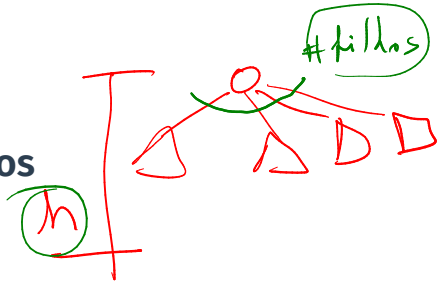


# Algoritmos Parametrizados

Árvores de busca limitadas







---

Lehilton Pedrosa

Segundo Semestre de 2016

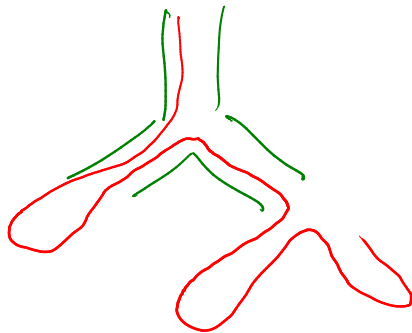
Instituto de Computação – Unicamp

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

# Ramificação

---

Ideia: *backtracking*

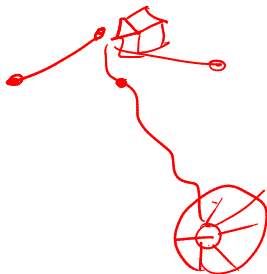


Ideia: *backtracking*

- Fazer uma sequência de decisões

Ideia: *backtracking*

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



## Ideia: *backtracking*

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

## 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

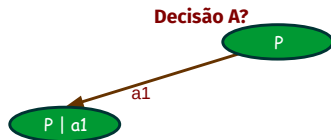
# Árvore de busca



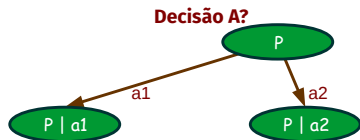
**Decisão A?**



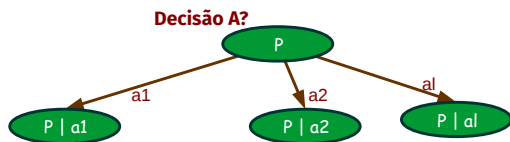
# Árvore de busca



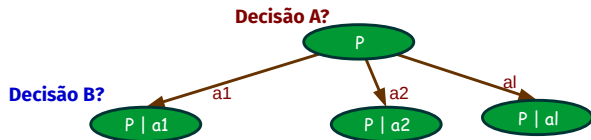
# Árvore de busca



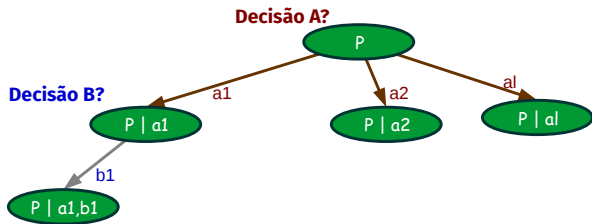
# Árvore de busca



# Árvore de busca

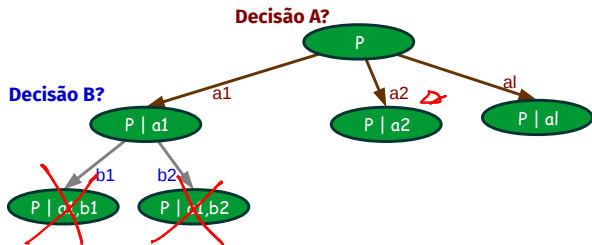


# Árvore de busca

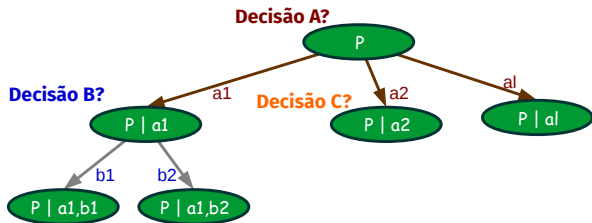




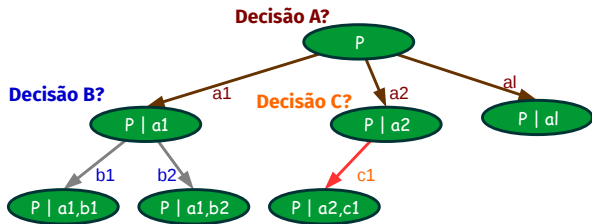
# Árvore de busca



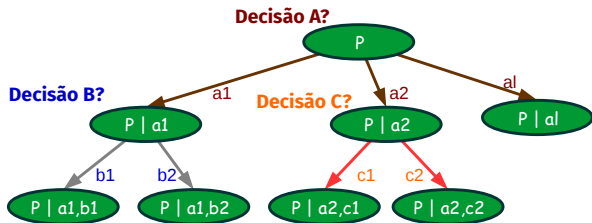
# Árvore de busca



# Árvore de busca



# Árvore de busca



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

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

- Tamanho da árvore é pequeno

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

- Tamanho da árvore é pequeno (função de  $k$ )

**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



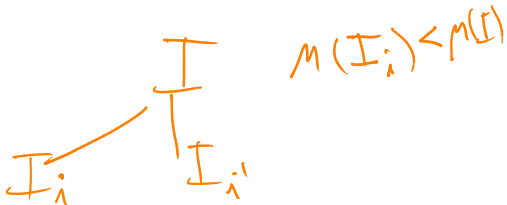
# Árvores limitadas

**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  $k$ )

Em cada nó da árvores executamos uma ramificação

Em cada nó da árvores executamos uma ramificação

## Ramificação de $l$

Crie instâncias  $l_1, \dots, l_e$  tais que:

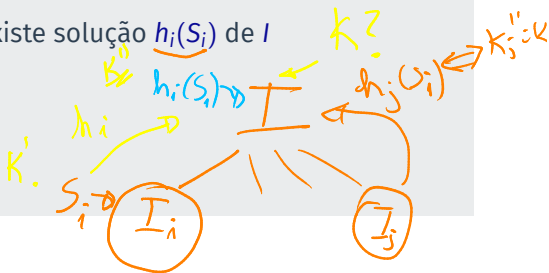
# Ramificação

Em cada nó da árvore executamos uma **ramificação**

## Ramificação de $I$

Crie instâncias  $I_1, \dots, I_\ell$  tais que:

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



Em cada nó da árvores executamos uma ramificação

## Ramificação de $I$

Crie instâncias  $I_1, \dots, I_\ell$  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;

Em cada nó da árvores executamos uma **ramificação**

## Ramificação de $I$

Crie instâncias  $I_1, \dots, I_\ell$  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.  $\ell$  é uma função do parâmetro  $\mu(I)$  somente;



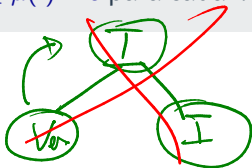


Em cada nó da árvores executamos uma **ramificação**

## Ramificação de $l$

Crie instâncias  $l_1, \dots, l_\ell$  tais que:

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



## Limitando a altura

Um algoritmo de ramificação tem altura **limitada**:

Um algoritmo de ramificação tem altura limitada:

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

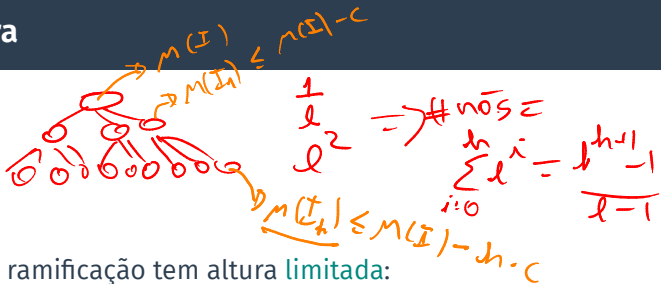
↳ quando  $\mu(I) < 0$ ,  
temos uma folha

Um algoritmo de ramificação tem altura **limitada**:

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

# Limitando a altura

nível 0 :  
 nível 1 :  
 nível 2 :



Um algoritmo de ramifica\u00e7\u00e3o tem altura limitada:

- se  $\mu(I) < 0$ , ent\u00e3o inst\u00e2ncia \u00e9 f\u00e1cil;
- grau de ramifica\u00e7\u00e3o  $l \leq g(M(I))$  de um n\u00f3 \u00e9 limitado;
- altura \u00e9 limitada

$$\Rightarrow \# \text{ n\u00f3s} \leq \frac{l^{\frac{M(I)}{c} + 1} - 1}{l - 1} \leq g(M(I))$$

## Framework para 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)

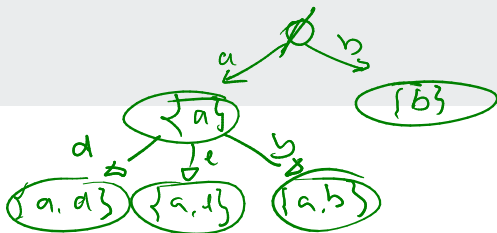
$\leq f(k)$

## 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

*que contém um elemento da solução*



## 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

---

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  $O(n\sqrt{m})$  (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)  $\Rightarrow$  TOTAL:  $n\sqrt{m} + \underline{4^k k^{\mathcal{O}(1)}}$

## 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

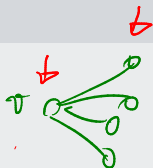
## Recapitulando:

- 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; (1)
- ou  $N(v)$  está na solução. (2)



## Observação

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



## Ramificação

- Escolha  $v \in V(G)$

## Ramificação

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

## Ramificação

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



## 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:

## Ramificação

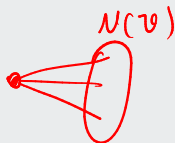
- Escolha  $v \in V(G)$
- Temos  $S = \{v\} \cup N(v)$  contém elemento de uma solução;
- Ramificamos em:
  1. Se  $v$  está na solução:
    - 1.1 remova  $v$  de  $G$ ;

## 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  $k \leftarrow k - 1$

## 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:



## Ramificação

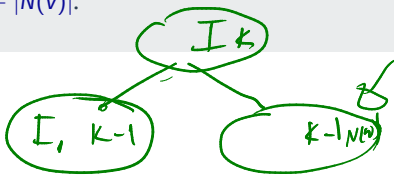
- Escolha  $v \in V(G)$
- Temos  $S = \{v\} \cup N(v)$  contém elemento de uma solução;
- Ramificamos em:
  1. Se  $v$  está na solução:
    - 1.1 remova  $v$  de  $G$ ;
    - 1.2 faça  $k \leftarrow k - 1$
  2. Se  $N(v)$  está na solução:
    - 2.1 remova  $N[v]$  de  $G$ ;

## Ramificação

- Escolha  $v \in V(G)$  *~ isolado*
- 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  $k \leftarrow k - |N(v)|$ .

$$M(I) = k$$

$$\exists c > 0$$
$$c = 1?$$



## 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  $k \leftarrow k - |N(v)|$ .

## 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

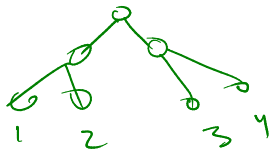
- cada nó da árvore gasta tempo  $\mathcal{O}(1)$

## Tempo de execução

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

# Tempo de execução

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



$$2 \cdot k - 1$$

# Tempo de execução

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

## Observação

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

# Tempo de execução

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

## Observação

Dada uma árvore de ramificação  $\mathcal{T}$  com  $\ell$  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 tempo  $n^{\mathcal{O}(1)}$
- seja  $\tau(k)$  o número de nós
- **TOTAL:**  $\tau(k)n^{\mathcal{O}(1)}$ .

## Observação

*Dada uma árvore de ramificação  $\mathcal{T}$  com  $\ell$  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 tempo  $n^{\mathcal{O}(1)}$
- seja  $\tau(k)$  o número de nós
- **TOTAL:**  $\tau(k)n^{\mathcal{O}(1)}$ .

## Observação

*Dada uma árvore de ramificação  $\mathcal{T}$  com  $\ell$  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

$$T(i) = \left\{ \right.$$



# Tempo de execução

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

## Observação

*Dada uma árvore de ramificação  $\mathcal{T}$  com  $\ell$  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 tempo  $n^{\mathcal{O}(1)}$
- seja  $\tau(k)$  o número de nós
- **TOTAL:**  $\tau(k)n^{\mathcal{O}(1)}$ .

## Observação

*Dada uma árvore de ramificação  $\mathcal{T}$  com  $\ell$  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

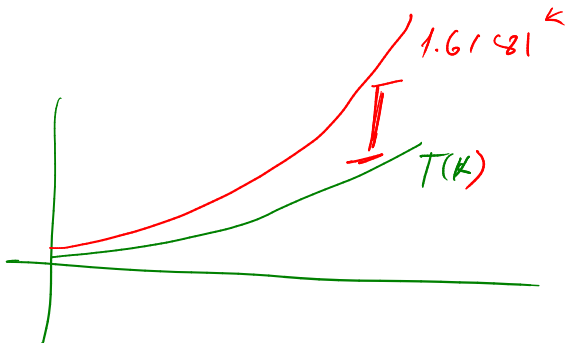
## Lema

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

Indução:  $k = 0, 1$   $T(k) = 1 \leq 1,6181^k$  OK

Seja  $k \geq 2$

$$\begin{aligned} T(k) &\leq T(k-1) + T(k-2) \\ \text{P.h.} &\leq 1,6181^{k-1} + 1,6181^{k-2} \\ &= 1,6181^{k-2} (1,6181 + 1) \\ &< 1,6181^{k-2} (1,6181)^2 = 1,6181^k \quad \square \end{aligned}$$



$$T(K) \leq \sigma (1.6181^K)$$

$$T(K) \geq \omega (1.6180^K)$$

$$\Rightarrow T(K) = \Theta(1.6181^K \cdot \underline{K}^{\Theta(1)})$$

## Obtendo a base da exponencial

## Obtendo a base da exponencial

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

$$T(k) = T(k-1) + T(k-2)$$

$$? \leq c \cdot \lambda^{k-1} + c \lambda^{k-2}$$

$$= c \lambda^{k-2} (\lambda + 1)$$

$$? \leq c \lambda^k \iff \lambda + 1 \leq \lambda^2$$

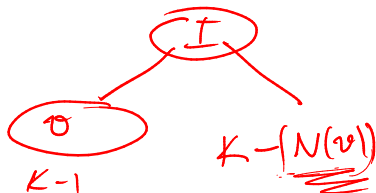
$\Rightarrow$  entao  $\lambda$  é mínimo quando  $\lambda + 1 = \lambda^2$

$$\Rightarrow \lambda = \phi$$

**Observação:** a base impacta muito no tempo de execução

**Observação:** a base impacta muito no tempo de execução

**Conclusão:** tentar melhorar ao máximo o passo de ramificação





**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 diminuimos  $k$  em  $|N(v)| \geq 2$

**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 diminuimos  $k$  em  $|N(v)| \geq 2$ ;
- mas se  $|N(v)| > 2$ ?

**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 diminuimos  $k$  em  $|N(v)| \geq 2$ ;
- **mas se  $|N(v)| > 2$ ?** a árvore é menor!

**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 diminuimos  $k$  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  $v \in V(G)$ .*

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

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

$$T(i) =$$

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

$$T(i) = \left\{ \begin{array}{l} T(i-1) + T(i-3) \end{array} \right.$$

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

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



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

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

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  $\mathcal{O}(n\sqrt{m} + \underline{1.4656} k^{\mathcal{O}(1)})$ .*

# 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 <sup>da árvore</sup> 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 árvore 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 árvore tem  $p$  subproblemas;
- os parâmetros de cada subproblema são  $k - d_1, k - d_2, \dots, k - d_p$ .

# 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:

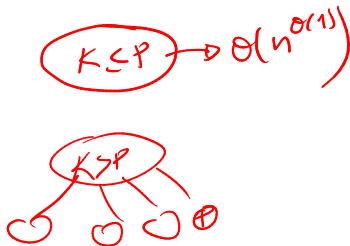
- cada nó da árvore tem  $p$  subproblemas;
- os parâmetros de cada subproblema são  $k - d_1, k - 2, \dots, k - p$ .

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

- 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  $(d_1, d_2, \dots, d_p)$ :

# 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  $(d_1, d_2, \dots, d_p)$ :

$$T(k) =$$

# 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  $(d_1, d_2, \dots, d_p)$ :

$$T(k) = \left\{ \right.$$



# 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  $(d_1, d_2, \dots, d_p)$ :

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

# 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  $(d_1, d_2, \dots, d_p)$ :

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

## Resolvendo a recorrência

- Usamos indução

# Resolvendo a recorrência

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

$$T(k) = T(k-d_1) + T(k-d_2) + \dots + T(k-d_p) \\ \leq c \cdot \lambda^{k-d_1} + c \cdot \lambda^{k-d_2} + \dots + c \cdot \lambda^{k-d_p}$$

$$? \leq c \lambda^k$$

Seja  $d = \max\{d_1, d_2, \dots, d_p\}$

$$\lambda^{k-(k-d)} \geq \lambda^{k-d_1} + \dots + \lambda^{k-d_p}$$

## Resolvendo a recorrência

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

Obtemos:

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

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

## Resolvendo a recorrência

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

Obtemos:

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

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

Reescrevemos como:

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

## Resolvendo a recorrência

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

Obtemos:

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

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

Reescrevemos como:

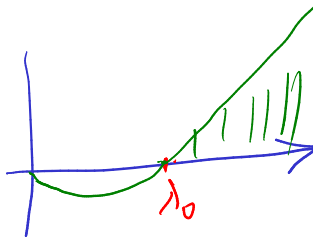
$$P(\lambda) := \lambda^d - \lambda^{d-d_1} - \lambda^{d-d_2} - \dots - \lambda^{d-d_p} \geq 0$$

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

# Calculando

Seja  $\lambda_0$  solução de  $P(\lambda) = 0$ :

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





Seja  $\lambda_0$  solução de  $P(\lambda) = 0$ :

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

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

$$h \leq g(M(I))$$



## Conjunto de vértices de retroalimentação

---

# Conjunto de vértices de retroalimentação mínimo

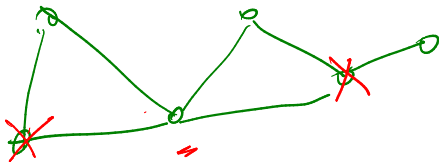
(Feedback Vertex Set)



## 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



Consideramos o problema com **multigrafos**, i.e.:

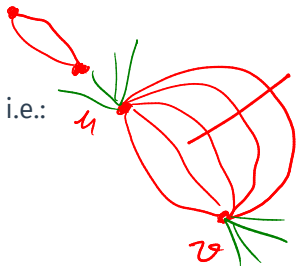
- podem existir **laços**
- podem existir arestas múltiplas

**Redução FVS.1:** Se houver laço  $v$  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 laço 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 laço em  $v$ , remova  $v$  e diminua  $k$ .

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

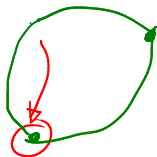
**Redução FVS.3:** Se um vértice  $v$  tem grau  $d(v) \leq 1$ , remova  $v$ .





# Instância simplificada

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



# Instância simplificada

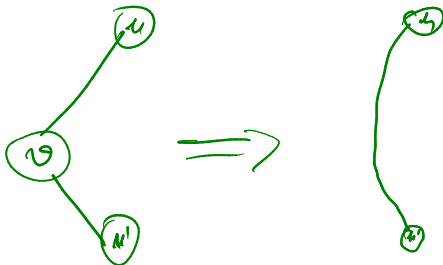
**Redução FVS.4:** Se FVS.1 não se aplica e  $v$  tem grau  $d(v) = 2$ :

- 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  $v$ .



1.  $u = u'$



2.  $uu' \in E(G)$



# 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.

# 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.

$$k = 0$$

# 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 exhaustivamente 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 exhaustivamente 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  $v$ .

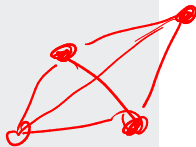
Após as reduções, podemos obter uma instância trivial.

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

## Propriedades

Após executar exhaustivamente as reduções, obtemos  $G$

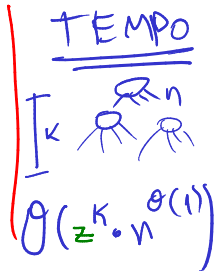
1. sem laços;
2. somente com arestas simples ou duplas; e
3. com grau mínimo 3.



$$n \text{ vértices}; d_v \geq 3 \Rightarrow |E(G)| \geq \frac{n \cdot 3}{2}$$

Ideia :

— Backtracking.



Maurro's  $(b, k)$

0) Se  $k \leq 0$  a resposta é não, sim!

1) Liste os vertices  $v_1, v_2, \dots$   
em ordem crescente de grau.

2) P comece  $v_i$  (com  $i \leq z$ )  $\rightarrow 3k$   
Remova  $v_i$   
Se MAURRO'S  $(b - v_i, k - 1)$ , sim!

3) Responda NÃO

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

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

## Vértices pesados

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

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

$$z = \underline{3k}$$

- Defina  $V_{3k} = \{v_1, \dots, v_{3k}\}$ .

# Vértices pesados

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

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

- Defina  $V_{3k} \equiv \{v_1, \dots, v_{3k}\}$ .

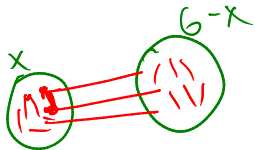
## Lema

*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  $V_{3k}$ .*

Prova Seja  $x$  uma sol. ótima (i.e.  $|X|=k$ )

$$\text{Fato: } \sum_{v \in X} (d_v - 1) \geq |E(G)| - |V(G)| + 1$$

$$\rightarrow b-x \text{ é } \text{foresta} \Rightarrow |E(b-x)| \leq |V(b-x)| - 1 \leq |V(G)| - |X| + 1$$



$$|E(G)| \leq |E(b-x)| + \sum_{v \in X} d_v \leq$$

$$\leq |V(G)| - 1 + \sum_{v \in X} (d_v - 1)$$

□

OB) Mostrar q existe um número  
pequeno num solução  $X$

$$\Rightarrow \text{Sup. } X \cap V_{3K} = \emptyset \Rightarrow \forall v \in V \Rightarrow d_v \leq \min_{u \in V_{3K}} d_u$$

Temos:  $\sum_{v_i \in V_{3K}} (d_{v_i} - 1) = (d_{v_1} - 1) + \dots + (d_{v_k} - 1) +$   
 $(d_{v_{k+1}} - 1) + \dots + (d_{v_{2k}} - 1) +$   
 $(d_{v_{2k+1}} - 1) + \dots + (d_{v_{3k}} - 1)$   
 $\geq 3 \sum_{v \in X} (d_v - 1) \geq 3 (|E(G)| + |V(G)| + 1)$   
*↳ pelo FATO*

Também:  $X \subseteq V \setminus V_{3K} \Rightarrow \sum_{v \in V \setminus V_{3K}} (d_v - 1) \geq \sum_{v \in X} (d_v - 1)$   
 $\geq (|E(G)| - |V(G)| + 1)$

Somando:

$$\sum_{v \in V} (d_v - 1) \geq 4(|E(G)| - |V(G)| + 1)$$

$$\Rightarrow 2|E| - |V| \geq 4|E| - 4|V| + 4$$

$$\Rightarrow 3|V| > 2|E| = \sum_{v \in V} d_v \geq 3 \cdot |V|$$

↳



## Teorema

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

sem algoritmo!

