma unidade

Dividindo: Como um separador X_t divide uma árvore de Steiner H ?

- somente um subconjunto $X \subseteq X_t$ está em H
- $G[X]$ contém componentes conexas (que particionam X)

Quebrando o problema

Simplificando: evitando subproblemas vazios:

- escolha terminal $u^* \in K$ e adicione a cada bag
- largura da decomposição aumenta somente em uma unidade

Dividindo: Como um separador X_t divide uma árvore de Steiner H ?

- somente um subconjunto $X \subseteq X_t$ está em H
- $G[X]$ contém componentes conexas (que particionam X)
- $K \cap V_t \subseteq H[V_t]$

Subproblema

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X .

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

1. a floresta F contém exatamente componentes C_1, \dots, C_q com $V(C_s) \cap X_t = P_s$, para cada $s \in \{1, \dots, q\}$;

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

1. a floresta F contém exatamente componentes C_1, \dots, C_q com $V(C_s) \cap X_t = P_s$, para cada $s \in \{1, \dots, q\}$;
2. $X_t \cap V(F) = X$;

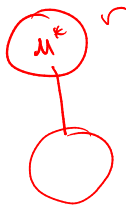
Subproblema

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

1. a floresta F contém exatamente componentes C_1, \dots, C_q com $V(C_s) \cap X_t = P_s$, para cada $s \in \{1, \dots, q\}$;
2. $X_t \cap V(F) = X$;
3. $K \cap V_t \subseteq V(F)$.

$$X = \{u^*\}$$
$$\mathcal{P} = \{\{u^*\}\}$$



$$C_r = C$$

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

1. a floresta F contém exatamente componentes C_1, \dots, C_q com $V(C_s) \cap X_t = P_s$, para cada $s \in \{1, \dots, q\}$;
2. $X_t \cap V(F) = X$;
3. $K \cap V_t \subseteq V(F)$.

- se não há solução, marcamos $c[t, S, \mathcal{P}] = +\infty$

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

1. a floresta F contém exatamente componentes C_1, \dots, C_q com $V(C_s) \cap X_t = P_s$, para cada $s \in \{1, \dots, q\}$;
2. $X_t \cap V(F) = X$;
3. $K \cap V_t \subseteq V(F)$.

- se não há solução, marcamos $c[t, S, \mathcal{P}] = +\infty$
- o problema original é

Subproblema

Seja t um nó, $X \subseteq X_t$ e $\mathcal{P} = \{P_1, P_2, \dots, P_q\}$ uma partição de X . Então $c[t, X, \mathcal{P}]$ é o menor número de arestas de uma floresta F que:

1. a floresta F contém exatamente componentes C_1, \dots, C_q com $V(C_s) \cap X_t = P_s$, para cada $s \in \{1, \dots, q\}$;
2. $X_t \cap V(F) = X$;
3. $K \cap V_t \subseteq V(F)$.

- se não há solução, marcamos $c[t, S, \mathcal{P}] = +\infty$
- o problema original é $c[r, \{u^*\}, \{\{u^*\}\}]$

Folha t :

Folha t :

- Temos $X_t = \{u^*\}$

Folha t :

- Temos $X_t = \{u^*\}$
 - (a) Se $X = \emptyset$, então $\mathcal{P} = \emptyset$ e $c[t, X, \mathcal{P}] = +\infty$

Folha t :

- Temos $X_t = \{u^*\}$
 - (a) Se $X = \emptyset$, então $\mathcal{P} = \emptyset$ e $c[t, X, \mathcal{P}] = +\infty$
 - (b) Se $S = \{u^*\}$, então $\mathcal{P} = \{\{u^*\}\}$ e $c[t, X, \mathcal{P}] = 0$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

(a) Se $u \notin X$ ou $v \notin X$,

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

(a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$,

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

(a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$

(b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:
 \Rightarrow existe uma solução com a mesma partição em $X_{t'}$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:
 \Rightarrow existe uma solução com a mesma partição em $X_{t'}$
 - uv faz parte de uma solução:

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:
 - \Rightarrow existe uma solução com a mesma partição em $X_{t'}$
 - uv faz parte de uma solução:
 - $\Rightarrow u$ e v estavam em componentes distintas em $X_{t'}$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:
 \Rightarrow existe uma solução com a mesma partição em $X_{t'}$
 - uv faz parte de uma solução:
 $\Rightarrow u$ e v estavam em componentes distintas em $X_{t'}$

Definimos:

$$c[t, X, \mathcal{P}] =$$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:
 \Rightarrow existe uma solução com a mesma partição em $X_{t'}$
 - uv faz parte de uma solução:
 $\Rightarrow u$ e v estavam em componentes distintas em $X_{t'}$

Definimos:

$$c[t, X, \mathcal{P}] = \min \left\{ c[t', X, \mathcal{P}], \right.$$

Preenchendo a tabela: nó de introdução de aresta

Nó t introduz vértice uv :

- (a) Se $u \notin X$ ou $v \notin X$, daí $c[t, X, \mathcal{P}] = +\infty$
- (b) Se $u, v \in X$ e não existe $P \in \mathcal{P}$ com $u, v \in P$, daí $c[t, X, \mathcal{P}] = +\infty$
- (c) Senão, existe \hat{P} com $u, v \in \hat{P}$:
 - uv não faz parte da solução:
 \Rightarrow existe uma solução com a mesma partição em $X_{t'}$
 - uv faz parte de uma solução:
 $\Rightarrow u$ e v estavam em componentes distintas em $X_{t'}$

Definimos:

$$c[t, X, \mathcal{P}] = \min \left\{ c[t', X, \mathcal{P}], \min_{\mathcal{P}'} c[t', X, \mathcal{P}'] \right\},$$

onde \mathcal{P}' contém as mesmas partes \mathcal{P} , exceto que \hat{P} foi dividido em P_u, P_v , com $u \in P_u$ e $v \in P_v$

Preenchendo a tabela: nó de esquecimento e junção

Nó t esquece w :

Nó t esquece w :

$$c[t, X, \mathcal{P}] = \min \left\{ c[t', X, \mathcal{P}], \min_{\mathcal{P}'} c[t', X \cup \{w\}, \mathcal{P}'] \right\},$$

onde \mathcal{P}' contém as mesmas partes \mathcal{P} , exceto que foi adicionado w a alguma parte P

Preenchendo a tabela: nó de esquecimento e junção

Nó t esquece w :

$$c[t, X, \mathcal{P}] = \min \left\{ c[t', X, \mathcal{P}], \min_{\mathcal{P}'} c[t', X \cup \{w\}, \mathcal{P}'] \right\},$$

onde \mathcal{P}' contém as mesmas partes \mathcal{P} , exceto que foi adicionado w a alguma parte P

Nó t junta t_1 e t_2 :

Preenchendo a tabela: nó de esquecimento e junção

Nó t esquece w :

$$c[t, X, \mathcal{P}] = \min \left\{ c[t', X, \mathcal{P}], \min_{\mathcal{P}'} c[t', X \cup \{w\}, \mathcal{P}'] \right\},$$

onde \mathcal{P}' contém as mesmas partes \mathcal{P} , exceto que foi adicionado w a alguma parte P

Nó t junta t_1 e t_2 :

$$c[t, X, \mathcal{P}] = \min_{\mathcal{P}_1, \mathcal{P}_2} c[t_1, X, \mathcal{P}_1] + c[t_2, X, \mathcal{P}_2],$$

onde \mathcal{P}_1 e \mathcal{P}_2 são todas as partições que, quando juntadas, não induzem componentes cíclicas

Preenchendo a tabela: nó de esquecimento e junção

Nó t esquece w :

$$c[t, X, \mathcal{P}] = \min \left\{ c[t', X, \mathcal{P}], \min_{\mathcal{P}'} c[t', X \cup \{w\}, \mathcal{P}'] \right\},$$

onde \mathcal{P}' contém as mesmas partes \mathcal{P} , exceto que foi adicionado w a alguma parte P

Nó t junta t_1 e t_2 :

$$c[t, X, \mathcal{P}] = \min_{\mathcal{P}_1, \mathcal{P}_2} c[t_1, X, \mathcal{P}_1] + c[t_2, X, \mathcal{P}_2],$$

onde \mathcal{P}_1 e \mathcal{P}_2 são todas as partições que, quando juntadas, não induzem componentes cíclicas

Teorema

Dado um grafo G , terminais $K \subseteq V(G)$ e uma decomposição em árvore de G com largura k , então podemos encontrar uma Árvore de Steiner Mínima que conecta K com tempo de execução $k^{O(k)} \cdot n$.

Diversos problemas

Diversos problemas admitem **FPT** quando a largura de árvore é pequena.

Os algoritmos de programação dinâmica podem ser usados para obter:

- **Algoritmos exponenciais simples:**

Diversos problemas

Diversos problemas admitem **FPT** quando a largura de árvore é pequena.

Os algoritmos de programação dinâmica podem ser usados para obter:

- **Algoritmos exponenciais simples:**
 - Cobertura por vértice, conjunto independente máximo, Conjunto dominante, Corte máximo, Transversal de Ciclo Ímpar, q -Coloração.

Diversos problemas

Diversos problemas admitem **FPT** quando a largura de árvore é pequena.

Os algoritmos de programação dinâmica podem ser usados para obter:

- **Algoritmos exponenciais simples:**
 - Cobertura por vértice, conjunto independente máximo, Conjunto dominante, Corte máximo, Transversal de Ciclo Ímpar, q -Coloração.
- **Algoritmos super-exponenciais (com dependência linear):**

Diversos problemas

Diversos problemas admitem **FPT** quando a largura de árvore é pequena.

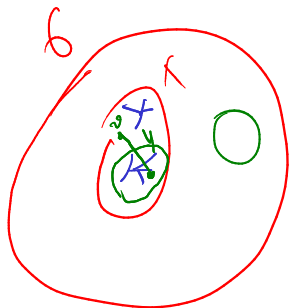
Os algoritmos de programação dinâmica podem ser usados para obter:

- **Algoritmos exponenciais simples:**
 - Cobertura por vértice, conjunto independente máximo, Conjunto dominante, Corte máximo, Transversal de Ciclo Ímpar, q -Coloração.
- **Algoritmos super-exponenciais (com dependência linear):**
 - Árvore de Steiner, Conjunto de retroalimentação de vértices, Caminho Hamiltoniano, Número cromático, Empacotamento de ciclos, Cobertura por Vértices conexa, Conjunto dominante conexo, conjunto de retroalimentação conexo.

Teorema de Courcèlle

Exemplo de fórmula **MSO₂** (Monadic Second Order):

$$\text{conn}(X) = \forall Y \subseteq V [(\exists u \in V u \in Y \wedge \exists v \in X v \notin Y) \\ \Rightarrow (\exists e \in E \exists u \in X \exists v \in X \text{inc}(u, e) \wedge \text{inc}(v, e) \wedge u \in Y \wedge v \notin Y)]$$



Para cada $Y \subseteq X$:

↳ Para cada $u \in V$:

↳ Para cada $v \in X$:

Se $u \in Y$:

↳ Para cada $e \in E$...

etc etc. .



Exemplo de fórmula **MSO₂** (*Monadic Second Order*):

$$\begin{aligned} \text{conn}(X) &= \forall_{Y \subseteq V} [(\exists_{u \in V} u \in Y \wedge \exists_{v \in X} v \notin Y) \\ &\Rightarrow (\exists_{e \in E} \exists_{u \in X} \exists_{v \in X} \text{inc}(u, e) \wedge \text{inc}(v, e) \wedge u \in Y \wedge v \notin Y)] \end{aligned}$$

Em português: Para todo subconjunto e vértices Y , se X contém tanto um vértice de Y como um vértice fora de Y , então existe uma aresta e cujos extremos u, v ambos pertencem a X , mas um deles está em Y e o outro está fora de Y .

Um “programa”

Fórmula:

Um “programa”

Fórmula:

- uma sequência de **símbolos**, **variáveis** e **predicados**.

Um “programa”

Fórmula:

- uma sequência de **símbolos**, **variáveis** e **predicados**.
- pode ser interpretada como um “programa” para verificar uma propriedade:

Um “programa”

Fórmula:

- uma sequência de **símbolos**, **variáveis** e **predicados**.
- pode ser interpretada como um “programa” para verificar uma propriedade: **o subgrafo induzido por X é conexo?**

Variáveis: podem representar

Variáveis: podem representar

- vértices e arestas de um grafo

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas (variáveis monádicas)

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas (variáveis monádicas)

parâmetros: os parâmetros ou **variáveis livres** são as variáveis que são externas à fórmula

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas (variáveis monádicas)

parâmetros: os parâmetros ou **variáveis livres** são as variáveis que são externas à fórmula, e.g.: X

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas (variáveis monádicas)

parâmetros: os parâmetros ou **variáveis livres** são as variáveis que são externas à fórmula, e.g.: X

predicados: os predicados são afirmações sobre determinado número de variáveis

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas (variáveis monádicas)

parâmetros: os parâmetros ou **variáveis livres** são as variáveis que são externas à fórmula, e.g.: X

predicados: os predicados são afirmações sobre determinado número de variáveis, e.g.: **inc**

Variáveis: podem representar

- vértices e arestas de um grafo
- **conjuntos** de vértices ou de arestas (variáveis monádicas)

parâmetros: os parâmetros ou **variáveis livres** são as variáveis que são externas à fórmula, e.g.: X

predicados: os predicados são afirmações sobre determinado número de variáveis, e.g.: **inc**

observação: se não permitirmos conjuntos de arestas, obteremos uma fórmula **MSO₁**

Operadores lógicos:

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall
 - testa uma condição sobre uma variável em **todas** avaliações do domínio

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall
 - testa uma condição sobre uma variável em **todas** avaliações do domínio, i.e.: falso se em alguma for falsa

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall
 - testa uma condição sobre uma variável em **todas** avaliações do domínio, i.e.: falso se em alguma for falsa
- **existencial:**

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall
 - testa uma condição sobre uma variável em **todas** avaliações do domínio, i.e.: falso se em alguma for falsa
- **existencial:**
 - símbolo \exists

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall
 - testa uma condição sobre uma variável em **todas** avaliações do domínio, i.e.: falso se em alguma for falsa
- **existencial:**
 - símbolo \exists
 - testa uma condição sobre uma variável em **alguma** avaliações do domínio

Operadores lógicos:

- operadores tradicionais: $\wedge, \vee, \neg, \Rightarrow$
- representam testes e condições

Quantificadores:

- **universal:**
 - símbolo \forall
 - testa uma condição sobre uma variável em **todas** avaliações do domínio, i.e.: falso se em alguma for falsa
- **existencial:**
 - símbolo \exists
 - testa uma condição sobre uma variável em **alguma** avaliações do domínio, i.e.: verdadeiro se em alguma for verdadeira

Definição formal

Distinguimos:

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“**parâmetro formal**”)

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“parâmetro formal”)
 - x^G é o elemento correspondente em G (“parâmetro real”)

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“parâmetro formal”)
 - x^G é o elemento correspondente em G (“parâmetro real”)
- **estrutura:** par $\langle G, \Sigma^G \rangle$ que representa um contexto

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“parâmetro formal”)
 - x^G é o elemento correspondente em G (“parâmetro real”)
- **estrutura:** par $\langle G, \Sigma^G \rangle$ que representa um contexto
 - Σ^G contém todas avaliações de Σ

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“parâmetro formal”)
 - x^G é o elemento correspondente em G (“parâmetro real”)
- **estrutura:** par $\langle G, \Sigma^G \rangle$ que representa um contexto
 - Σ^G contém todas avaliações de Σ
 - se ϕ é verdadeira com Σ^G , escrevemos

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“parâmetro formal”)
 - x^G é o elemento correspondente em G (“parâmetro real”)
- **estrutura:** par $\langle G, \Sigma^G \rangle$ que representa um contexto
 - Σ^G contém todas avaliações de Σ
 - se ϕ é verdadeira com Σ^G , escrevemos

$$\langle G, \Sigma^G \rangle \models \phi$$

Definição formal

Distinguimos:

- **sintaxe:** regras para uma expressão estar bem formada
- **semântica:** significado dado a uma fórmula e elementos
 - depende do contexto
- **assinatura:** conjunto Σ de variáveis livres
 - $x \in \Sigma$ contém um tipo associado (“parâmetro formal”)
 - x^G é o elemento correspondente em G (“parâmetro real”)
- **estrutura:** par $\langle G, \Sigma^G \rangle$ que representa um contexto
 - Σ^G contém todas avaliações de Σ
 - se ϕ é verdadeira com Σ^G , escrevemos

$$\langle G, \Sigma^G \rangle \models \phi$$

i.e., $\langle G, \Sigma^G \rangle$ modela ϕ .

Fórmula atômica

Uma fórmula **MSO₂** é definida recursivamente.

Base: Uma fórmula atômica é:

Uma fórmula **MSO₂** é definida recursivamente.

Base: Uma fórmula atômica é:

- $u \in X$, para cada $u, X \in \Sigma$, que representam vértice (aresta) e conjunto de vértices (arestas)

Uma fórmula **MSO₂** é definida recursivamente.

Base: Uma fórmula atômica é:

- $u \in X$, para cada $u, X \in \Sigma$, que representam vértice (aresta) e conjunto de vértices (arestas)
- $\text{inc}(u, e)$, para cada $u, e \in \Sigma$, que representam vértice e aresta

Uma fórmula **MSO₂** é definida recursivamente.

Base: Uma fórmula atômica é:

- $u \in X$, para cada $u, X \in \Sigma$, que representam vértice (aresta) e conjunto de vértices (arestas)
- $\text{inc}(u, e)$, para cada $u, e \in \Sigma$, que representam vértice e aresta
- $x = y$, para cada $x, y \in \Sigma$

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1$

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1, \phi_1 \wedge \phi_2$

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$
- $\phi_1 \Rightarrow \phi_2$

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$
- $\phi_1 \Rightarrow \phi_2$

Quantificação: dada fórmula ψ com assinatura Σ' com $v \in \Sigma'$, então também é fórmula com assinatura $\Sigma = \Sigma' \setminus \{v\}$:

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$
- $\phi_1 \Rightarrow \phi_2$

Quantificação: dada fórmula ψ com assinatura Σ' com $v \in \Sigma'$, então também é fórmula com assinatura $\Sigma = \Sigma' \setminus \{v\}$:

- $\phi_v = \forall_{v \in V} \psi$

Compondo fórmulas

Uma fórmula **MSO₂** é definida recursivamente.

Operações: dadas fórmulas ϕ_1, ϕ_2 , também são fórmulas:

- $\neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$
- $\phi_1 \Rightarrow \phi_2$

Quantificação: dada fórmula ψ com assinatura Σ' com $v \in \Sigma'$, então também é fórmula com assinatura $\Sigma = \Sigma' \setminus \{v\}$:

- $\phi_v = \forall_{v \in V} \psi$
- $\phi_\exists = \exists_{v \in V} \psi$

$$v \in X \wedge (u \in Y \rightarrow v \in Z)$$
$$\Sigma' = \{v, x, u, y, z\}$$

$$\exists_{v \in V} (v \in X \wedge (u \in Y \rightarrow v \in Z))$$

Podemos escreve notação equivalente quando for conveniente, e.g.:

Podemos escreve notação equivalente quando for conveniente, e.g.:

- $\exists_{v \in X} \psi$ ^{variável} é o mesmo que $\exists_{v \in V} (v \in X) \wedge \psi$

Podemos escreve notação equivalente quando for conveniente, e.g.:

- $\exists_{v \in X} \psi$ é o mesmo que $\exists_{v \in V} (v \in X) \wedge \psi$
- $\text{adj}(u, v)$ é $(u \neq v) \wedge (\exists_{e \in E} \text{inc}(u, e) \wedge \text{inc}(v, e))$

Podemos escreve notação equivalente quando for conveniente, e.g.:

- $\exists_{v \in X} \psi$ é o mesmo que $\exists_{v \in V} (v \in X) \wedge \psi$
- $\text{adj}(u, v)$ é $(u \neq v) \wedge (\exists_{e \in E} \text{inc}(u, e) \wedge \text{inc}(v, e))$
- $X \subseteq Y$ é $\forall_{v \in V} [(v \in X) \Rightarrow (v \in Y)]$

Podemos escreve notação equivalente quando for conveniente, e.g.:

- $\exists_{v \in X} \psi$ é o mesmo que $\exists_{v \in V} (v \in X) \wedge \psi$
- $\text{adj}(u, v)$ é $(u \neq v) \wedge (\exists_{e \in E} \text{inc}(u, e) \wedge \text{inc}(v, e))$
- $X \subseteq Y$ é $\forall_{v \in V} [(v \in X) \Rightarrow (v \in Y)]$

G é 3-colorível:

Exercício : tentar escrever antes de ler!

Teorema

Seja ϕ uma fórmula **MSO**₂ com assinatura Σ e G um grafo com uma avaliação Σ^G . Suponha que seja dada uma decomposição em árvore de G com largura t . Então existe um algoritmo que verifica se ϕ é satisfeita para G em tempo $f(\|\phi\|, t) \cdot n$, para alguma função f computável.

⇒ \exists coloração de FPT comparando a largura de árvore t .

Teorema de Courcelle em problemas de otimização

O teorema anterior não fornece um **FPT** para o problema de Cobertura por Vértices no parâmetro t diretamente!

$$\Psi = \exists x_1 \exists x_2 \dots \exists x_k \left(\forall e \left[\text{incl}(x_1, e) \vee \text{incl}(x_2, e) \vee \dots \vee \text{incl}(x_k, e) \right] \right)$$

$$\|\Psi\| \leq g(k)$$

\Rightarrow teorema implica alg
em tempo $f(k, t) \cdot n$

Teorema de Courcelle em problemas de otimização

O teorema anterior não fornece um **FPT** para o problema de Cobertura por Vértices no parâmetro t diretamente!

- Seja $\alpha(x_1, x_2, \dots, x_p) = a_0 + a_1x_1 + \dots + a_px_p$

Teorema de Courcelle em problemas de otimização

O teorema anterior não fornece um **FPT** para o problema de Cobertura por Vértices no parâmetro t diretamente!

- Seja $\alpha(x_1, x_2, \dots, x_p) = a_0 + a_1x_1 + \dots + a_px_p$
- Considere ϕ que contenha variáveis livres sobre conjuntos (monádicas) X_1, \dots, X_p de vértices ou arestas.

Teorema de Courcelle em problemas de otimização

O teorema anterior não fornece um **FPT** para o problema de Cobertura por Vértices no parâmetro t diretamente!

- Seja $\alpha(x_1, x_2, \dots, x_p) = a_0 + a_1x_1 + \dots + a_px_p$
- Considere ϕ que contenha variáveis livres sobre conjuntos (monádicas) X_1, \dots, X_p de vértices ou arestas.
- Considere o problema: minimizar $\alpha(|X_1|, \dots, |X_p|)$ tal que ϕ vale para G

Eg: $\text{MIN } |X| \text{ tq}$
 $\phi = \exists x \subseteq V (\forall e \exists v \in x \text{ inc}(v, e)).$
+ uma constante

Teorema de Courcelle em problemas de otimização

O teorema anterior não fornece um **FPT** para o problema de Cobertura por Vértices no parâmetro t diretamente!

- Seja $\alpha(x_1, x_2, \dots, x_p) = a_0 + a_1x_1 + \dots + a_px_p$
- Considere ϕ que contenha variáveis livres sobre conjuntos (monádicas) X_1, \dots, X_p de vértices ou arestas.
- Considere o problema: minimizar $\alpha(|X_1|, \dots, |X_p|)$ tal que ϕ vale para G

Teorema

Existe um algoritmo que resolve o problema acima em tempo $f(\|\phi\|, t) \cdot n$, para alguma função f computável.

Caracterizações Alternativas

Graph searching

Graph searching

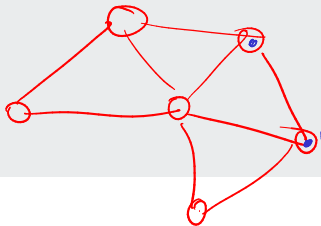
- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:

Graph searching

- ~~Um~~ ^{Variar} ~~detetive~~ (buscador) segue um ^{programa} programa, que é sequência de dois movimentos:
 - **posicionar:** um detetive livre posiciona-se em um determinado vértice

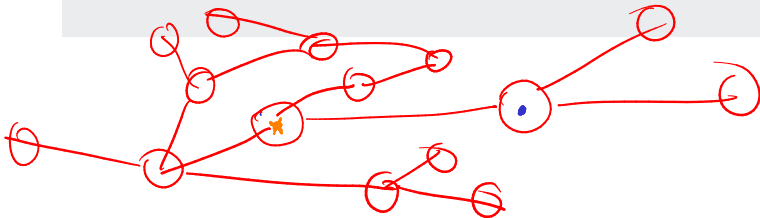
Graph searching

- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:
 - **posicionar**: um detetive livre posiciona-se em um determinado vértice
 - **remove**: um detetive posicionado se retira e fica livre



Graph searching

- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:
 - **posicionar**: um detetive livre posiciona-se em um determinado vértice
 - **remove**: um detetive posicionado se retira e fica livre
- O **fugitivo** está em um vértice e se move:
 - por arestas em velocidade **ilimitada**
 - não pode se **chocar** com um detetive posicionado



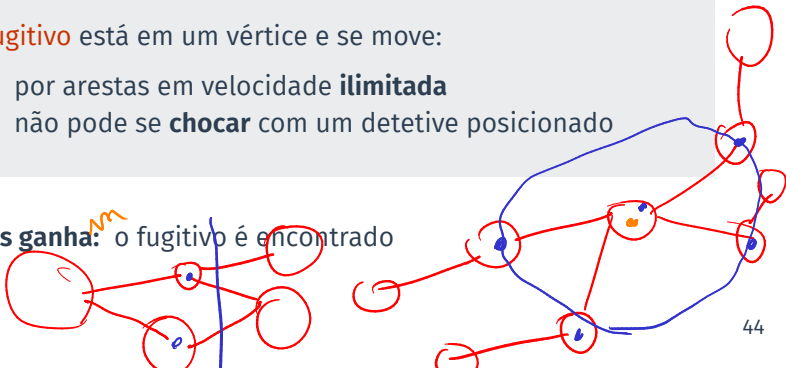
Graph searching

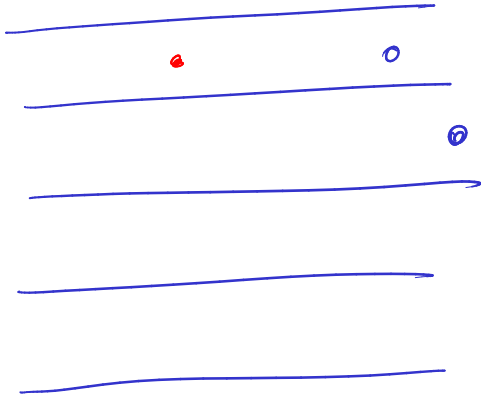
- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:
 - **posicionar**: um detetive livre posiciona-se em um determinado vértice
 - **remove**: um detetive posicionado se retira e fica livre
- O **fugitivo** está em um vértice e se move:
 - por arestas em velocidade **ilimitada**
 - não pode se **chocar** com um detetive posicionado

Graph searching

- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:
 - **posicionar**: um detetive livre posiciona-se em um determinado vértice
 - **remove**: um detetive posicionado se retira e fica livre
- O **fugitivo** está em um vértice e se move:
 - por arestas em velocidade **ilimitada**
 - não pode se **chocar** com um detetive posicionado

Detetives ganha: o fugitivo é encontrado





Graph searching

- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:
 - **posicionar**: um detetive livre posiciona-se em um determinado vértice
 - **remove**: um detetive posicionado se retira e fica livre
- O **fugitivo** está em um vértice e se move:
 - por arestas em velocidade **ilimitada**
 - não pode se **chocar** com um detetive posicionado

Detetives ganha: o fugitivo é encontrado

Fugitivo ganha: consegue fugir indefinidamente

Graph searching

- Um **detetive** (buscador) segue um **programa**, que é sequência de dois movimentos:
 - **posicionar**: um detetive livre posiciona-se em um determinado vértice
 - **remove**: um detetive posicionado se retira e fica livre
- O **fugitivo** está em um vértice e se move:
 - por arestas em velocidade **ilimitada**
 - não pode se **chocar** com um detetive posicionado

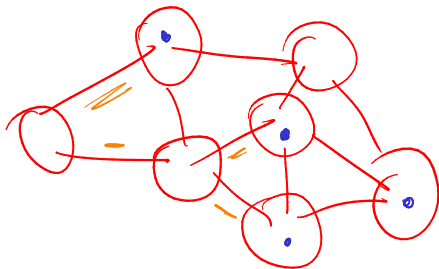
Detetives ganha: o fugitivo é encontrado

Fugitivo ganha: consegue fugir indefinidamente

Número de busca: menor número de detetives necessários

Contaminação

Se o grafo não tem vértices isolado, podemos imaginar que todas as arestas pelas quais o fugitivo pode andar estão **contaminadas com gás** que flui pelas arestas;



Contaminação

Se o grafo não tem vértices isolado, podemos imaginar que todas as arestas pelas quais o fugitivo pode andar estão **contaminadas com gás** que flui pelas arestas;

- uma aresta é descontaminada se dois vértices forem ocupados pelos **buscadores**

Contaminação

Se o grafo não tem vértices isolado, podemos imaginar que todas as arestas pelas quais o fugitivo pode andar estão **contaminadas com gás** que flui pelas arestas;

- uma aresta é descontaminada se dois vértices forem ocupados pelos **buscadores**
- uma aresta pode ser recontaminada se houver um caminho livre de uma aresta contaminada

Contaminação

Se o grafo não tem vértices isolado, podemos imaginar que todas as arestas pelas quais o fugitivo pode andar estão **contaminadas com gás** que flui pelas arestas;

- uma aresta é descontaminada se dois vértices forem ocupados pelos **buscadores**
- uma aresta pode ser recontaminada se houver um caminho livre de uma aresta contaminada

Teorema

7.3

Para todo grafo G , se k buscadores podem limpar G , então k buscadores podem limpar G de tal maneira que nenhuma aresta volte a se recontaminar.

Definição (Grafo de intervalos)

Um grafo G é de intervalo se:

Definição (Grafo de intervalos)

Um grafo G é de intervalo se:

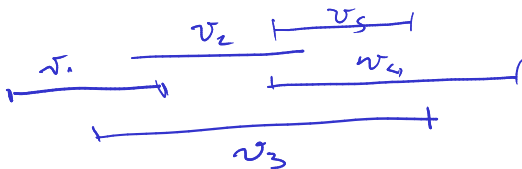
- para cada $v \in V$, existe intervalo real $I_v = [l_v, r_v]$

Grafo de intervalos

Definição (Grafo de intervalos)

Um grafo G é de intervalo se:

- para cada $v \in V$, existe intervalo real $I_v = [l_v, r_v]$
- para cada $u, v \in V, u \neq v$, temos^S que $uv \in E$ sss $I_v \cap I_u \neq \emptyset$.



Definição (Grafo de intervalos)

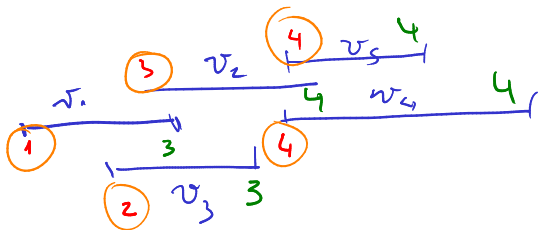
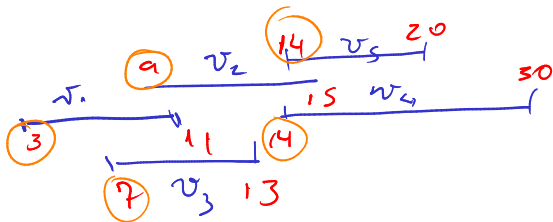
Um grafo G é de intervalo se:

- para cada $v \in V$, existe intervalo real $I_v = [l_v, r_v]$
- para cada $u, v \in V, u \neq v$, temos que $uv \in E$ sss $I_v \cap I_u \neq \emptyset$.
- os intervalos associados são ^{uma} representação de G

Definição (Grafo de intervalos)

Um grafo G é de intervalo se:

- para cada $v \in V$, existe intervalo real $I_v = [l_v, r_v]$
 - para cada $u, v \in V, u \neq v$, temos que $uv \in E$ sss $I_v \cap I_u \neq \emptyset$.
-
- os intervalos associados são a **representação** de G
 - a representação é **canônica** se $\{l_v : v \in V\} = \{1, 2, \dots, n\}$



Lembrando: G' é **supergrafo** de G , ou $G \subseteq G'$, se:

Largura de intervalos

Lembrando: G' é **supergrafo** de G , ou $G \subseteq G'$, se:

- $V(G) \subseteq V(G')$, e
- $E(G) \subseteq E(G')$.

Definição

A largura de intervalo de um grafo G é definida como:

$$\text{largura-intervalo}(G) = \min\{\omega(G') : G \subseteq G' \wedge G' \text{ é de intervalo}\}$$

→ ω = mensuração de intervalo

Largura de intervalos

Lembrando: G' é **supergrafo** de G , ou $G \subseteq G'$, se:

- $V(G) \subseteq V(G')$, e
- $E(G) \subseteq E(G')$.

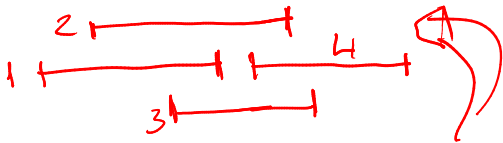
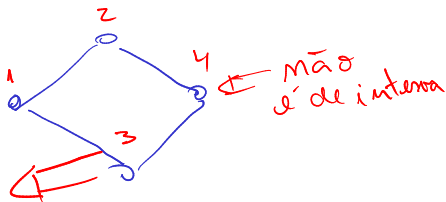
Definição

A largura de intervalo de um grafo G é definida como:

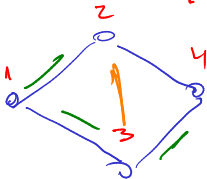
$$\text{largura-intervalo}(G) = \min\{\omega(G') : G \subseteq G' \wedge G' \text{ é de intervalo}\}$$

isso é, menor valor **número de clique** de um supergrafo G' de G que é de intervalo.

Ex $C_4 = 6$



G'



$\omega(G') = 3$

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

- (i) O número de busca de G é no máximo $k + 1$;*

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

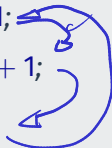
- (i) O número de busca de G é no máximo $k + 1$;*
- (ii) A largura de intervalo de G é no máximo $k + 1$;*

Equivalências de largura de caminho

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

- (i) O número de busca de G é no máximo $k + 1$;
- (ii) A largura de intervalo de G é no máximo $k + 1$;
- (iii) A largura de caminho de G é no máximo k .



(i) \Rightarrow (ii)

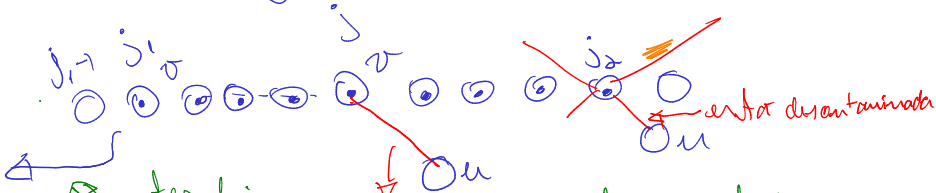
- Sup. q G n tem vertices isolados (SP.6)
- Considere a busca de 7.13 completa.
- Seja x_1, x_2, \dots, x_p os vertices marcados em ordem.
- Como todo vertex x_i tem uma aresta incidente, então v aparece no programa, pois sua aresta precisa ser desorientada.
- Podemos supor (SP6) q todo vertex x_i marcado.

Alg Existe programa em que todo vertex v é inserido e removido exatamente 1 vez.

Seja v . Seja j o momento em que uma arista incidente em v foi descontaminada pela i ésima vez.

\Rightarrow Seja j_1 t.q. $x_{j_1} = v$ $1 \leq j_1 \leq j$ e $j_1 \in \text{máx.}$

\Rightarrow Seja j_2 t.q. $x_{j_2} = v$ $j \leq j_2 \leq \text{mín.}$



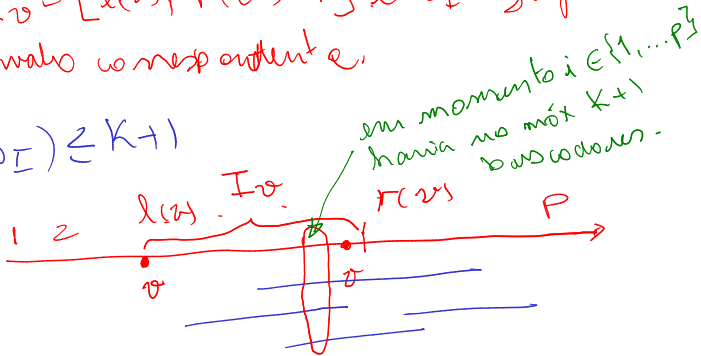
Antes de j_1 um detetive em v não descontaminava nada pela mínima multiplicidade de j , portanto esse não.

de pois de j_2 , nenhuma arista é recontaminada na vizinhança de j_2 e nem está contaminada do contrário, contaminaria (v, u) - brevemente movimentos post.

Seja $l(v)$ o momento que v é inserido e
 $r(v)$ o momento que v é removido.

Seja $I_v = [l(v), r(v) + 1]$ e G_I o grafo
de intervalos correspondente.

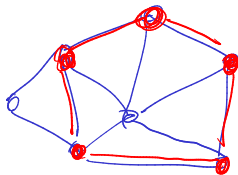
$$1) w(G_I) \leq K+1$$



2) $G_I \cong G$: todas arestas $uv \in G$ foram
simples, daí existe momento i em que
 u e v estavam ativos, daí $i \in I_u \cap I_v \Rightarrow uv \in E(G)$
 \square

Grafos cordais

Um grafo G é **cordal** se não contém um ciclo induzido de tamanho maior que 3.



← não é cordal

em particular, todo G de intervalo é cordal

Um grafo G é **cordal** se não contém um ciclo induzido de tamanho maior que 3.

⇒ todo ciclo com tamanho 4 ou mais tem uma **corda**.

Um grafo G é **cordal** se não contém um ciclo induzido de tamanho maior que 3.

⇒ todo ciclo com tamanho 4 ou mais tem uma **corda**.

Definição

A largura cordal de um grafo G é definida como:

$$\text{largura-cordal}(G) = \min\{\omega(G') : G \subseteq G' \wedge G' \text{ é cordal}\}$$

Um grafo G é **cordal** se não contém um ciclo induzido de tamanho maior que 3.

⇒ todo ciclo com tamanho 4 ou mais tem uma **corda**.

Definição

A largura cordal de um grafo G é definida como:

$$\text{largura-cordal}(G) = \min\{\omega(G') : G \subseteq G' \wedge G' \text{ é cordal}\}$$

isso é, menor valor **número de clique** de um supergrafo G' de G que é cordal.

Graph searching 2

Considere a seguinte busca:

Graph searching 2

Considere a seguinte busca:

- Um conjunto de **policiais** em um conjunto de vértices

Graph searching 2

Considere a seguinte busca:

- Um conjunto de **policiais** em um conjuntos de vértices
- Existe um **ladrão** cuja posição é conhecida em um vértice

Graph searching 2

Considere a seguinte busca:

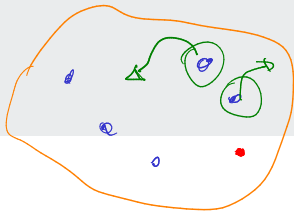
- Um conjunto de **policiais** em um conjuntos de vértices
- Existe um **ladrão** cuja posição é conhecida em um vértice
- Em um movimento, acontece o seguinte:

Outra busca em grafos

Graph searching 2

Considere a seguinte busca:

- Um conjunto de **policiais** em um conjuntos de vértices
- Existe um **ladrão** cuja posição é conhecida em um vértice
- Em um movimento, acontece o seguinte:
 - um subconjunto de policiais pode tomar um helicóptero e declara seus destinos

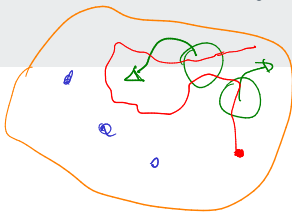


Outra busca em grafos

Graph searching 2

Considere a seguinte busca:

- Um conjunto de **policiais** em um conjuntos de vértices
- Existe um **ladrão** cuja posição é conhecida em um vértice
- Em um movimento, acontece o seguinte:
 - um subconjunto de policiais pode tomar um helicóptero e declara seus destinos
 - enquanto isso o ladrão pode mover-se sem passar por policiais, conhecendo a localização de todos policiais

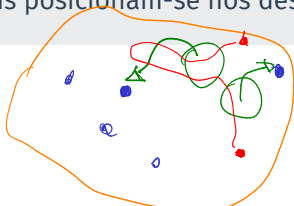


Outra busca em grafos

Graph searching 2

Considere a seguinte busca:

- Um conjunto de **policiais** em um conjuntos de vértices
- Existe um **ladrão** cuja posição é conhecida em um vértice
- Em um movimento, acontece o seguinte:
 - um subconjunto de policiais pode tomar um helicóptero e declara seus destinos
 - enquanto isso o ladrão pode mover-se sem passar por policiais, conhecendo a localização de todos policiais
 - os policiais posicionam-se nos destinos declarados



Outra busca em grafos

Graph searching 2

Considere a seguinte busca:

- Um conjunto de **policiais** em um conjunto de vértices
- Existe um **ladrão** cuja posição é conhecida em um vértice
- Em um movimento, acontece o seguinte:
 - um subconjunto de policiais pode tomar um helicóptero e declara seus destinos
 - enquanto isso o ladrão pode mover-se sem passar por policiais, conhecendo a localização de todos policiais
 - os policiais posicionam-se nos destinos declarados

Número de busca: menor número de policiais necessários para capturar ladrão.

Dizemos que conjuntos $A, B \subseteq V(G)$ tocam-se; *Se:*

Dizemos que conjuntos $A, B \subseteq V(G)$ tocam-se: *se!*

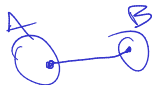
- $A \cap B \neq \emptyset$, ou

Dizemos que conjuntos $A, B \subseteq V(G)$ tocam se:

- $A \cap B \neq \emptyset$, ou



- existe $u \in A$ e $v \in B$ e $uv \in E(G)$.



Dizemos que conjuntos $A, B \subseteq V(G)$ tocam se:

- $A \cap B \neq \emptyset$, ou
- existe $u \in A$ e $v \in B$ e $uv \in E(G)$.

Definição (Bramble)

Uma *bramble* (espinheiro) \mathcal{B} é uma família de conjuntos de vértices tais que:

- conjuntos tocam-se dois a dois;
- cada conjunto induz uma componente conexa.

Dizemos que conjuntos $A, B \subseteq V(G)$ tocam se:

- $A \cap B \neq \emptyset$, ou
- existe $u \in A$ e $v \in B$ e $uv \in E(G)$.

Definição (Bramble)

Uma *bramble* (espinheiro) \mathcal{B} é uma família de conjuntos de vértices tais que:

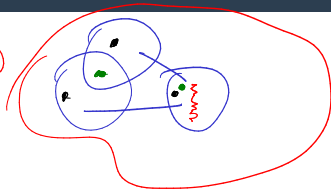
- conjuntos tocam-se dois a dois;
- cada conjunto induz uma componente conexa.

- $C \subseteq V(G)$ cobre uma \mathcal{B} se cada conjunto intersepta C

Espinheiro

Dizemos que conjuntos $A, B \subseteq V(G)$ tocam se:

- $A \cap B \neq \emptyset$, ou
- existe $u \in A$ e $v \in B$ e $uv \in E(G)$.



Definição (Bramble)

Uma *bramble* (espinheiro) \mathcal{B} é uma família de conjuntos de vértices tais que:

- conjuntos tocam-se dois a dois;
- cada conjunto induz uma componente conexa.

• cobre \mathcal{B}

• cobre \mathcal{B}

- $C \subseteq V(G)$ **cobre** uma \mathcal{B} se cada conjunto intercepta C
- a **ordem** de \mathcal{B} é o menor número de vértices que cobrem \mathcal{B}

Equivalências de largura de árvore

Teorema

Para todo $k \geq 0$ e grafo G :

- a largura de árvore de G é pelo menos k SSS
- G contém uma bramble de ordem pelo menos $k + 1$.

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

Equivalências de largura de árvore

Teorema

Para todo $k \geq 0$ e grafo G :

- a largura de árvore de G é pelo menos k **SSS**
- G contém uma bramble de ordem pelo menos $k + 1$.

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

- (i) A largura de árvore de G é no máximo k .

Equivalências de largura de árvore

Teorema

Para todo $k \geq 0$ e grafo G :

- a largura de árvore de G é pelo menos k **SSS**
- G contém uma bramble de ordem pelo menos $k + 1$.

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

- (i) A largura de árvore de G é no máximo k .
- (ii) A largura cordal de G é no máximo $k + 1$.

Equivalências de largura de árvore

Teorema

Para todo $k \geq 0$ e grafo G :

- a largura de árvore de G é pelo menos k SSS
- G contém uma bramble de ordem pelo menos $k + 1$.

Teorema

Para todo grafo G e $k \geq 0$, são equivalentes:

- (i) A largura de árvore de G é no máximo k .
- (ii) A largura cordal de G é no máximo $k + 1$.
- (iii) $k + 1$ policiais podem capturar um ladrão visível em G .

Equivalências de largura de árvore

Teorema

Para todo $k \geq 0$ e grafo G :

- a largura de árvore de G é pelo menos k SSS
- G contém uma bramble de ordem pelo menos $k + 1$.

Teorema

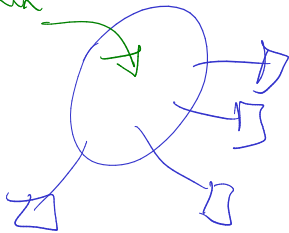
Para todo grafo G e $k \geq 0$, são equivalentes:

- (i) A largura de árvore de G é no máximo k .
- (ii) A largura cordal de G é no máximo $k + 1$.
- (iii) $k + 1$ policiais podem capturar um ladrão visível em G .
- (iv) Não existe bramble com ordem maior que $k + 1$ em G .

Deixam 7,6 pedoneres.

(Aprox. de larg. de ánsore)

Arv. Gen. Mínima
com K vért.



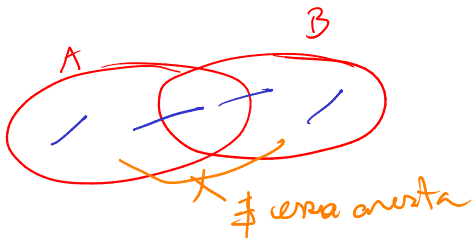
MAX # folhas
na árvore.

Obtendo uma decomposição em árvore

Separadores

Relembrando:

- **Separação:** (A, B) é uma separação de $A \cup B = V(G)$ se não há arestas entre $A \setminus B$ e $B \setminus A$



Separadores

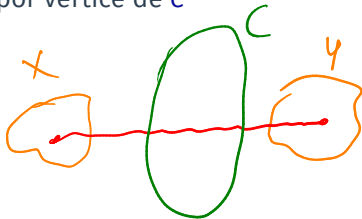
Relembrando:

- **Separação:** (A, B) é uma separação de $A \cup B = V(G)$ se não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:**

Separadores

Relembrando:

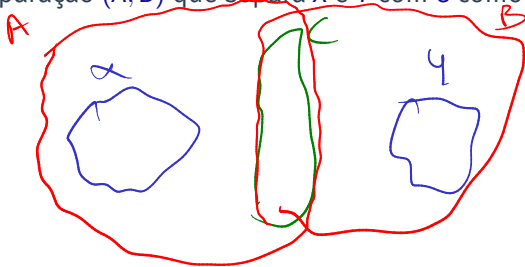
- **Separação:** (A, B) é uma separação de $A \cup B = V(G)$ se não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:**
 - dizemos que $C \subseteq V(G)$ **separa** X e Y se todo caminho X - Y passa por vértice de C



Separadores

Relembrando:

- **Separação:** (A, B) é uma separação de $A \cup B = V(G)$ se não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:**
 - dizemos que $C \subseteq V(G)$ **separa** X e Y se todo caminho X – Y passa por vértice de C
 - se C separa X e Y , então podemos encontrar uma separação (A, B) que separa X e Y com C como separador



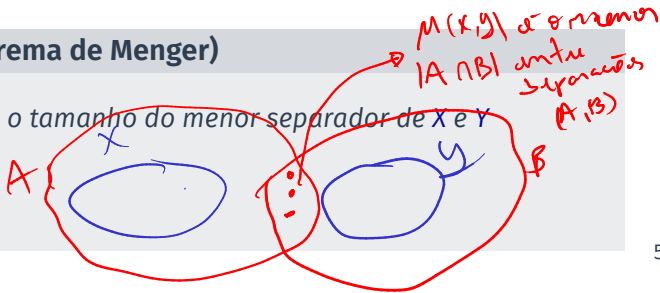
Separadores

Relembrando:

- **Separação:** (A, B) é uma separação de $A \cup B = V(G)$ se não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:**
 - dizemos que $C \subseteq V(G)$ **separa** X e Y se todo caminho $X-Y$ passa por vértice de C
 - se C separa X e Y , então podemos encontrar uma separação (A, B) que separa X e Y com C como separador

Teorema (Teorema de Menger)

- Seja $\mu(X, Y)$ o tamanho do menor separador de X e Y



Separadores

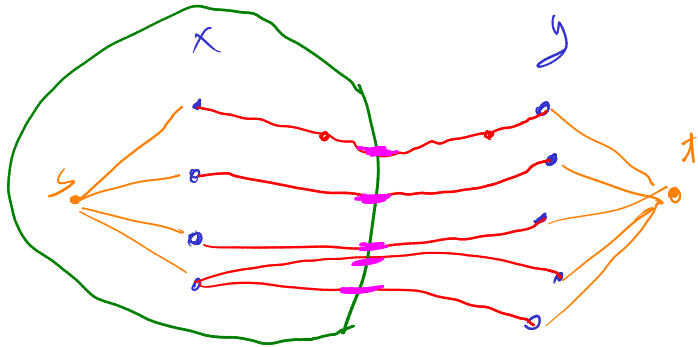
Relembrando:

- **Separação:** (A, B) é uma separação de $A \cup B = V(G)$ se não há arestas entre $A \setminus B$ e $B \setminus A$
- **Separador:**
 - dizemos que $C \subseteq V(G)$ **separa** X e Y se todo caminho X - Y passa por vértice de C
 - se C separa X e Y , então podemos encontrar uma separação (A, B) que separa X e Y com C como separador

Teorema (Teorema de Menger)

- Seja $\mu(X, Y)$ o tamanho do menor separador de X e Y
- Então $\mu(X, Y)$ é igual ao maior número de caminhos disjuntos nos vértices entre X e Y .

\Rightarrow Basta resolver junto para encontrar separação de ordem $\mu(X, Y)$



Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{\mathcal{O}(k^3)} \cdot n$ e:

- *ou constroi uma decomposição em árvore de largura k ;*

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{O(k^3)} \cdot n$ e:

- *ou* constroi uma decomposição em árvore de largura k ;
- *ou* conclui que a largura de árvore de G é maior que k .

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{\mathcal{O}(k^3)} \cdot n$ e:

- *ou constroi uma decomposição em árvore de largura k ;*
- *ou conclui que a largura de árvore de G é maior que k .*

Observações:

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{\mathcal{O}(k^3)} \cdot n$ e:

- *ou constroi uma decomposição em árvore de largura k ;*
- *ou conclui que a largura de árvore de G é maior que k .*

Observações:

- o algoritmo é linear em n ;

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{O(k^3)} \cdot n$ e:

- *ou* constroi uma decomposição em árvore de largura k ;
- *ou* conclui que a largura de árvore de G é maior que k .

Observações:

- o algoritmo é linear em n ;
- mas **não** é de exponencial simples em k ;

$$\begin{array}{l} 2^{\theta(2^3)} \approx 2^8 \\ 3^{\theta(3^3)} \approx 3^{27} \\ 4^{4^4} \approx 10^{38} \end{array}$$

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{\mathcal{O}(k^3)} \cdot n$ e:

- *ou* constroi uma decomposição em árvore de largura k ;
- *ou* conclui que a largura de árvore de G é maior que k .

Observações:

- o algoritmo é linear em n ;
- mas **não** é de exponencial simples em k ;
- o fator com k pode ser dominante

Obtendo uma decomposição

Existe um algoritmo **FPT** para obter uma decomposição em árvore:

Teorema (Boadlander)

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $k^{\mathcal{O}(k^3)} \cdot n$ e:

- *ou* constroi uma decomposição em árvore de largura k ;
- *ou* conclui que a largura de árvore de G é maior que k .

Observações:

- o algoritmo é linear em n ;
- mas **não** é de exponencial simples em k ;
- o fator com k pode ser dominante

Alternativa: obter decomposições “quase” ótimas

Obtendo uma decomposição aproximada

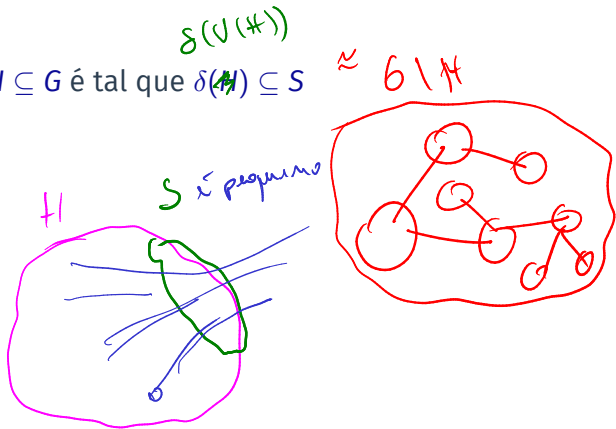
Como obter uma decomposição “pequena” de G ?

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S \approx G/H$



Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

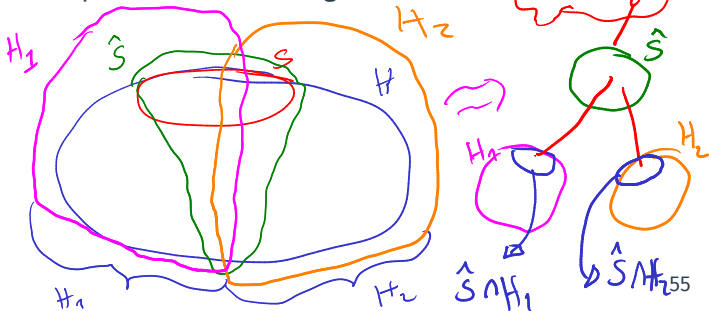
1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. Obtemos:

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. **Obtemos:**
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G



Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. Obtemos:
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G
4. Gostaríamos:

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. **Obtemos:**
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G
4. **Gostaríamos:**
 - $|V(H_1) \cap \hat{S}|$ e $|V(H_2) \cap \hat{S}|$ sejam “pequenos”

(ainda menor que S)

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. **Obtemos:**
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G
4. **Gostaríamos:**
 - $|V(H_1) \cap \hat{S}|$ e $|V(H_2) \cap \hat{S}|$ sejam “pequenos”
 - i.e., com tamanho menor que ck para c constante

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. **Obtemos:**
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G
4. **Gostaríamos:**
 - $|V(H_1) \cap \hat{S}|$ e $|V(H_2) \cap \hat{S}|$ sejam “pequenos”
 - i.e., com tamanho menor que ck para c constante
5. **Novos objetivos:**

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. **Obtemos:**
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G
4. **Gostaríamos:**
 - $|V(H_1) \cap \hat{S}|$ e $|V(H_2) \cap \hat{S}|$ sejam “pequenos”
 - i.e., com tamanho menor que ck para c constante
5. **Novos objetivos:**
 - dividir S entre H_1 e H_2

Obtendo uma decomposição aproximada

Como obter uma decomposição “pequena” de G ?

Ideia:

1. Suponha que $H \subseteq G$ é tal que $\delta(H) \subseteq S$
2. Quebramos H em H_1, H_2 com algum separador $\hat{S} \supseteq S$
3. **Obtemos:**
 - $V(H_1) \cap \hat{S}$ separa H_1 do resto do grafo G
 - $V(H_2) \cap \hat{S}$ separa H_2 do resto do grafo G
4. **Gostaríamos:**
 - $|V(H_1) \cap \hat{S}|$ e $|V(H_2) \cap \hat{S}|$ sejam “pequenos”
 - i.e., com tamanho menor que ck para c constante
5. **Novos objetivos:**
 - dividir S entre H_1 e H_2 *balanço na divisão*
 - encontrar \hat{S} não muito grande

Separadores balanceados

- seja $w : V(G) \rightarrow \mathbb{R}_+$ e $w(X) = \sum_{u \in X} w(u)$ para $X \subseteq V(G)$

Separadores balanceados

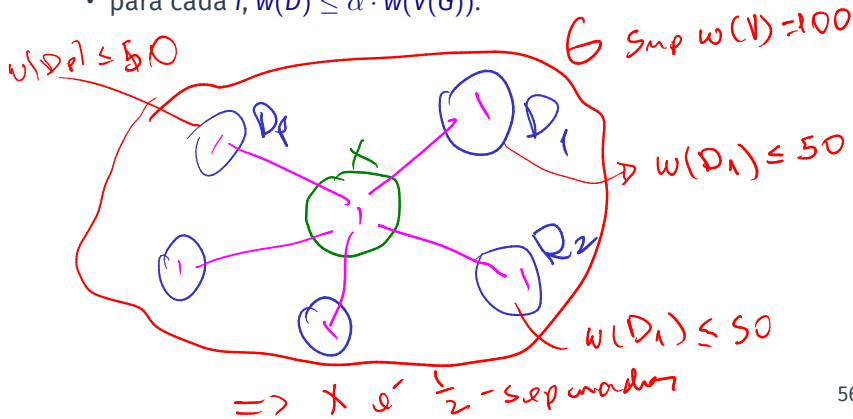
- seja $w : V(G) \rightarrow \mathbb{R}_+$ e $w(X) = \sum_{u \in X} w(u)$ para $X \subseteq V(G)$
- $X \subseteq V(G)$ é um α -separador balanceado em G se:

Separadores balanceados

- seja $w : V(G) \rightarrow \mathbb{R}_+$ e $w(X) = \sum_{u \in X} w(u)$ para $X \subseteq V(G)$
- $X \subseteq V(G)$ é um α -separador balanceado em G se:
 - $G - X$ contém componentes conexas D_1, \dots, D_p

Separadores balanceados

- seja $w : V(G) \rightarrow \mathbb{R}_+$ e $w(X) = \sum_{u \in X} w(u)$ para $X \subseteq V(G)$
- $X \subseteq V(G)$ é um α -separador balanceado em G se:
 - $G - X$ contém componentes conexas D_1, \dots, D_p
 - para cada i , $w(D_i) \leq \alpha \cdot w(V(G))$.

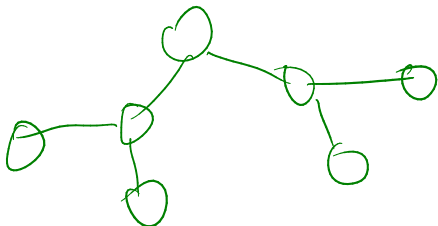


Separadores balanceados

- seja $w : V(G) \rightarrow \mathbb{R}_+$ e $w(X) = \sum_{u \in X} w(u)$ para $X \subseteq V(G)$
- $X \subseteq V(G)$ é um α -separador balanceado em G se:
 - $G - X$ contém componentes conexas D_1, \dots, D_p
 - para cada i , $w(D_i) \leq \alpha \cdot w(V(G))$.

Lema

Seja G com largura de árvore k e $w : V(G) \rightarrow \mathbb{R}_+$. Então existe um $\frac{1}{2}$ -separador balanceado X com $|X| \leq k + 1$.



Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

- $w(A \setminus B) \leq \alpha \cdot w(V(G))$ e

Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

- $w(A \setminus B) \leq \alpha \cdot w(V(G))$ e
- $w(B \setminus A) \leq \alpha \cdot w(V(G))$.

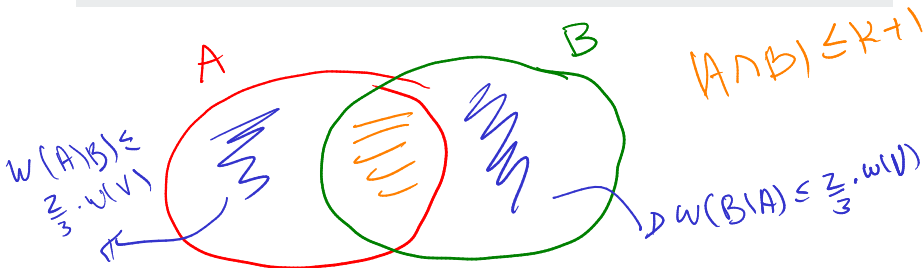
Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

- $w(A \setminus B) \leq \alpha \cdot w(V(G))$ e
- $w(B \setminus A) \leq \alpha \cdot w(V(G))$.

Lema

Seja G com largura de árvore k e $w : V(G) \rightarrow \mathbb{R}_+$. Então existe uma $\frac{2}{3}$ -separação balanceada (A, B) com ordem no máximo $k + 1$.



Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

- $w(A \setminus B) \leq \alpha \cdot w(V(G))$ e
- $w(B \setminus A) \leq \alpha \cdot w(V(G))$.

Lema

Seja G com largura de árvore k e $w : V(G) \rightarrow \mathbb{R}_+$. Então existe uma $\frac{2}{3}$ -separação balanceada (A, B) com ordem no máximo $k + 1$.

Corolário

Seja G com largura de árvore k e $S \subseteq V(G)$. Então existe uma partição S_A, S_B de S tal que:

Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

- $w(A \setminus B) \leq \alpha \cdot w(V(G))$ e
- $w(B \setminus A) \leq \alpha \cdot w(V(G))$.

Lema

Seja G com largura de árvore k e $w : V(G) \rightarrow \mathbb{R}_+$. Então existe uma $\frac{2}{3}$ -separação balanceada (A, B) com ordem no máximo $k + 1$.

Corolário

Seja G com largura de árvore k e $S \subseteq V(G)$. Então existe uma partição S_A, S_B de S tal que:

- $k + 2 \leq |S_A|, |S_B| \leq 2k + 2$ e

Separações balanceadas

Uma separação (A, B) é um α -balanceada se:

- $w(A \setminus B) \leq \alpha \cdot w(V(G))$ e
- $w(B \setminus A) \leq \alpha \cdot w(V(G))$.

Lema

Seja G com largura de árvore k e $w : V(G) \rightarrow \mathbb{R}_+$. Então existe uma $\frac{2}{3}$ -separação balanceada (A, B) com ordem no máximo $k + 1$.

Corolário

Seja G com largura de árvore k e $S \subseteq V(G)$. Então existe uma partição S_A, S_B de S tal que:

- $k + 2 \leq |S_A|, |S_B| \leq 2k + 2$ e
- $\mu_G(S_A, S_B) \leq k + 1$.

$$49 \quad |S| = 3k + 4$$

Teorema

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $\mathcal{O}(8^k k^2 \cdot n^2)$ e:

- ou constroi uma decomposição em árvore de largura k ;

$4k+4$

Teorema

Existe algoritmo que, dado um grafo G com n vértices e um inteiro k , executa em tempo $\mathcal{O}(8^k k^2 \cdot n^2)$ e:

- ou constroi uma decomposição em árvore de largura ~~k~~ ;
- ou conclui que a largura de árvore de G é maior que k .

$4k \rightarrow 4$

Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

SAÍDA: a decomposição de $G[W]$
em árvore com largura $\leq K+4$
e raiz r com $x_r \in S$.
(se concluir que $f_w(b) > K$.)

Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

INVARIANTES:

Algoritmo

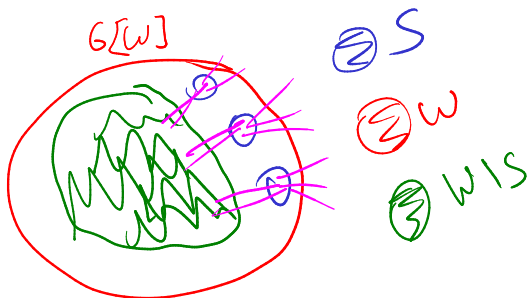
ENTRADA: $W \subseteq V(G), S \subseteq W$

INVARIANTES: (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;

Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

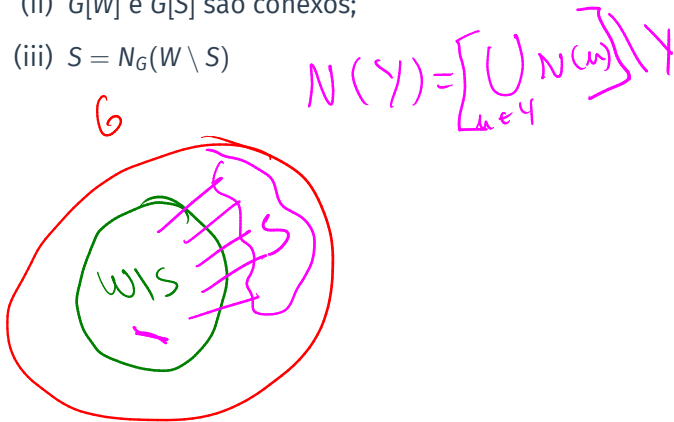
- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
w \setminus s
 - (ii) $G[W]$ e $G[\setminus]$ são conexos;



Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$



Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$

decompose(W, S):

1. Construa um conjunto \hat{S} com as propriedades:

Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$

decompose(W, S):

1. Construa um conjunto \hat{S} com as propriedades:
 - (a) $S \subsetneq \hat{S} \subseteq W$;

Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$

decompose(W, S):

1. Construa um conjunto \hat{S} com as **propriedades**:
 - (a) $S \subsetneq \hat{S} \subseteq W$;
 - (b) $|\hat{S}| \leq 4k + 5$;

Algoritmo

ENTRADA: $W \subseteq V(G)$, $S \subseteq W$

- INVARIANTES:**
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$

decompose(W, S):

1. Construa um conjunto \hat{S} com as **propriedades**:
 - (a) $S \subsetneq \hat{S} \subseteq W$;
 - (b) $|\hat{S}| \leq 4k + 5$;
 - (c) cada componente de $G[W \setminus \hat{S}]$ tem até $3k + 4$ vizinho em \hat{S}

Algoritmo

ENTRADA: $W \subseteq V(G)$, $S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$

decompose(W, S):

1. Construa um conjunto \hat{S} com as propriedades:
 - (a) $S \subsetneq \hat{S} \subseteq W$;
 - (b) $|\hat{S}| \leq 4k + 5$;
 - (c) cada componente de $G[W \setminus \hat{S}]$ tem até $3k + 4$ vizinho em \hat{S}
2. Sejam D_1, \dots, D_p componentes de $G[W \setminus \hat{S}]$

Algoritmo

ENTRADA: $W \subseteq V(G), S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$



decompose(W, S):

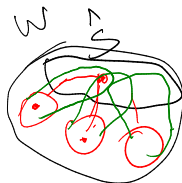
1. Construa um conjunto \hat{S} com as propriedades:
 - (a) $S \subsetneq \hat{S} \subseteq W$;
 - (b) $|\hat{S}| \leq 4k + 5$;
 - (c) cada componente de $G[W \setminus \hat{S}]$ tem até $3k + 4$ vizinho em \hat{S}
2. Sejam D_1, \dots, D_p componentes de $G[W \setminus \hat{S}]$
 - 2.1 Para cada $i, \mathcal{T}_i \leftarrow \text{decompose}(N_G[D_i], N_G(D_i))$

em partição \hat{S}
 $N_G(D_i) \subseteq \hat{S}$

Algoritmo

ENTRADA: $W \subseteq V(G)$, $S \subseteq W$

- INVARIANTES:
- (i) $|S| \leq 3k + 4$ e $W \setminus S \neq \emptyset$;
 - (ii) $G[W]$ e $G[S]$ são conexos;
 - (iii) $S = N_G(W \setminus S)$



decompose(W, S):

1. Construa um conjunto \hat{S} com as propriedades:
 - (a) $S \subsetneq \hat{S} \subseteq W$;
 - (b) $|\hat{S}| \leq 4k + 5$;
 - (c) cada componente de $G[W \setminus \hat{S}]$ tem até $3k + 4$ vizinho em \hat{S}
2. Sejam D_1, \dots, D_p componentes de $G[W \setminus \hat{S}]$
 - 2.1 Para cada i , $\mathcal{T}_i \leftarrow \text{decompose}(N_G[D_i], N_G(D_i))$
 - 2.2 Crie uma nó r com conjunto $X_r = \hat{S}$
 - 2.3 Devolva $\mathcal{T}_{W,S}$ com raiz r e filhos \mathcal{T}_i , para cada i

o Princípio sup. q. incontáveis \hat{S} com (a), (b), (c).

Ag. Du fato \mathcal{I}_{ws} é de comp. com largura $4k+4$.

\Rightarrow Para cada D_i vale:

(i) por (c) temos $|D_i| \leq 3k+4 \checkmark$

(ii) $G[D_i]$ é conexo e $D_i = \underbrace{N[D_i]}_{W_i} \setminus \underbrace{N(D_i)}_{S_i}$

$G[N[D_i]]$ é conexo

(iii) $S_i = N(D_i) = N(\underbrace{N[D_i]}_{W_i} \setminus \underbrace{N(D_i)}_{S_i}) = N(W_i \setminus S_i)$.

\Rightarrow Temos $|X_r| = |\hat{S}| \leq 4k+5$ (por (b))

e todo outro nó t tem $X_t \leq 4k+5$ por indução.

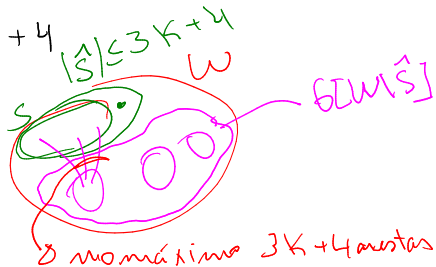
$\Rightarrow X_r$ de fato separa as subdecomposições.

a afirmação está OK.

• Vamos obter \hat{S} .

Caso 1: Se $|S| < 3K+4$

\Rightarrow basta adicionar
um vértice
arbitrário



Caso 2: Senão, temos $|S| = 3K+4$

\Rightarrow temos que $tw(G[W]) \leq tw(G)$

\Rightarrow Se $tw(G) \leq K$, pelo corolário
existe partição de S , (S_A, S_B) tq:

- $K+2 \leq |S_A|, |S_B| \leq 2K+2$

- $M(S_A, S_B) \leq tw(G[W]) + 1 \leq K+1$

• Não sabemos quem é $S_A \cap S_B$, mas $S_A \cup S_B = S$ e $|S| \leq 3K+4$ então enumeramos as 2^{3K+4} partições.

• Para cada parte $S_A \cap S_B$ verificamos se existe $K+1$ caminhos disjuntos em vértices usando o fluxo.

\Rightarrow Se encontrei, continue.

\Rightarrow Se não encontrei, então $tw(b) > K$ e termine o algoritmo.

• Encontramos separação (A, B) com separador $A \cap B$ tal que $|A \cap B| \leq K+1$

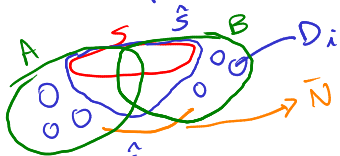
• defina $\hat{S} = S \cup (A \cap B)$.

Vamos mostrar q \hat{S} satisfaz b, c, a:

$$(b) |\hat{S}| = |SU(A \cap B)| \leq |S| + |A \cap B| \in$$

$$\in \underbrace{3k+4}_{\text{invariante}} + k-1 = 4k+5.$$

(c) Seja D_i uma comp. conexa $G[W|\hat{S}]$.



Temos que $D_i \subseteq A \cap \hat{S}$ ou
 $D_i \subseteq B \cap \hat{S}$.

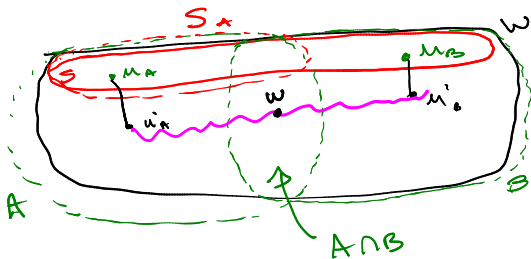
Pontamento:

$$\text{ou } N(D_i) \cap \hat{S} \subseteq A \cap B \cup SA$$

$$= |N(D_i) \cap \hat{S}| \in k+1 + 2k+2 \in 3k+3$$

ou $N(D_i) \cap \hat{S} \subseteq A \cap B \cup S \Rightarrow$

$$(a) \hat{S} \neq S$$



- $|S_A|, |S_B| \geq K+2$ e $|A \cap B| \leq K+1$
 $\Rightarrow \exists m_A \in S_A \setminus (A \cap B)$ e $\exists u_B \in S_B \setminus (A \cap B)$.
- Sejam m'_A e m'_B vizinhos de m_A e m_B em $W \setminus S$,
 onde existe pois $G[W \setminus S]$ é conexo a $W \setminus S$.
- Como $A \cap B$ separa S_A e $S_B \Rightarrow m'_A \in A$ e $m'_B \in B$.
- Considere um caminho entre m'_A e m'_B em
 $G[W \setminus S]$, onde existe pois $G[W \setminus S]$ é conexo.
- Portanto existe $w \in A \cap B \setminus S$, já que $A \cap B$ separa
 m'_A e m'_B .

Tempo de cada aresta:

\Rightarrow em cada chamada da recursão associamos um vértice novo ao último do S.

\Rightarrow cada vértice só é associado a uma chamada no máximo.

\Rightarrow Portanto, há no máximo n chamadas.

Tempo por nó: \rightarrow cada aresta 2^{3K+4} fluxos

\rightarrow cada fluxo gera

$\rightarrow (n+m)$ por iteração

$\rightarrow (K+1)$ iterações.

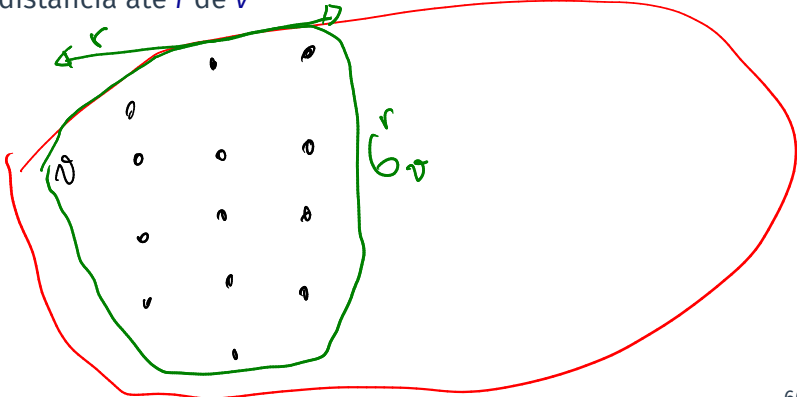
Se $m > Kn$, $tw(G) > K$. Supomos que $m \leq Kn$.

$$\text{TOTAL} = \underbrace{2^{3K+4}}_{\text{em cada}} \cdot \underbrace{K^2 n}_{\text{fluxo}} \cdot \frac{n}{\text{nós}} = \Theta(8^K K^2 n^2)$$

Shifting

Seja $r \geq 0$ e v um vértice de G .

- denotamos por G_v^r o subgrafo induzido por vértices com distância até r de v

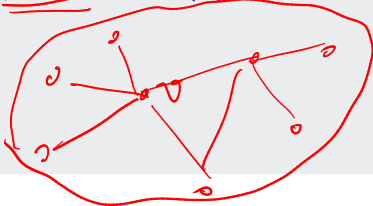


Seja $r \geq 0$ e v um vértice de G .

- denotamos por G_v^r o subgrafo induzido por vértices com distância até r de v

Teorema

Seja G um grafo planar e $v \in V(G)$. Então



Seja $r \geq 0$ e v um vértice de G .

- denotamos por G_v^r o subgrafo induzido por vértices com distância até r de v

Teorema

Seja G um grafo planar e $v \in V(G)$. Então

- $tw(G_v^r) \leq 3r + 1$

Seja $r \geq 0$ e v um vértice de G .

- denotamos por G_v^r o subgrafo induzido por vértices com distância até r de v

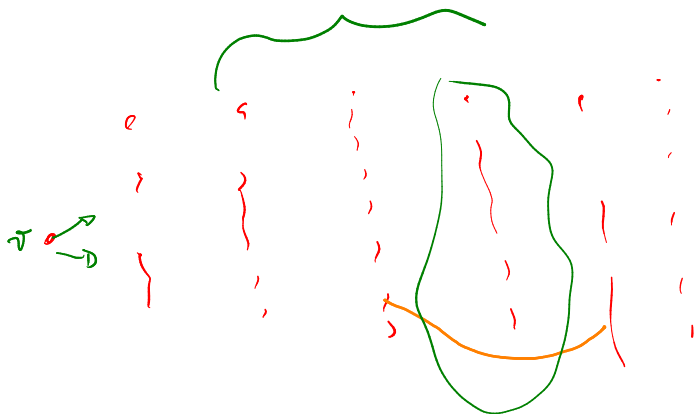
Teorema

Seja G um grafo planar e $v \in V(G)$. Então

- $tw(G_v^r) \leq 3r + 1$;
- *uma decomposição em árvore com largura $3r + 1$ pode ser encontrada em tempo polinomial.*

Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .



Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

- L_i são os vértices u tais que $d(v, u) = i$;

Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

- L_i são os vértices u tais que $d(v, u) = i$;
- $G_{i,i+j-1} = G[L_i \cup L_{i+1} \cup \dots \cup L_{i+j-1}]$.

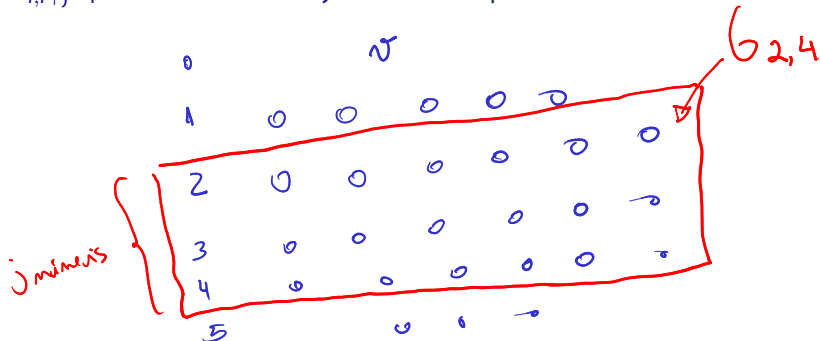
Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

- L_i são os vértices u tais que $d(v, u) = i$;
- $G_{i,i+j-1} = G[L_i \cup L_{i+1} \cup \dots \cup L_{i+j-1}]$.

$G_{i,i+j-1}$ é uma fatia com j camadas a partir de i



Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

- L_i são os vértices u tais que $d(v, u) = i$;
- $G_{i,i+j-1} = G[L_i \cup L_{i+1} \cup \dots \cup L_{i+j-1}]$.

$G_{i,i+j-1}$ é uma fatia com j camadas a partir de i

Corolário

Para cada $i \geq 0$ e $j \geq 1$:

Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

- L_i são os vértices u tais que $d(v, u) = i$;
- $G_{i,i+j-1} = G[L_i \cup L_{i+1} \cup \dots \cup L_{i+j-1}]$.

$G_{i,i+j-1}$ é uma fatia com j camadas a partir de i

Corolário

Para cada $i \geq 0$ e $j \geq 1$:

- $tw(G_{i,i+j-1}) \leq 3j + 1$;

Fatiamento do grafo

Consideramos uma fatia de um grafo planar G a partir de um vértice v .

Seja $i \geq 0$ e $j \geq 1$:

- L_i são os vértices u tais que $d(v, u) = i$;
- $G_{i,i+j-1} = G[L_i \cup L_{i+1} \cup \dots \cup L_{i+j-1}]$.

$G_{i,i+j-1}$ é uma fatia com j camadas a partir de i

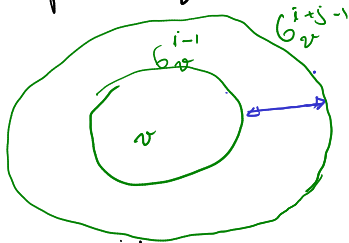
Corolário

Para cada $i \geq 0$ e $j \geq 1$:

- $tw(G_{i,i+j-1}) \leq 3j + 1$;
- *uma tal decomposição pode ser obtida em tempo polinomial.*

Prova

- Considere o grupo G_v^{i+j-1}
- Observe que G_v^{i-1} é um subgrupo induzido.



- Conjugue G_v^{i-1} em v^* e obtenha H :
 - H é planar
 - $V(G_{i, i+j-1}^v)$ estão a distância $2i+j$ de v^* em H .
- Pelo teorema anterior, obtenha uma decomposição de H com largura $3 \cdot j + 1$
- Adapte essa decomposição removendo v^* .

D

Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

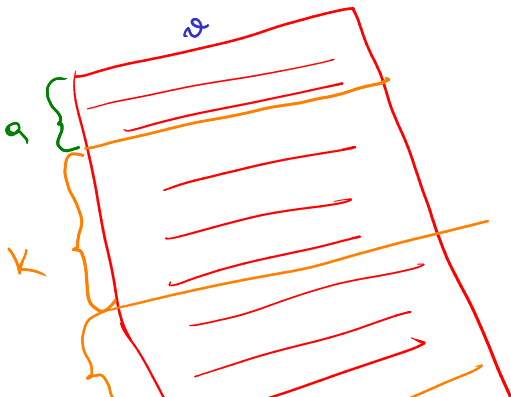
1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$

Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:



Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:
 - 2.1 removemos uma camada;

Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:
 - 2.1 removemos uma camada;
 - 2.2 cortamos uma fatia com k camadas.

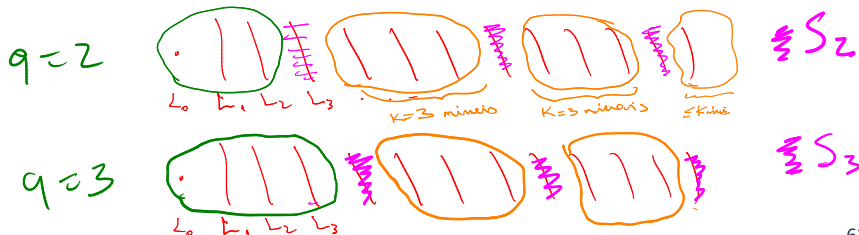


Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:
 - 2.1 removemos uma camada;
 - 2.2 cortamos uma fatia com k camadas.
3. Repetimos o mesmo procedimento para cada $q = 0, \dots, k$



Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:
 - 2.1 removemos uma camada;
 - 2.2 cortamos uma fatia com k camadas.
3. Repetimos o mesmo procedimento para cada $q = 0, \dots, k$ (i.e., deslocamos as camadas)

Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:
 - 2.1 removemos uma camada;
 - 2.2 cortamos uma fatia com k camadas.
3. Repetimos o mesmo procedimento para cada $q = 0, \dots, k$ (i.e., deslocamos as camadas)

Pontos-chaves:

- em algum deslocamento, nenhum dos k vértices da solução foi removido;

pois S_0, S_1, \dots, S_k
são disjuntas e, portanto, existe
 S_i que não contém vértice da sol⁶²

Método Shifting

Suponha que temos um problema de encontrar um subconjunto de até k vértices.

Ideia:

1. Primeiro cortamos uma fatia com q camadas, $0 \leq q \leq k$
2. Fatiamos o resto de G fazendo:
 - 2.1 removemos uma camada;
 - 2.2 cortamos uma fatia com k camadas.
3. Repetimos o mesmo procedimento para cada $q = 0, \dots, k$ (i.e., deslocamos as camadas)

Pontos-chaves:

- em algum deslocamento, nenhum dos k vértices da solução foi removido;
- o grafo resultante de cada deslocamento tem largura de árvore pequena

Shifting lemma

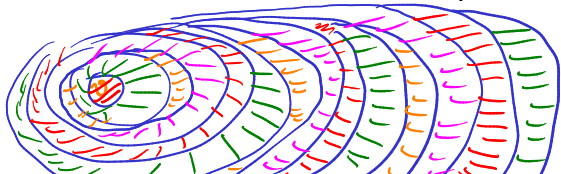
Lema

Seja G planar e $k \geq 0$. Então o conjunto de vértices de G pode ser particionado em $k + 1$ subconjuntos (possivelmente vazios) tais que a **remoção** de qualquer um deles induz um grafo com largura de árvore no máximo $3k + 1$. Ainda, a partição bem como uma tal decomposição pode ser obtida em tempo polinomial.

Prova: • Suponha G a ser conexo.

• Seja $S_j = \text{conj. de vértices à distância } a(k+1)+j$ para algum $a=0,1,\dots$

$j=1$
 $k=3$



S_0
 S_1
 S_2
 S_3

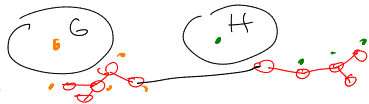
• Observa que $(S_j)_{0 \leq j \leq k}$ particiona $V(G)$.

• Assim, $G - S_j$ é umizo disjunta de

$$G_{(k+1)a+j+1, (k+1)a+1+j-1} \text{ e } G_{j-1}$$

• Para cada um deles construímos uma decomposição em árvore com altura $3 \cdot k + 1$. (pelo lema anterior)

• Agora basta conectar as decomposições arbitrariamente.



• Se G for desconexo:

- obtenho uma decomp. para cada comp.
- depois conecto arbitrariamente. \square

Problema do isomorfismo de subgrafo

Relembrando: No problema do isomorfismo, dado um grafo H com tamanho fixo k e um grafo G , existe um subgrafo de G isomorfo a H ?

Problema do isomorfismo de subgrafo

Relembrando: No problema do isomorfismo, dado um grafo H com tamanho fixo k e um grafo G , existe um subgrafo de G isomorfo a H ?

Primeiro-passo: encontrar um algoritmo **FPT** para o caso particular do problema com largura de árvore fixa.

Problema do isomorfismo de subgrafo

Relembrando: No problema do isomorfismo, dado um grafo H com tamanho fixo k e um grafo G , existe um subgrafo de G isomorfo a H ?

Primeiro-passo: encontrar um algoritmo **FPT** para o caso particular do problema com largura de árvore fixa.

Lema

Existe um algoritmo que, dado um grafo G e um grafo H , decide se existe um subgrafo de G isomorfo a H .

Problema do isomorfismo de subgrafo

Relembrando: No problema do isomorfismo, dado um grafo H com tamanho fixo k e um grafo G , existe um subgrafo de G isomorfo a H ?

Primeiro-passo: encontrar um algoritmo **FPT** para o caso particular do problema com largura de árvore fixa.

Lema

Existe um algoritmo que, dado um grafo G e um grafo H , decide se existe um subgrafo de G isomorfo a H . O tempo de execução é $f(|V(H)|, tw(G)) \cdot |V(G)|^{O(1)}$, onde f é uma função computável que depende de $|V(H)|$.

k largura de árvore.

Obtendo um FPT com parâmetro k em grafos planares

Teorema

*Existe um algoritmo que, dado um grafo **planar** G e um grafo H , decide se existe um subgrafo de G isomorfo a H .*

Obtendo um FPT com parâmetro k em grafos planares

Teorema

Existe um algoritmo que, dado um grafo *planar* G e um grafo H , decide se existe um subgrafo de G isomorfo a H . O tempo de execução é $f(|V(H)|) \cdot |V(G)|^{O(1)}$, onde f é uma função computável que depende de $|V(H)|$.

FPT SÓ em k

Prova

- Como vemos o shift, Lemma 2 obtemos conjuntos $(S_j)_{0 \leq j \leq K}$
- Sup. que exista $\hat{H} \in G$ tq $\hat{H} \cong H$.
- Como $|V(\hat{H})| = K$, exista S_j tq $S_j \cap V(\hat{H}) = \emptyset$.
- Portanto $\hat{H} \in G[V \setminus S_j]$.
- Como temos uma d.c. em átomos de $G - S_j$ com largura $3K+1$, usamos o algoritmo anterior e decidimos se de fato $\hat{H} \in G - S_j$.
- Testamos para cada $G - S_j, 0 \leq j \leq K$, obtemos o resultado.

Problema da Bissecção Mínima

Dado um grafo G com n vértices e um inteiro k .

Problema da Bissecção Mínima

Dado um grafo G com n vértices e um inteiro k .

Pergunta: Existe uma partição de $V(G)$ em A e B tal que :

- $\lfloor n/2 \rfloor \leq |A|, |B| \leq \lceil n/2 \rceil$ e

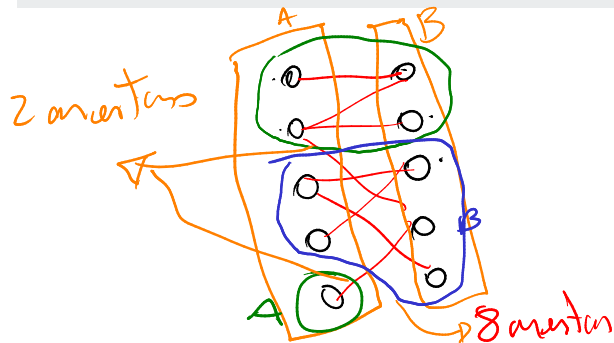
Bissecção mínima

Problema da Bissecção Mínima

Dado um grafo G com n vértices e um inteiro k .

Pergunta: Existe uma partição de $V(G)$ em A e B tal que :

- $\lfloor n/2 \rfloor \leq |A|, |B| \leq \lceil n/2 \rceil$ e
- o número de arestas entre A e B é no máximo k ?



Bissecção mínima

Problema da Bissecção Mínima

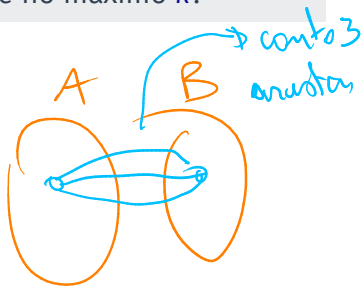
Dado um grafo G com n vértices e um inteiro k .

Pergunta: Existe uma partição de $V(G)$ em A e B tal que :

- $\lfloor n/2 \rfloor \leq |A|, |B| \leq \lceil n/2 \rceil$ e
- o número de arestas entre A e B é no máximo k ?

Consideraremos uma versão mais geral:

- Pode haver arestas múltiplas;



Bissecção mínima

Problema da Bissecção Mínima

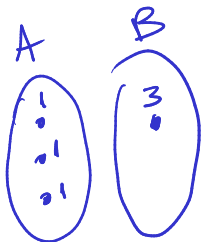
Dado um grafo G com n vértices e um inteiro k .

Pergunta: Existe uma partição de $V(G)$ em A e B tal que :

- $\lfloor n/2 \rfloor \leq |A|, |B| \leq \lceil n/2 \rceil$ e \leftarrow
- o número de arestas entre A e B é no máximo k ?

Consideraremos uma versão mais geral:

- Pode haver arestas múltiplas;
- Cada vértice v tem um peso inteiro $w(v) \geq 0$;



Problema da Bissecção Mínima

Dado um grafo G com n vértices e um inteiro k .

Pergunta: Existe uma partição de $V(G)$ em A e B tal que :

- $\lfloor n/2 \rfloor \leq |A|, |B| \leq \lceil n/2 \rceil$ e
- o número de arestas entre A e B é no máximo k ?

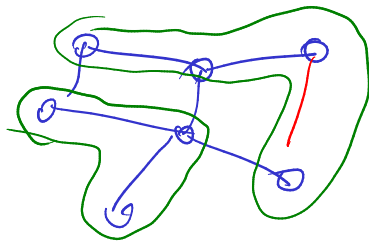
Consideraremos uma versão mais geral:

- Pode haver arestas múltiplas;
- Cada vértice v tem um peso inteiro $w(v) \geq 0$;
- A e B são tais que $\lfloor w(V(G))/2 \rfloor \leq s(A), s(B) \leq \lceil w(V(G))/2 \rceil$

Um lema de shifting com contração de arestas

Lema

Seja G planar e $k \geq 0$. Então o conjunto de arestas de G pode ser particionado em $k + 1$ subconjuntos tais que a contração de qualquer um deles induz um grafo com largura de árvore no máximo ck , para alguma constante $c > 0$. Ainda, a partição bem como uma tal decomposição pode ser obtida em tempo polinomial.



Um lema de shifting com contração de arestas

Lema

Seja G planar e $k \geq 0$. Então o conjunto de arestas de G pode ser particionado em $k + 1$ subconjuntos tais que a **contração** de qualquer um deles induz um grafo com largura de árvore no máximo ck , para alguma constante $c > 0$. Ainda, a partição bem como uma tal decomposição pode ser obtida em tempo polinomial.

Combinamos com um algoritmo para largura de árvore fixa.

Um lema de shifting com contração de arestas

Lema

Seja G planar e $k \geq 0$. Então o conjunto de arestas de G pode ser particionado em $k + 1$ subconjuntos tais que a **contração** de qualquer um deles induz um grafo com largura de árvore no máximo ck , para alguma constante $c > 0$. Ainda, a partição bem como uma tal decomposição pode ser obtida em tempo polinomial.

Combinamos com um algoritmo para largura de árvore fixa.

Lema

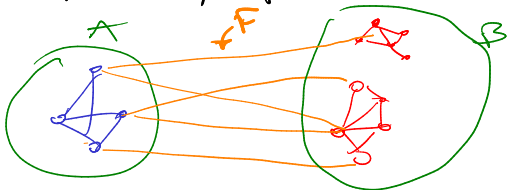
O Problema da Bisseção Mínima pode ser resolvido em tempo $2^t \cdot W \cdot n^{O(1)}$ em multigrafo com n vértices com largura de árvore t e cujo maior peso de um vértice é W .

FPT para Bisseção Mínima planar

Teorema

Bisseção Mínima em grafos planares pode ser resolvida em tempo $2^{O(k)} \cdot W \cdot n^{O(1)}$, onde W é o maior peso de um vértice.

Prova: \Leftarrow Sup. que exista solução pl K -bisseção (A, B) .



- Como conds o lema anterior e obtemos uma partição de $E(G)$, $(S_j)_{0 \leq j \leq K}$.
- Como a sol F tem no máx K arestas, existe $S_j \neq \emptyset$ $S_j \cap F = \emptyset$.

• Realize as seguintes operações para $(u, v) \in S_j$.

1) Contraiu (u, v) em y

2) Remova os laços

3) $w(y) = w(u) + w(v)$.

• Seja b_j o grafo obtido. Claramente, se

b tem K -bisseção, então b_j .

No sentido contrário, se (A', B') é uma K -biss.

de b_j , então obtemos uma K -biss. de b (A, B)

bastando p/ isso desconectar os vértices de A' e B' .

• Portanto, a redução é válida:

b tem K -biss $\Leftrightarrow b_j$ tem K -biss.

• Agora, para cada b_j , temos uma decomposição em árvore de largura $< K$ e usamos o algoritmo anterior.

□

The

End