Desafios de Programação

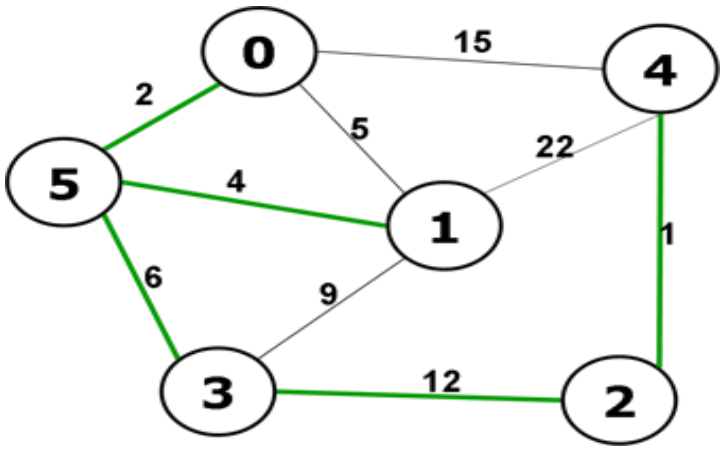
Prof. Eduardo Theodoro

Universidade Federal de Mato Grosso do Sul (UFMS)

Motivação

Problema

Dado um conjunto de computadores, onde cada par de computadores pode ser ligado através de uma quantidade de fibra ótica, encontrar uma rede que interconecte todos os computadores de forma a utilizar a menor quantidade de fibra ótima possível.



O Problema da Árvore Geradora Mínima - The Minimum Spanning Tree

- Este problema se resume em encontrar, em um grafo com pesos nas arestas, um subgrafo gerador conexo de custo mínimo, ou seja, uma árvore geradora de custo mínimo.
- O algoritmo que vamos estudar para resolver este problema é de autoria de Kruskal (1956).

Aplicações

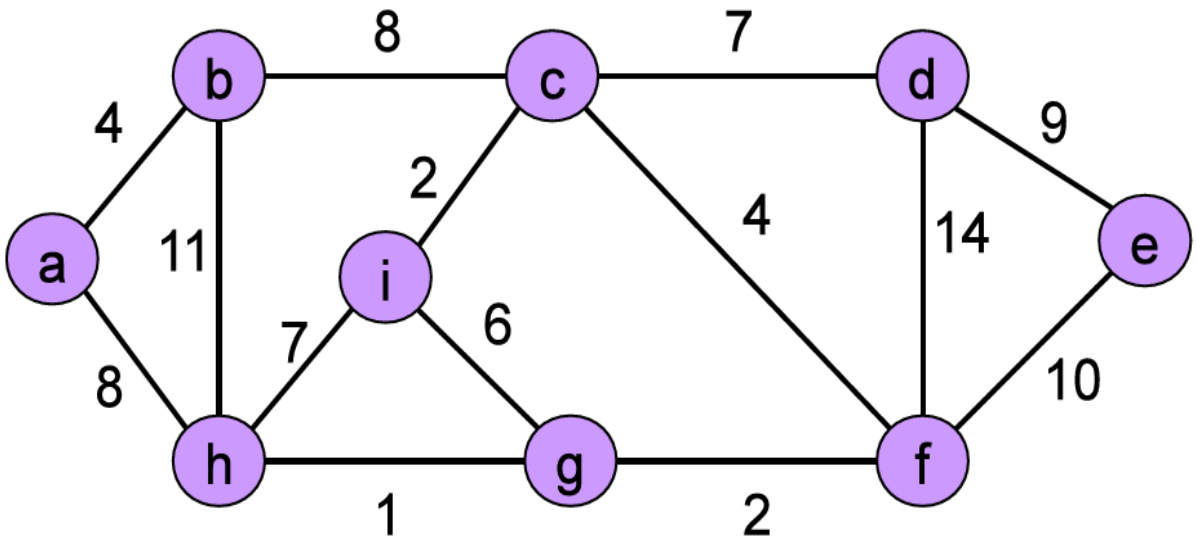
1. Projeto de redes de telecomunicações (redes de computadores, redes de telefonia, etc..)
2. Projeto de rodovias, ferrovias.
3. Projeto de redes de transmissão de energia.

Algoritmo de Kruskal

1. Dado um grafo $G = (V, E)$, considere o grafo $T = (V(G), \emptyset)$;
2. $S \leftarrow E(G)$;
3. Enquanto $|T| < n - 1$ faça
   - "remova uma aresta e de peso mínimo de S";
   - Se e liga duas componentes distintas de T então adicione e à T; Senão descarte e;

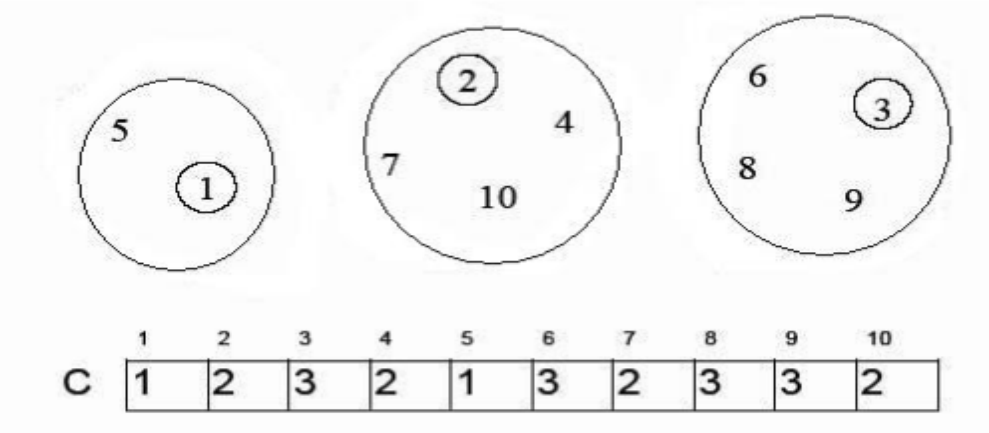
Exercício

Encontre a árvore geradora mínima.



Estrutura de Dados - Conjuntos Disjuntos (Union-Find)

- Uma estrutura de dados conjuntos-disjuntos é uma coleção $S = \{S_1, \ldots, S_k\}$ de conjuntos dinâmicos disjuntos.
- Cada conjunto S_k é identificado por um representante, que é um membro do conjunto.



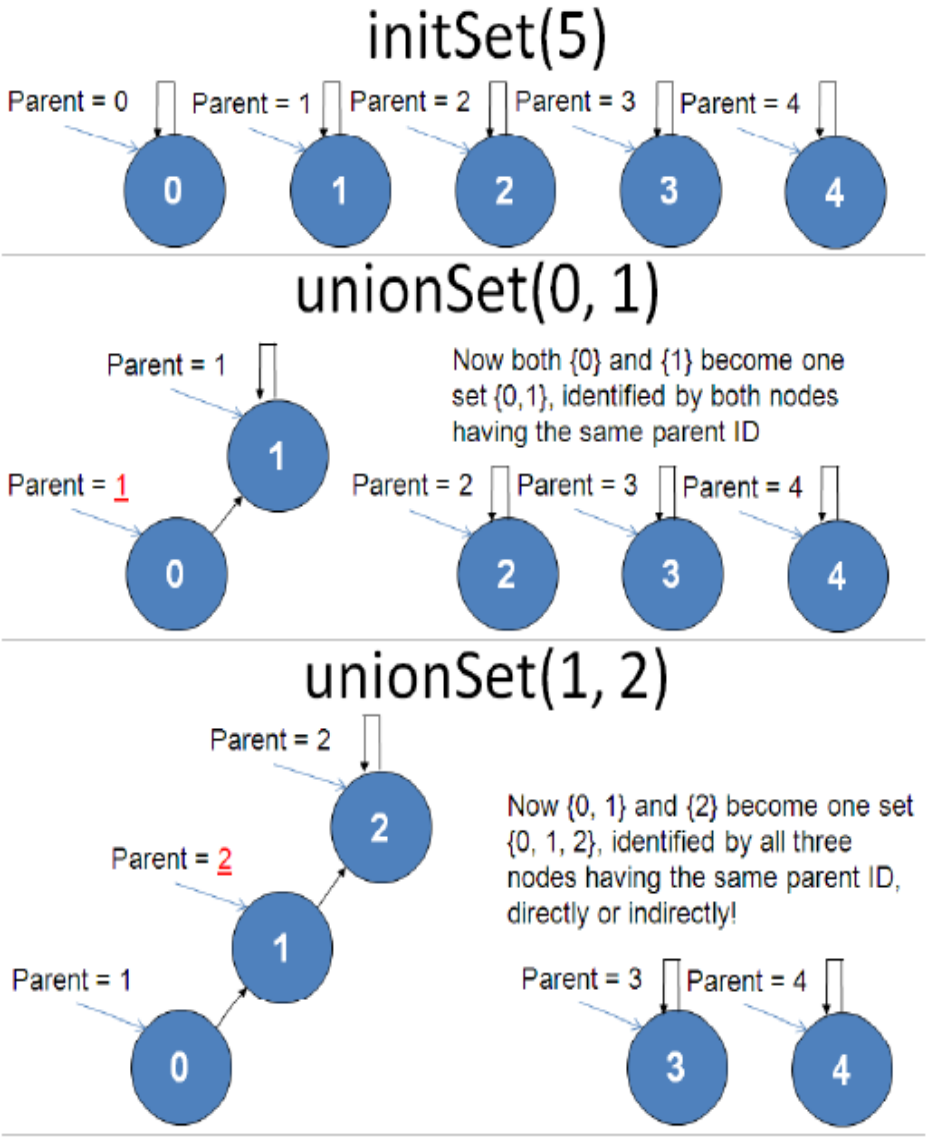
Estrutura de Dados - Conjuntos Disjuntos (Union-Find)

- A ideia principal então da estrutura é manter um representante para cada conjunto!
- Esta informação é armazenada em um vetor de inteiros $pset[i]$, que diz qual é o representante do conjunto que contém i.
- Inicialmente, $pset[i] = i$. **(initSet(n))**

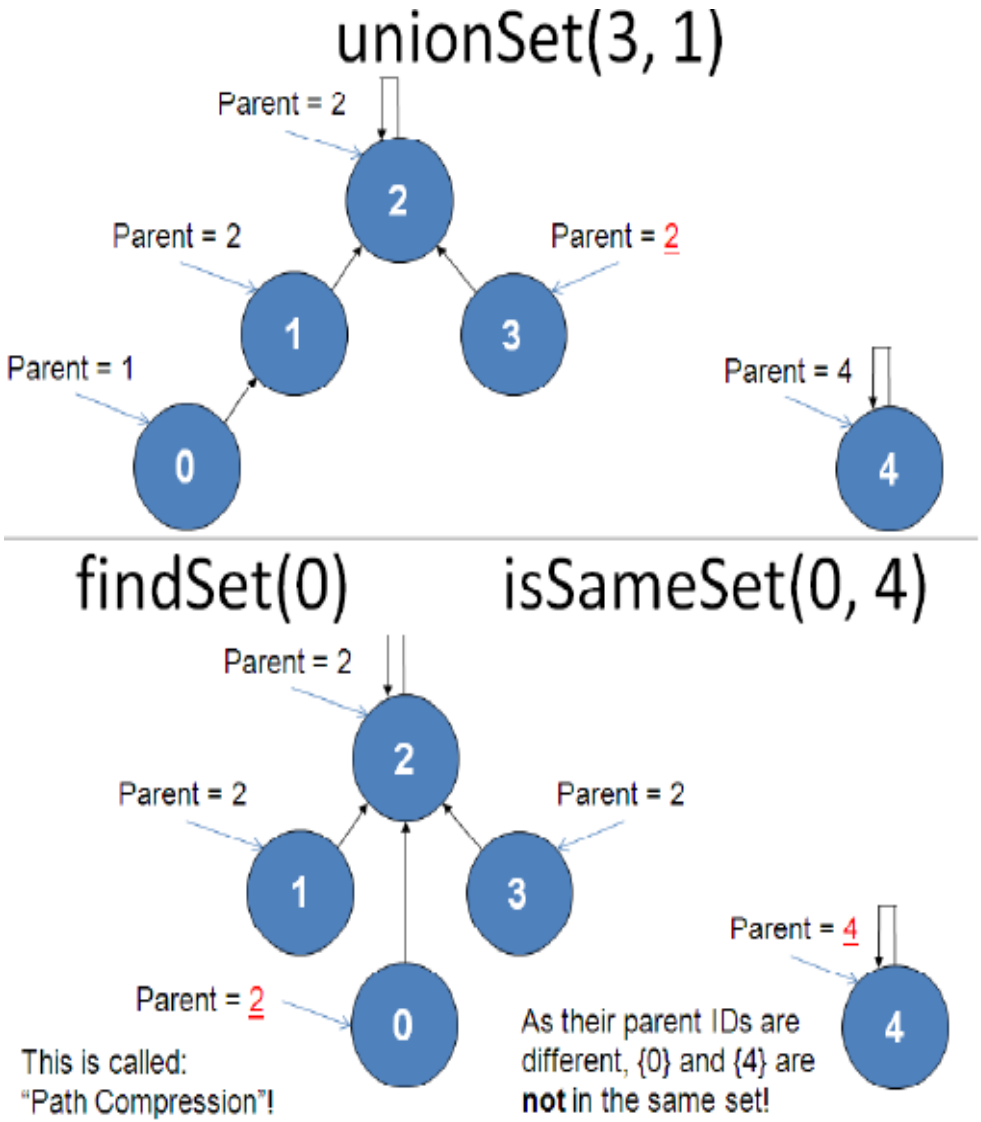
Estrutura de Dados - Conjuntos Disjuntos (Union-Find)

- Se desejamos unir dois conjuntos i e j, chamamos a função $unionSet(i, j)$ que faz ambos os elementos i e j terem o mesmo representante -direto ou indiretamente.
- Isto é feito através da chamada de $findSet(j)$, que procura o representante do elemento j e atribui seu valor a $pset[findSet(i)]$, ou seja, $pset[findSet(i)] = findSet(j)$.
- A função $findSet(i)$ simplesmente busca recursivamente enquanto $pset[i]$ não for igual a i. Então, quando ele encontra o representante do conjunto, digamos x, ele comprime o caminho fazendo com que $pset[i] = x$. Logo, chamadas sucessivas de $findSet(i)$ serão feitas em $O(1)$. Esta estratégia é chamada de **Path Compression**.

Exemplo


initSet(5)
Parent = 0
Parent = 1
Parent = 2
Parent = 3
Parent = 4
0
1
2
3
4
unionSet(0, 1)
Parent = 1
Parent = 1
0
1
Now both {0} and {1} become one set {0,1}, identified by both nodes having the same parent ID
Parent = 2
Parent = 3
Parent = 4
2
3
4
unionSet(1, 2)
Parent = 2
Parent = 2
Parent = 1
0
1
2
Now {0, 1} and {2} become one set {0, 1, 2}, identified by all three nodes having the same parent ID, directly or indirectly!
Parent = 3
Parent = 4
3
4

Exemplo


unionSet(3, 1)
Parent = 2
Parent = 2
Parent = 2
Parent = 1
Parent = 4
2
1
3
0
4
findSet(0)
isSameSet(0, 4)
Parent = 2
Parent = 2
Parent = 2
Parent = 2
Parent = 4
2
1
3
0
4
This is called:
"Path Compression"!
As their parent IDs are different, {0} and {4} are **not** in the same set!

Código

```
void initSet(vector<int>& pset){
  fore(i, pset.size()) pset[i] = i;
}

int findSet(vector<int>& pset, int i){
  if(pset[i]==i) return i;
  pset[i] = findSet(pset, pset[i]);
  return pset[i];

}

void unionSet(vector<int>& pset, int i, int j){
  pset[findSet(pset, i)] = findSet(pset, j);
}
```