

DCA0204, Módulo 6

Árvores

Daniel Aloise

baseado em slides do prof. Leo Liberti, École Polytechnique, França

DCA, UFRN

Sumário

- Introdução
- Definições e propriedades
- Árvores geradoras
- Listando árvores
- Árvores na psicologia e linguagens
- Busca em Profundidade - Depth-First Search (DFS)

Introdução

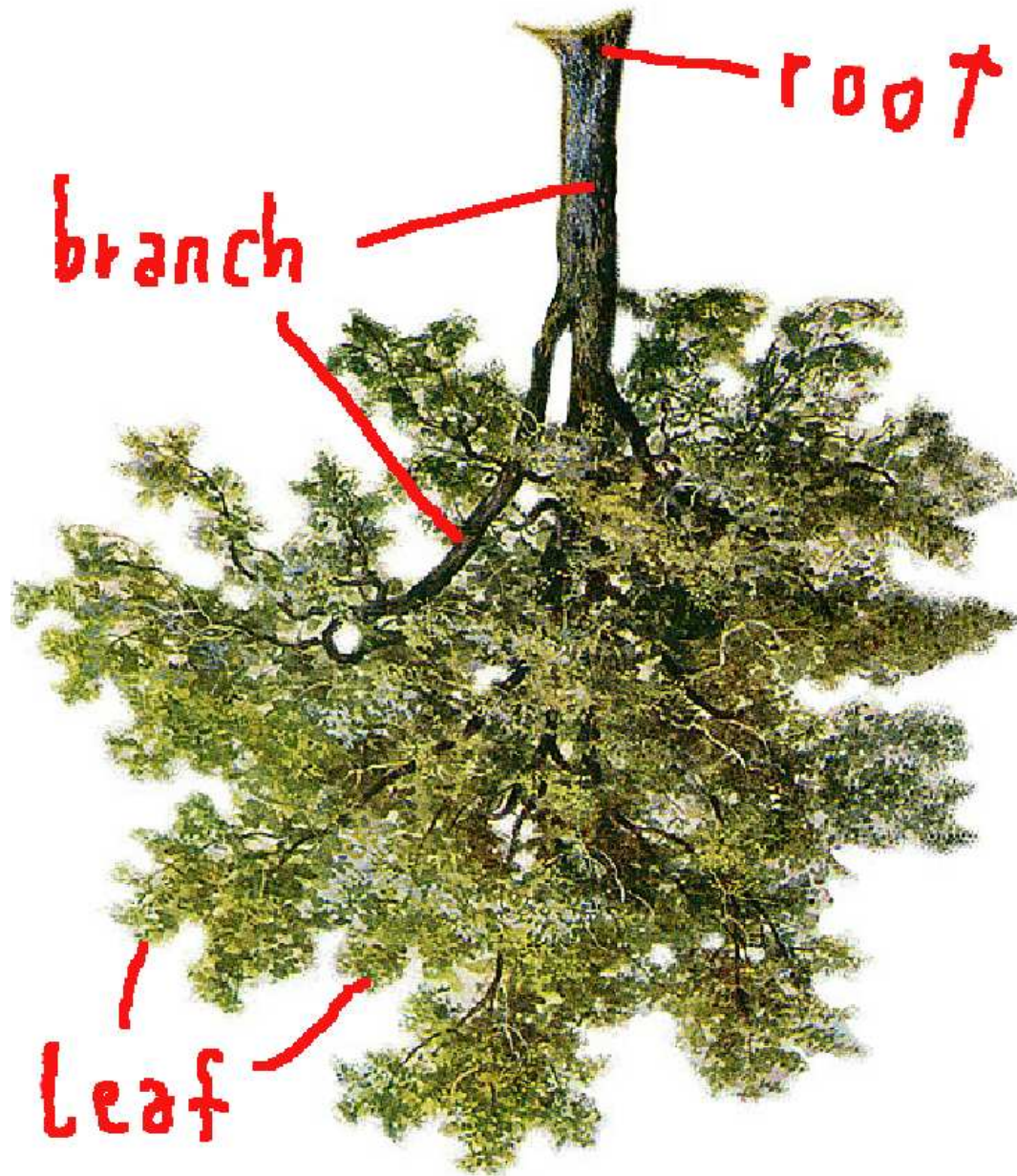
Árvores



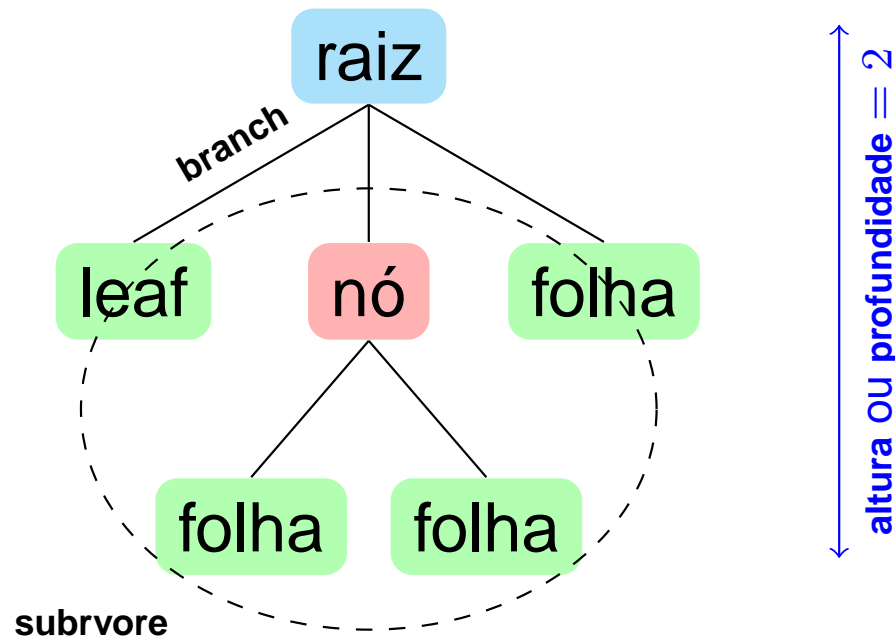
Como nós desenhamos elas



Nomenclatura



Representação gráfica

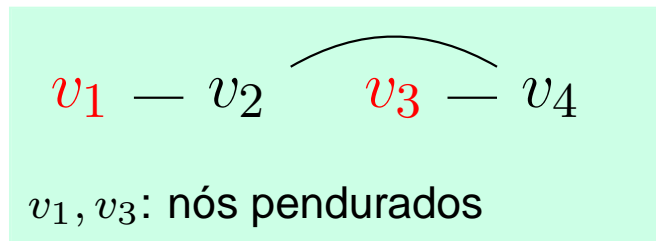


altura/profundidade = comprimento do caminho mais longo [raiz → folha]

Definição matemática de uma árvore

Árvore: um grafo conexo $G = (V, E)$ sem ciclos

- Possui nó é especificado como **raiz**.
- Todo nó que aparece como parte de uma única aresta é chamado de um **nó pendurado**



- Um n pendurado que não é uma raiz é chamdo de **folha**
- As arestas de uma árvore são chamados **ramos**

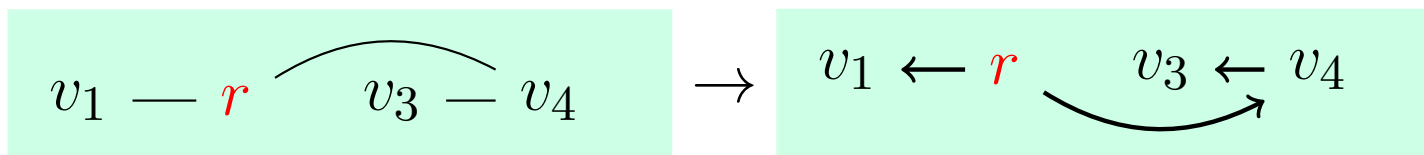
Orientações

- **Orientação de uma árvore** $T = (V, E)$ com raiz $r \in V$: dígrafo $U = (V, A)$ s.a.:

$$\forall \{u, v\} \in E \quad (u, v) \in A \text{ XOR } (v, u) \in A.$$

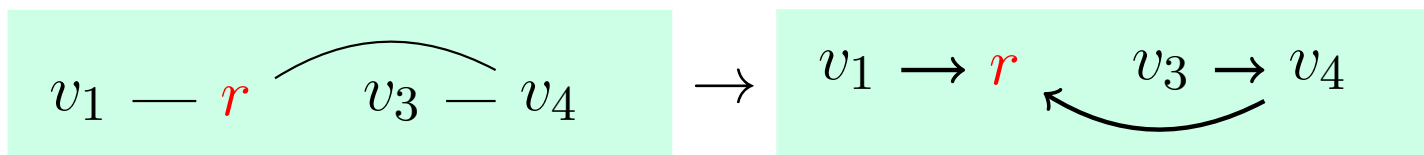
- **orientação exterior:** uma orientação s.a.:

$$\forall \ell \in V \quad \text{leaf}(\ell) \rightarrow \exists \text{ caminho em } U : r \rightarrow \ell.$$



- **orientação interior S.a.:**

$$\forall \ell \in V \quad \text{leaf}(\ell) \rightarrow \exists \text{ caminho em } U : \ell \rightarrow r.$$



Uma árvore tem $|V| - 1$ arestas

Thm.

Uma árvore T em um conjunto V tem $|V| - 1$ arestas

Proof

Seja $m(T)$ o número de arestas em T

Mostre que $m(T) = |V| - 1$ por indução em $|V|$

Se $|V| = 2$, uma relação minimamente conectada requer uma aresta

Hipótese de indução: Suponha $m(T) = |V| - 2$ para todas as árvores T com $|V| - 1$ nós

Seja T qualquer árvore com V nós

Qualquer árvore precisa ter pelo menos um nó folha (por quê?)

Uma vez que ℓ é uma folha, ele é incidente a apenas uma aresta e

Considere a árvore $T' = T \setminus \{e\}$ em $V' = V \setminus \{\ell\}$

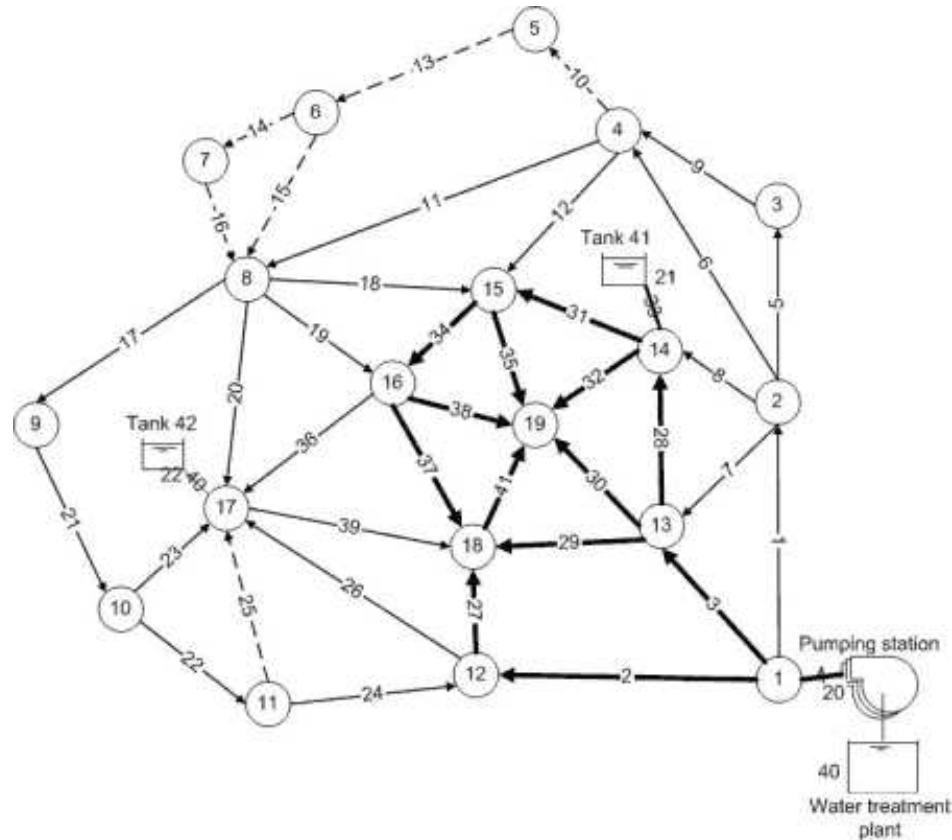
Uma vez que $|V'| = |V| - 1$, $m(T') = |V| - 2$ pela hipótese de indução

Desta forma, T possui exatamente $m(T) = m(T \cup \{e\}) = m'(T) + 1 = |V| - 1$ arestas

Árvores geradoras

Distribuição de eletricidade/água

- Cabos/dutos precisam atingir todos os nós do grafo.
- O principal custo está na instalação dos cabos/dutos.



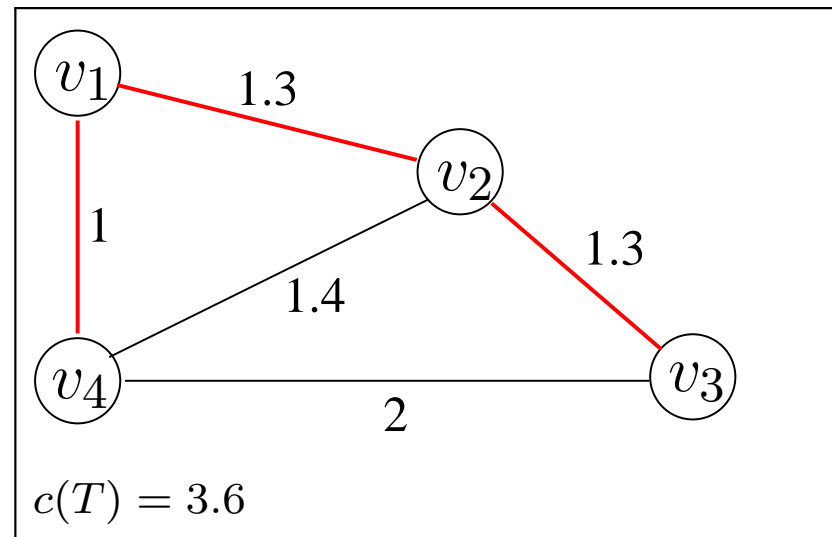
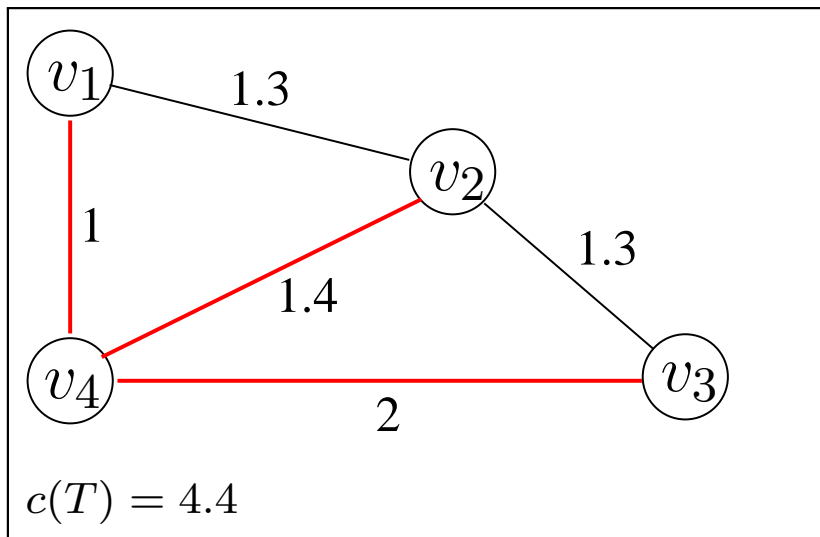
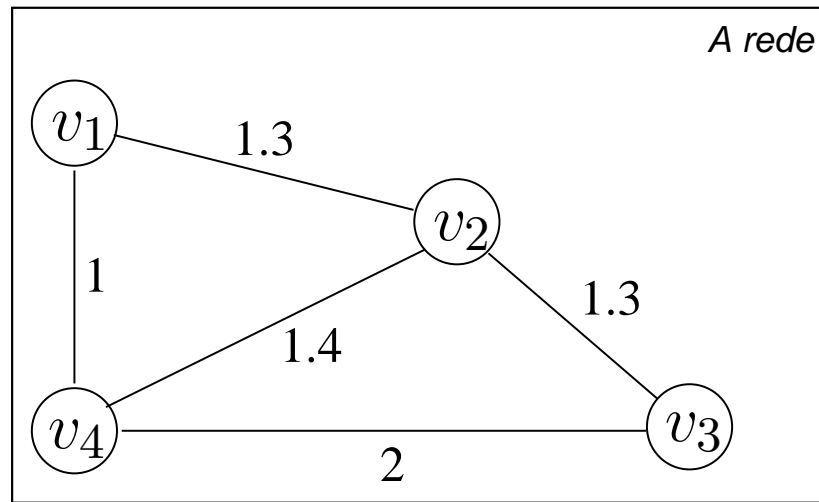
Árvores geradoras

- Custo é otimizado se o material puder ser distribuído para todos os lugares gastando o mínimo possível
- Uma árvore em $U \subseteq V$ é **geradora** se $U = V$
- Se cada aresta e na rede tem custo c_e , o custo de T é

$$c(T) = \sum_{e \in E(T)} c_e$$

Problema da Árvore Geradora Mínima (AGM): Dado um grafo ponderado $G = (V, E)$, encontre a árvore geradora de peso mínimo em G

Exemplo



Algoritmo de Kruskal: um esboço

- requer estrutura de dados *union-find*
- Seja E o conjunto de arestas do grafo
 - 1: $T = \emptyset$
 - 2: Ordene as arestas em E em ordem crescente de peso
 - 3: **while** $|T| < |V| - 1$ **do**
 - 4: Considere a próxima aresta e da sequencia;
 - 5: **if** $T \cup \{e\}$ **não tem ciclo** **then**
 - 6: $T \leftarrow T \cup \{e\};$
 - 7: **end if**
 - 8: $E \leftarrow E \setminus \{e\};$
 - 9: **end while**
- Ao fim, T tem $|V| - 1$ arestas e não tem ciclos.
- \Rightarrow Uma árvore por definição.

Complexidade de pior caso da melhor implementação:
 $O(m \log n)$

Definições

- Uma *partição* de um conjunto M é uma coleção M_1, \dots, M_k de subconjuntos de M com as seguintes propriedades:
 - $M_i \cap M_j = \emptyset$ para $i \neq j$
 - $M = M_1 \cup \dots \cup M_k$.
- No algoritmo de Kruskal, a floresta particiona V .
- Algumas componentes são triviais: nós isolados.

Definições

- O algoritmo de Kruskal realiza duas operações na partição:
 - Testa se dois nós estão no mesmo subconjunto (subárvore).
 - Junta dois subconjuntos em um (através da inserção de uma aresta).
- A estrutura de dados *Union-find* fornece estas duas operações para uma partição de elementos de maneira eficiente.

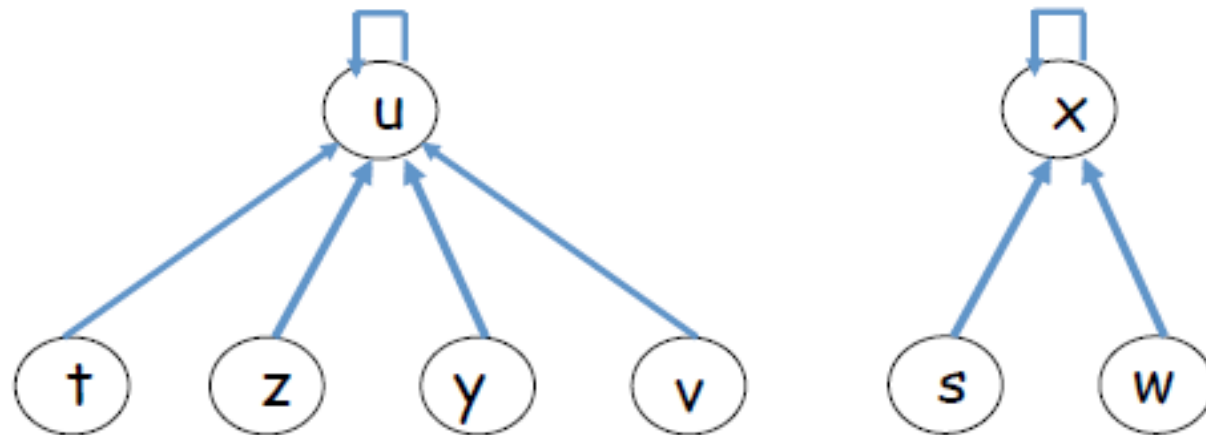
Union-find

- Inicialmente, cada elemento é um subconjunto.
- Cada subconjunto escolhe um dos seus elementos como *representante* (feito pela ED não pelo usuário).
- A função $find(i)$ retorna o representante do subconjunto contendo i .
- Logo, dois elementos estão no mesmo subconjunto se os seus representantes são iguais.

Union-find

- A operação $link(i,j)$ aplicada aos representantes de dois subconjuntos distintos *une* os subconjuntos.
- Solução simples: cada subconjunto é representado por uma *árvore*.
 - Raiz – representante do subconjunto.
- Cada elemento armazena o seu pai nesta árvore (array *parent*).
- Para as raízes, temos *self-loops*.

Union-find



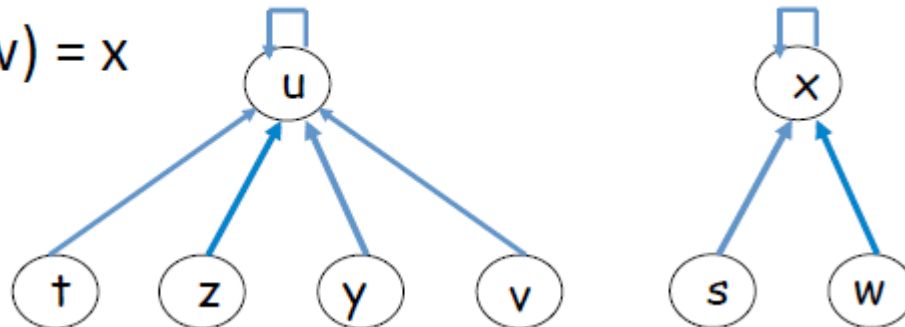
$$S_1 = \{t, \underline{u}, v, y, z\} \quad S_2 = \{s, w, \underline{x}\}$$

Union-find

- A implementação de $\text{find}(i)$ é trivial.
- Seguimos as referências de parent até que encontramos um self-loop (localizado no representante de i).

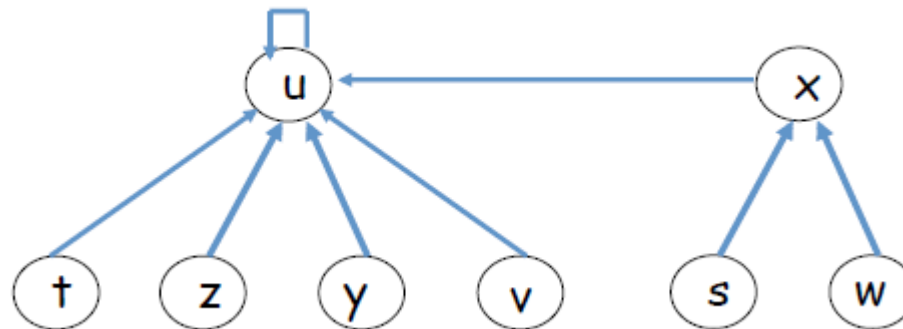
$\text{Find}(z) = u$

$\text{Find}(w) = x$



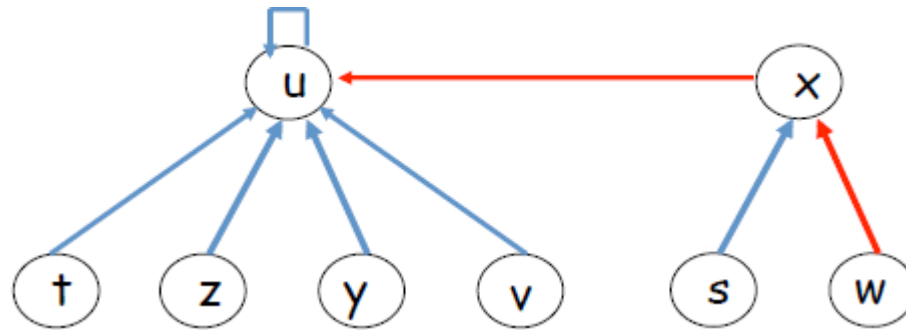
Union-find

- A implementação de $link(i,j)$ é também simples.
- Fazemos com que um representante seja pai do outro representante.
- $link(u,x)$



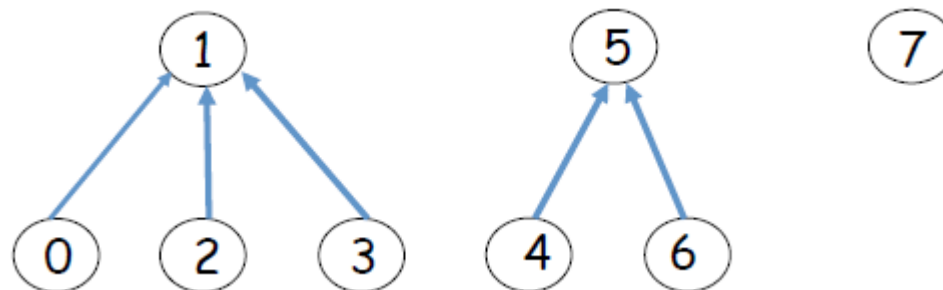
Union-find

- $find(w)$



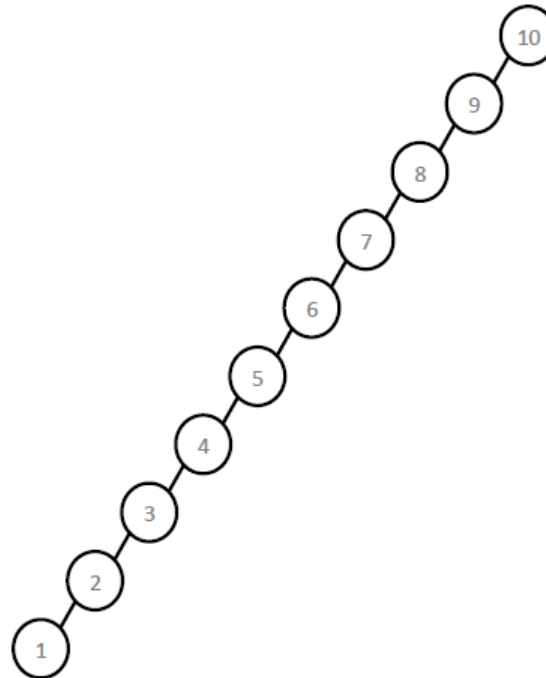
- Exemplo

Elem.	0	1	2	3	4	5	6	7
parent	1	-1	1	1	5	-1	5	-1



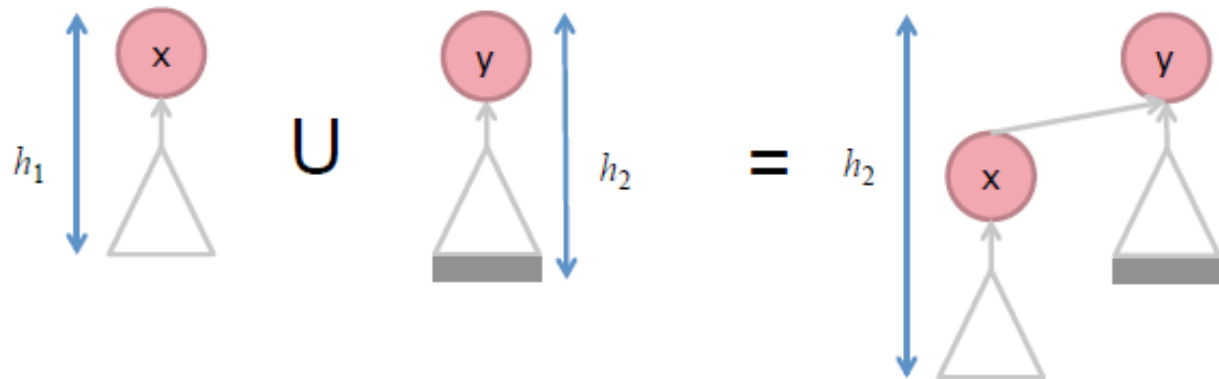
Problemas

- O algoritmo apresentado é correto mas ineficiente.
- As referências *parent* podem formar cadeias longas que são atravessadas muitas e muitas vezes pelas operações *find*.
- Pior caso: $O(n)$



Batendo no ângulo...

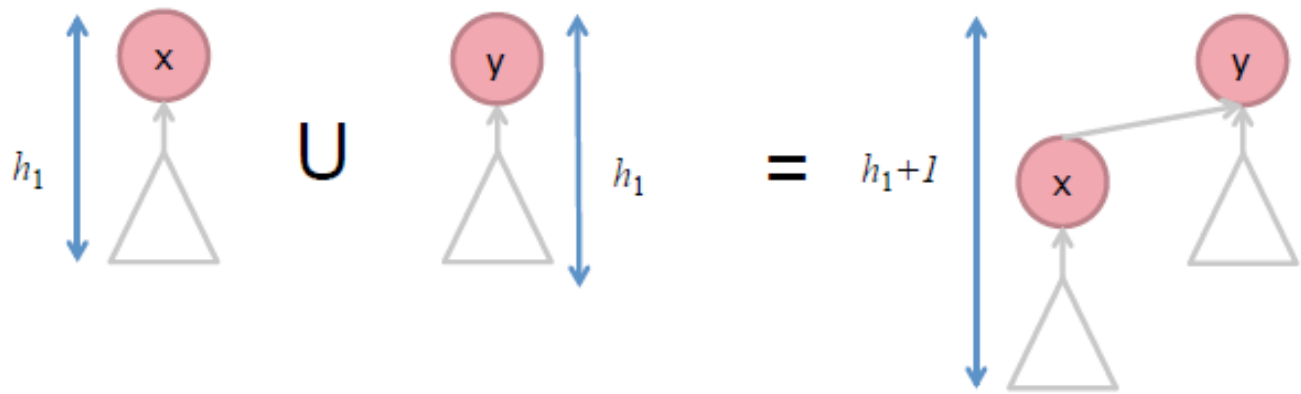
- Duas otimizações podem ser feitas.
- No algoritmo de *união por altura*, verifica-se a altura de cada conjunto para determinar a união.



- A árvore de menor altura é *pendurada* na de maior altura.
- A altura da árvore de maior altura não cresce!

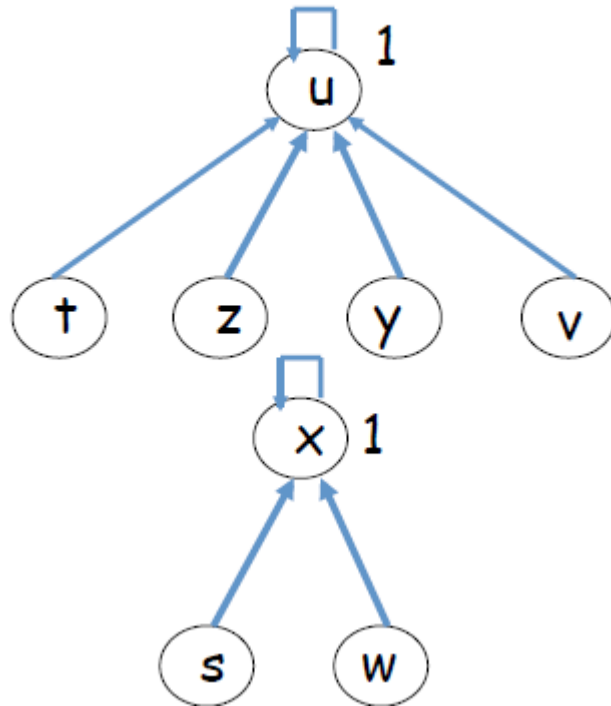
Batendo no ângulo...

- Se as alturas forem iguais, a ordem não importa, e a altura aumentará de um.

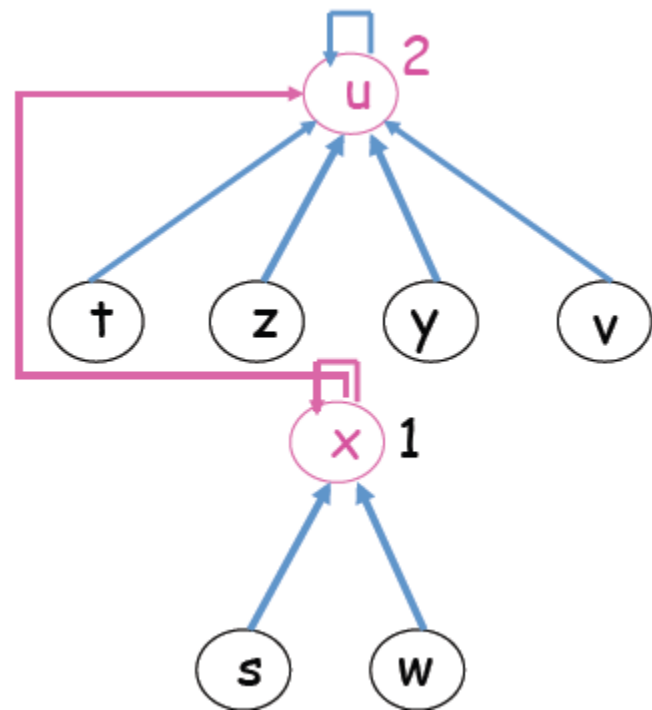


Exemplo

$\{ \{t, u, v, y, z\}, \{s, w, x\} \}$



$link(u, x)$



As alturas também podem ser armazenadas em arrays.

Batendo no ângulo...

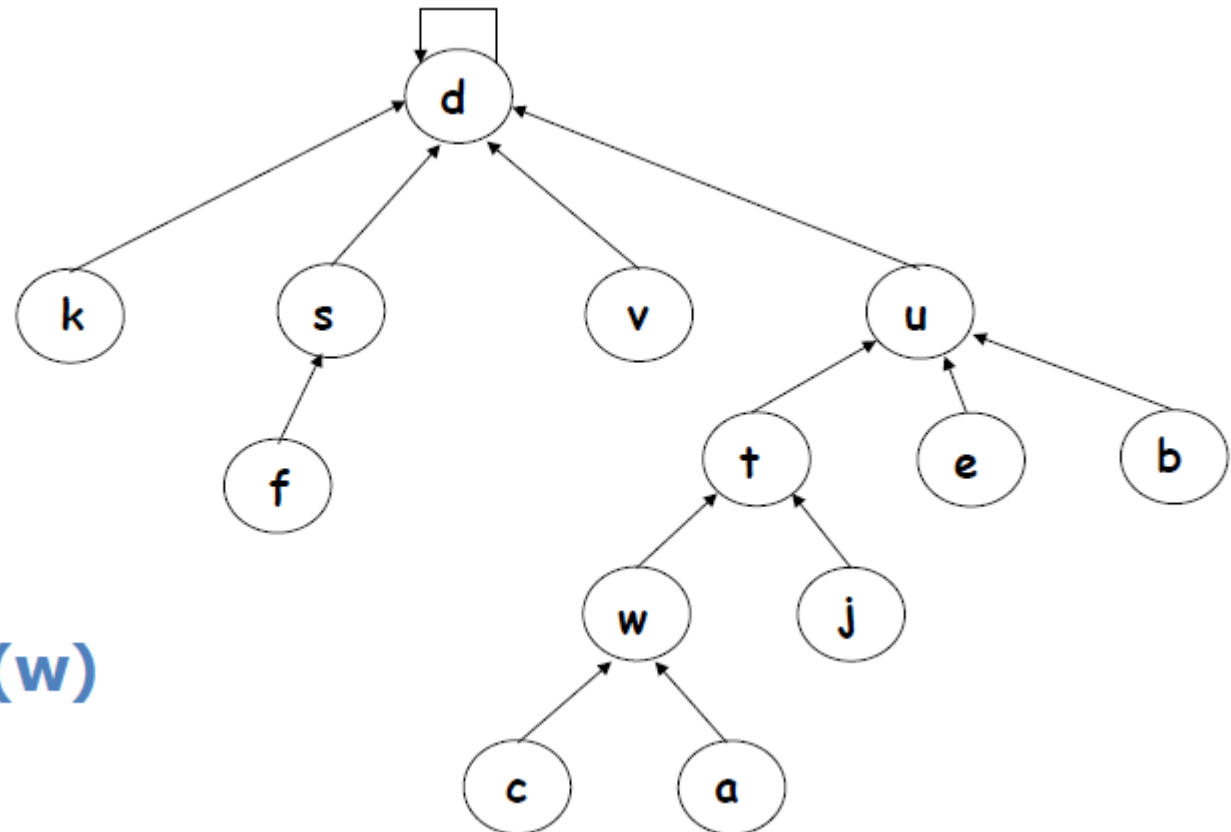
A união por altura garante que a altura = profundidade de qualquer árvore da *union-find* não excede $\log n$

Prova: Uma árvore da *union-find* com altura k contém pelo menos 2^k elementos. Isto é certamente verdade para $k=0$. A altura de uma árvore cresce de $k-1$ para k quando ela recebe um filho de altura $k-1$. Logo a árvore tinha pelo menos 2^{k-1} elementos antes da operação de *link* e recebeu pelo menos mais 2^{k-1} elementos, totalizando depois da operação pelo menos 2^k elementos.

$$n \geq |V(T)| \geq 2^k \therefore k \leq \log n$$

Batendo no ângulo...

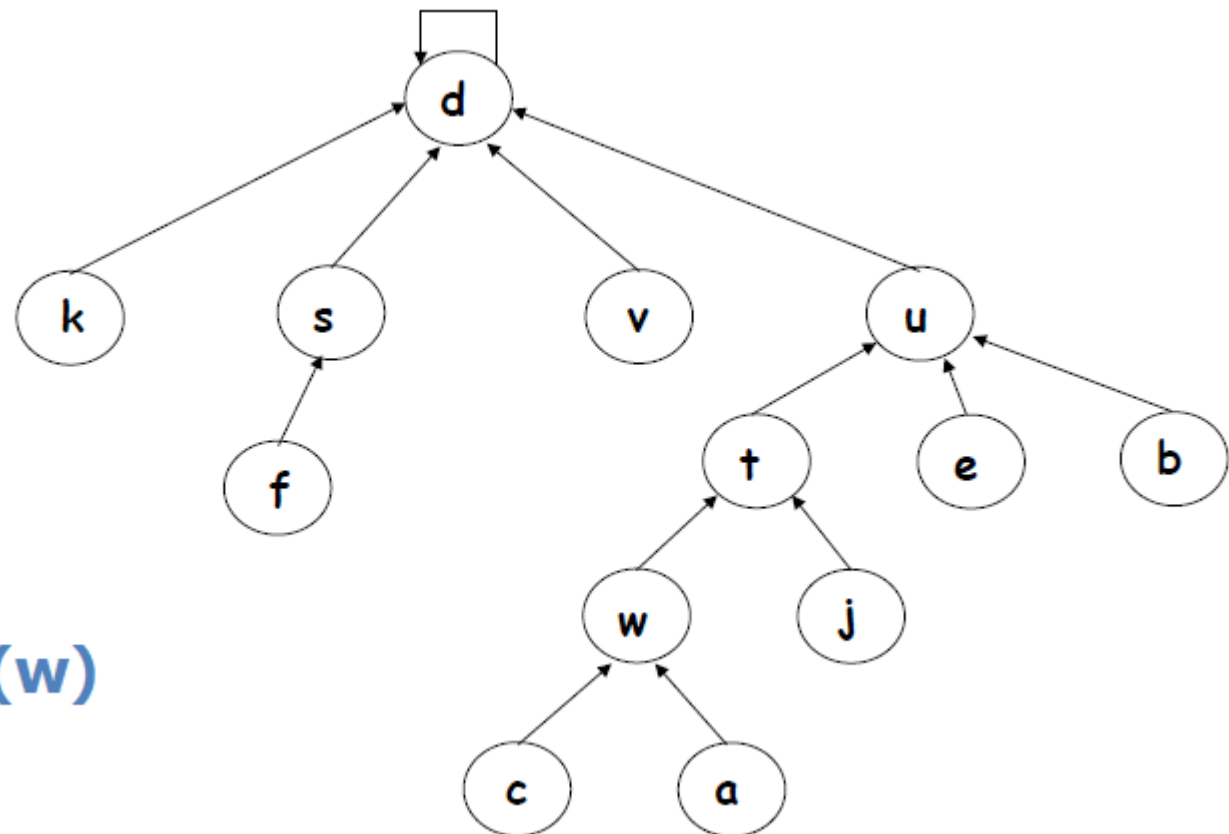
- Outra otimização é a *compressão de caminhos*



Find(w)

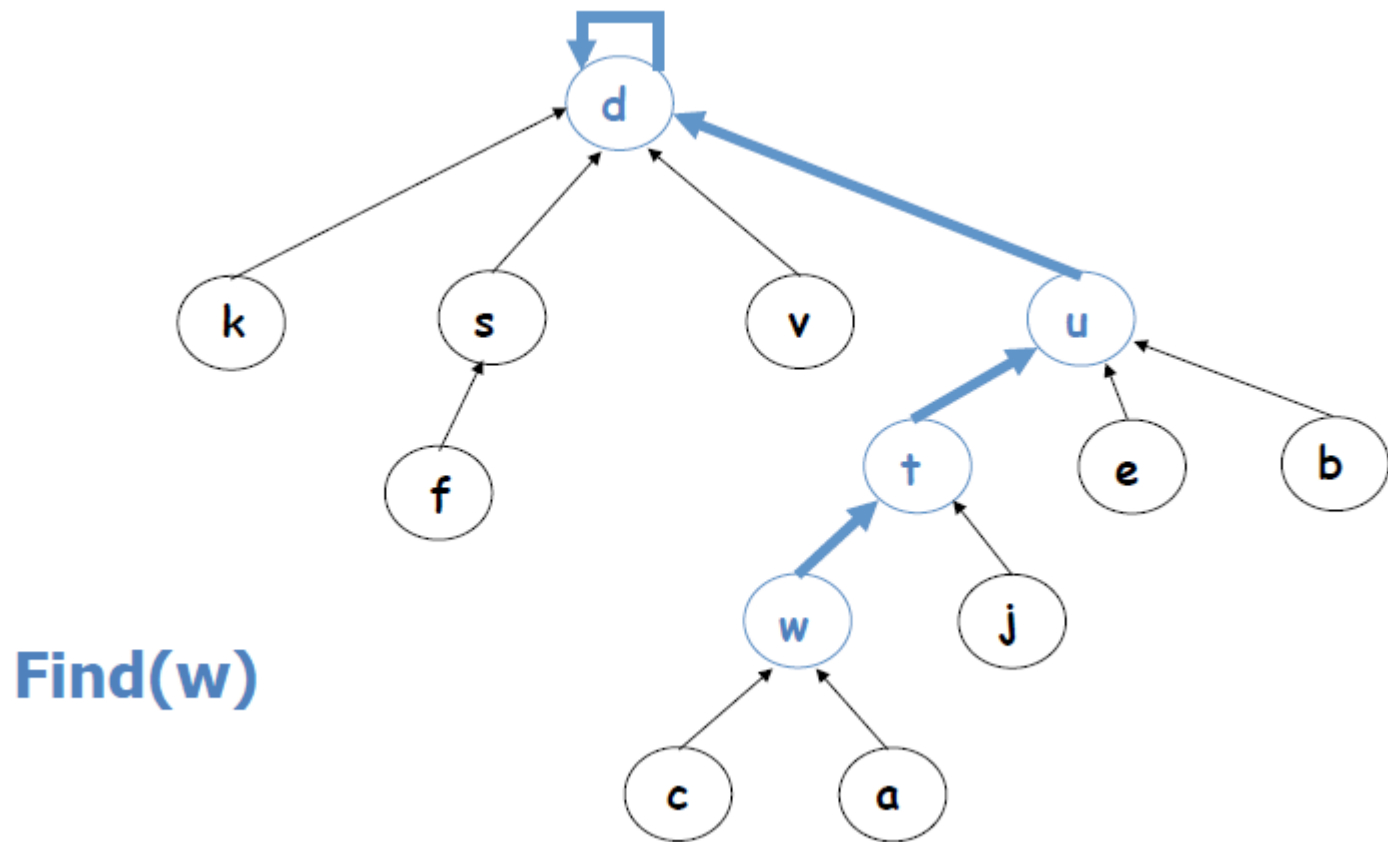
Batendo no ângulo...

- Outra otimização é a *compressão de caminhos*

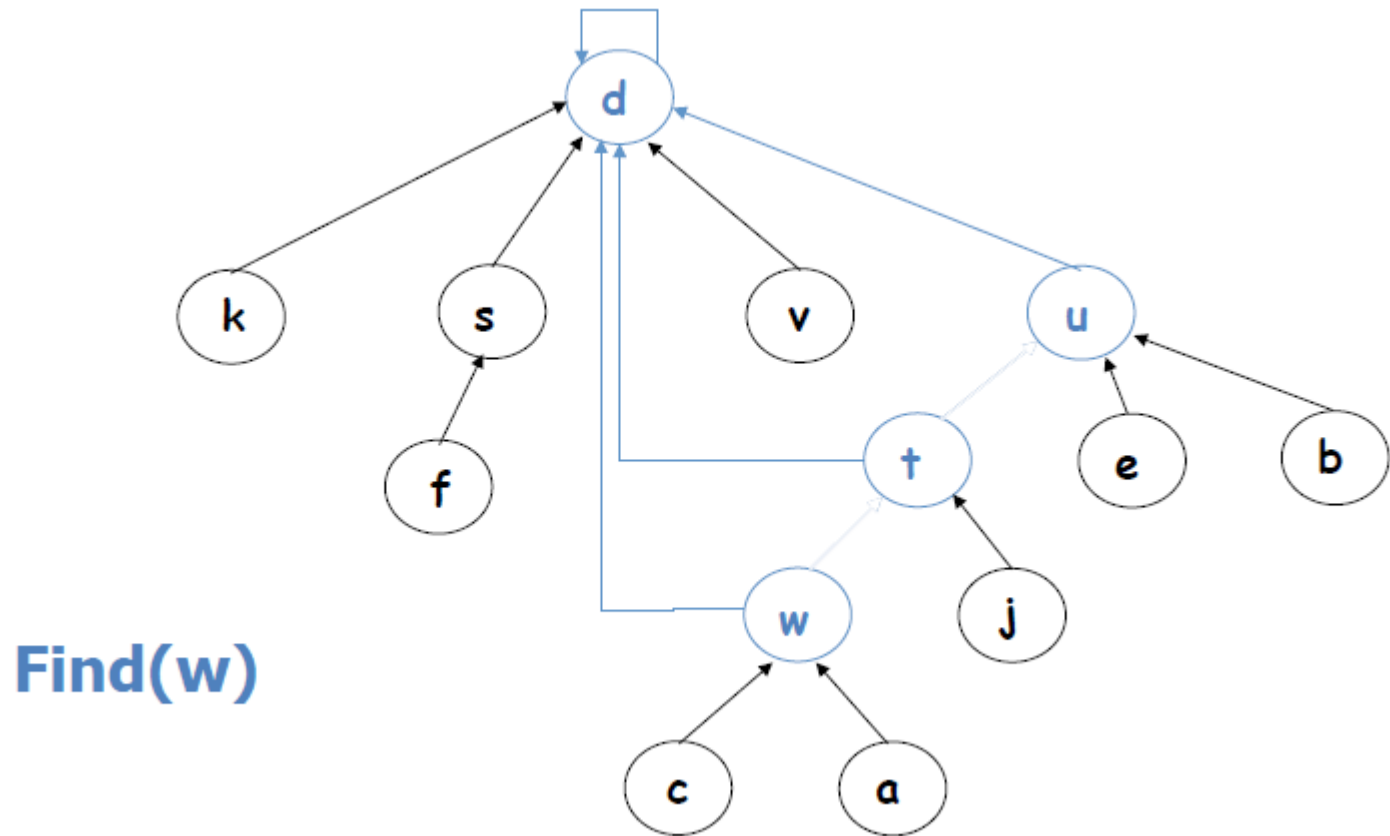


Find(w)

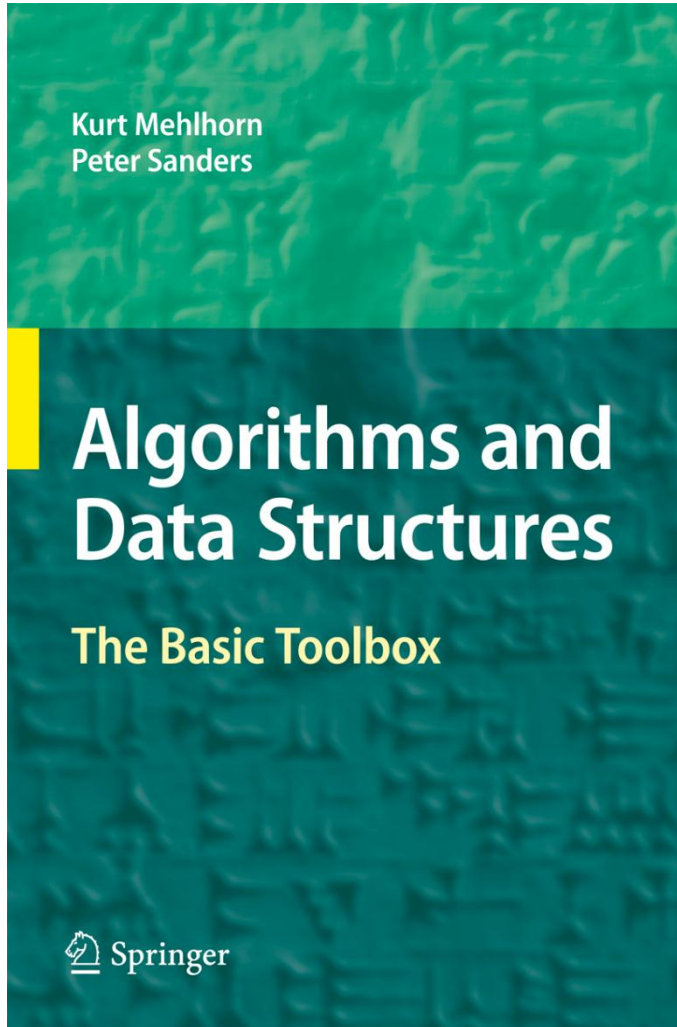
Batendo no ângulo...



Batendo no ângulo...



Pseudo-código



- Página 223 do livro do Mehlhorn & Sanders

Corretude do algoritmo de Kruskal

Thm.

O algoritmo de Kruskal encontra a AGM em um grafo conexo $G = (V, E)$ ponderado

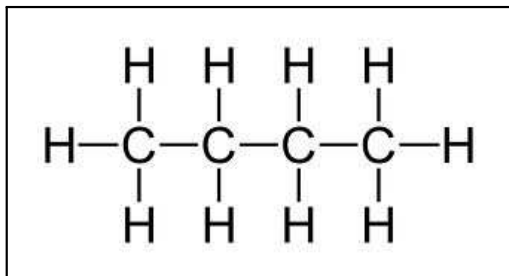
Proof

- Vamos supor a sequência ordenada e^* de arestas $(e_1, e_2, \dots, e_i, \dots, e_{n-1})$ fornecida pelo algoritmo de Kruskal.
- Suponha que a afirm. seja falsa \Rightarrow existe aresta na sequência e^* que não está presente na AGM (sem perda de generalidade, vamos supor também que a AGM seja única).
- Suponhamos que e_i seja a primeira aresta em e^* que não está presente na AGM.
- A adição de e_i à AGM gera um ciclo (pois a AGM é uma árvore!). Além disso, qualquer aresta neste ciclo possui peso menor do que o peso de e_i (por quê?).
- \Rightarrow Qualquer aresta x neste ciclo é considerada antes do que e_i por Kruskal. Mais do que isso, ela tinha que ter sido selecionada! (por quê??).
- Chegamos a uma contradição.

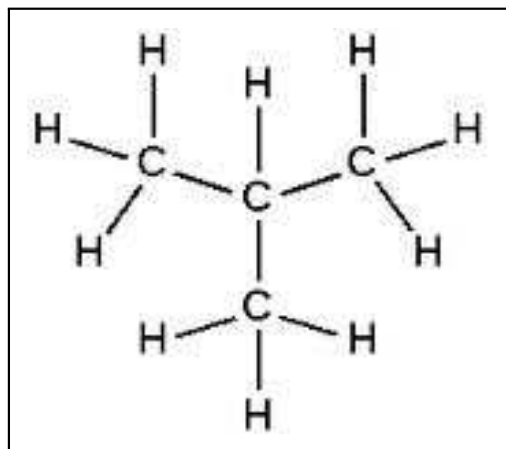
Árvores químicas

Descrições moleculares

- Até metade do século 19, as pessoas pensavam que moléculas fossem completamente definidas por suas fórmulas atômicas
- Ex. parafinas são C_kH_{2k+2}
- Então as pessoas começaram a notar que diferentes ligações resultavam em substâncias com diferentes propriedades: **isômeros**



butano



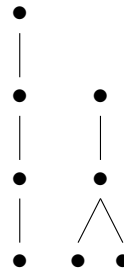
isobutano

Listando isômeros

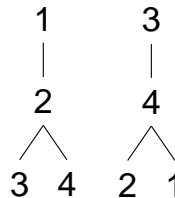
- Carbonos tem valência 4 (eles podem ser incidentes a 4 arestas)
- Hidrogênios tem valência 1 (eles podem ser incidentes a 1 aresta)
- Encontrando isômeros da parafina na metade do século 19:
 - liste todas as árvores com $n = 3k + 2$ nós
 - remova aquelas cujas valências não batem com a fórmula química da parafina
- Como nós listamos todas as árvores? Quantas existem?

Listando árvores rotuladas

- Duas interpretações possíveis
- Estas duas são diferentes (*árvores não-rotuladas*):



- Estas duas são diferentes (*árvores rotuladas*):



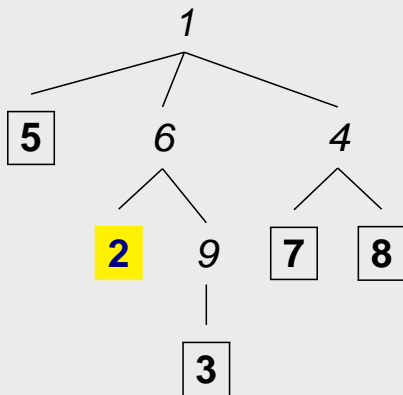
- Contar/listar árvores rotuladas é mais fácil do que árvores não rotuladas
- Existem mais árvores rotuladas do que não rotuladas (por quê?)

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



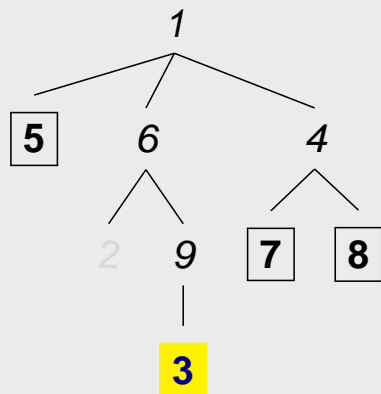
$$L(T) = \{5, 2, 3, 7, 8\}, v = 2, t = (6)$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



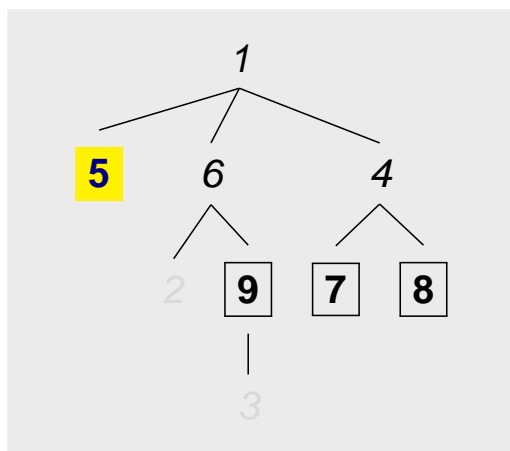
$$L(T) = \{5, 3, 7, 8\}, v = 3, t = (6, 9)$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



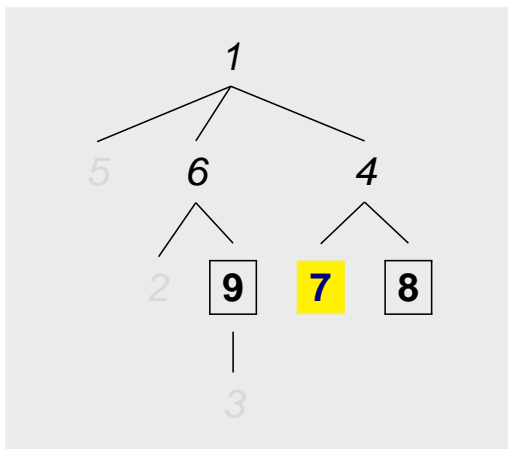
$$L(T) = \{5, 7, 8, 9\}, v = 5, t = (6, 9, 1)$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



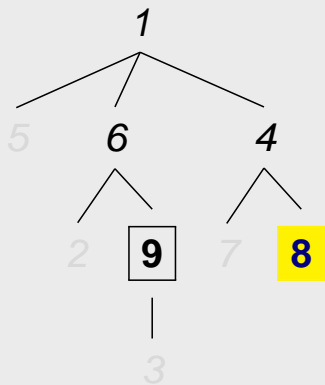
$$L(T) = \{7, 8, 9\}, v = 7, t = (6, 9, 1, 4)$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



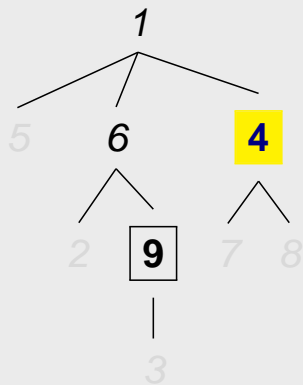
$$L(T) = \{8, 9\}, v = 8, t = (6, 9, 1, 4, 4)$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



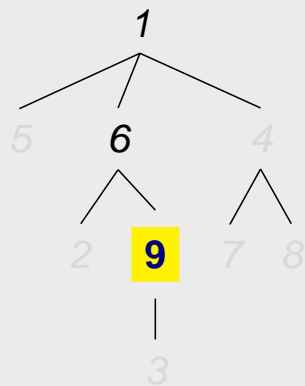
$$L(T) = \{9, 4\}, v = 4, t = (6, 9, 1, 4, 4, \mathbf{1})$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



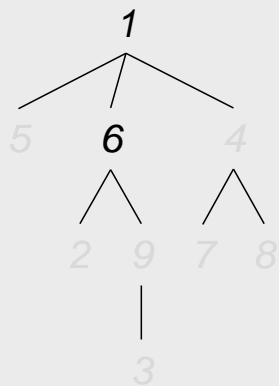
$$L(T) = \{9\}, v = 9, t = (6, 9, 1, 4, 4, 1, \mathbf{6})$$

Sequências de Prüfer

Mapeia árvore em seq. numérica

- Para uma árvore T seja $L(T)$ o conjunto de nós folha de T

```
1: for  $k \in \{1, \dots, |V| - 2\}$  do  
2:    $v = \min L(T)$ ;  
3:   seja  $e$  a única aresta incidente a  $v$ ;  
4:   seja  $t_k \neq v$  o outro nó incidente a  $e$ ;  
5:    $T \leftarrow T \setminus \{v\}$ ;  
6: end for  
7: return  $t = (t_1, \dots, t_{|V|-2})$ 
```



$$L(T) = \emptyset, t = (6, 9, 1, 4, 4, 1, 6)$$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

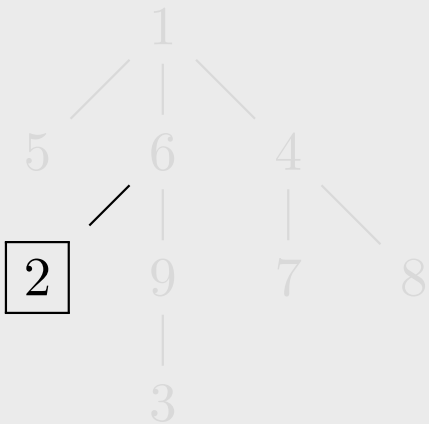


$$V \setminus t = \{2, 3, 5, 7, 8\},$$
$$t = (6, 9, 1, 4, 4, 1, 6)$$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

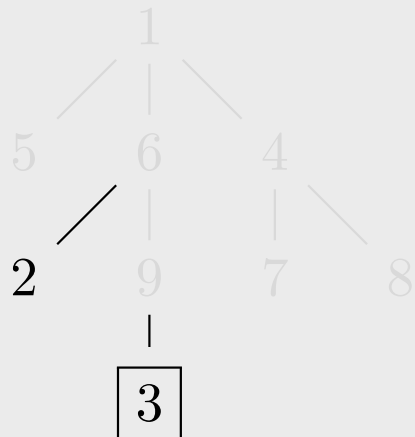


$V \setminus t = \{\boxed{2}, 3, 5, 7, 8\}$, $\ell = 2$,
 $t = (\textcolor{red}{6}, 9, 1, 4, 4, 1, 6)$, edge $\{2, 6\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

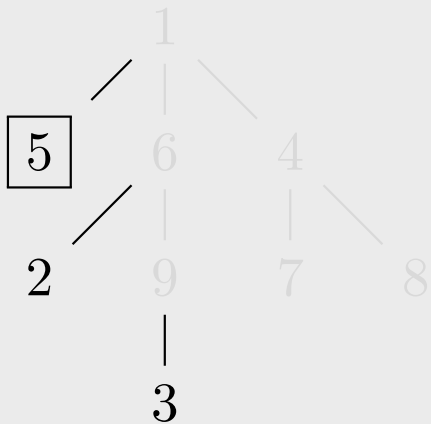


$V \setminus t = \{\boxed{3}, 5, 7, 8\}$, $\ell = 3$,
 $t = (\textcolor{red}{9}, 1, 4, 4, 1, 6)$, **edge** $\{3, 9\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

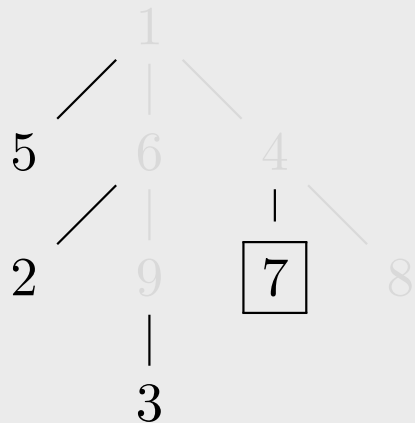


$V \setminus t = \{\boxed{5}, 7, 8, \textcolor{blue}{9}\}$, $\ell = 5$, $t = (\textcolor{red}{1}, 4, 4, 1, 6)$,
edge $\{5, 1\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

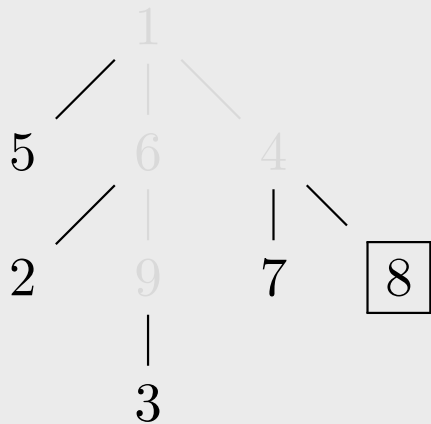


$V \setminus t = \{\boxed{7}, 8, 9\}$, $\ell = 7$, $t = (\textcolor{red}{4}, 4, 1, 6)$,
edge $\{7, 4\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

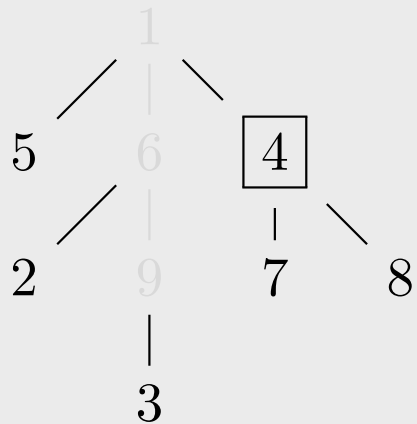


$V \setminus t = \{\boxed{8}, 9\}$, $\ell = 8$, $t = (4, 1, 6)$,
edge $\{8, 4\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

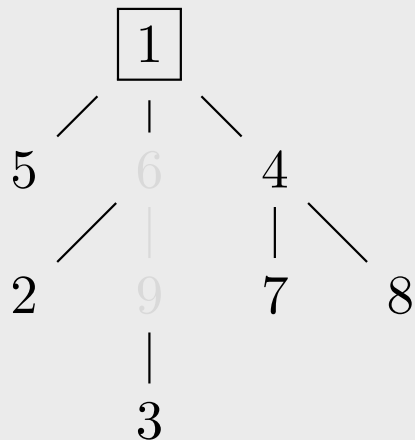


$V \setminus t = \{9, \boxed{4}\}$, $\ell = 4$, $t = (\textcolor{red}{1}, 6)$,
edge $\{4, 1\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela

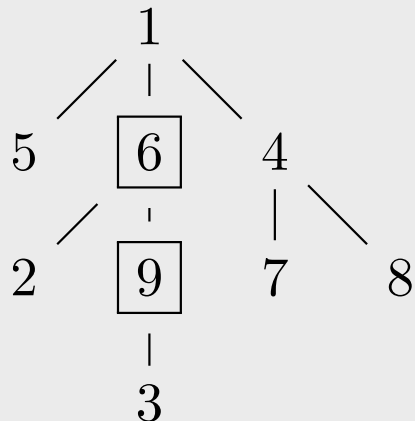


$V \setminus t = \{\boxed{1}, 9\}$, $\ell = 1$, $t = (\textcolor{red}{6})$,
edge $\{1, 6\}$

De volta às árvores

Mapeamento de seq.
numérica em árvore

1. Dada uma sequência de prüfer t em V
2. Encontre o menor índice ℓ em $V \setminus t$
3. Adicione $\{\ell, t_1\}$ a T
4. Remova t_1 de t
5. Remova ℓ de V
6. Repita a partir do Step 2 até que $t = \emptyset$
7. Neste ponto $|V \setminus t| = 2$ (é uma aresta): adicione ela



$V \setminus t = \{\boxed{6}, 9\},$
edge $\{6, 9\}$

Bijeção

Thm.

[Prüfer, 1918] Existe uma bijeção entre árvores e sequências

Thm.

[Cayley 1889] Seja $|V| = n$. Existem n^{n-2} árvores rotuladas em V

Psicologia e linguagem natural

Uma observação

- A maioria dos estudantes *(e não apenas os estudantes!)* acham arrays, listas, mapas, filas e pilhas “mais fáceis” do que árvores

Uma observação

- A maioria dos estudantes *(e não apenas os estudantes!)* acham arrays, listas, mapas, filas e pilhas “mais fáceis” do que árvores
- Tese 1: **a representação gráfica**
As pessoas são acostumadas a ler sequências mais do que a ler árvores

Uma observação

- A maioria dos estudantes *(e não apenas os estudantes!)* acham arrays, listas, mapas, filas e pilhas “mais fáceis” do que árvores
- Tese 1: **a representação gráfica**
As pessoas são acostumadas a ler sequências mais do que a ler árvores
- Tese 2: **iteração vs. recursão**
 - Sequências são modelos de **iteração** e árvores são modelos de **recursão**
 - A maioria das pessoas pensa iterativamente (?)

Uma observação

- A maioria dos estudantes *(e não apenas os estudantes!)* acham arrays, listas, mapas, filas e pilhas “mais fáceis” do que árvores
- Tese 1: **a representação gráfica**
As pessoas são acostumadas a ler sequências mais do que a ler árvores
- Tese 2: **iteração vs. recursão**
 - Sequências são modelos de **iteração** e árvores são modelos de **recursão**
 - A maioria das pessoas pensa iterativamente (?)
- Tese 3: **árvores requerem decisões**
 - Todo nó tem ≤ 1 próximo nó em uma sequência
nós de árvores podem ter mais do que um subnó
 - \Rightarrow Percorrer uma sequência: não envolve decisões
 - \Rightarrow Explorar uma árvore: qual deve ser o próximo subnó processado?

Linguagens e gramáticas

- Lembra de *substantivos, adjetivos, verbos transitivos* da escola?
- Analisar sentenças significa identificar seus componentes gramaticais
- Podemos analisar estes componentes recursivamente:

<u>sentença</u>	→	<i>sujeitos verbo</i>
<i>sujeito</i>	→	<i>sujeito sujeitos</i>
<i>sujeito</i>	→	<i>subst.</i>
		<i>artigo subst.</i>
		<i>adjetivos subst.</i>
		<i>artigo adjetivos subst.</i>
<i>adjetivos</i>	→	<i>adjetivo adjetivos</i>
<i>verbo</i>	→	<i>...</i>

Árvores de análise

O macio, peludo gato ron-
rona

<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...

Árvores de análise

O macio, peludo gato ronrona

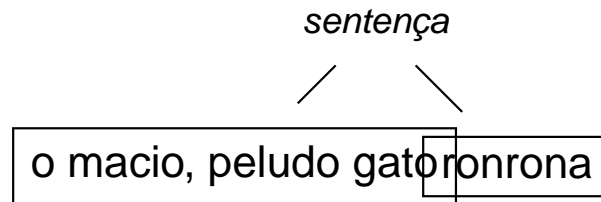
<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...

o macio, peludo gato ronrona

Árvores de análise

O macio, peludo gato ronrona

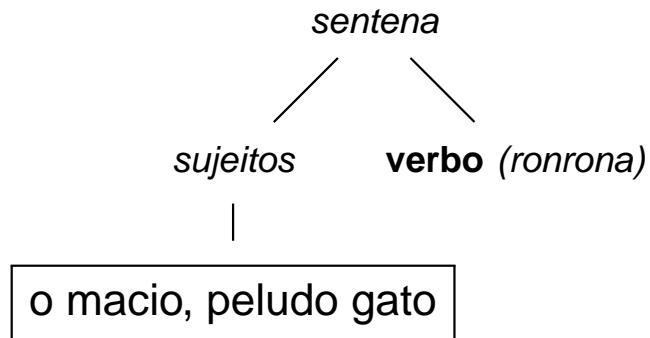
<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...



Árvores de análise

O macio, peludo gato ronrona

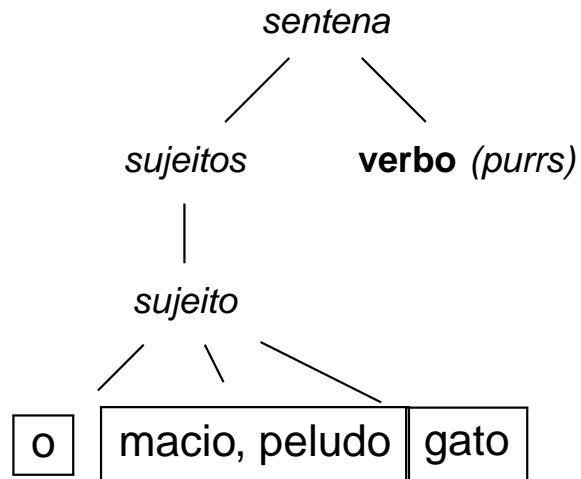
<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...



Árvores de análise

O macio, peludo gato ron-
rona

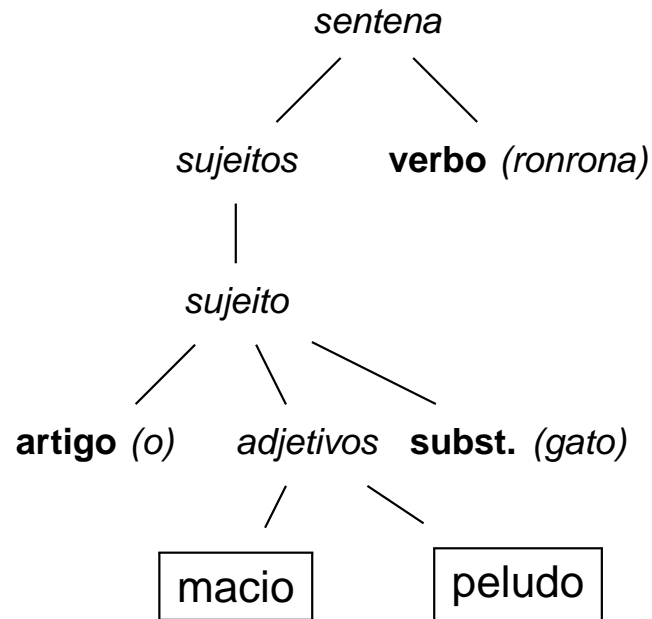
<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...



Árvores de análise

O macio, peludo gato ronrona

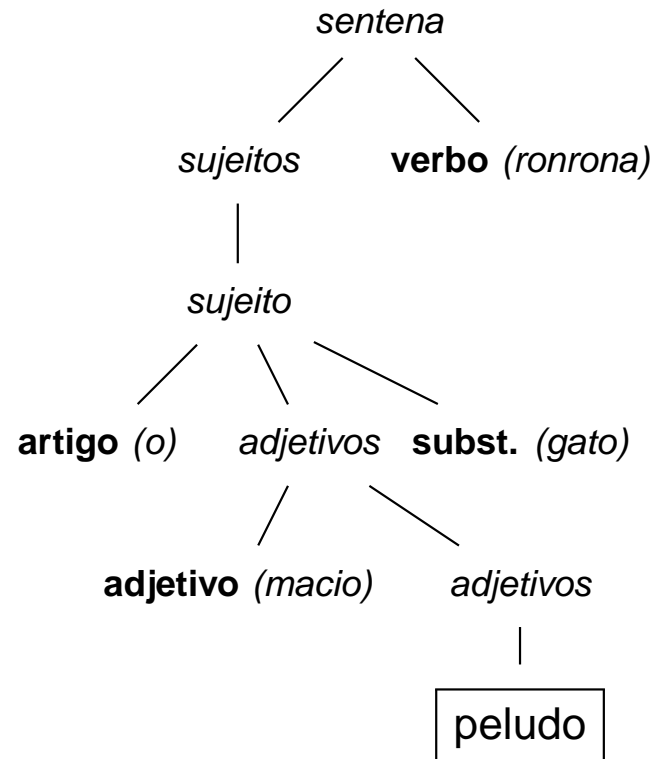
<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...



Árvores de análise

O macio, peludo gato ronrona

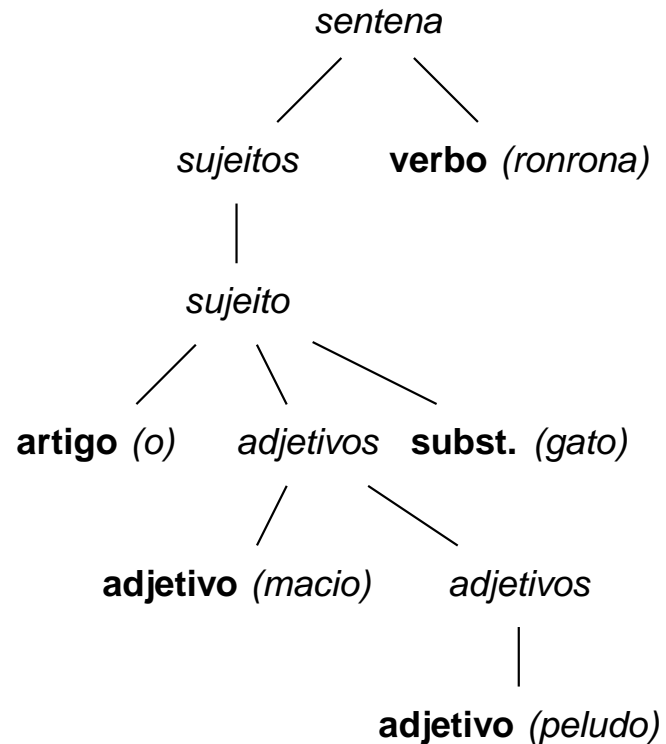
<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...



Árvores de análise

O macio, peludo gato ronrona

<u>sentença</u>	→	sujeitos verbo
sujeitos	→	sujeito sujeitos
sujeito	→	subst.
		artigo subst.
		adjetivos subst.
		artigo adjetivos subst.
adjetivos	→	adjetivo adjetivos
verbo	→	...



Linguagens formais e naturais

- Se existe mais de uma árvore de análise para uma sentença, a gramática é **ambígua**
- Se as diferentes árvores de análise para uma sentença levam a diferentes interpretações, a linguagem em si é ambígua
- Linguagens não-ambíguas são chamadas **formais**
(ex. *C/C++*, *Java*,...)
- Linguagens ambíguas são chamadas **naturais**
(ex. *Inglês*, *Português*,...)

Exploração de árvores

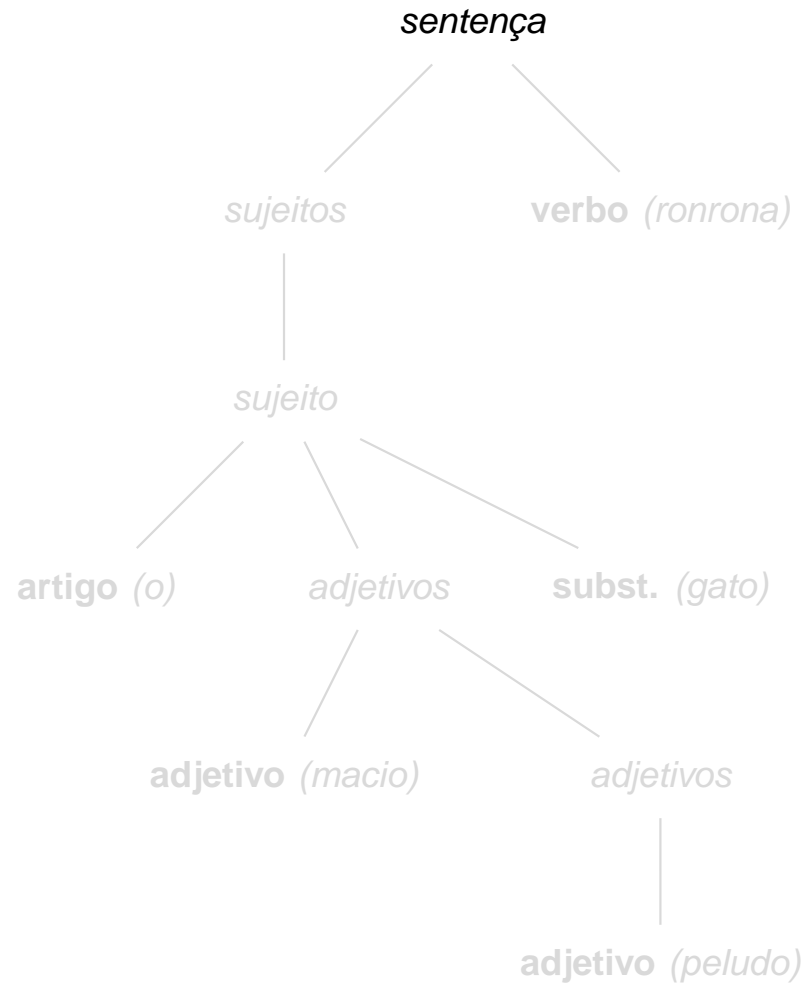
- Breadth-First Search (BFS — visto no Módulo 2)
encontra o caminho pra fora de um labirinto no menor número de passos
- Depth-First Search (DFS)
encontra o caminho pra fora do labirinto
- DFS: chamada recursiva $\text{dfs}(\text{node } v)$:
 - 1: realize (opcional) uma ação em v ;
 - 2: **for** todos os subnós u de v **do**
 - 3: $\text{dfs}(u)$;
 - 4: **end for**
 - 5: realize (opcional) uma ação em v ;
- DFS é $\text{dfs}(\text{raiz})$

Tese [século XX]: nosso cérebro trata sentenças como labirintos, e usa DFS para encontrar uma saída (i.e., analisa elas)

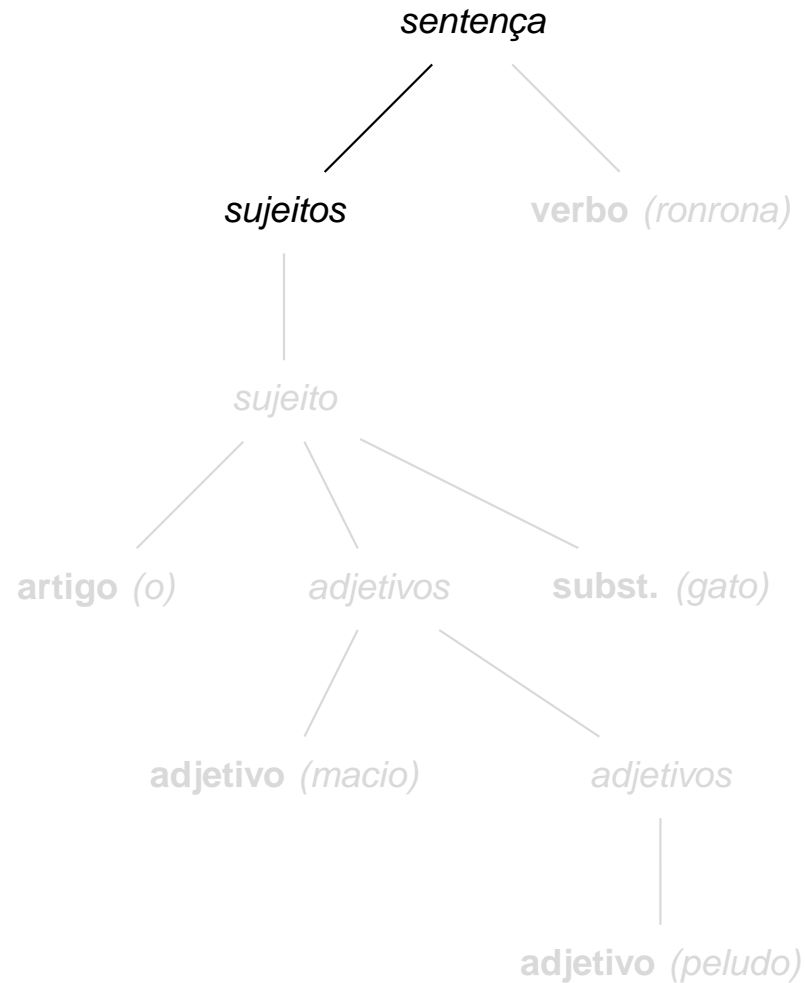
DFS: explorando uma árvore de análise



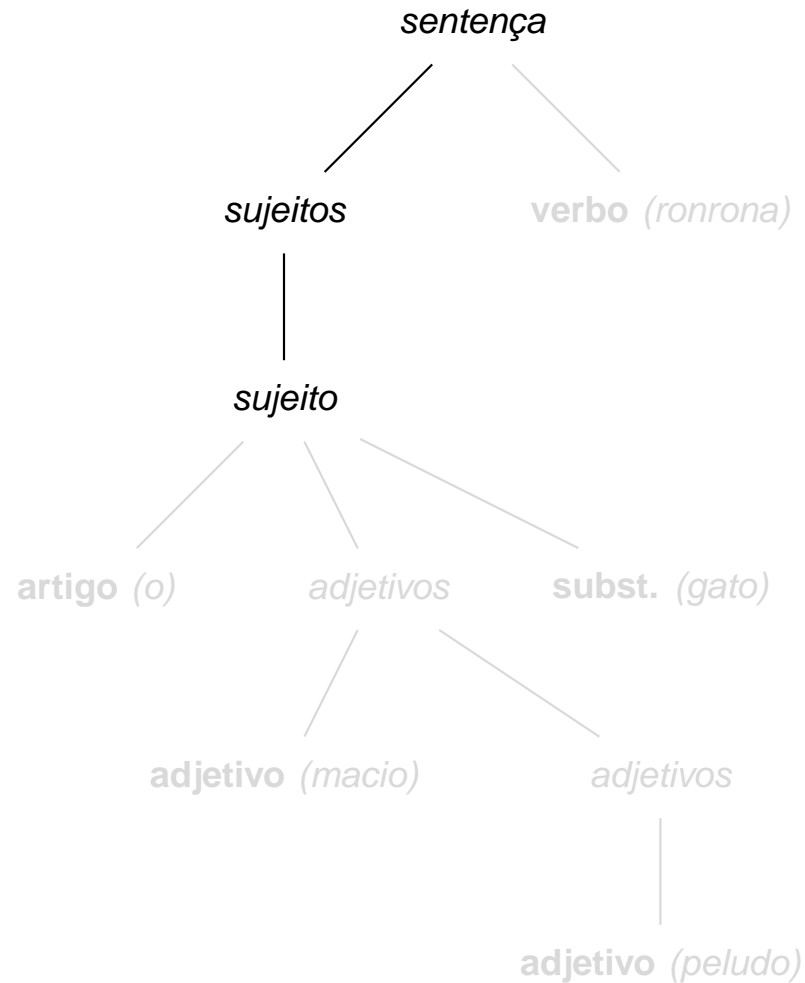
DFS: explorando uma árvore de análise



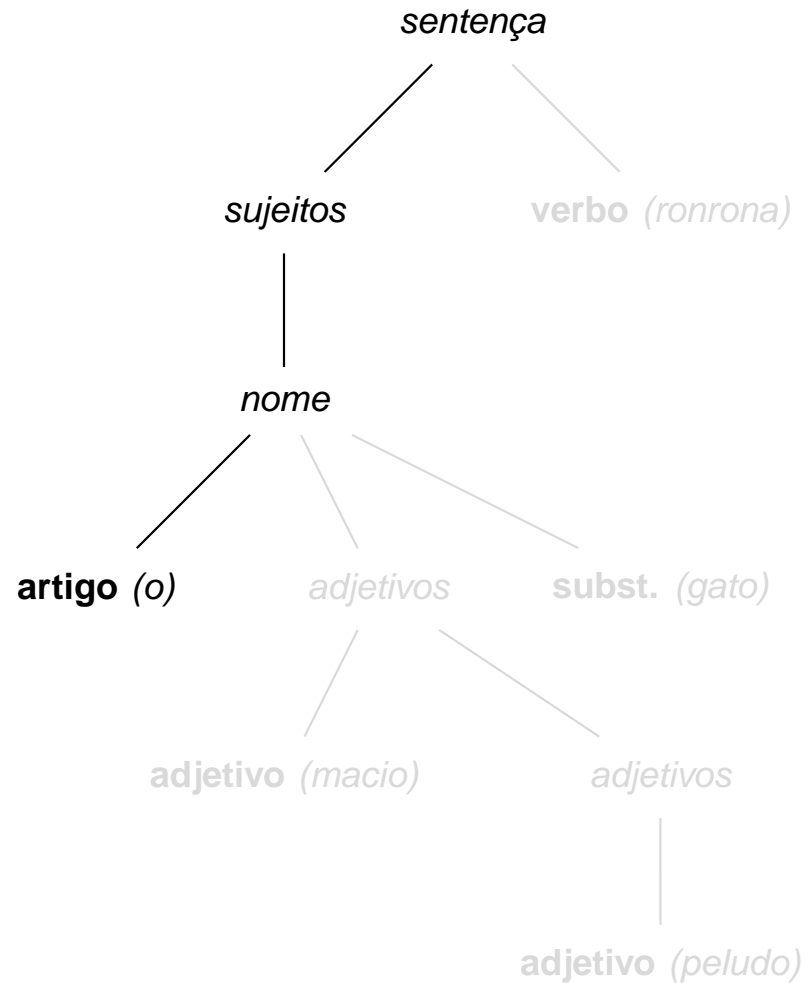
DFS: explorando uma árvore de análise



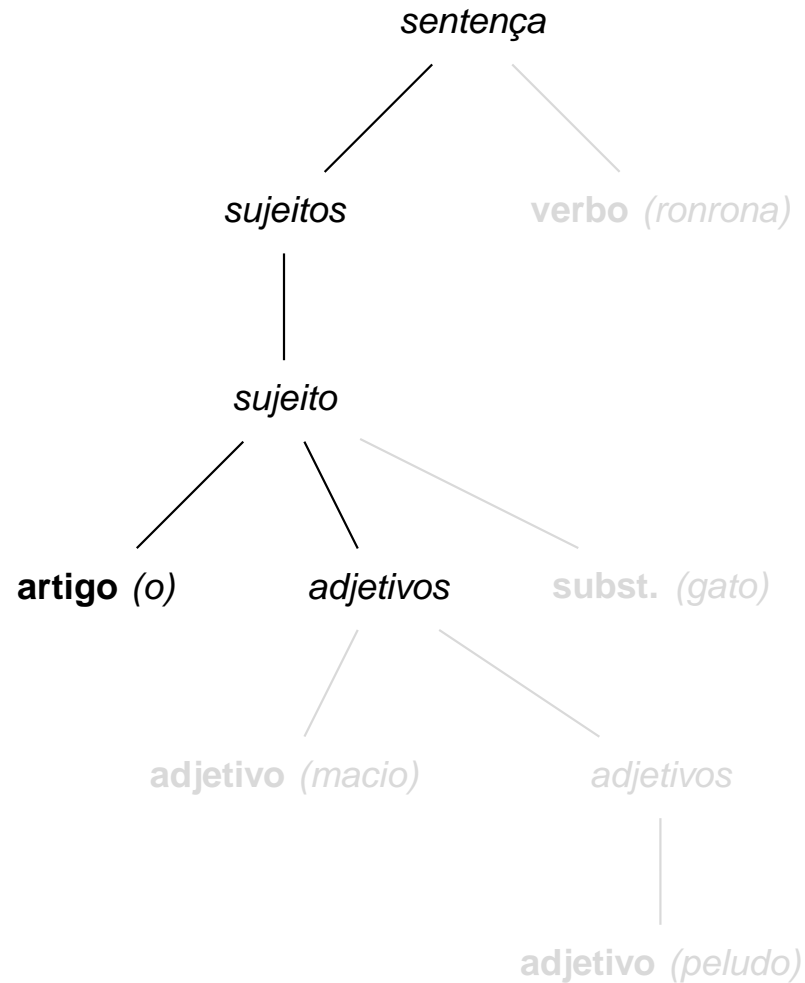
DFS: explorando uma árvore de análise



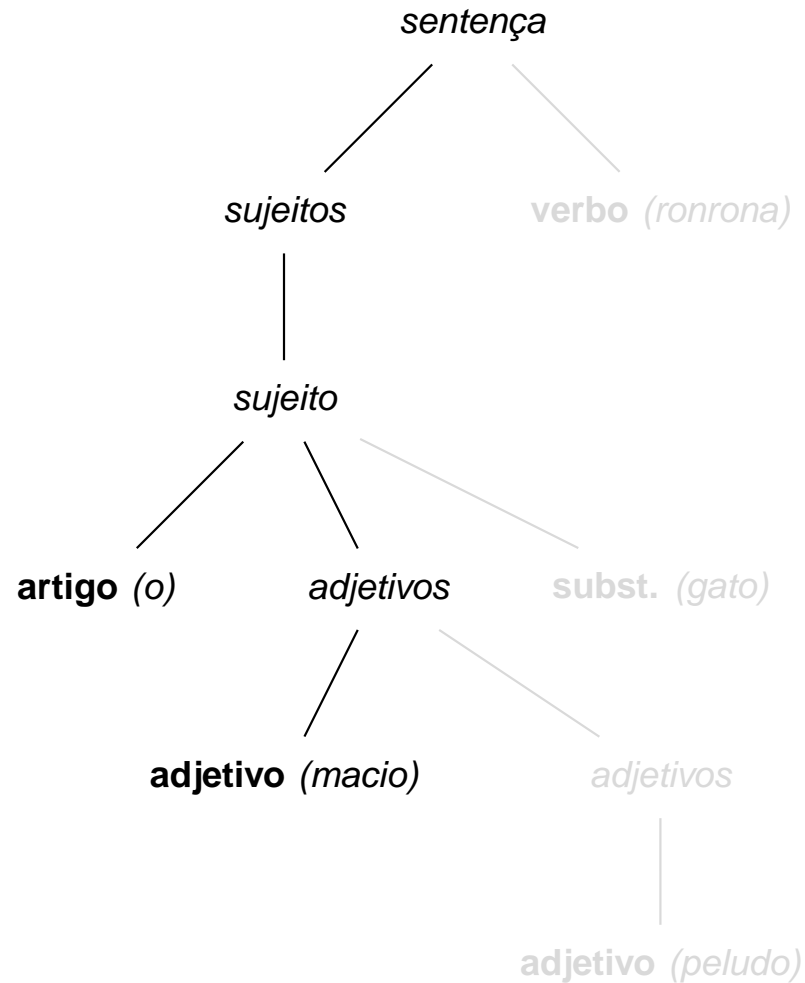
DFS: explorando uma árvore de análise



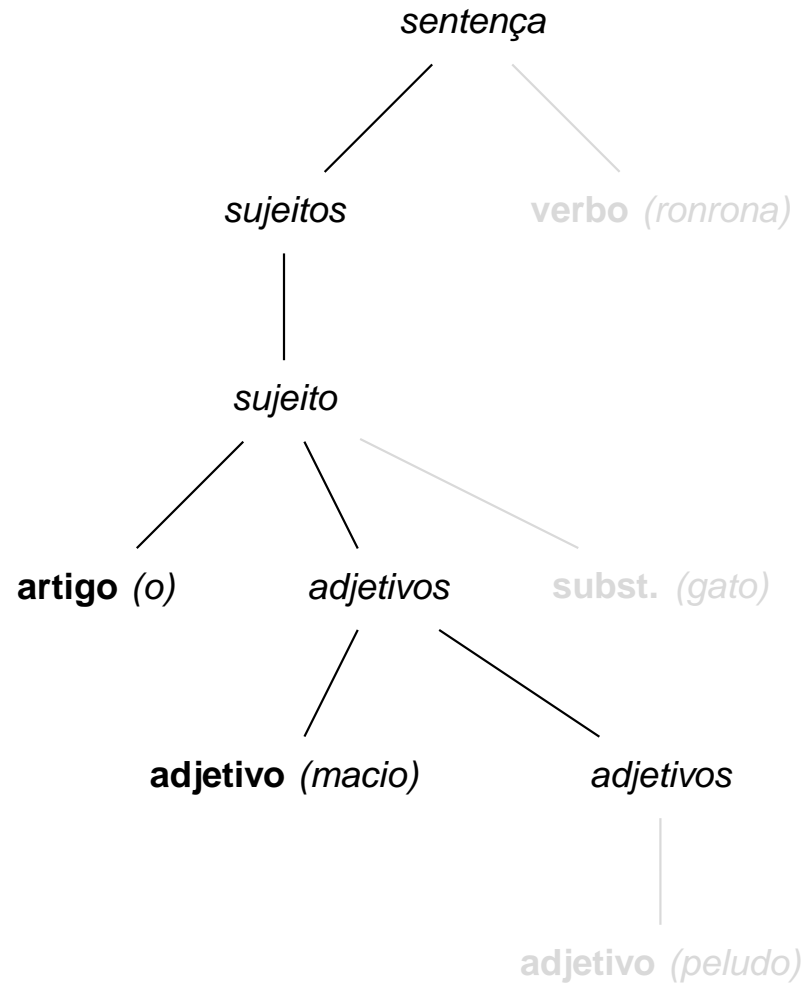
DFS: explorando uma árvore de análise



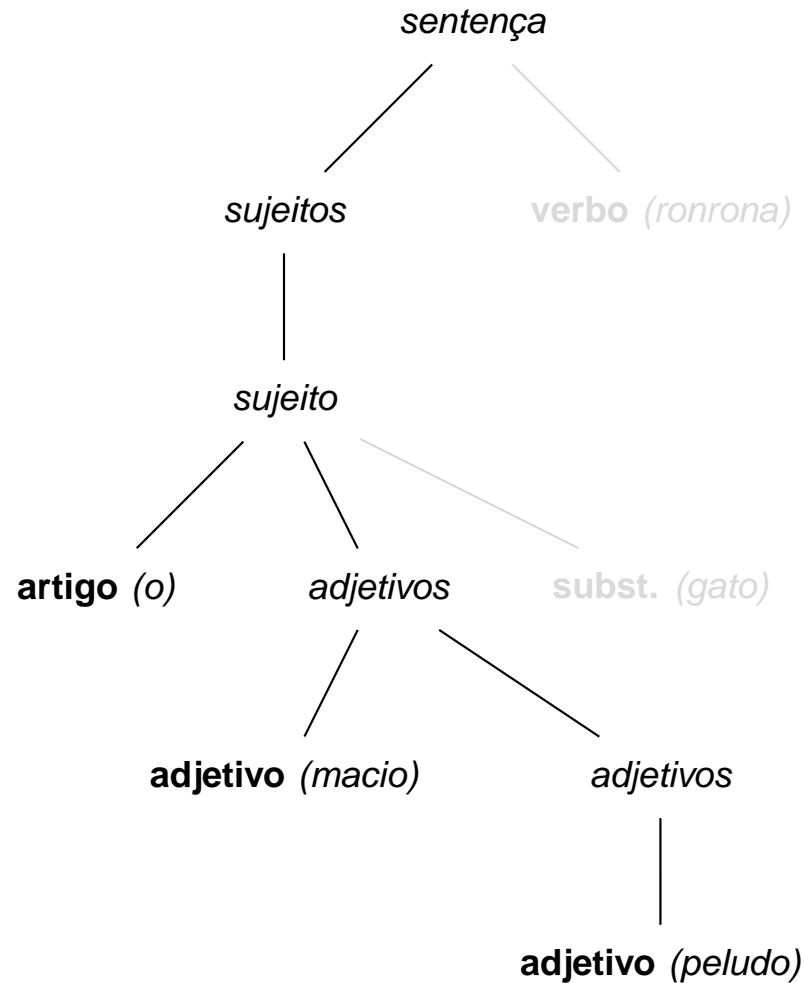
DFS: explorando uma árvore de análise



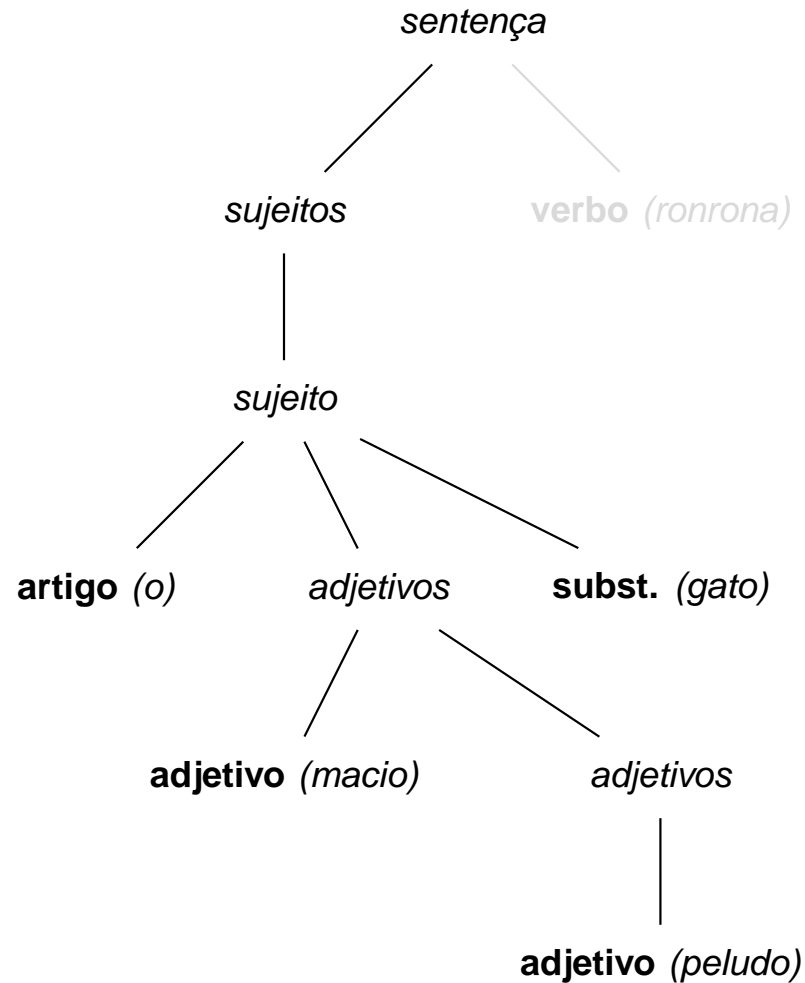
DFS: explorando uma árvore de análise



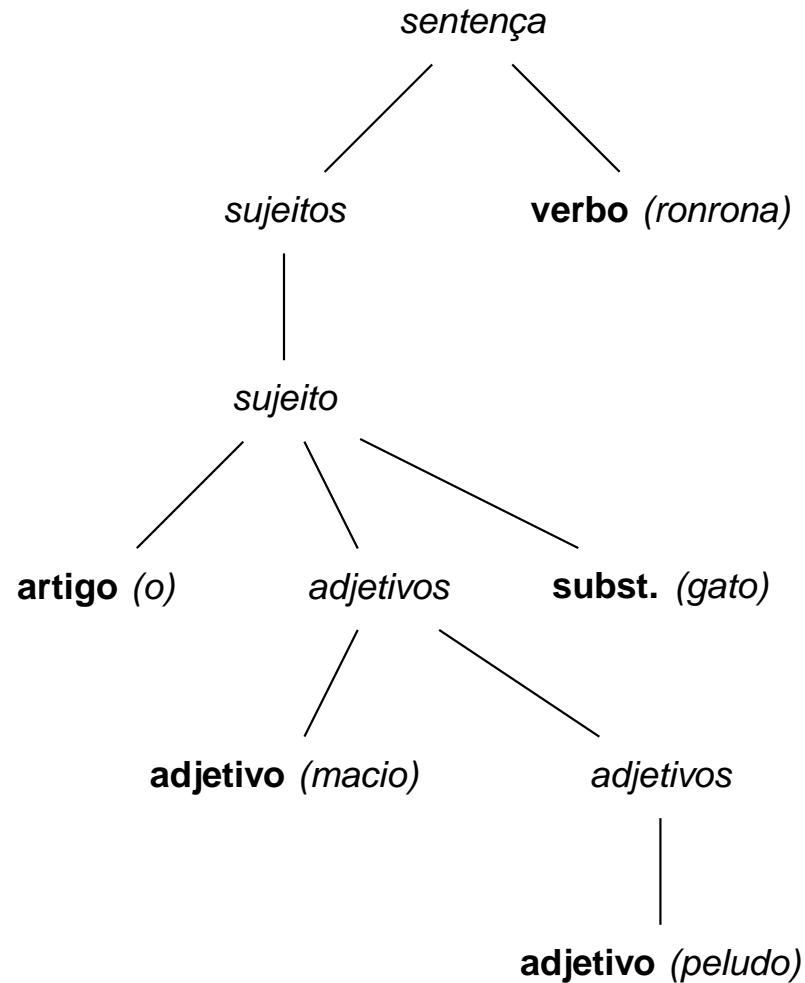
DFS: explorando uma árvore de análise



DFS: explorando uma árvore de análise



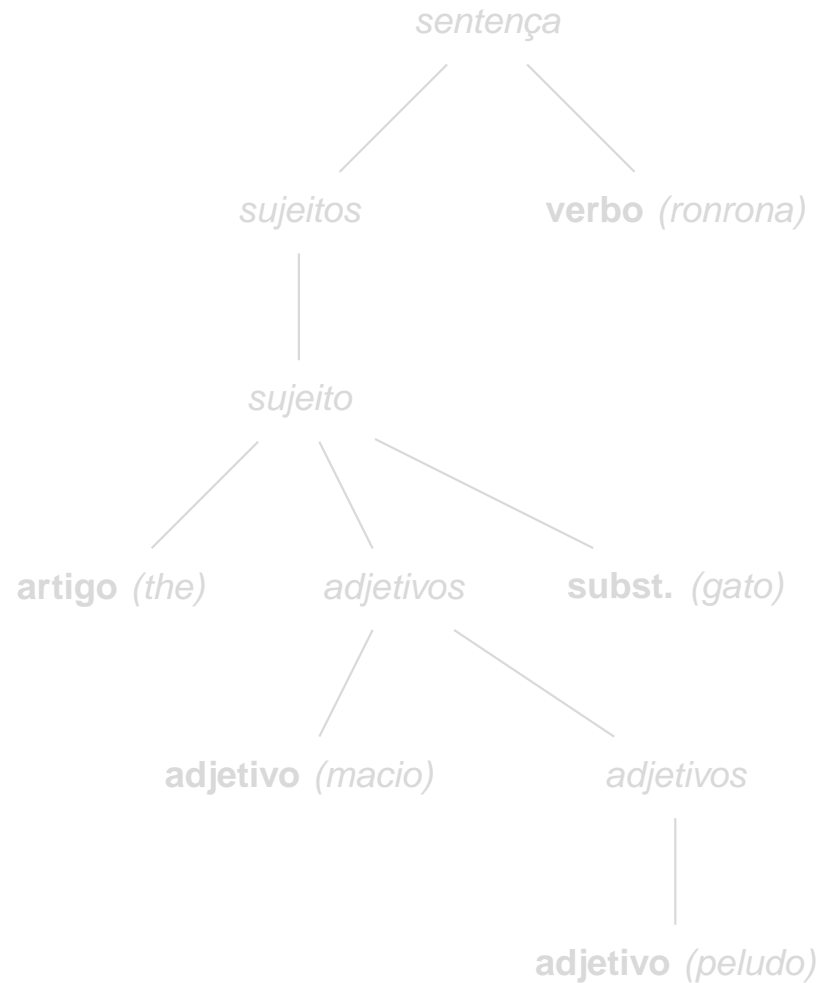
DFS: explorando uma árvore de análise



Quanta memória?

- Quanto precisamos “lembrar” durante a DFS?
- Note que o código recursivo não faz uso explícito de memória
- Do módulo 3, lembre que recursão é implementada utilizando pilha
- Qual o tamanho máximo da pilha ao se explorar uma árvore com DFS?

DFS em árvores de análise: memória



0

DFS em árvores de análise: memória



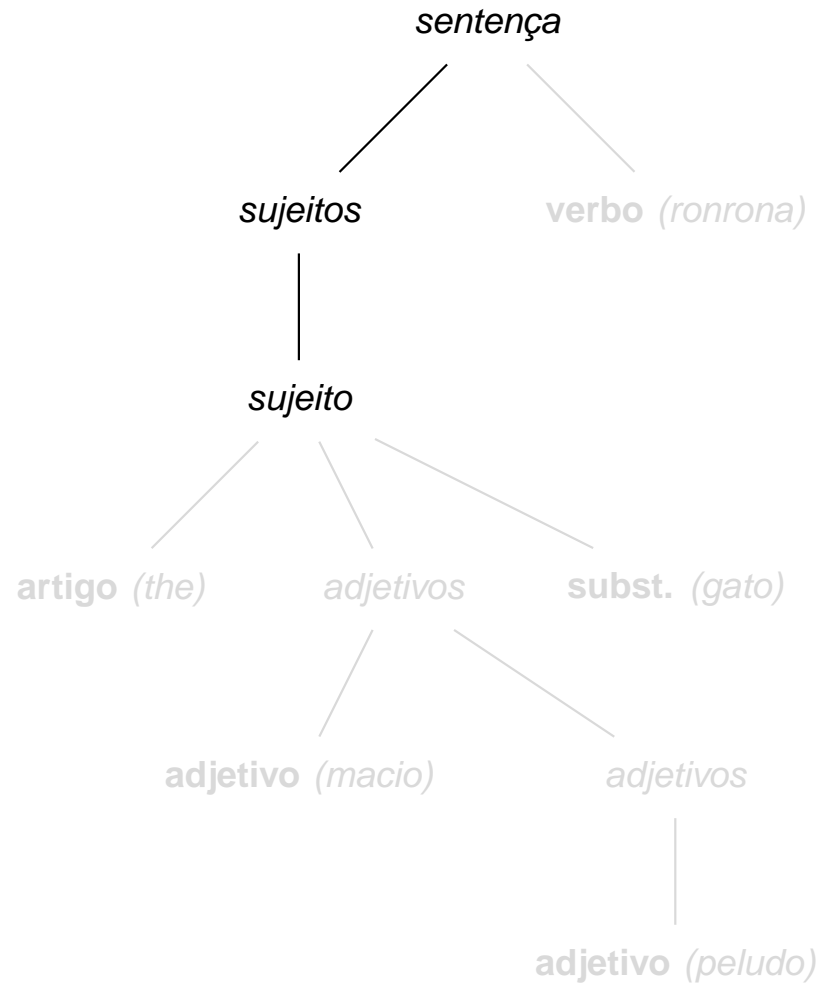
0

DFS em árvores de análise: memória



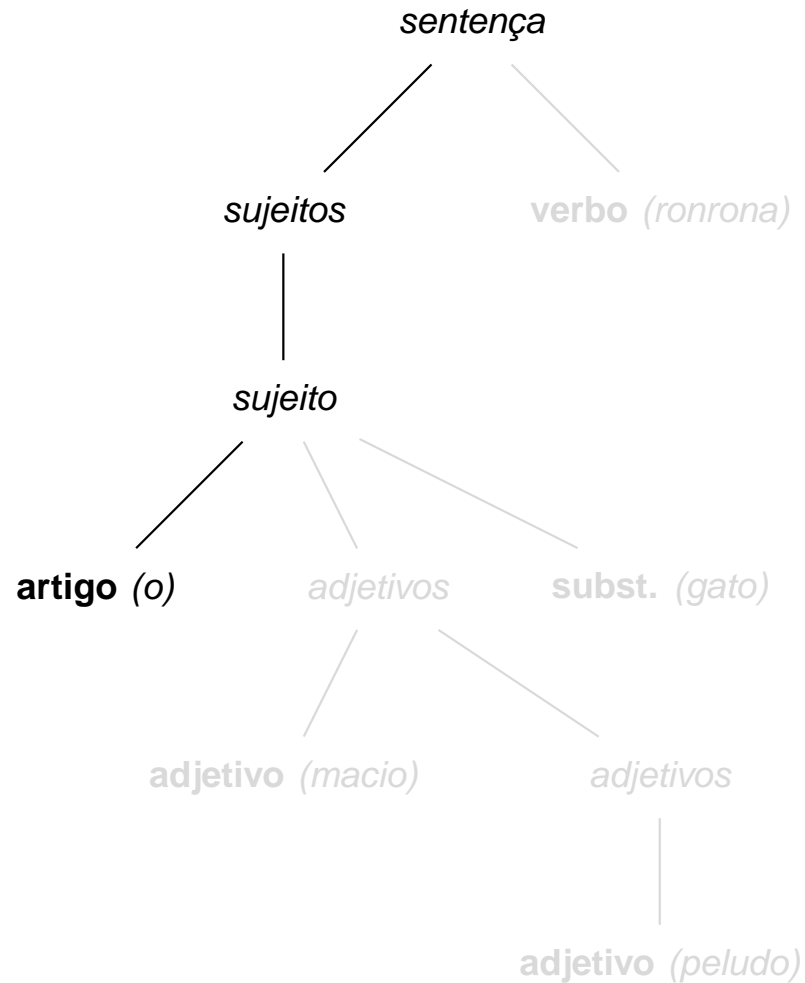
1

DFS em árvores de análise: memória



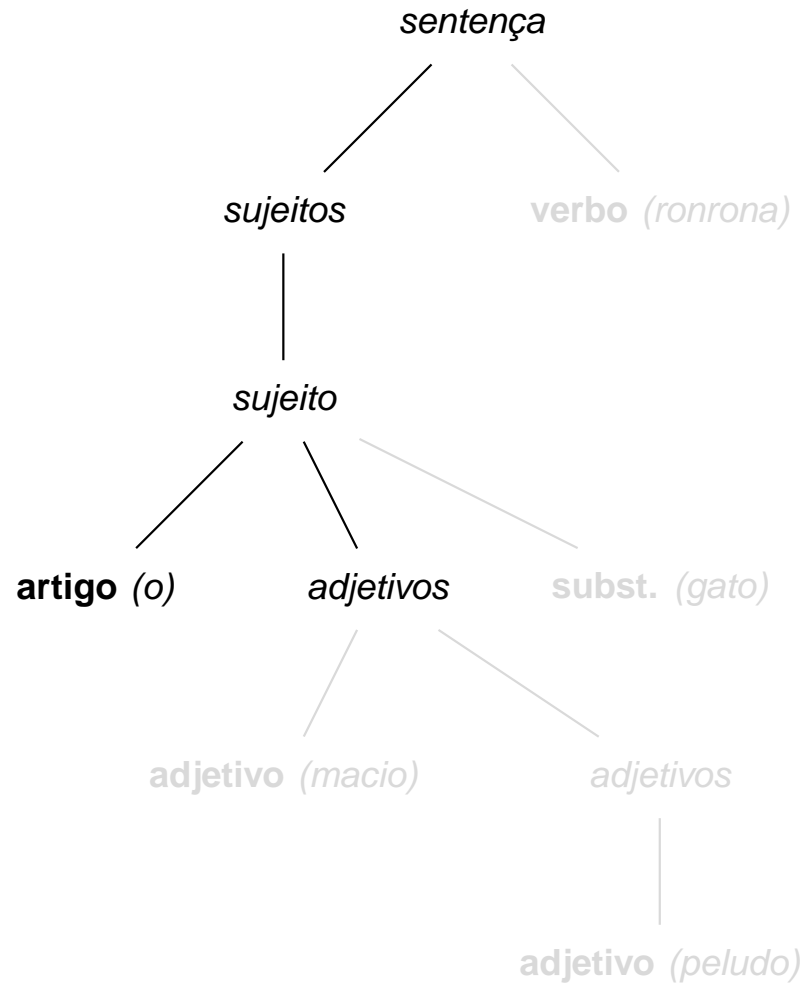
2

DFS em árvores de análise: memória



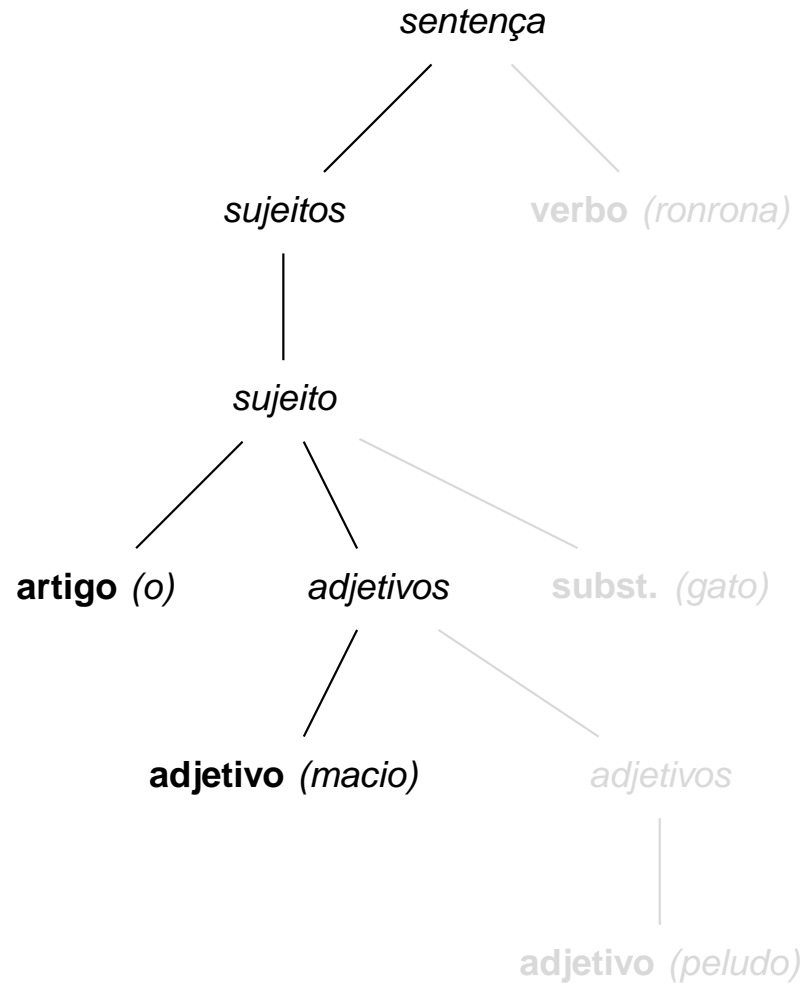
3

DFS em árvores de análise: memória



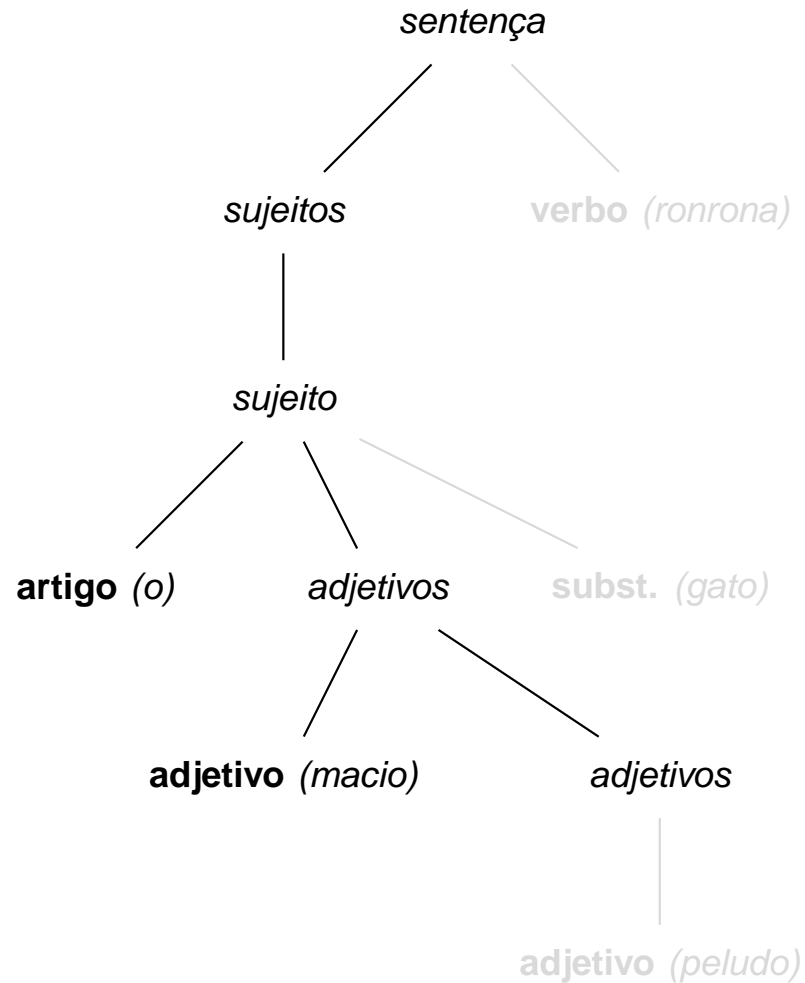
3

DFS em árvores de análise: memória



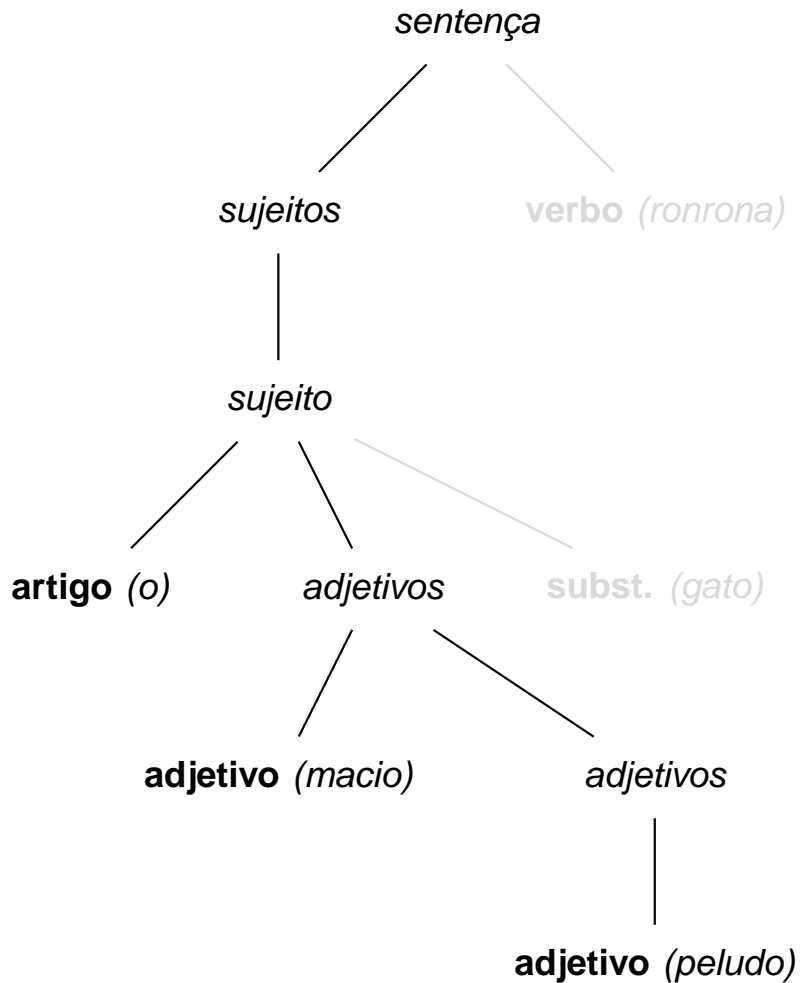
4

DFS em árvores de análise: memória



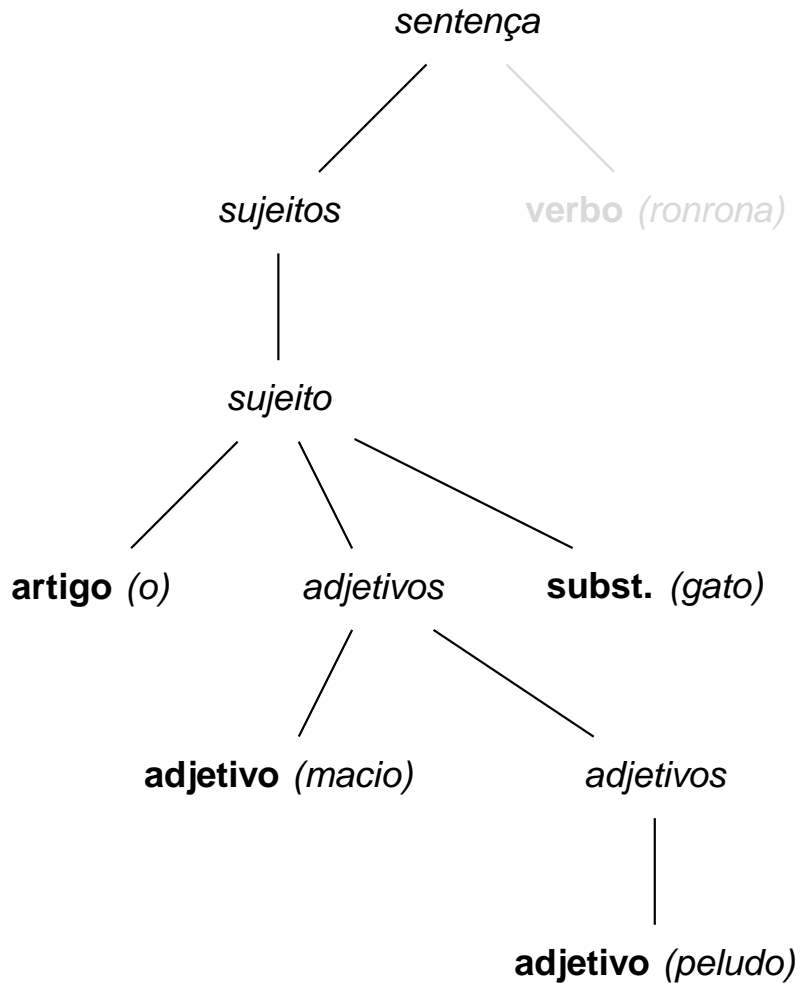
4

DFS em árvores de análise: memória



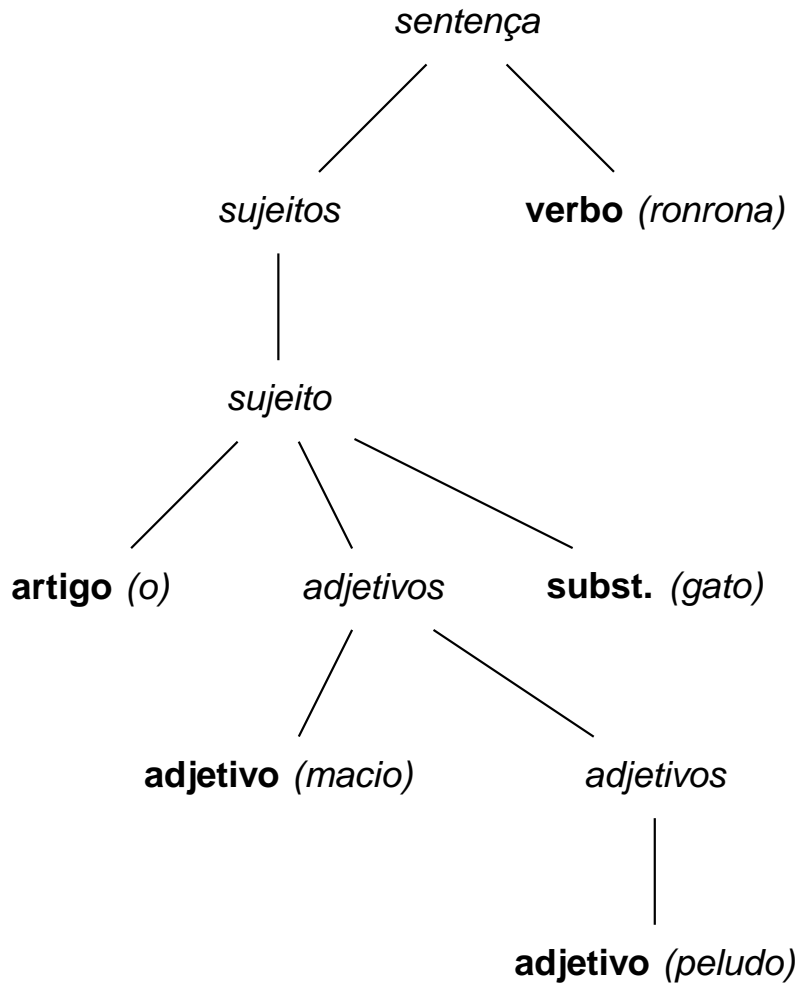
max=5 **5**

DFS em árvores de análise: memória



max=5 3

DFS em árvores de análise: memória



max=5

1

Memória e profundidade

A quantidade de memória é igual à profundidade da árvore

profundidade = caminho mais longo da raiz até uma folha

Busca em profundidade (DFS)

Exame de um (Dí)grafo

- DFS explora os nós de uma árvore a partir da raiz, e visita cada nó (conectado) apenas uma vez
- Generalização: examina os nós de um dígrafo (ou os vértices de um grafo) a partir do nó s

Require: $G = (V, A)$, $s \in V$, $R = \{s\}$, $Q = \{s\}$

```
1: while  $Q \neq \emptyset$  do  
2:   escolha  $v \in Q$  //  $v$  examinado  
3:    $Q \leftarrow Q \setminus \{v\}$   
4:   for  $w \in N^+(v) \setminus R$  do  
5:      $R \leftarrow R \cup \{w\}$   
6:      $Q \leftarrow Q \cup \{w\}$   
7:   end for  
8: end while
```

O algoritmo é correto

Thm.

Se existe um caminho orientado P de s a $z \in V$, então o algoritmo examina z

Proof

- Suponha que a afirm. seja falsa, então $\exists (x, y) \in P$ com $x \in R$ e $y \notin R$ (c.c., por indução no tamanho do passeio, $z \in R$ pelo Step 5 e portanto em Q pelo Step 6)
- Pelo Step 6, x foi adicionado a Q
- O algoritmo não para antes de eliminar x de Q no Step 3 em alguma iteração
- Quando isto acontece os nós em $N^+(v)$ são adicionados a R e a Q .
- Portanto $y \notin N^+(x)$, o que implica $(x, y) \notin P$, o que resulta em uma contradição.

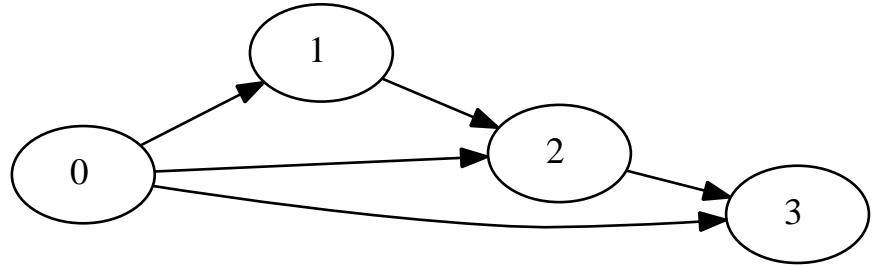
Armazenando um grafo

- Visto no Módulo 1: use *array denteado* (também chamado **lista de adjacência**)

$$N^+(0) = (1, 2, 3)$$

$$N^+(1) = (2)$$

$$N^+(2) = (3)$$



- Visto no módulo 2: use *lista de arcos*

$$L = ((0, 1), (0, 2), (0, 3), (1, 2), (2, 3))$$

Eficiência diferente para diferentes algoritmos

O algoritmo leva $O(n + m)$

Thm.

Se o dígrafo está codificado com listas de adjacência, o algoritmo leva tempo proporcional a $O(n + m)$ no pior caso

Proof

● **Cada nó é considerado uma única vez:**

● Sempre que um nó x é eliminado de Q , ele foi previamente inserido pelo Step 6, o que significa que ele também foi adicionado a R no Step 5

● No Step 4, x nunca é readicionado a Q

● **Cada arco (x, y) é considerado uma única vez:**

● Quando $x = v$ no Step 2 então $y \in N^+(x)$, então ou $y = w$ no Step 4 ou verifica-se que $y \in R$

● Em ambos os casos, a relação (x, y) foi considerada uma única vez

A escolha de $v \in Q$

- No Step 2, a escolha de $v \in Q$ determina a ordem na qual os nós são examinados
- Podemos alterar isto usando diferentes estruturas de dados para implementar o conjunto Q
- Duas estruturas de dados são comumente usadas:

1. Pilhas

DFS : isto corresponde à ordem (LIFO)

2. Filas

BFS: isto corresponde à ordem (FIFO)

Fim do Módulo 6