Grafos - Árvore Espalhada Mínima

Tópicos Especiais em Algoritmos - Ciência da Computação

INSTITUTO FEDERAL
Brasília

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília, Campus Taguatinga

Sumário

Sumário

Motivação

- Suponha que tenhamos uma infraestrutura de rede montada.
- Várias máquinas estão conectadas à outras através de diversos roteadores.
- Ao mesmo tempo, a economia de energia se tornou uma situação crítica nos dias de hoje.
- Como você faria para continuar permitindo a comunicação de quaisquer computadores com menor custo possível?
- Quais roteadores você desativaria?
- Qual a estrutura obtida?

Motivação

- O problema da árvore espalhada mínima visa resolver este tipo de problemas.
- Queremos um subgrafo acíclico e conexo de menor custo (árvore de menor custo).
- Existem algoritmos bem conhecidos para resolução deste problema, tais como:
  - Algoritmo de Prim.
  - Algoritmo de Kruskal.
- No entanto, vamos examinar algumas definições antes de atacar o problema.

Árvore Espalhada Mínima

Custo de uma Árvore

- O custo de uma árvore é dado pelo somatório do custo de suas arestas:

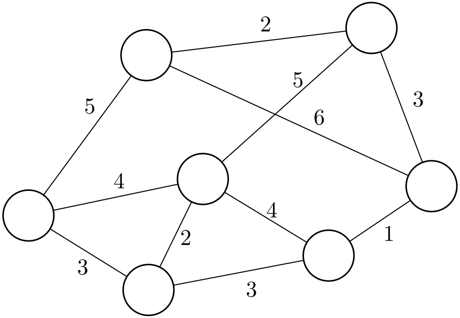
$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

Árvore Espalhada Mínima

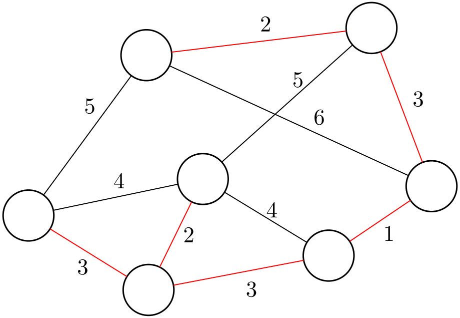
Árvore Espalhada Mínima

- Entrada: um grafo $G = (V, E)$.
- Saída: uma árvore de G com custo mínimo.

Árvore Espalhada Mínima



Árvore Espalhada Mínima



Árvore Espalhada Mínima

Menor custo

- Qual a estratégia para chegar no menor custo possível?

Árvore Espalhada Mínima

Menor custo

- Qual a estratégia para chegar no menor custo possível?
- Inicialmente, todo vértice é uma componente conexa.

Árvore Espalhada Mínima

Menor custo

- Qual a estratégia para chegar no menor custo possível?
- Inicialmente, todo vértice é uma componente conexa.
- Adicione arestas de menor custo possível de modo que conecte componentes conexas.

Árvore Espalhada Mínima

Menor custo

- Qual a estratégia para chegar no menor custo possível?
- Inicialmente, todo vértice é uma componente conexa.
- Adicione arestas de menor custo possível de modo que conecte componentes conexas.
- Jamais adicione uma aresta de custo maior que conecte as mesmas componentes.

Árvore Espalhada Mínima

Menor custo

- Qual a estratégia para chegar no menor custo possível?
- Inicialmente, todo vértice é uma componente conexa.
- Adicione arestas de menor custo possível de modo que conecte componentes conexas.
- Jamais adicione uma aresta de custo maior que conecte as mesmas componentes.
- Jamais forme um ciclo!

Árvore Espalhada Mínima

Algorithm 1: GENERIC-MST(G)

```
T ← ∅
while T não for uma árvore do
    Encontre uma aresta (u, v) que é segura para T
    Adicione a aresta (u, v) à T
```

Àrvore Espalhada Mínima

- Como escolher uma aresta segura?
- Veremos duas abordagens básicas.

Sumário

Algoritmo de Prim

- O algoritmo de Prim de certa forma se parece muito com o algoritmo de Dijkstra.
- Começamos de um nó arbitrário como o único nó de nossa árvore.
- Escolhemos sempre as arestas de menor custo para adicionarmos à árvore a partir dos nós previamente inseridos na árvore.
- Da mesma forma que no algoritmo de Dijkstra, precisamos de uma estrutura de dados eficiente.
- Ao contrário do algoritmo de Dijkstra, o algoritmo de Prim não tem problemas com arestas de custo negativo.

Algoritmo de Prim

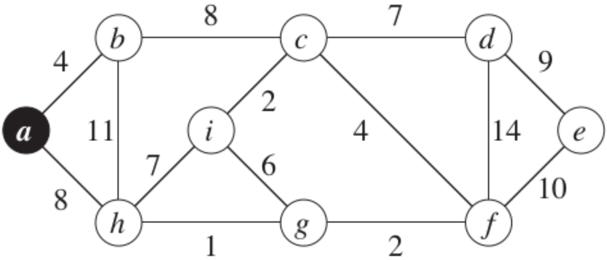


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

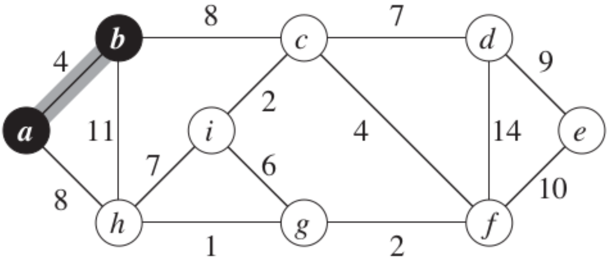


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

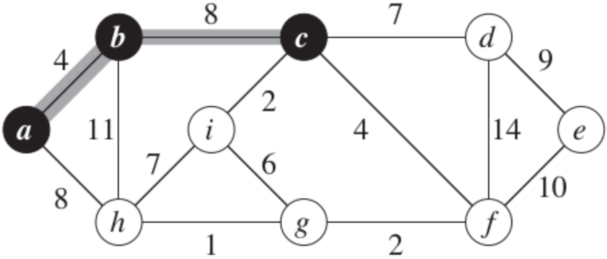


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

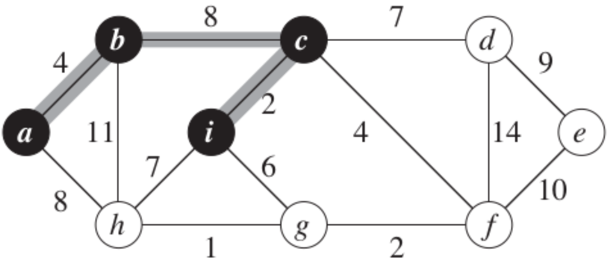


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

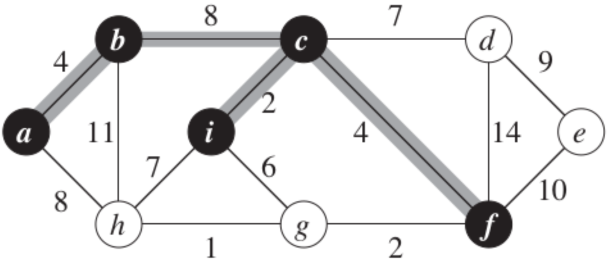


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

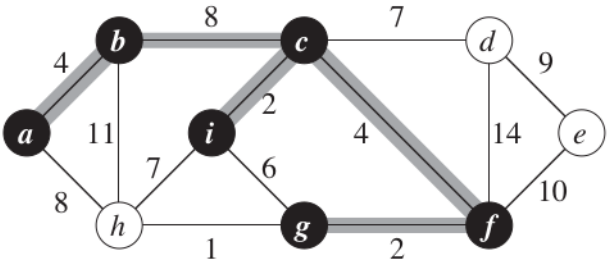


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

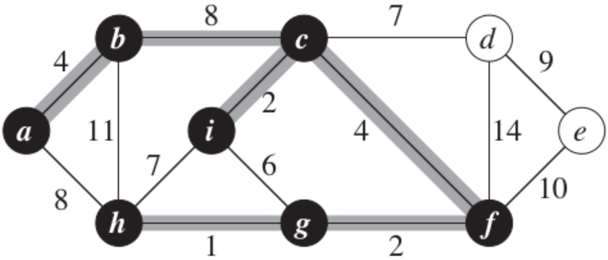


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

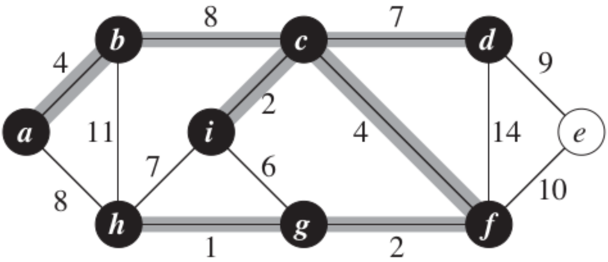


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

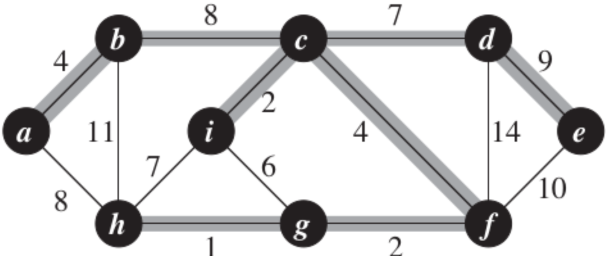


Figura: Execução do Algoritmo de Prim

Algoritmo de Prim

Algorithm 2: INITIALIZE-PRIM(G, s)

Input: G, s

```
for all( v ∈ V )
    v.d ← ∞
    v.π ← NULL
    v.color ← white
s.d ← 0
```

Algoritmo de Prim

Algorithm 3: $\textsc{Prim}(G, s, w)$

Input: G, s, w

Output: T, á árvore espalha mínima de G que contém o vértice s

$\textsc{initialize-prim}(G, s)$
$Q.\textsc{insert-update}(s)$
$T \leftarrow \emptyset$
$last \leftarrow s$
while $\neg Q.\textsc{empty}()$ **do**
  $u \leftarrow Q.\textsc{extract-min}()$
  $u.color \leftarrow$ **black**
  if($last \neq s$) $T.\textsc{append}((last, u))$
  $last \leftarrow u$
  for all($(u, v) \in E \wedge v.color =$ **white**)
    if($v.d > w(u, v)$)
      $v.d \leftarrow w(u, v)$
      $v.\pi \leftarrow u$
      $Q.\textsc{insert-update}(v)$
return T

Algoritmo de Djikstra

Complexidade

- Usando vetores: $\Theta(|V|^2 + |E|)$.
- Usando heap: $\Theta(|V| \log |V| + |E| \log |V|)$.
- Usando heap de fibonacci: $\Theta(|V| \log |V| + |E|)$.
- O que você vai usar em grafos densos? E em grafos esparsos?

Sumário

Algoritmo de Kruskal

- O algoritmo de Kruskal tem uma filosofia bem simples.
- Ordena-se as arestas em ordem crescente de custo.
- Para cada aresta (u, v) em ordem crescente, acrescente a aresta (u, v) na árvore espalhada mínima somente se u e v não estão na mesma componente conexa.
- Isso evita a formação de ciclos.
- Seja t_s o custo da ordenação, t_e o custo de varrer as arestas e t_{uf} o custo de verificar se dois vértices estão em uma mesma componente conexa.
- O custo total do algoritmo é $\Theta(t_s + t_e \cdot t_{uf})$.

Algoritmo de Kruskal

- Assumindo que:
  - O tempo de ordenação leve $t_s \in \Theta(|E| \lg |E|) = \Theta(|E| \lg |V|)$.
- O custo total do algoritmo é $\Theta(|E| \lg |V| + |E| \cdot t_{uf})$
- Caso uma estrutura eficiente que possibilite checar se dois vértices estejam na mesma componente conexa seja utilizada, conseguimos um algoritmo $\Theta(|E| \lg |V|)$.

Sumário

3. Kruskal
   - Union-find
   - Algoritmo de Kruskal

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Suponha $\mathcal{S} = (\{s_0\}, \{s_1\}, \ldots, \{s_n\})$ conjuntos *singleton*.
- Deseja-se implementar duas operações básicas:
  - $\textsc{union}(A, B)$, une dois conjuntos A e B.
  - $\textsc{find}(x)$, diz em qual conjunto encontra-se o elemento x.
- É interessante observar que, após um número qualquer de sucessivas aplicações de $\textsc{union}$, o que se tem são conjuntos disjuntos, isto é, conjuntos cuja interseção é vazia.

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Para simplificar, para cada conjunto, vamos eleger um representante.
- Assim, dois elementos estão na mesma coleção se e somente se eles possuem o mesmo representante.
- Inicialmente, cada singleton tem como representante ele mesmo.
- Desta forma, a operação de $\textsc{Find}(x)$ pode simplesmente retornar o representante do conjunto no qual x está incluído.
- Assim, dados dois elementos, x e y, conseguimos saber se eles estão no mesmo conjunto verificando se $\textsc{Find}(x) = \textsc{Find}(y)$.

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Podemos modelar isso computacionalmente através de uma floresta.
- Cada *singleton* x inicialmente faz parte de uma árvore e possui dois valores:
  - $x.parent$: diz quem é o pai de x (inicialmente é ele mesmo).
  - $x.rank$ diz o rank de x, que é uma cota superior da altura da árvore com raiz em x.
- Para achar o representante do conjunto que x está incluso, basta seguir os ponteiros de $parent$ até chegar na raiz.
- Logo, já sabemos implementar FIND.

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

Algorithm 4: $\textsc{Find}(x)$

Input: x

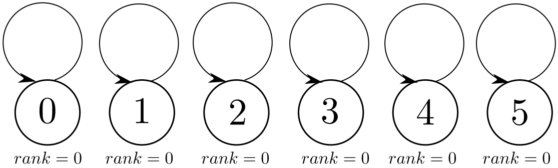
Output: representante do conjunto de x

if($x.parent \neq x$)
  return $\textsc{Find}(x.parent)$

return x

Union-find

Estruturas de Dados Para Conjuntos Disjuntos



Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Para implementar a operação de $\textsc{union}(x, y)$, achamos o representante dos conjuntos que x e y se encontram, chame-os de x' e y'. Se $x' \neq y'$, significa que x e y estão em conjuntos diferentes. Logo só é necessário unir as duas árvores fazendo com que y' vire filho de x' ou x' vire filho de y'.
- Qual opção escolher?
- Escolhemos a árvore de maior $rank$ para abrigar a de menor $rank$.

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

Algorithm 5: UNION(x, y)

Input: x, y

```
x' ← FIND(x)
y' ← FIND(y)
if( x' ≠ y' )
    if( x'.rank > y'.rank )
        y'.parent ← x'
    else
        x'.parent ← y'
        if( x'.rank = y'.rank )
            y'.rank++
```

Estruturas de Dados para Conjuntos Disjuntos

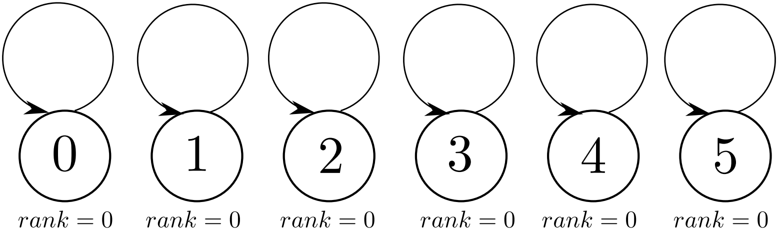


Figura: Estado inicial.

Estruturas de Dados para Conjuntos Disjuntos

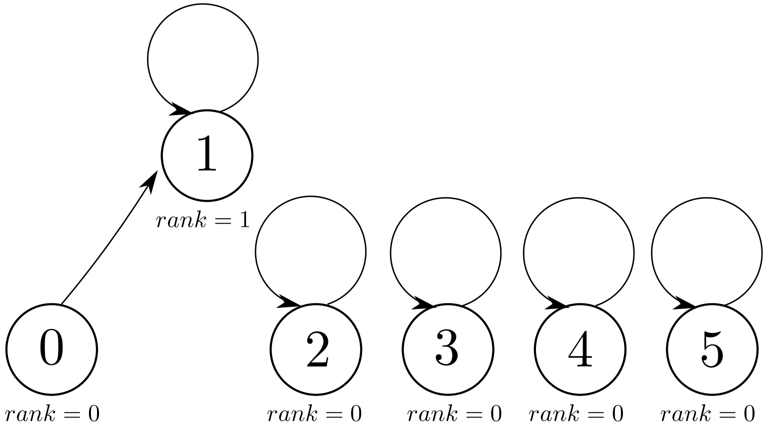


Figura: $\textsc{Union}(0, 1)$

Estruturas de Dados para Conjuntos Disjuntos

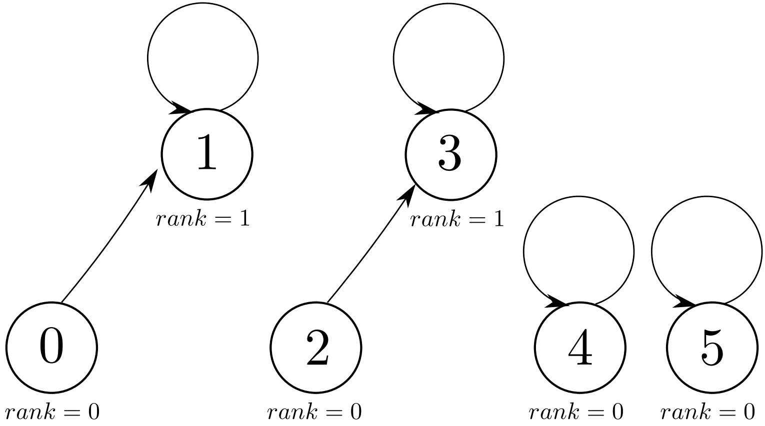


Figura: $\textsc{union}(2, 3)$

Estruturas de Dados para Conjuntos Disjuntos

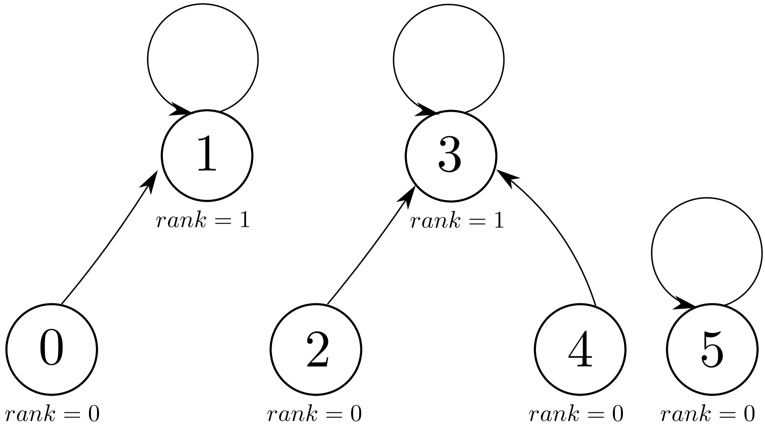


Figura: $\text{UNION}(2, 4)$

Estruturas de Dados para Conjuntos Disjuntos

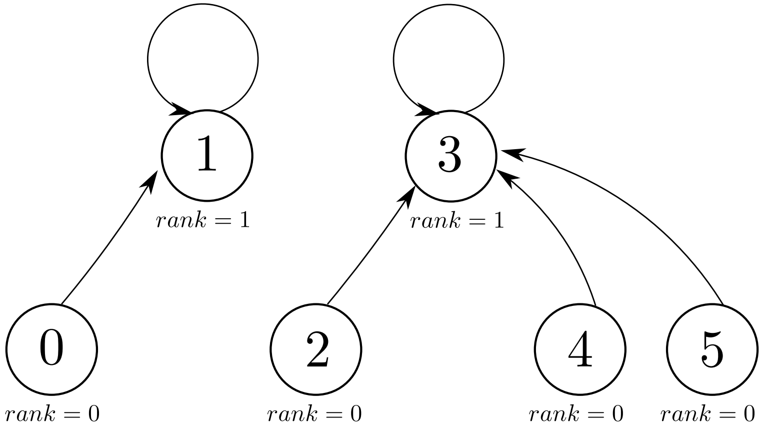


Figura: $\textsc{union}(2, 5)$

Estruturas de Dados para Conjuntos Disjuntos

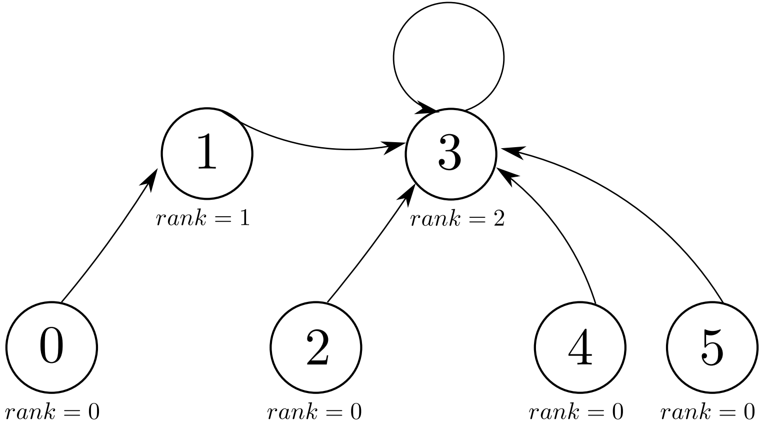


Figura: $\textsc{union}(0, 3)$

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Da forma como estão implementados UNION e FIND, a árvore final pode ter altura $\Theta(n)$, o que ocasionaria um tempo linear para as consultas.
- Contudo é possível melhorar esse tempo significativamente ao utilizar uma técnica chamada de **path-compression**.
- Ao realizar a consulta de $\text{FIND}(x)$, atualizamos x e os demais nós no caminho até a raiz para apontarem imediatamente para o representante do conjunto, isto é, a raiz.

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Dadas M operações de UNION ou FIND complexidade final amortizada utilizando a técnica de **path-compression** é:

$$\Theta(M\alpha(n)) \subsetneq \Theta(M \lg^* n) \subsetneq \Theta(M \lg n)$$

em que α é a função de Ackermann inversa.

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

Algorithm 6: $\textsc{Find}(x)$ com path-compression

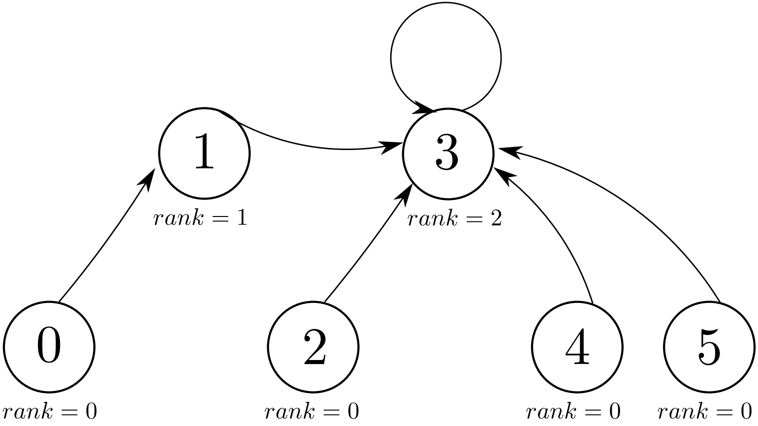
Input: x

Output: representante do conjunto de x

if($x.parent \neq x$)
    $x.parent \leftarrow \textsc{Find}(x.parent)$

return x

Estruturas de Dados para Conjuntos Disjuntos



Estruturas de Dados para Conjuntos Disjuntos

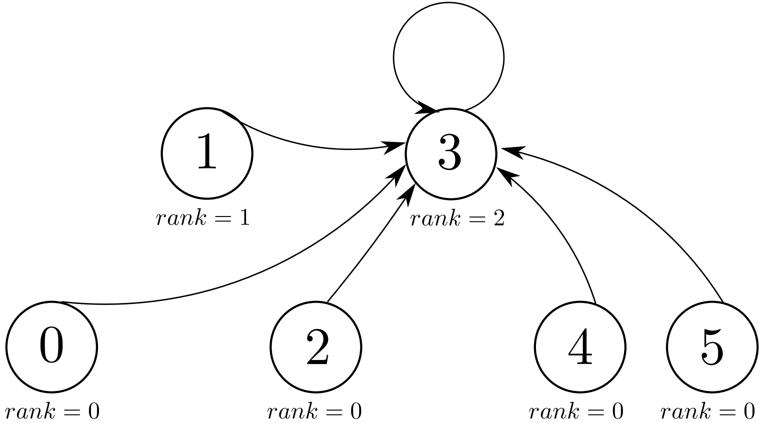


Figura: $\textsc{find}(0)$

Union-find

Estruturas de Dados Para Conjuntos Disjuntos

- Todas as operações que sucedem uma compressão de caminho se beneficiarão do tamanho reduzido da altura.
- Note que, após a compressão de caminho, o $rank$ da raiz não corresponde mais a altura da árvore, então $rank$ deve ser sempre interpretado como uma cota superior para a altura.

Sumário

3. Kruskal
   - Union-find
   - Algoritmo de Kruskal

Algoritmo de Kruskal

- Suponha agora a existência de uma estrutura de dados UF que provê as operações de $\textsc{union}(x, y)$, $\textsc{find}(x)$ como discutido anteriormente e $\textsc{sameSet}(x, y)$. Esta última operação responde se x e y estão no mesmo conjunto e para respondê-la é necessário apenas verificar se $\textsc{find}(x) = \textsc{find}(y)$.
- O algoritmo de Kruskal inicializa os vértices, numerados e 0 a $|V| - 1$ em n singletons.
- Esta inicialização pode ser feita facilmente por um método $\textsc{makeSet}$.

Algoritmo de Kruskal

- Em seguida, as arestas são ordenadas em ordem crescente do peso.
- Por fim, para cada aresta (u, v) na ordem dada, pegamos ela para a árvore espalhada desde que $UF.\textsc{sameSet}(u, v)$ seja falso, isto é, desde que u e v estejam em componentes conexas distintas. Em seguida, u e v são colocadas na mesma componente conexa através de $UF.\textsc{union}(u, v)$.

Algoritmo de Kruskal

Algorithm 7: $\textsc{kruskal}(G, w)$

Input: G, w

Output: T, a árvore (ou floresta) espalhada mínima de G

$UF.\textsc{makeSet}(n)$

$T \leftarrow \emptyset$

$E' \leftarrow \textsc{sort}(E, w)$ `// ordena-se as arestas pelo custo`

for all($(u, v) \in E'$)

    if($\neg UF.\textsc{sameSet}(u, v)$)

        $T.\textsc{append}((u, v))$

        $UF.\textsc{union}(u, v)$

return T

Algoritmo de Kruskal

Complexidade

- $\Theta(|E| \lg |V|)$.
- O algoritmo é dominado pela ordenação das arestas.
- Se o custo das arestas são inteiros na faixa $[0, k]$, pode-se considerar utilizar um algoritmo de ordenação pseudolinear e obter a complexidade de $\Theta(k + |E| \cdot \alpha(|V|))$.

Sumário

Considerações

- É possível computar a árvore espalhada mínima de um grafo em tempo $O(|E| \lg |V|)$.
- A estrutura para conjuntos disjuntos utilizada no algoritmo de Kruskal não serve apenas para esse problema, é uma estrutura que pode ser utilizada em outros contextos.
- No algoritmo de Prim, caso seja necessário computar a floresta de custo mínimo, basta repetir o algoritmo para cada componente conexa.