Algoritmos Parametrizados

Árvores de busca limitadas

Lehilton Pedrosa

Segundo Semestre de 2018

Instituto de Computação – Unicamp

Roteiro

1. Ramificação
2. Cobertura por vértices
3. Recorrências
4. Conjunto de vértices de retroalimentação

Ramificação

Árvore de busca

Ideia: *backtracking*

Árvore de busca

Ideia: *backtracking*

- Fazer uma sequência de decisões

Árvore de busca

Ideia: *backtracking*

- Fazer uma sequência de decisões
- Em cada decisão, obter um subproblema

Árvore de busca

Ideia: *backtracking*

- Fazer uma sequência de decisões
- Em cada decisão, obter um subproblema
- Uma folha representa uma solução

Árvore de busca

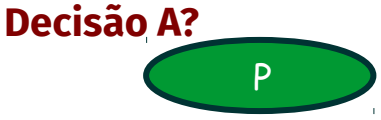
Ideia: *backtracking*

- Fazer uma sequência de decisões
- Em cada decisão, obter um subproblema
- Uma folha representa uma solução (ou a inexistência dela)

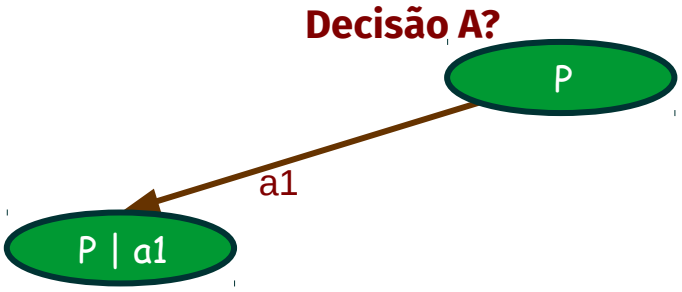
Árvore de busca

Árvore de busca

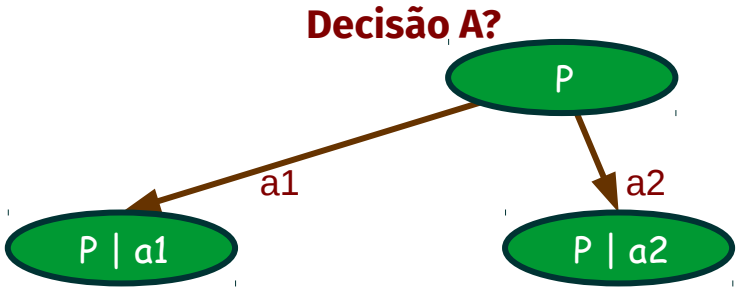
Árvore de busca



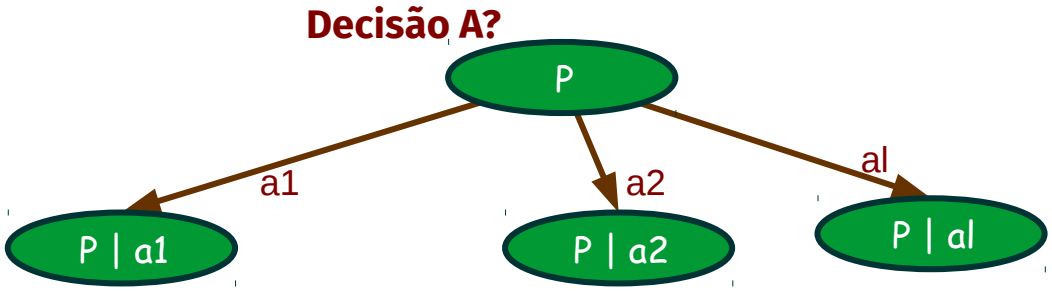
Árvore de busca



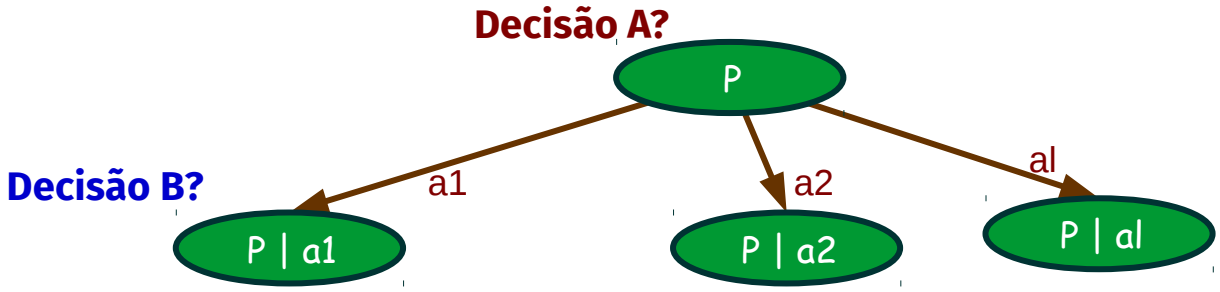
Árvore de busca



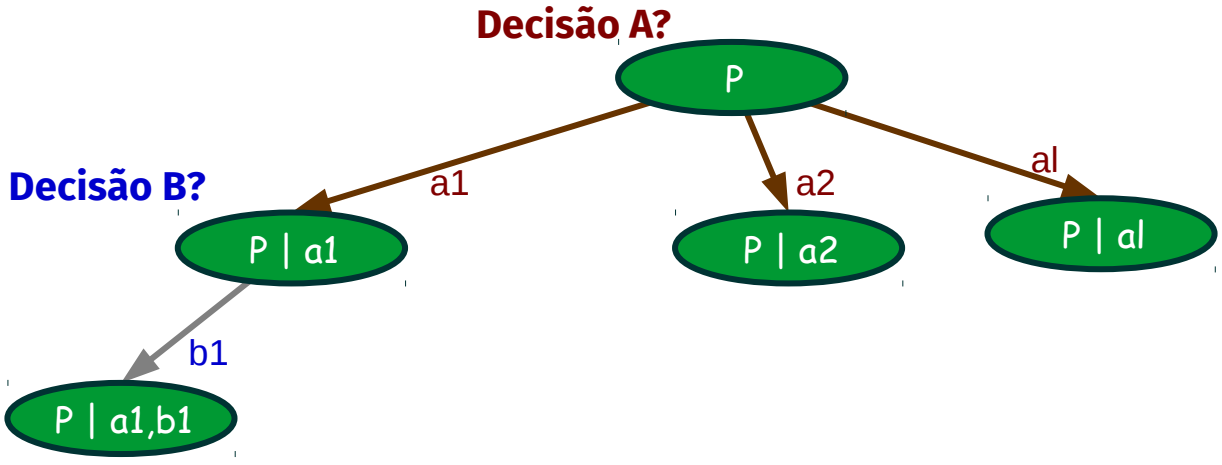
Árvore de busca



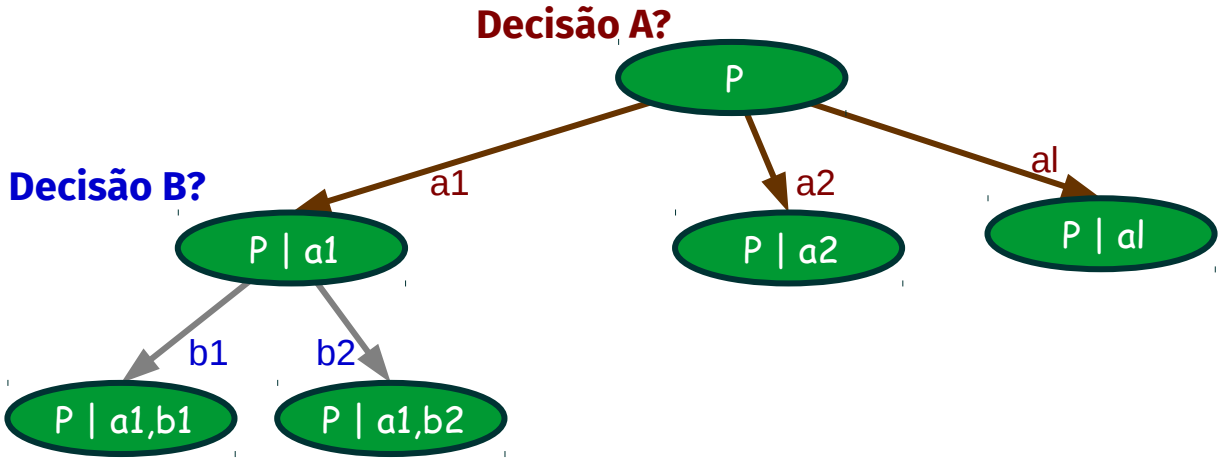
Árvore de busca



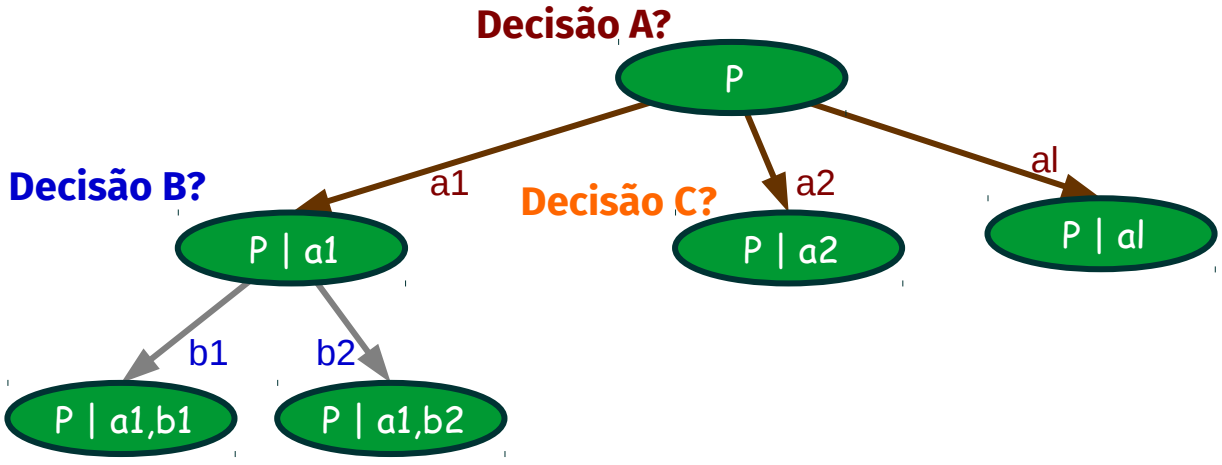
Árvore de busca



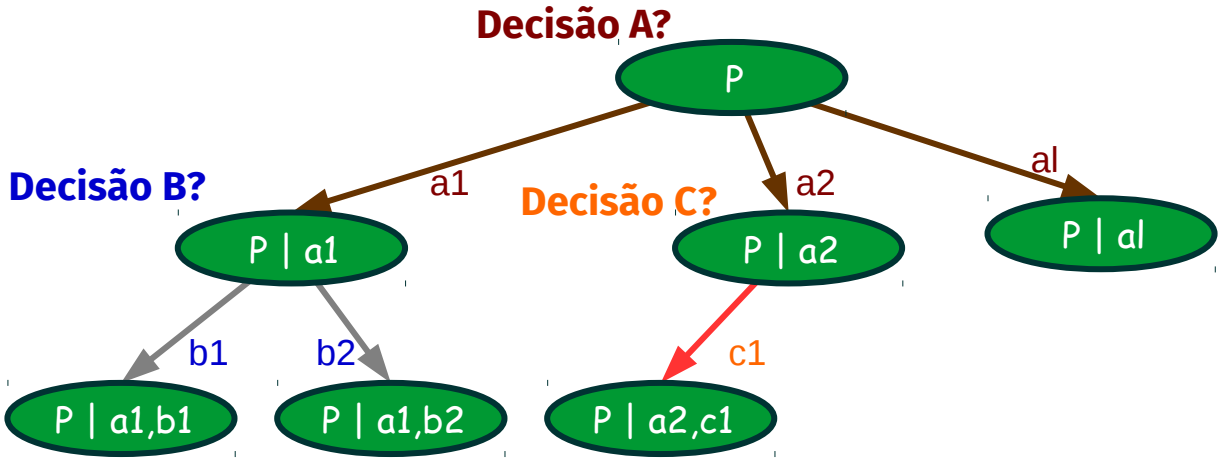
Árvore de busca



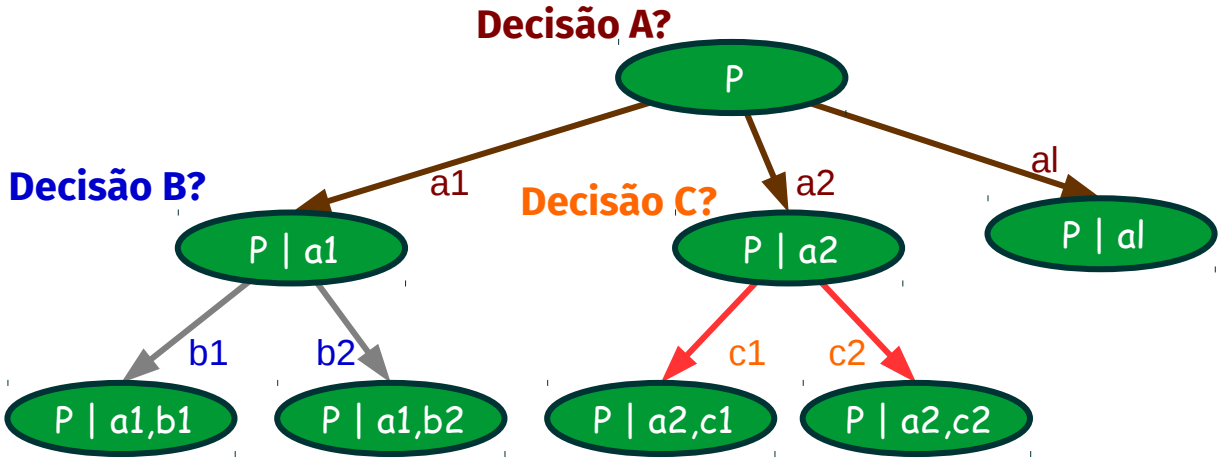
Árvore de busca



Árvore de busca



Árvore de busca



Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

- Tamanho da árvore é pequeno

Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

- Tamanho da árvore é pequeno (função de k)

Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

- Tamanho da árvore é pequeno (função de k)
- Tempo gasto em cada nó é polinomial

Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

- Tamanho da árvore é pequeno (função de k)
- Tempo gasto em cada nó é polinomial

Vamos considerar:

- Uma instância I de um problema de minimização

Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

- Tamanho da árvore é pequeno (função de k)
- Tempo gasto em cada nó é polinomial

Vamos considerar:

- Uma instância I de um problema de minimização
- Uma medida $\mu(I)$:

Árvores limitadas

Pergunta: Como usar árvores de busca para obter **FPT**?

- Tamanho da árvore é pequeno (função de k)
- Tempo gasto em cada nó é polinomial

Vamos considerar:

- Uma instância I de um problema de minimização
- Uma medida $\mu(I)$:
  (queremos que $\mu(I)$ dependa só do parâmetro k)

Ramificação

Em cada nó da árvores executamos uma ramificação

Ramificação

Em cada nó da árvores executamos uma ramificação

Ramificação de I

Crie instâncias I_1, ..., I_ℓ tais que:

Ramificação

Em cada nó da árvores executamos uma ramificação

Ramificação de I

Crie instâncias I_1, ..., I_ℓ tais que:

1. dada solução S_i de I_i, existe solução $h_i(S_i)$ de I

Ramificação

Em cada nó da árvores executamos uma ramificação

Ramificação de I

Crie instâncias I_1, ..., I_ℓ tais que:

1. dada solução S_i de I_i, existe solução $h_i(S_i)$ de I e $\{h_i(S)\}_{1 \leq i \leq \ell}$ contém solução ótima;

Ramificação

Em cada nó da árvores executamos uma ramificação

Ramificação de I

Crie instâncias I_1, ..., I_ℓ tais que:

1. dada solução S_i de I_i, existe solução $h_i(S_i)$ de I e $\{h_i(S)\}_{1 \leq i \leq \ell}$ contém solução ótima;
2. ℓ é uma função do parâmetro $\mu(I)$ somente;

Ramificação

Em cada nó da árvores executamos uma ramificação

Ramificação de I

Crie instâncias I_1, ..., I_ℓ tais que:

1. dada solução S_i de I_i, existe solução $h_i(S_i)$ de I e $\{h_i(S)\}_{1 \leq i \leq \ell}$ contém solução ótima;
2. ℓ é uma função do parâmetro $\mu(I)$ somente;
3. existe $c > 0$ tal que $\mu(I_i) \leq \mu(I) - c$ para cada i.

Limitando a altura

Um algoritmo de ramificação tem altura limitada:

Limitando a altura

Um algoritmo de ramificação tem altura limitada:

- se $\mu(I) < 0$, então instância é fácil;

Limitando a altura

Um algoritmo de ramificação tem altura limitada:

- se $\mu(I) < 0$, então instância é fácil;
- grau de ramificação ℓ de um nó é limitado;

Limitando a altura

Um algoritmo de ramificação tem altura limitada:

- se $\mu(I) < 0$, então instância é fácil;
- grau de ramificação ℓ de um nó é limitado;
- altura é limitada

Framework para FPT

Árvores e FPT

Framework para FPT

Solução é um subconjunto de algum universo U:

Framework para FPT

Solução é um subconjunto de algum universo U:

1. Identificamos $S \subseteq U$ pequeno

Framework para FPT

Solução é um subconjunto de algum universo U:

1. Identificamos $S \subseteq U$ pequeno (em tempo polinomial)

Framework para FPT

Solução é um subconjunto de algum universo U:

1. Identificamos $S \subseteq U$ pequeno (em tempo polinomial)
2. Adivinhamos qual elemento de S está em uma solução ótima

Framework para FPT

Solução é um subconjunto de algum universo U:

1. Identificamos $S \subseteq U$ pequeno (em tempo polinomial)
2. Adivinhamos qual elemento de S está em uma solução ótima
3. Garantimos que a medida $\mu(I)$ diminui

Framework para FPT

Solução é um subconjunto de algum universo U:

1. Identificamos $S \subseteq U$ pequeno (em tempo polinomial)
2. Adivinhamos qual elemento de S está em uma solução ótima
3. Garantimos que a medida $\mu(I)$ diminui
   (i.e., o "número" de decisões que faltam diminui)

Cobertura por vértices

Cobertura por vértices

Recapitulando:

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$
- Núcleo: $2k$ vértices em tempo $\mathcal{O}(n\sqrt{m})$

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$
- Núcleo: $2k$ vértices em tempo $\mathcal{O}(n\sqrt{m})$ (usando técnicas de LP)

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$
- Núcleo: $2k$ vértices em tempo $\mathcal{O}(n\sqrt{m})$ (usando técnicas de LP)
- Algoritmo exato: $\mathcal{O}(4^k k^{\mathcal{O}(1)})$ (para k fixo)

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$
- Núcleo: $2k$ vértices em tempo $\mathcal{O}(n\sqrt{m})$ (usando técnicas de LP)
- Algoritmo exato: $\mathcal{O}(4^k k^{\mathcal{O}(1)})$ (para k fixo)

Observação

Para $v \in V(G)$:

- *ou v está na solução;*

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$
- Núcleo: $2k$ vértices em tempo $\mathcal{O}(n\sqrt{m})$ (usando técnicas de LP)
- Algoritmo exato: $\mathcal{O}(4^k k^{\mathcal{O}(1)})$ (para k fixo)

Observação

Para $v \in V(G)$:

- *ou v está na solução;*
- *ou $N(v)$ está na solução.*

Cobertura por vértices

Recapitulando:

- Queremos encontrar $X \subseteq V(G)$ tal que $E[G - X] = \emptyset$
- Núcleo: $2k$ vértices em tempo $\mathcal{O}(n\sqrt{m})$ (usando técnicas de LP)
- Algoritmo exato: $\mathcal{O}(4^k k^{\mathcal{O}(1)})$ (para k fixo)

Observação

Para $v \in V(G)$:

- *ou v está na solução;*
- *ou $N(v)$ está na solução.*

Observação

Se, para todo $v \in V(G)$, $d(v) \leq 1$, então Cobertura por vértices é polinomial.

Ramificando

Ramificação

- Escolha $v \in V(G)$

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     1.1 remova v de G;

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     1.1 remova v de G;
     1.2 faça $k \leftarrow k - 1$

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     1.1 remova v de G;
     1.2 faça $k \leftarrow k - 1$
  2. Se $N(v)$ está na solução:

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     - 1.1 remova v de G;
     - 1.2 faça $k \leftarrow k - 1$
  2. Se $N(v)$ está na solução:
     - 2.1 remova $N[v]$ de G;

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     - 1.1 remova v de G;
     - 1.2 faça $k \leftarrow k - 1$
  2. Se $N(v)$ está na solução:
     - 2.1 remova $N[v]$ de G;
     - 2.2 faça $k \leftarrow k - |N(v)|$.

Ramificando

Ramificação

- Escolha $v \in V(G)$
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     - 1.1 remova v de G;
     - 1.2 faça $k \leftarrow k - 1$
  2. Se $N(v)$ está na solução:
     - 2.1 remova $N[v]$ de G;
     - 2.2 faça $k \leftarrow k - |N(v)|$.

Ramificando

Ramificação

- Escolha $v \in V(G)$ com grau máximo;
- Temos $S = \{v\} \cup N(v)$ contém elemento de uma solução;
- Ramificamos em:
  1. Se v está na solução:
     1.1 remova v de G;
     1.2 faça $k \leftarrow k - 1$
  2. Se $N(v)$ está na solução:
     2.1 remova $N[v]$ de G;
     2.2 faça $k \leftarrow k - |N(v)|$.

Tempo de execução

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Observação

Dada uma árvore de ramificação $\mathcal{T}$ com ℓ folhas, então o número de nós de $\mathcal{T}$ é no máximo $2\ell - 1$.

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Observação

Dada uma árvore de ramificação $\mathcal{T}$ com ℓ folhas, então o número de nós de $\mathcal{T}$ é no máximo $2\ell - 1$.

- Seja $T(k)$ o número de folhas na árvore de ramificação da Cobertura por vértices

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Observação

Dada uma árvore de ramificação $\mathcal{T}$ com ℓ folhas, então o número de nós de $\mathcal{T}$ é no máximo $2\ell - 1$.

- Seja $T(k)$ o número de folhas na árvore de ramificação da Cobertura por vértices

$$T(i) =$$

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Observação

Dada uma árvore de ramificação $\mathcal{T}$ com ℓ folhas, então o número de nós de $\mathcal{T}$ é no máximo $2\ell - 1$.

- Seja $T(k)$ o número de folhas na árvore de ramificação da Cobertura por vértices

$$T(i) = \begin{cases}$$

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Observação

Dada uma árvore de ramificação $\mathcal{T}$ com ℓ folhas, então o número de nós de $\mathcal{T}$ é no máximo $2\ell - 1$.

- Seja $T(k)$ o número de folhas na árvore de ramificação da Cobertura por vértices

$$T(i) = \begin{cases} T(i-1) + T(i-2) & \text{se } i \geq 2; \\ \end{cases}$$

Tempo de execução

- cada nó da árvore gasta temo $n^{\mathcal{O}(1)}$
- seja $\tau(k)$ o número de nós
- TOTAL: $\tau(k)n^{\mathcal{O}}(1)$.

Observação

Dada uma árvore de ramificação $\mathcal{T}$ com ℓ folhas, então o número de nós de $\mathcal{T}$ é no máximo $2\ell - 1$.

- Seja $T(k)$ o número de folhas na árvore de ramificação da Cobertura por vértices

$$T(i) = \begin{cases} T(i-1) + T(i-2) & \text{se } i \geq 2; \\ 1 & \text{caso contrário.} \end{cases}$$

Calculando

Lema

Para todo $k \geq 0$, $T(k) \leq 1{,}6181^k$.

Obtendo a base da exponencial

Obtendo a base da exponencial

Ideia: vamos tentar fazer $T(k) \leq c\,\lambda^k$.

Olhando

Observação: a base impacta muito no tempo de execução

Olhando

Observação: a base impacta muito no tempo de execução

Conclusão: tentar melhorar ao máximo o passo de ramificação

Olhando

Observação: a base impacta muito no tempo de execução

Conclusão: tentar melhorar ao máximo o passo de ramificação

Quando ramificamos:

- se v não é parte da solução, então diminuímos k em $|N(v)| \geq 2$

Olhando

Observação: a base impacta muito no tempo de execução

Conclusão: tentar melhorar ao máximo o passo de ramificação

Quando ramificamos:

- se v não é parte da solução, então diminuímos k em $|N(v)| \geq 2$;
- mas se $|N(v)| > 2$?

Olhando

Observação: a base impacta muito no tempo de execução

Conclusão: tentar melhorar ao máximo o passo de ramificação

Quando ramificamos:

- se v não é parte da solução, então diminuímos k em $|N(v)| \geq 2$;
- mas se $|N(v)| > 2$? a árvore é menor!

Olhando

Observação: a base impacta muito no tempo de execução

Conclusão: tentar melhorar ao máximo o passo de ramificação

Quando ramificamos:

- se v não é parte da solução, então diminuímos k em $|N(v)| \geq 2$;
- mas se $|N(v)| > 2$? a árvore é menor!

Observação

Cobertura por vértices é polinomial quando $d(v) \leq 2$ para todo $v \in V(G)$.

Melhorando

Agora resolvemos um nó em tempo polinomial sempre que $\Delta(G) \leq 2$:

Melhorando

Agora resolvemos um nó em tempo polinomial sempre que $\Delta(G) \leq 2$:

$$T(i) =$$

Melhorando

Agora resolvemos um nó em tempo polinomial sempre que $\Delta(G) \leq 2$:

$$T(i) = \begin{cases}$$

Melhorando

Agora resolvemos um nó em tempo polinomial sempre que $\Delta(G) \leq 2$:

$$T(i) = \begin{cases} T(i-1) + T(i-3) & \text{se } i \geq 3; \\ \end{cases}$$

Melhorando

Agora resolvemos um nó em tempo polinomial sempre que $\Delta(G) \leq 2$:

$$T(i) = \begin{cases} T(i-1) + T(i-3) & \text{se } i \geq 3; \\ 1 & \text{caso contrário.} \end{cases}$$

Melhorando

Agora resolvemos um nó em tempo polinomial sempre que $\Delta(G) \leq 2$:

$$T(i) = \begin{cases} T(i-1) + T(i-3) & \text{se } i \geq 3; \\ 1 & \text{caso contrário.} \end{cases}$$

Teorema

Cobertura por vértices pode ser resolvida em tempo $\mathcal{O}(n\sqrt{m} + 1.4656\, k^{\mathcal{O}(1)})$.

Recorrências

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k
- obtemos uma recorrência

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k
- obtemos uma recorrência
- resolvemos a recorrências

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k
- obtemos uma recorrência
- resolvemos a recorrências

Vetor de ramificação

Suponha que:

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k
- obtemos uma recorrência
- resolvemos a recorrências

Vetor de ramificação

Suponha que:

- cada nó da arvore tem p subproblemas;

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k
- obtemos uma recorrência
- resolvemos a recorrências

Vetor de ramificação

Suponha que:

- cada nó da arvore tem p subproblemas;
- os parâmetros de cada subproblema são $k - d_1, k - 2, \ldots, k - p$.

Limitando árvores de busca por recorrência

Pergunta: Como limitar o tamanho da árvore de busca?

- representamos o tamanho como uma função T de k
- obtemos uma recorrência
- resolvemos a recorrências

Vetor de ramificação

Suponha que:

- cada nó da arvore tem p subproblemas;
- os parâmetros de cada subproblema são $k - d_1, k - 2, \ldots, k - p$.

Então o vetor $(d_1, d_2, \ldots, d_p)$ é chamado vetor de ramificação.

Recorrência geral

- cada nó tem um número constante p de ramos

Recorrência geral

- cada nó tem um número constante p de ramos
- para parâmetro $k \leq p$, presumimos que a instância é fácil (polinomial)

Recorrência geral

- cada nó tem um número constante p de ramos
- para parâmetro $k \leq p$, presumimos que a instância é fácil (polinomial)

Recorrência associada a $(d_1, d_2, \ldots, d_p)$:

Recorrência geral

- cada nó tem um número constante p de ramos
- para parâmetro $k \leq p$, presumimos que a instância é fácil (polinomial)

Recorrência associada a $(d_1, d_2, \ldots, d_p)$:

$T(k) =$

Recorrência geral

- cada nó tem um número constante p de ramos
- para parâmetro $k \leq p$, presumimos que a instância é fácil (polinomial)

Recorrência associada a $(d_1, d_2, \ldots, d_p)$:

$$T(k) = \begin{cases}$$

Recorrência geral

- cada nó tem um número constante p de ramos
- para parâmetro $k \leq p$, presumimos que a instância é fácil (polinomial)

Recorrência associada a $(d_1, d_2, \ldots, d_p)$:

$$T(k) = \begin{cases} T(k-d_1) + T(k-d_2) + \cdots + T(k-d_p) & \text{se } k \geq p; \\ \end{cases}$$

Recorrência geral

- cada nó tem um número constante p de ramos
- para parâmetro $k \leq p$, presumimos que a instância é fácil (polinomial)

Recorrência associada a $(d_1, d_2, \ldots, d_p)$:

$$T(k) = \begin{cases} T(k-d_1) + T(k-d_2) + \cdots + T(k-d_p) & \text{se } k \geq p; \\ 1 & \text{caso contrário.} \end{cases}$$

Resolvendo a recorrência

- Usamos indução

Resolvendo a recorrência

- Usamos indução
- Tentamos uma solução do tipo $T(k) \leq c \cdot \lambda^k$

Resolvendo a recorrência

- Usamos indução
- Tentamos uma solução do tipo $T(k) \leq c \cdot \lambda^k$

Obtemos:

$$\lambda^d \geq \lambda^{d-d_1} + \lambda^{d-d_2} + \cdots + \lambda^{d-d_p},$$

onde $d = \max\{d_1, \ldots, d_p\}$.

Resolvendo a recorrência

- Usamos indução
- Tentamos uma solução do tipo $T(k) \leq c \cdot \lambda^k$

Obtemos:

$$\lambda^d \geq \lambda^{d-d_1} + \lambda^{d-d_2} + \cdots + \lambda^{d-d_p},$$

onde $d = \max\{d_1, \ldots, d_p\}$.

Reescrevemos como:

$$\lambda^d - \lambda^{d-d_1} - \lambda^{d-d_2} - \cdots - \lambda^{d-d_p} \geq 0$$

Resolvendo a recorrência

- Usamos indução
- Tentamos uma solução do tipo $T(k) \leq c \cdot \lambda^k$

Obtemos:

$$\lambda^d \geq \lambda^{d-d_1} + \lambda^{d-d_2} + \cdots + \lambda^{d-d_p},$$

onde $d = \max\{d_1, \ldots, d_p\}$.

Reescrevemos como:

$$P(\lambda) := \lambda^d - \lambda^{d-d_1} - \lambda^{d-d_2} - \cdots - \lambda^{d-d_p} \geq 0$$

$P(\lambda)$ é o vetor característico de T.

Calculando

Seja λ_0 solução de $P(\lambda) = 0$:

- $P(\lambda) < 0$ para $0 < \lambda < \lambda_0$;
- $P(\lambda) > 0$ para $\lambda > \lambda_0$.

Calculando

Seja λ_0 solução de $P(\lambda) = 0$:

- $P(\lambda) < 0$ para $0 < \lambda < \lambda_0$;
- $P(\lambda) > 0$ para $\lambda > \lambda_0$.

Obtemos $T(k) \leq \mathcal{O}(\lambda_0^k)$

Conjunto de vértices de retroalimentação

Conjunto de vértices de retroalimentação mínimo

Conjunto de vértices de retroalimentação mínimo

Seja G um grafo. Um subconjunto de vértices $X \subseteq V(G)$ é chamado de conjunto de retroalimentação de G se $G - X$ é acíclico.

Decidir: existe conjunto de realimentação com no máximo k vértices?

Reduções simples

Consideramos o problema com multigrafos

Reduções simples

Consideramos o problema com multigrafos, i.e.:

- podem existir laços
- podem existir arestas múltiplas

Reduções simples

Consideramos o problema com multigrafos, i.e.:

- podem existir laços
- podem existir arestas múltiplas

Redução FVS.1: Se houver laça em v, remova v e diminua k.

Reduções simples

Consideramos o problema com multigrafos, i.e.:

- podem existir laços
- podem existir arestas múltiplas

Redução FVS.1: Se houver laça em v, remova v e diminua k.

Redução FVS.2: Se houver $t > 2$ cópias de uma aresta, remova $t - 2$ cópias.

Reduções simples

Consideramos o problema com multigrafos, i.e.:

- podem existir laços
- podem existir arestas múltiplas

Redução FVS.1: Se houver laça em v, remova v e diminua k.

Redução FVS.2: Se houver $t > 2$ cópias de uma aresta, remova $t - 2$ cópias.

Redução FVS.3: Se um vértice v tem grau $d(v) \leq 1$, remova v.

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Após as reduções, podemos obter uma instância trivial.

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Após as reduções, podemos obter uma instância trivial.

Redução FVS.5: Se $k < 0$, devolva não.

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Após as reduções, podemos obter uma instância trivial.

Redução FVS.5: Se $k < 0$, devolva não.

Propriedades

Após executar exaustivamente as reduções, obtemos G

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Após as reduções, podemos obter uma instância trivial.

Redução FVS.5: Se $k < 0$, devolva não.

Propriedades

Após executar exaustivamente as reduções, obtemos G

1. sem laços;

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Após as reduções, podemos obter uma instância trivial.

Redução FVS.5: Se $k < 0$, devolva não.

Propriedades

Após executar exaustivamente as reduções, obtemos G

1. sem laços;
2. somente com arestas simples ou duplas; e

Instância simplificada

Redução FVS.4: Se FVS.1 não se aplica e v tem grau $d(v) = 2$:

- adicione uma aresta entre os vértices $N(v)$;
- remova v.

Após as reduções, podemos obter uma instância trivial.

Redução FVS.5: Se $k < 0$, devolva não.

Propriedades

Após executar exaustivamente as reduções, obtemos G

1. sem laços;
2. somente com arestas simples ou duplas; e
3. com grau mínimo 3.

Vértices pesados

- Considere uma ordenação $(v_1, v_2, \ldots, v_n)$ de $V(G)$ tal que:

$$d(v_1) \geq d(v_2) \geq \cdots \geq d(v_n).$$

Vértices pesados

- Considere uma ordenação $(v_1, v_2, \ldots, v_n)$ de $V(G)$ tal que:

$$d(v_1) \geq d(v_2) \geq \cdots \geq d(v_n).$$

- Defina $V_3 = \{v_1, \ldots, v_{3k}\}$.

Vértices pesados

- Considere uma ordenação $(v_1, v_2, \ldots, v_n)$ de $V(G)$ tal que:

$$d(v_1) \geq d(v_2) \geq \cdots \geq d(v_n).$$

- Defina $V_3 = \{v_1, \ldots, v_{3k}\}$.

Lema

Seja G um grafo tal que $d(v) \geq 3$ para todo $v \in V(G)$. Todo conjunto de retroalimentação de G com tamanho até k tem pelo menos um vértice de V_{3k}.

FPT para Conjunto de retroalimentação

Teorema

Existe um algoritmo para o Conjunto de vértices de retroalimentação que executa em tempo $(3k)^k \cdot n^{\mathcal{O}(1)}$.