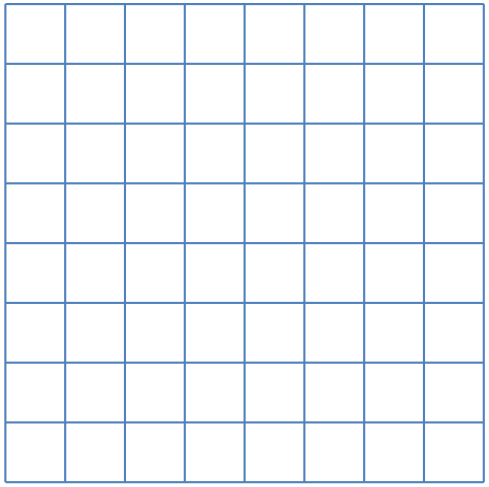
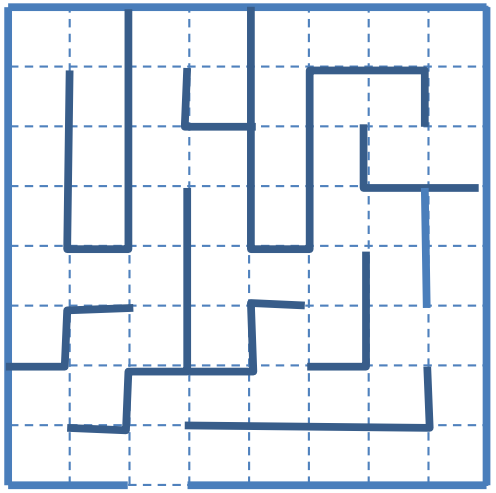
INF 1010
Estruturas de Dados Avançadas

Partições dinâmicas

Motivação:
Como criar um Labirinto?

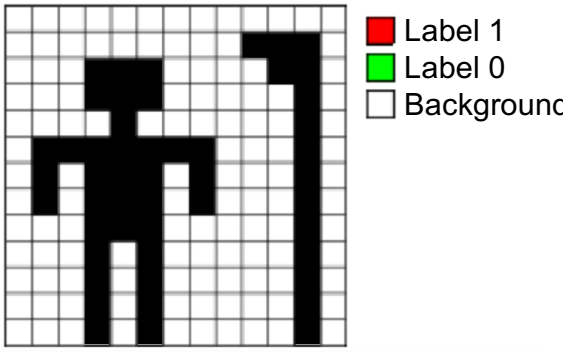




Algoritmo

```
Numere todas as células da grade de 0 a N*N-1;
Crie todas as paredes isolando todas as células;
Repita {
   Escolha randomicamente uma parede interior;
   Se as duas células adjacentes a esta parede,
       x e y, estiverem em diferentes partições
       remova a parede e faça uma união(x,y);
}
Até que todas as células estejam no na mesma
partição.
```

Motivação: Detecção de Componentes Conectadas

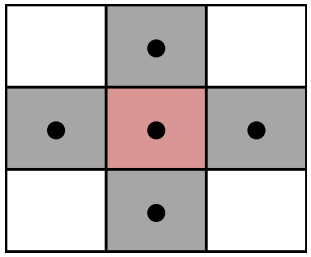


- **Objetivo:**

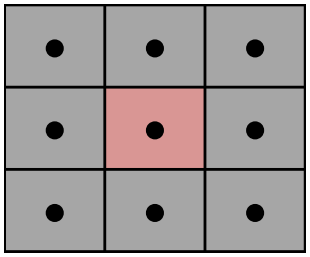
  detectar componentes conectadas encontra regiões de pixels que possuem o mesmo valor.

- Dois **pixels** são **conectados** quando:
  - eles são vizinhos
  - seus valores de cinza satisfazem um critério específico de similaridade

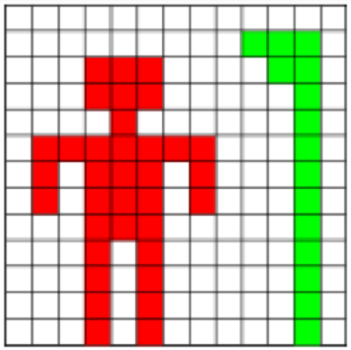
  Obs: imagem binária devem possuir o mesmo valor.



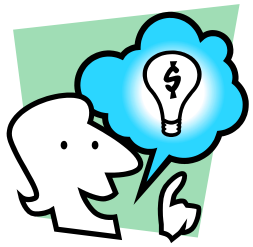
4 - conectado



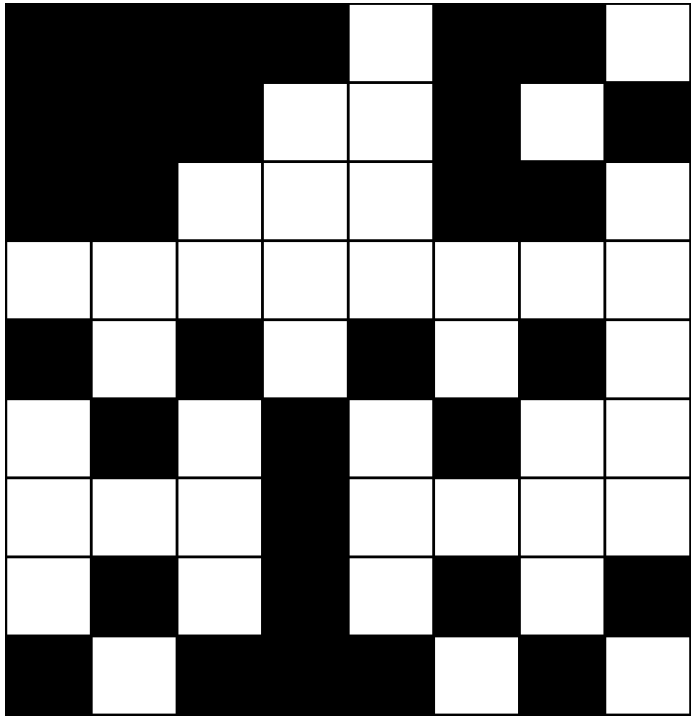
8 - conectado



Algoritmo



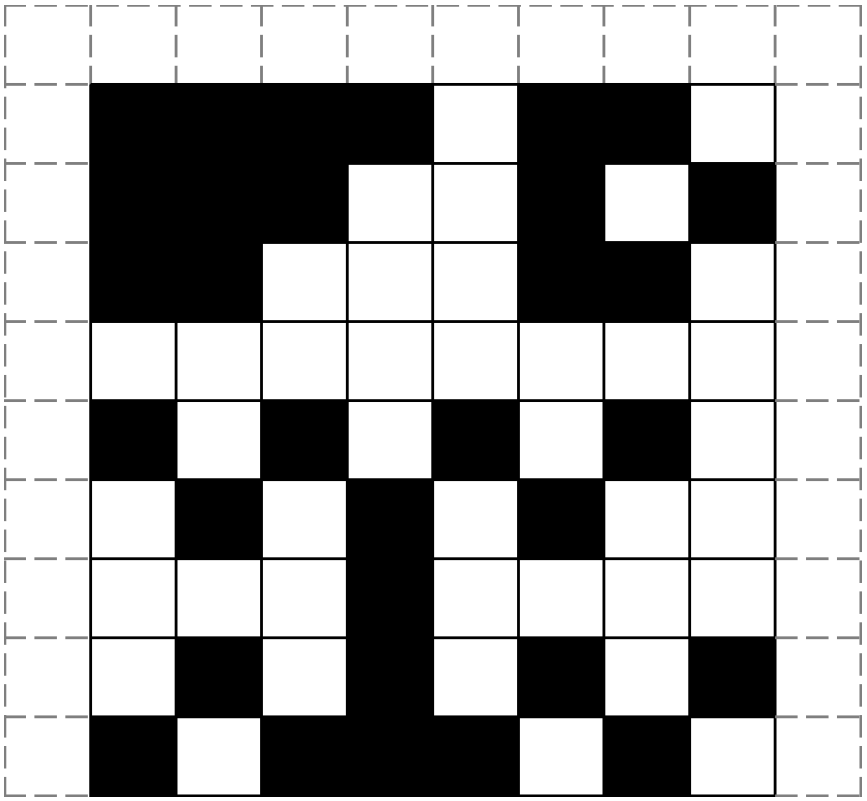
Percorrer a imagem e etiquetar as componentes.



0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		2		2		2	
	2		2		2		
			2				
	2		2		2		2
2		2	2	2		2	

Detectando componentes

Dada uma imagem binária $\mathbf{I}$.



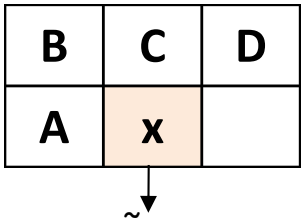
PARTE 1:

Varrer a imagem da esquerda para a direita e de cima para baixo avaliando cada pixel preto e atribuindo uma etiqueta conforme sua vizinhança.

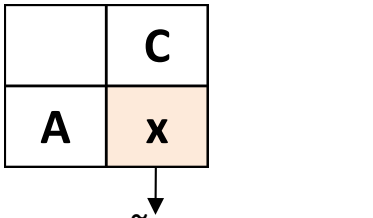
PARTE 2:

Percorrer a imagem mais uma vez atribuindo uma única etiqueta para cada componente conectada.

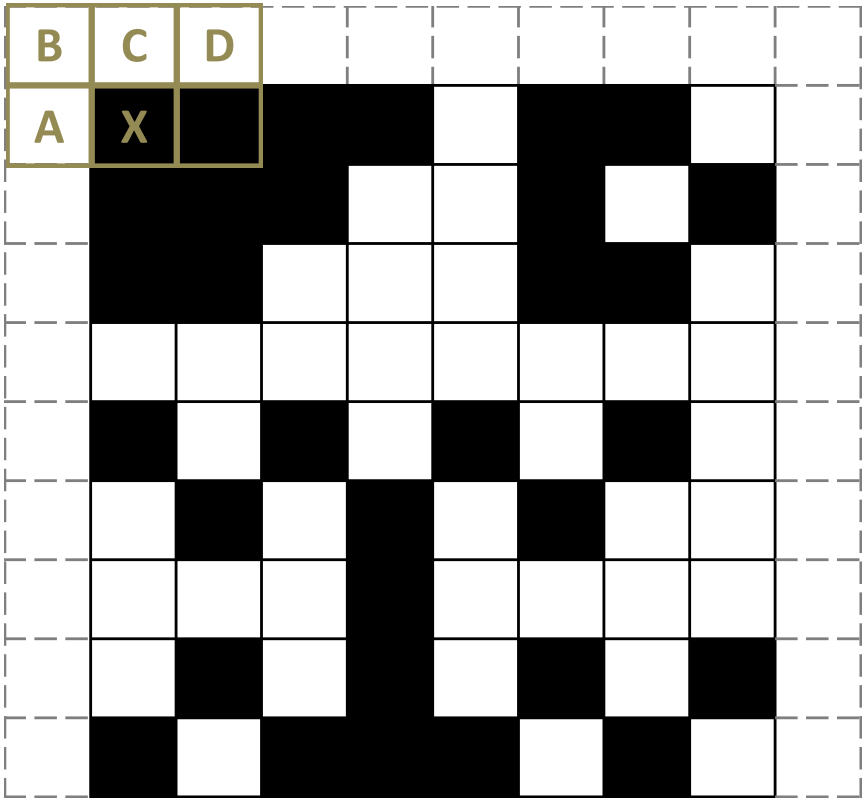
8-CONECTADA



4-CONECTADA



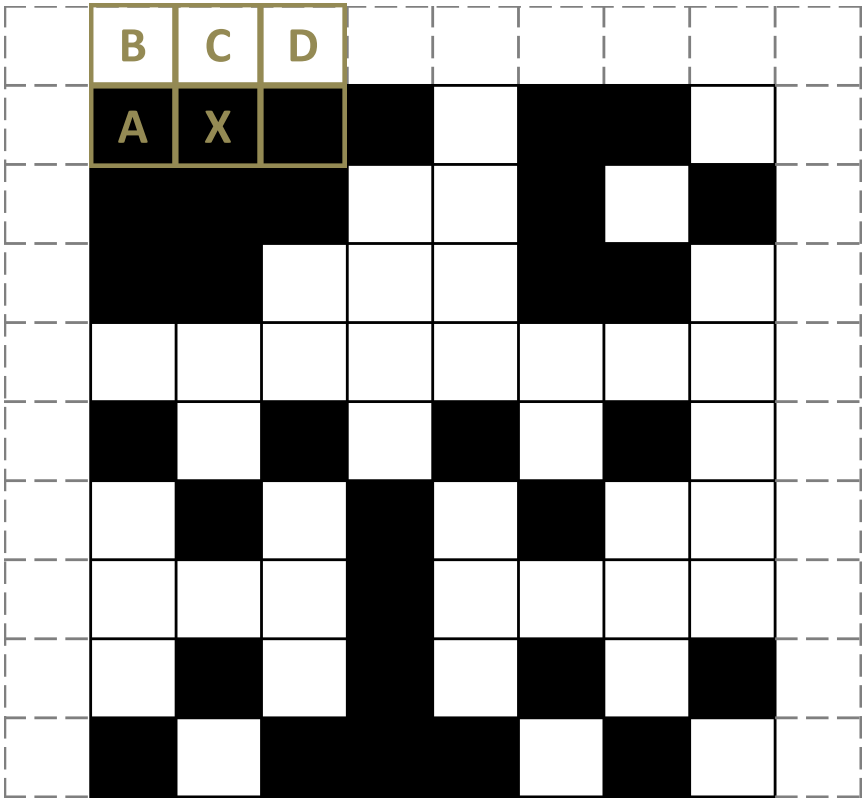
Detectando componentes



0							

Tabela de Equivalência

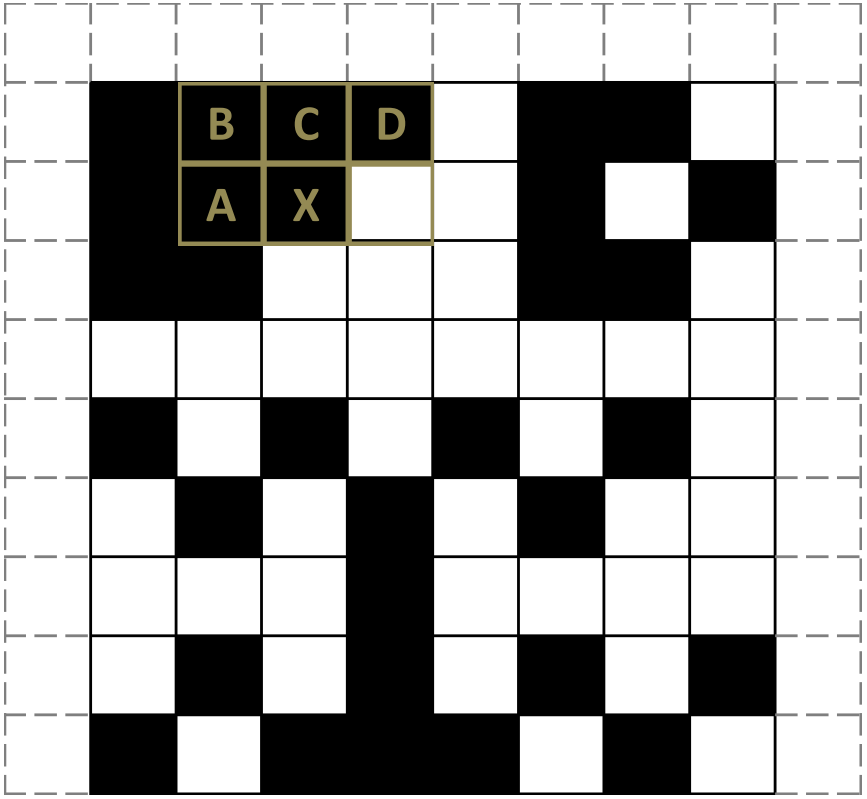
Detectando componentes



0	0						

Tabela de Equivalência

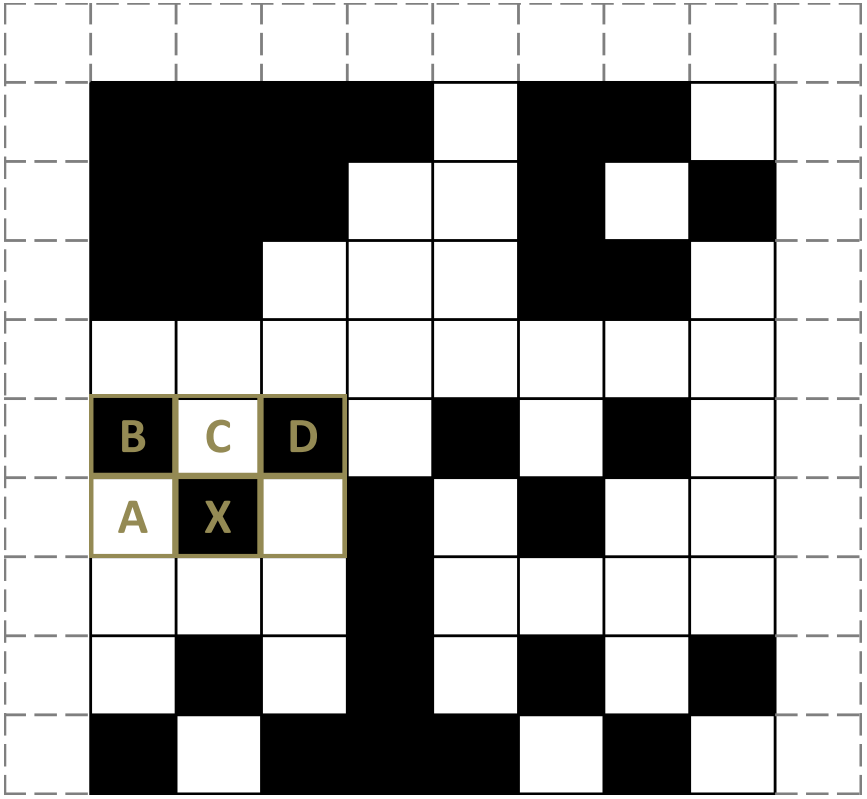
Detectando componentes



0	0	0	0		1	1	
0	0	0					

Tabela de Equivalência

Detectando componentes

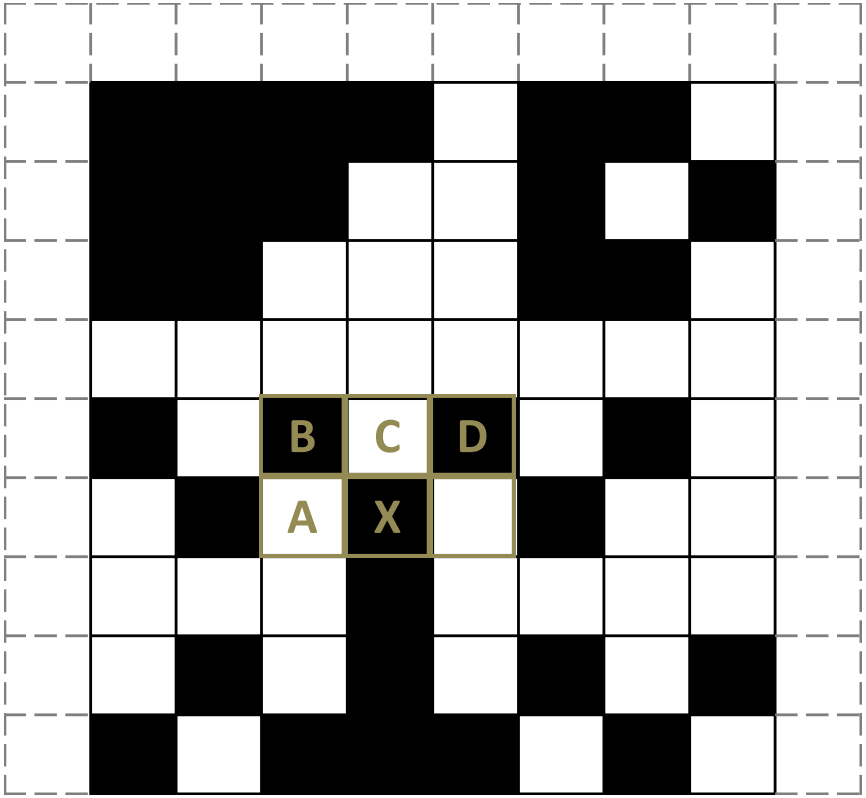


0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		3		4		5	
	2						

Tabela de Equivalência

2	3

Detectando componentes

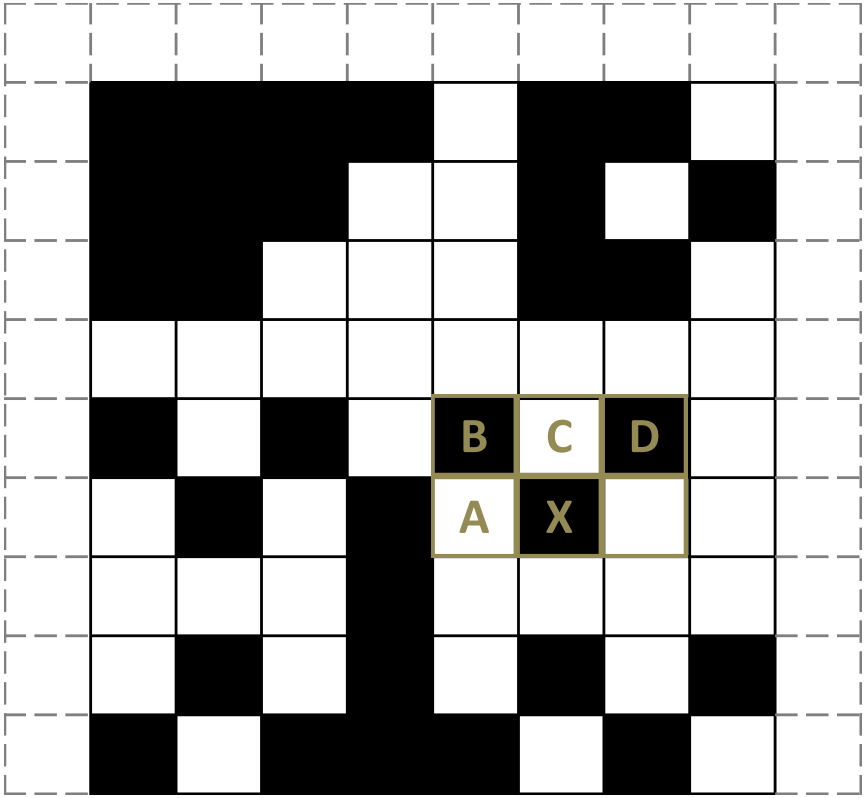


0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		3		4		5	
	2		3				

Tabela de Equivalência

2	3
3	4

Detectando componentes

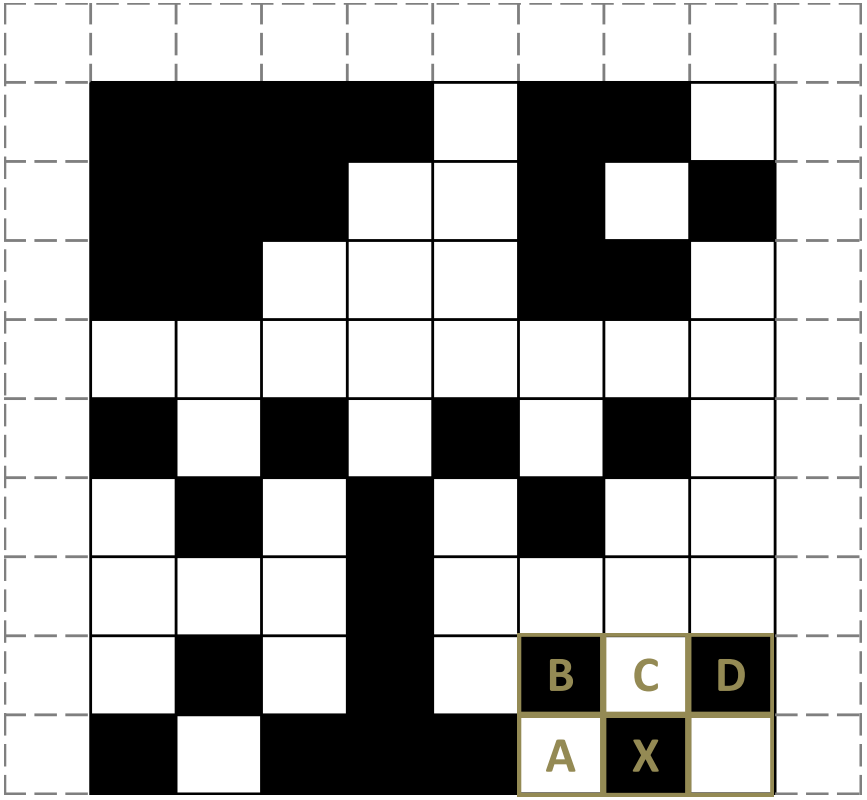


0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		3		4		5	
	2		3		4		

Tabela de Equivalência

2	3
3	4
4	5

Detectando componentes



0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		3		4		5	
	2		3		4		
			3				
	6		3		7		8
6		6	3	3		7	

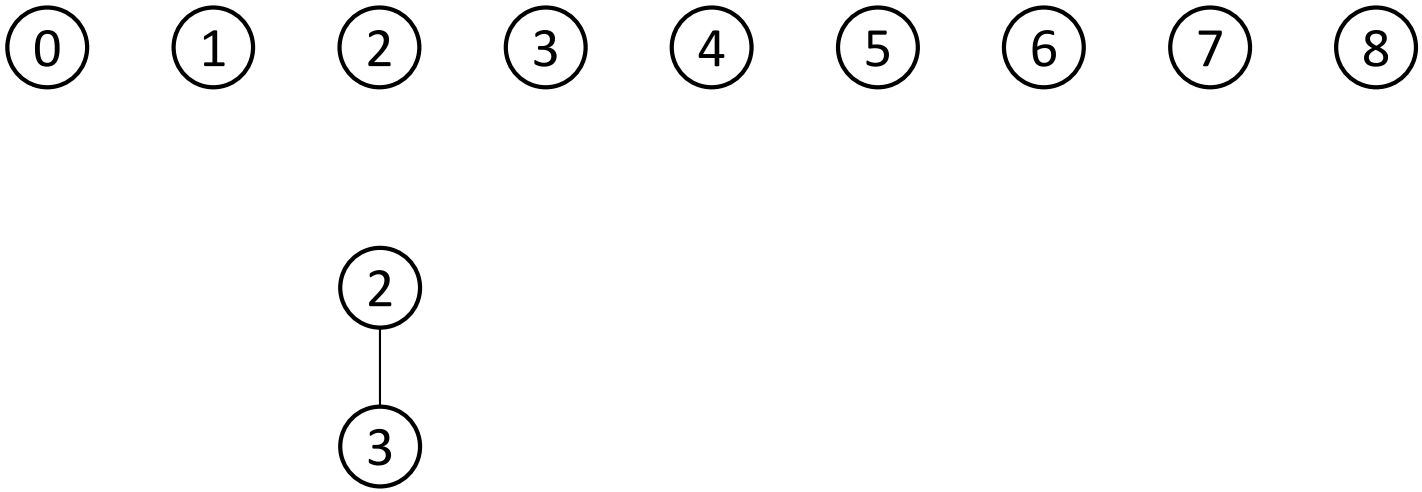
Tabela de Equivalência

2	3
3	4
4	5
6	3
6	3
3	7
7	8

Detectando componentes conectadas

Tabela de Equivalência

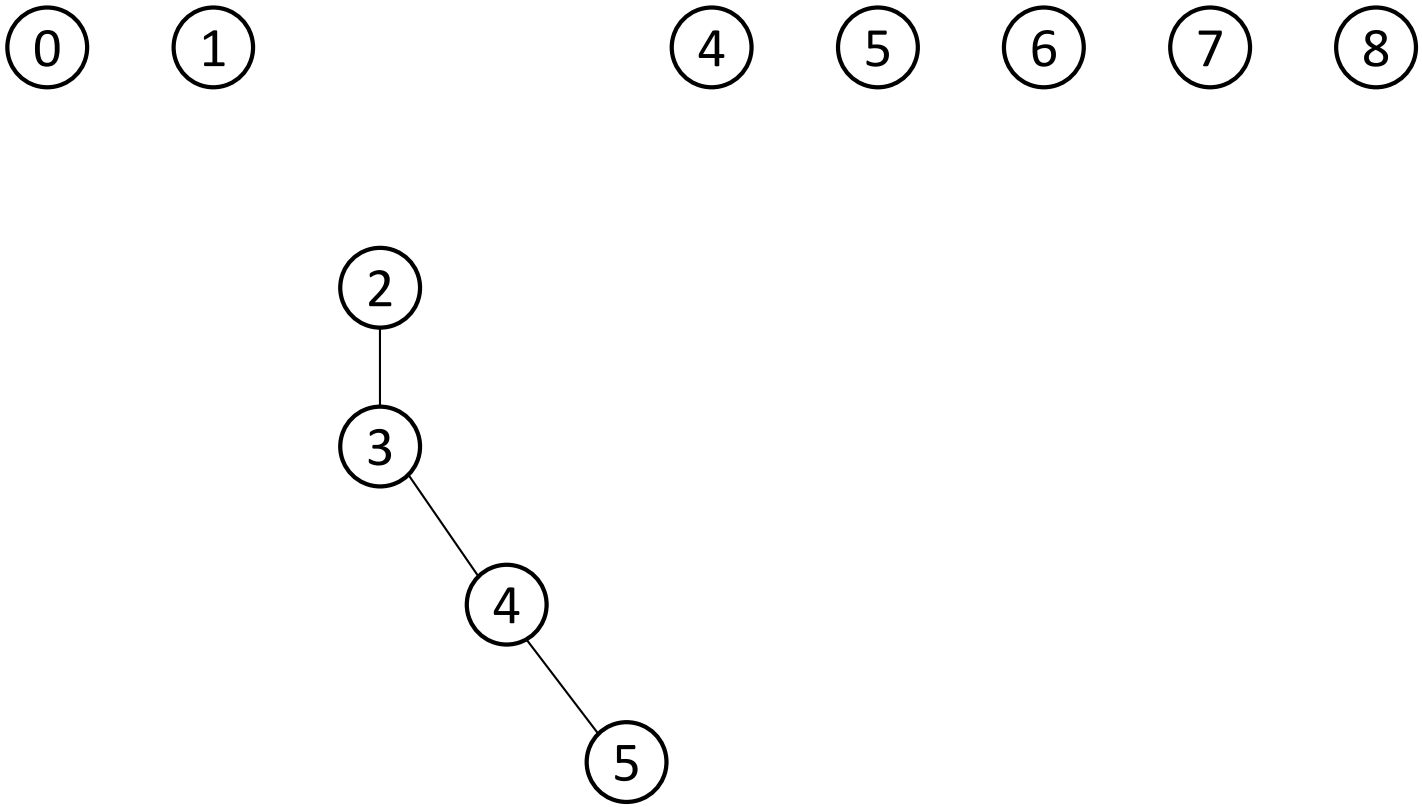
2	3
3	4
4	5
6	3
6	3
3	7
7	8



Detectando componentes conectadas

Tabela de Equivalência

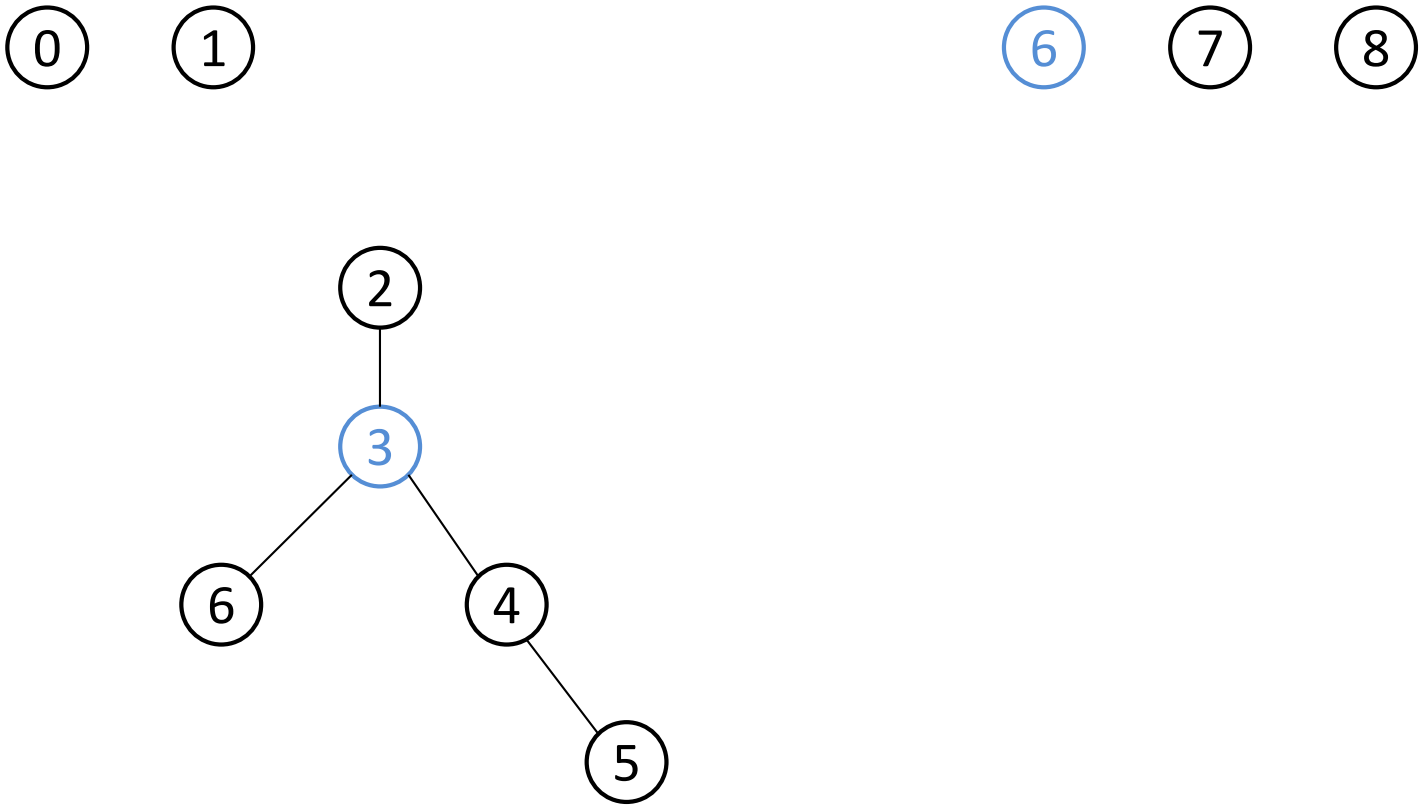
2	3
3	4
4	5
6	3
6	3
3	7
7	8



Detectando componentes conectadas

Tabela de Equivalência

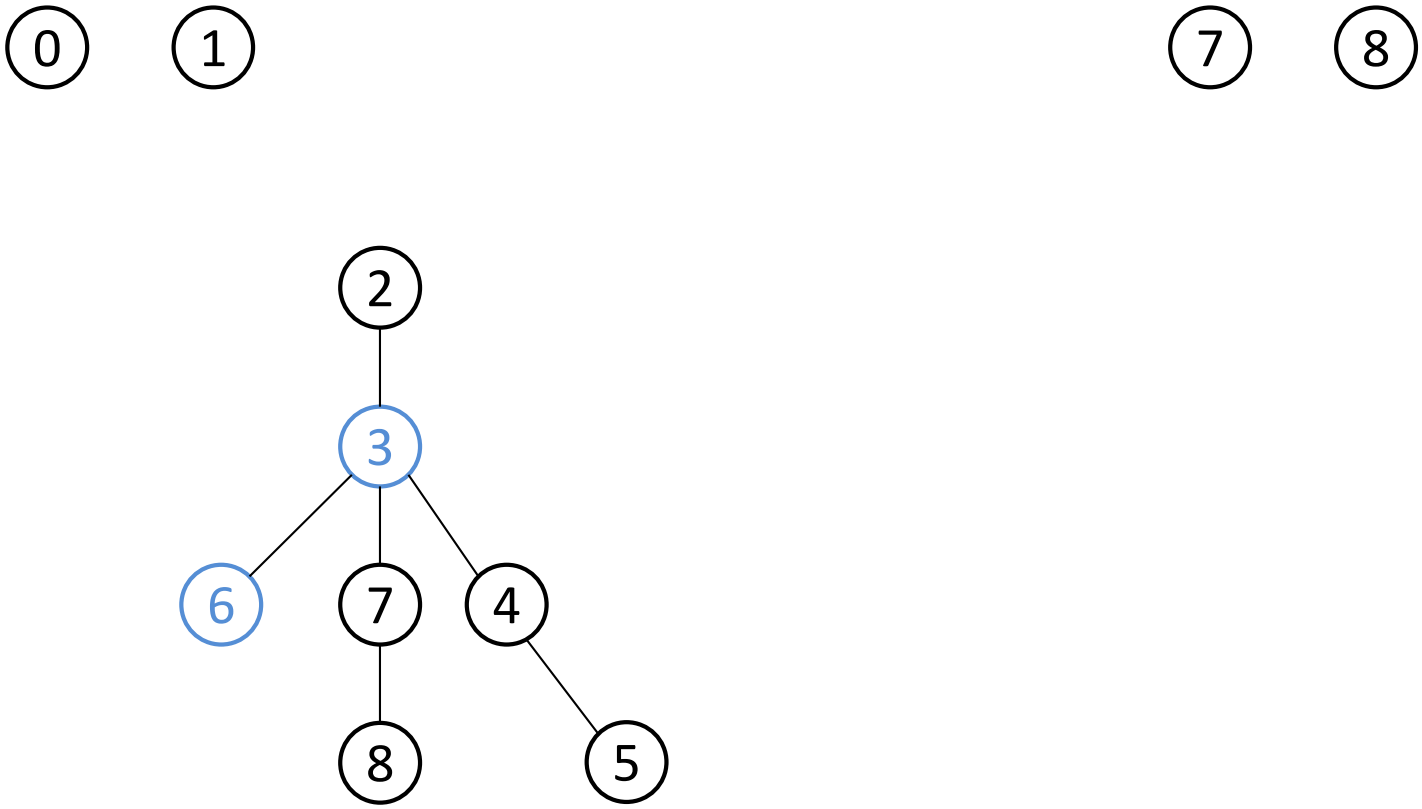
2	3
3	4
4	5
6	3
6	3
3	7
7	8



Detectando componentes conectadas

Tabela de Equivalência

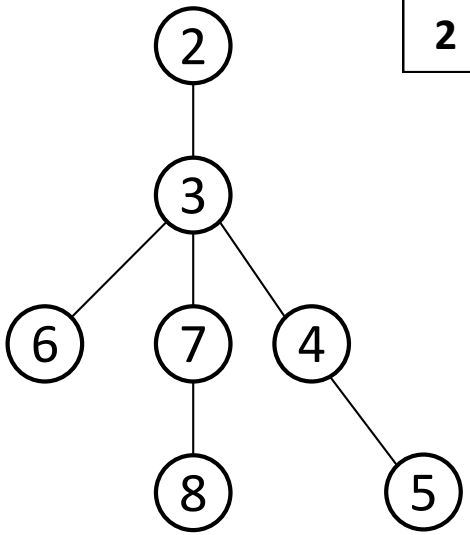
2	3
3	4
4	5
6	3
6	3
3	7
7	8



Detectando componentes

0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		3		4		5	
	2		3		4		
			3				
	6		3		7		8
6		6	3	3		7	

0	0	0	0		1	1	
0	0	0			1		1
0	0				1	1	
2		2		2		2	
	2		2		2		
			2				
	2		2		2		2
2		2	2	2		2	



Partições de Conjuntos Dinâmicas

Estrutura de dados de conjuntos-disjuntos

- Uma estrutura de dados de conjuntos-disjuntos (Disjoint-set data structure), também conhecida como "Union Find" ou ainda "Merge Find", é uma estrutura de dados que manipula um conjunto de elementos disjuntos particionados em subconjuntos.

Definições

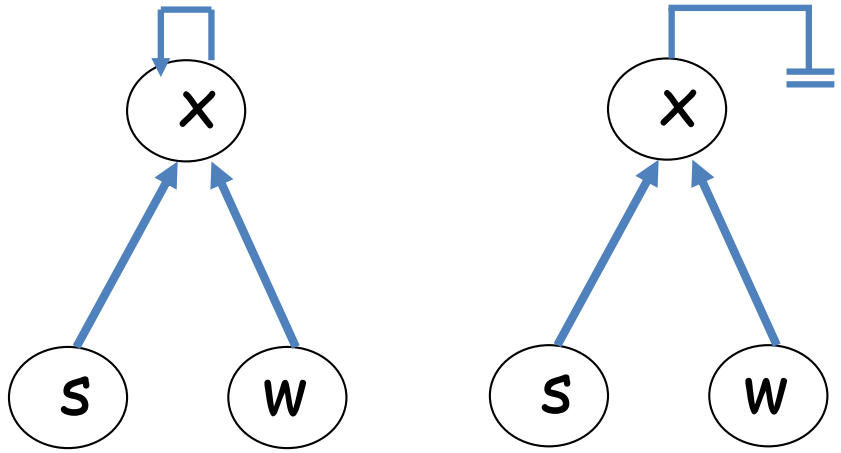
- Universo: $U = \{x_1, x_2, \ldots, x_n\}$
- Coleções: $C = \{S_1, \ldots, S_k\}$ de conjuntos disjuntos
- $S_i \subseteq U$ para qualquer i
- $\cup S_i = U$ (COBERTURA)
- $S_i \cap S_j = \phi$ sempre que $i \neq j$ (DISJUNTOS)
- Cada S_i é identificado por um de seus elementos $x \in S_i$ (**REP**RESENTANTE)

Operações Básicas

- MakeSet(x) : Cria um conjunto com um só elemento. Esta configuração também é conhecida como singleton;
  - $S_i = \{x\}$
- Find(x): Informa de que conjunto um elemento faz parte, para é retornado um elemento que representa o conjunto. Também útil para determinar se dois elementos estão no mesmo conjunto;
  - {k,j,n,o,y,**w**,x,z} := w
- Union(x,y): Junta dois conjuntos em um único. Substituindo assim S_i e S_j em C por um novo conjunto S_k.
  - $S_i \cup S_j = S_k$

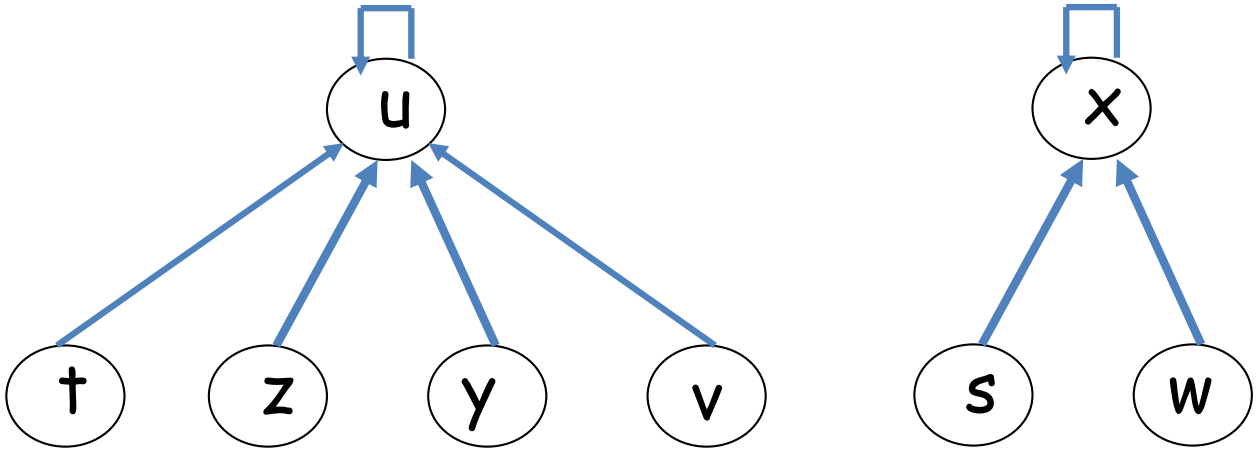
Representação Por Floresta
Árvore Reversa

- Uma forma tradicional de se representar e implementar estas estruturas é usando árvores reversas (Reversed Trees)
- Nesta configuração, cada nó aponta para o seu pai.
- Um nó da estrutura de dados Union-Find é um líder se ele é a raiz da árvore.
- Depois de um nó deixar de ser um líder (ou seja, o nó no topo de uma árvore), ele nunca pode se tornar um líder novamente.
- O apontamento do nó raiz pode ser definido de duas formas principais:
  - a raiz aponta para si mesmo
  - a raiz aponta para NULL



Exemplo de Conjuntos Disjuntos

- Um conjunto é representado pela sua raiz.

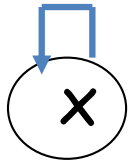


$S_1 = \{t,\underline{u},v,y,z\}$ $S_2 = \{s,w,\underline{x}\}$

Exemplo de Operações: MakeSet(x)

- O MakeSet(x) cria um conjunto com um elemento só (singleton).

MakeSet(x) = {x}

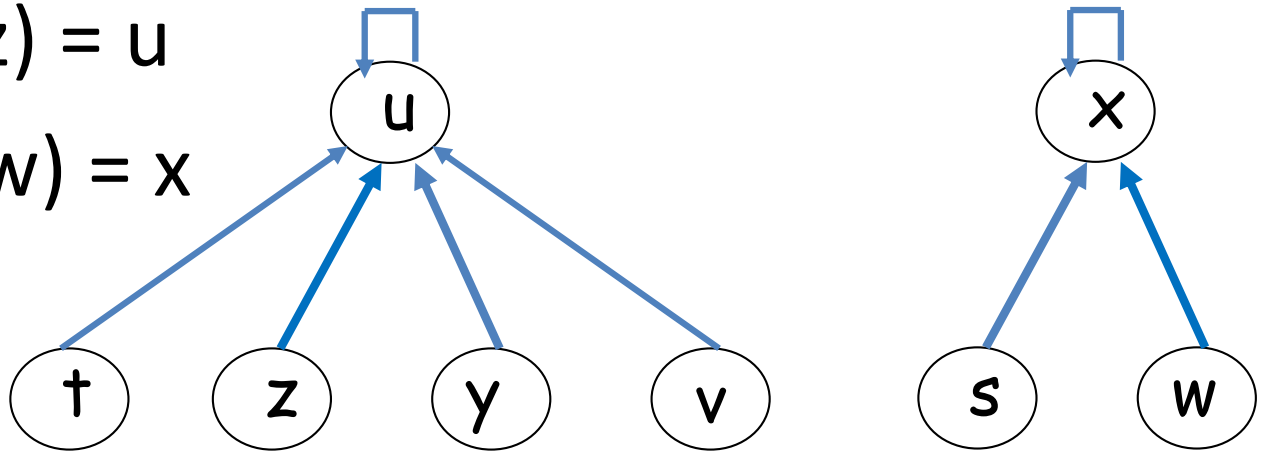


Exemplo de Operações: Find(x)

- O Find(x) retorna o elemento que representa o conjunto naquele instante. Assim o algoritmo busca a raiz da árvore que contém o elemento x.

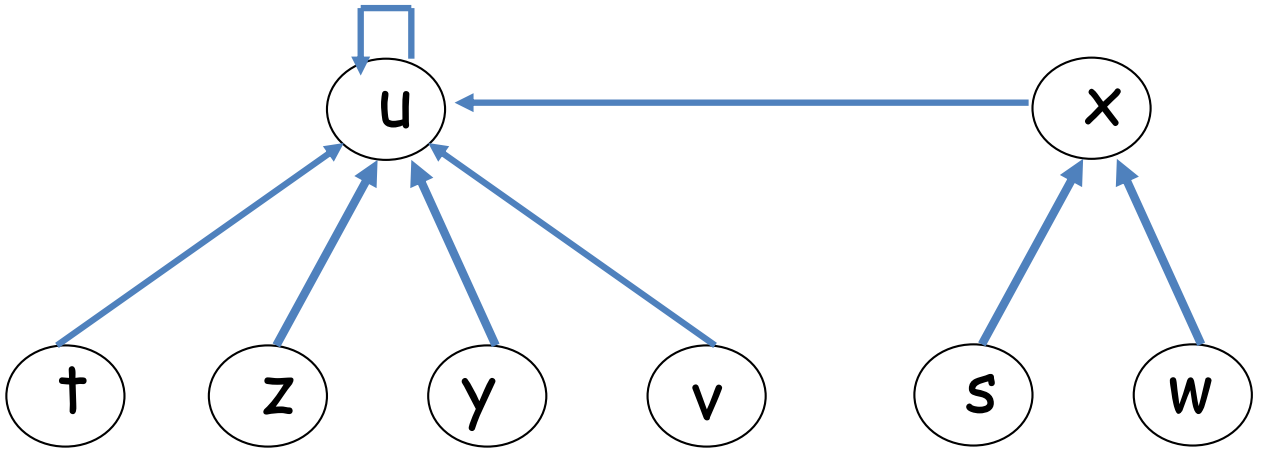
Find(z) = u

Find(w) = x



Exemplo de Operações: Union(u,x)

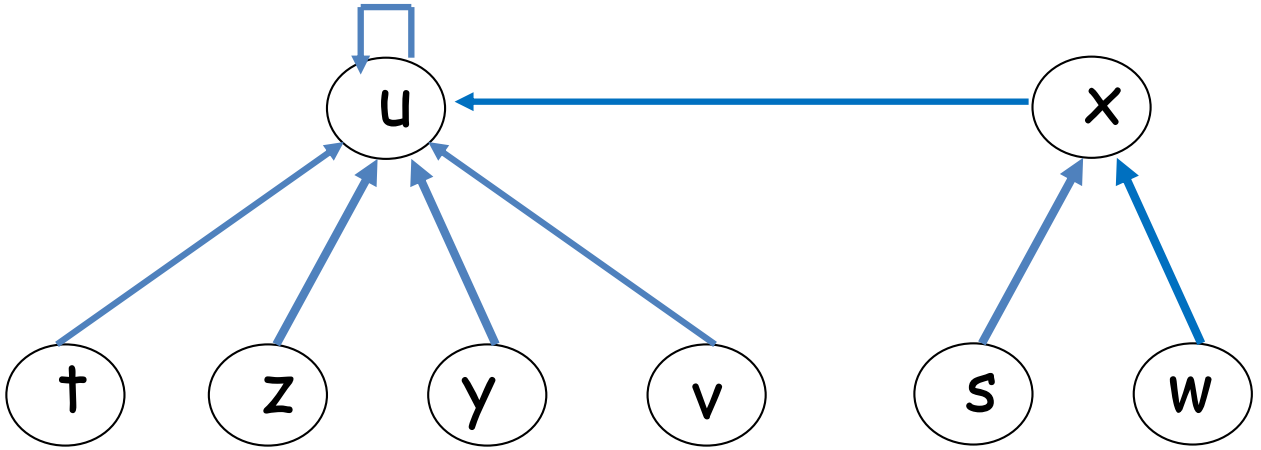
- O Union(u,x) une dois grupos fazendo com que a raiz de um grupo aponte para a raiz do outro grupo. Desta forma uma árvore vira a sub-árvore da outra.
- Union(u,x) => $S_1 \cup S_2$ = {t,**u**,v,y,z, s,w,x}.



Já é possível perceber aqui, que conforme as uniões forem realizada, uma árvore completamente degenerada pode ser criada, fazendo com que as operações de Find se realizem em tempo linear O(n)

Exemplo de Operações: Find(w)

- O comando de Find vai percorrendo toda a estrutura até encontrar o elemento que representa o conjunto.
- Find(w) = u



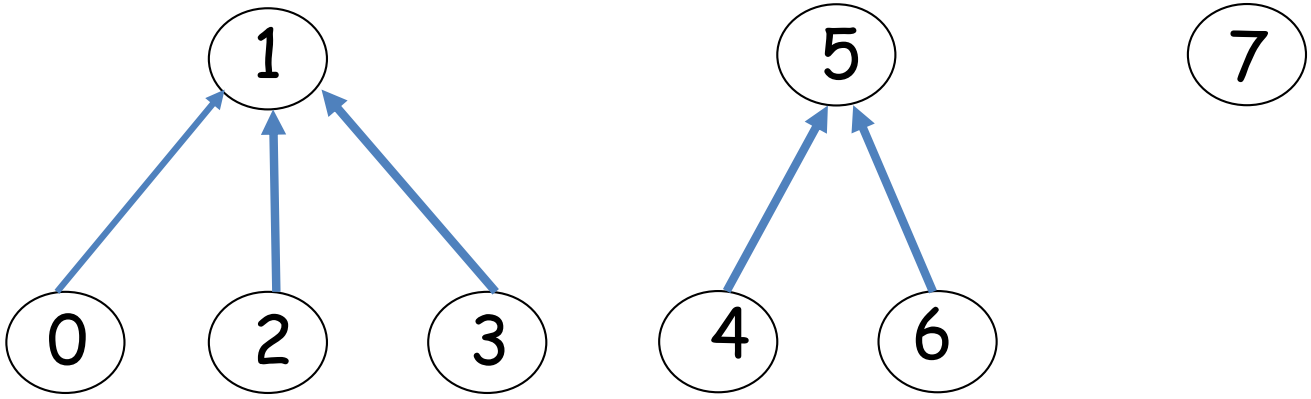
Representação Simplificada por Vetor

- Um exemplo didático de como estrutura de dados de conjuntos-disjuntos pode ser implementada é com um simples vetor contínuo. Cada elemento do vetor é diretamente mapeados para os elementos do universo. Cada elemento do vetor possui um apontador para o pai dele.

Representação Simplificada por Vetor

- Veja aqui um exemplo de conjuntos de alguns números. Note que neste exemplo a representação de raiz é quando o elemento apontando para -1.

elemento	0	1	2	3	4	5	6	7
ponteiro	1	-1	1	1	5	-1	5	-1



Algoritmo Simples de Union-Find

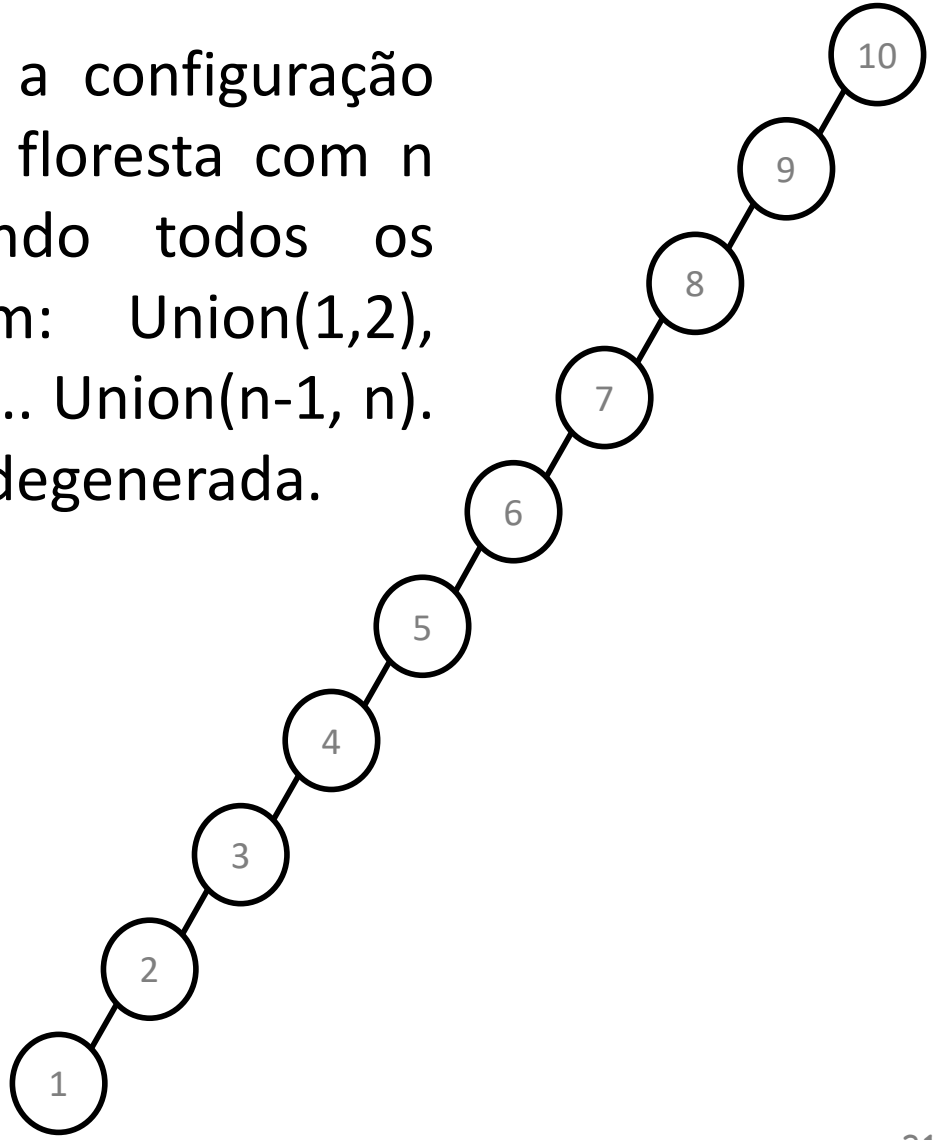
```
void create(int p[], int size) {
        for(int i=0;i<size;++i) p[i] = -1;
}

int find(int u){
        return (  ( p[u] == -1 ) ? u : find(p[u]) );
}

int union(int u, int v) {
        u = find(u);
        v = find(v)
        if (u!=v) p[u] = v;
}
```

Problemas do Algoritmo Simples

- Num primeiro instante a configuração inicial consiste de uma floresta com n nós (singletons). Unindo todos os elementos em ordem: Union(1,2), Union(2,3), Union(3,4), ... Union(n-1, n). Resulta em uma árvore degenerada.



Eficiência

- MakeSet
  - ACESSO DIRETO A NÓ : O(1)
- Find
  - NECESSÁRIO PERCORRER OS ELEMENTOS : O(n)
- Union
  - TEMPO CONSTANTE EM SI : O(1)
  - PORÉM É NECESSÁRIO ENCONTRAR OS REPRESENTANTES CASO NÃO SE TENHA CERTEZA DA ORDERM DE INSERÇÃO : O(n)

Eficiência dos Algoritmos

- Caso seja necessário unir todos os elementos e depois verificar cada um a que conjunto ele pertence, todos os elementos precisaria sofrem um Union e depois um Find.
- O tempo de uma **Union** é constante assim unir todos os elemento leva o tempo O(n). Mas cada execução do **find** segue uma sequência até a raiz do conjunto.
- Como o tempo para cada **find** de um elemento no nível i de uma árvore é O(i), o tempo total necessário para processar os n **find**s é:

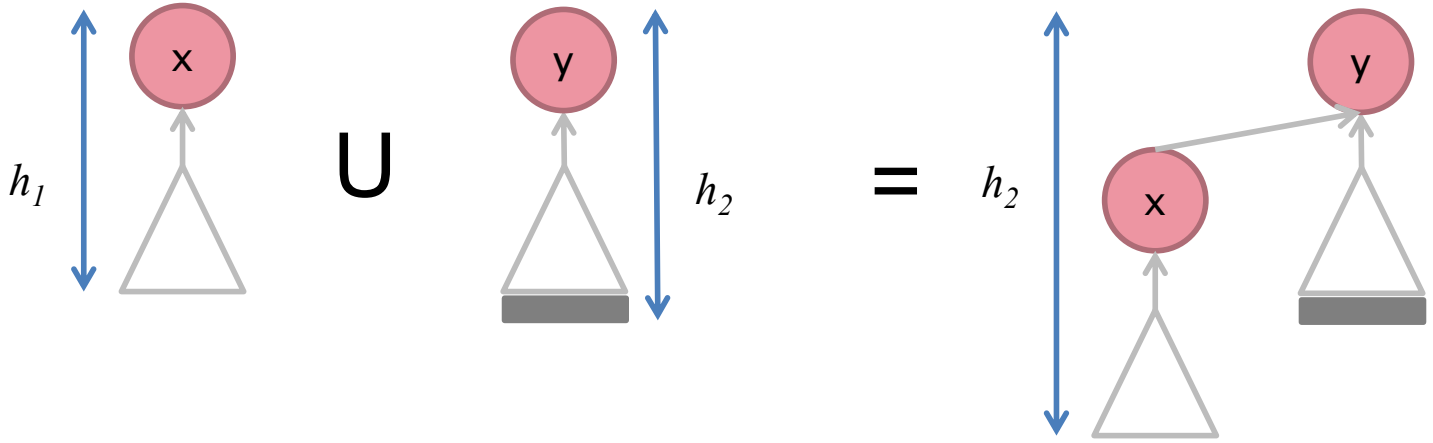
$$O(\sum_{i=1}^{n} i) = O(n^2)$$

Union Seguro

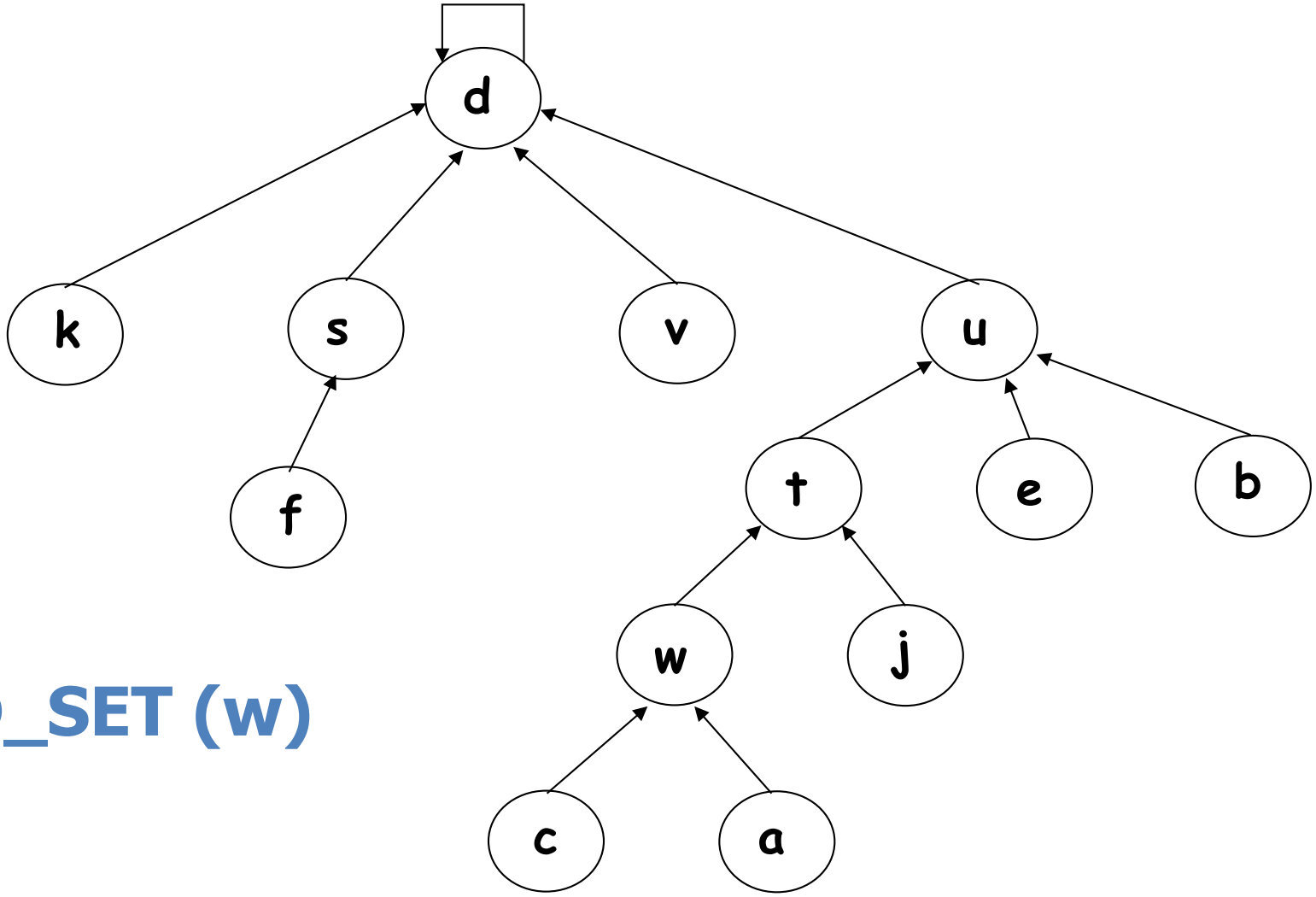
- Para fazer a união de dois grupos é necessário informar a raiz do conjunto, caso isso não seja feito o algoritmo ira ter um comportamente errado.
- Contudo fazer os Finds dentro da rotina de Union vai fazer com que ela se torne O(n).

Algoritmo União por Altura

- No algoritmos de união por altura é verificada a altura de cada conjunto para determinar a união.

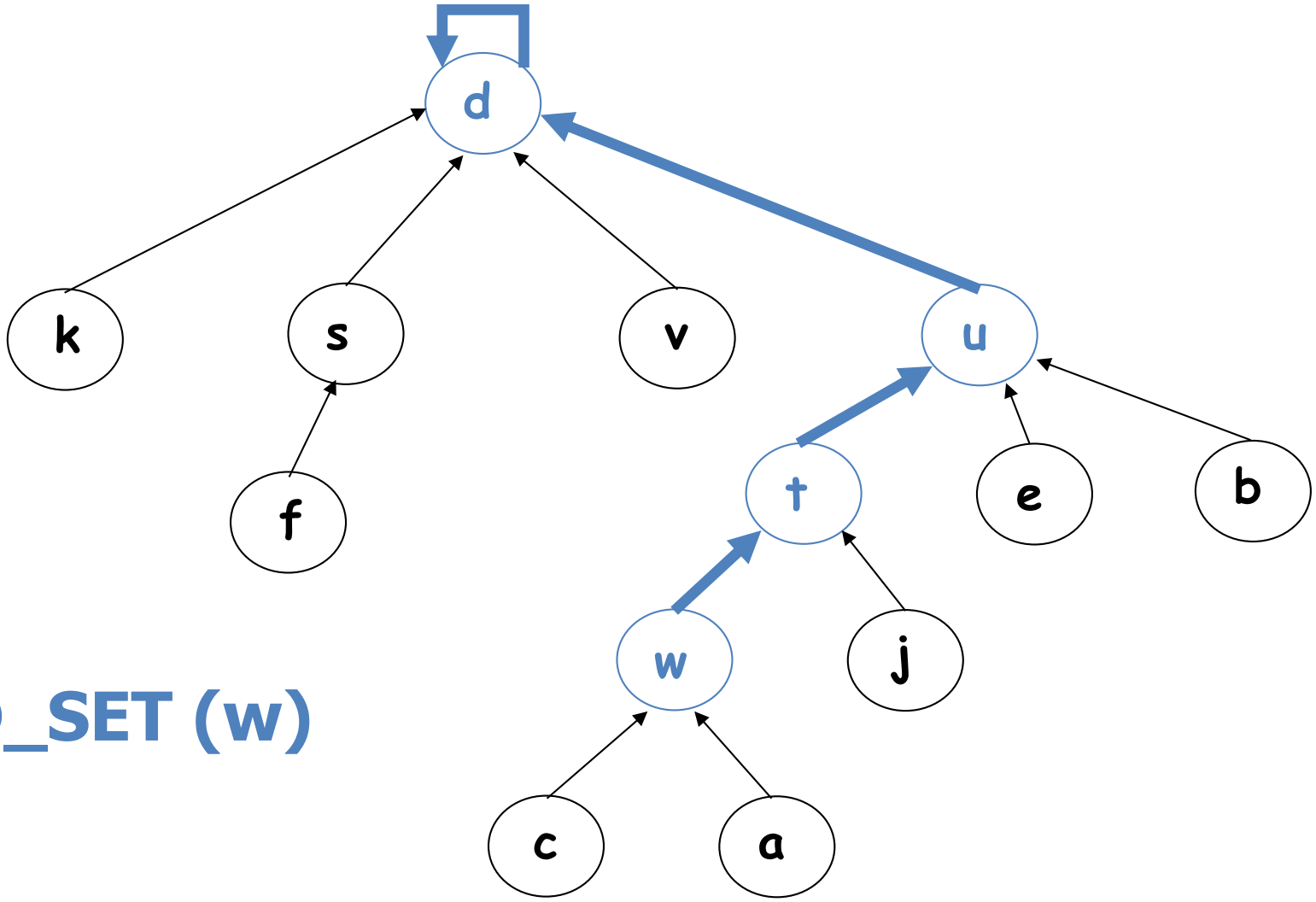


Compressão de Trajetória



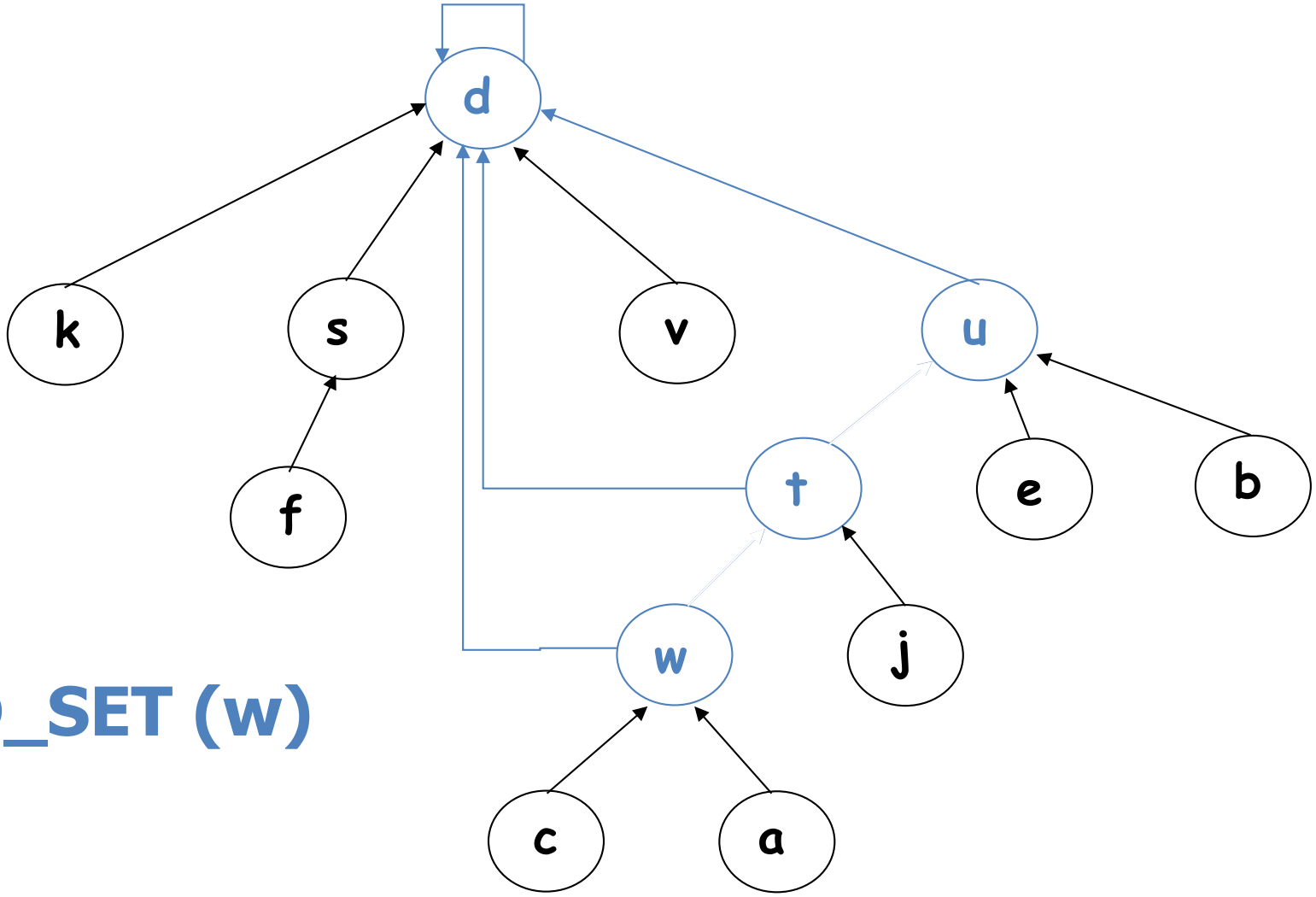
FIND_SET (w)

Compressão de Trajetória



FIND_SET (w)

Compressão de Trajetória



FIND_SET (w)

União Por Tamanho

- Da mesma forma que a união por altura a união por tamanho, minimiza a criação de uniões degeneradas, colocando sempre o menor no maior

Referências

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*. Second edition. The MIT Press, 2001.
- Halim, S, Halim, F. *Competitive Programming*. Lulu, 2010
- Horowitz, E., Sahni, S., Rajasekaran, S. *Computer Algorithms*. Computer Science Press, 1997.
- Weiss, M.K. – *Data Structures and Algorithm Analysis,* Benjamin/Cummins, 1992.
- en.wikipedia.org/wiki/Disjoint-set_data_structure