Disjoint Sets*

TEORIA DOS GRAFOS

*MATERIAL ADICIONAL

Nota

Esse conteúdo é auxiliar para os algoritmos de Árvore Geradora Mínima, porém não se limita unicamente a isso.

Conjuntos Disjuntos podem ser usados em diversas aplicações.

Alguns exemplos serão dados nesse material a título de curiosidade e para demonstrar a versatilidade dessa estrutura de dados.

Motivação (de Grafos)

O **algoritmo de Kruskal** para obter uma Árvore Geradora Mínima a partir de um grafo conexo não direcionado funciona basicamente assim:

1 – ordenar as arestas pelo peso

2 – inserir arestas uma de cada vez

3 - A princípio, todos os nós pertencem a uma árvore que contém apenas a si mesmos

4 - Se as duas extremidades de uma aresta pertencerem a árvores diferentes, adicionamos essa aresta ao conjunto de arestas resultante e conectamos as duas árvores em uma.

5 - Caso contrário, descartamos essa aresta.

6 - Quando atingirmos n-1 arestas, temos a Árvore Geradora Mínima do Grafo

Disjoint Sets

Nesse algoritmo, necessitamos iterativamente

- Verificar se dois vértices pertencem à mesma árvore
- Unificar duas árvores

Isso é exatamente o que a estrutura de Disjoint Sets faz!

Outros nomes:

Union-find

Merge-sets

Conjuntos Disjuntos

Union-Find

Na **Matemática**, conjuntos disjuntos são aqueles que não possuem elementos em comum, ou seja, a Interseção é vazia.

Na **Computação**, é uma estrutura de dados que acompanha um conjunto de elementos particionados em vários subconjuntos disjuntos (não sobrepostos).

Union-Find

Nessa estrutura, cada elemento possui *basicamente*

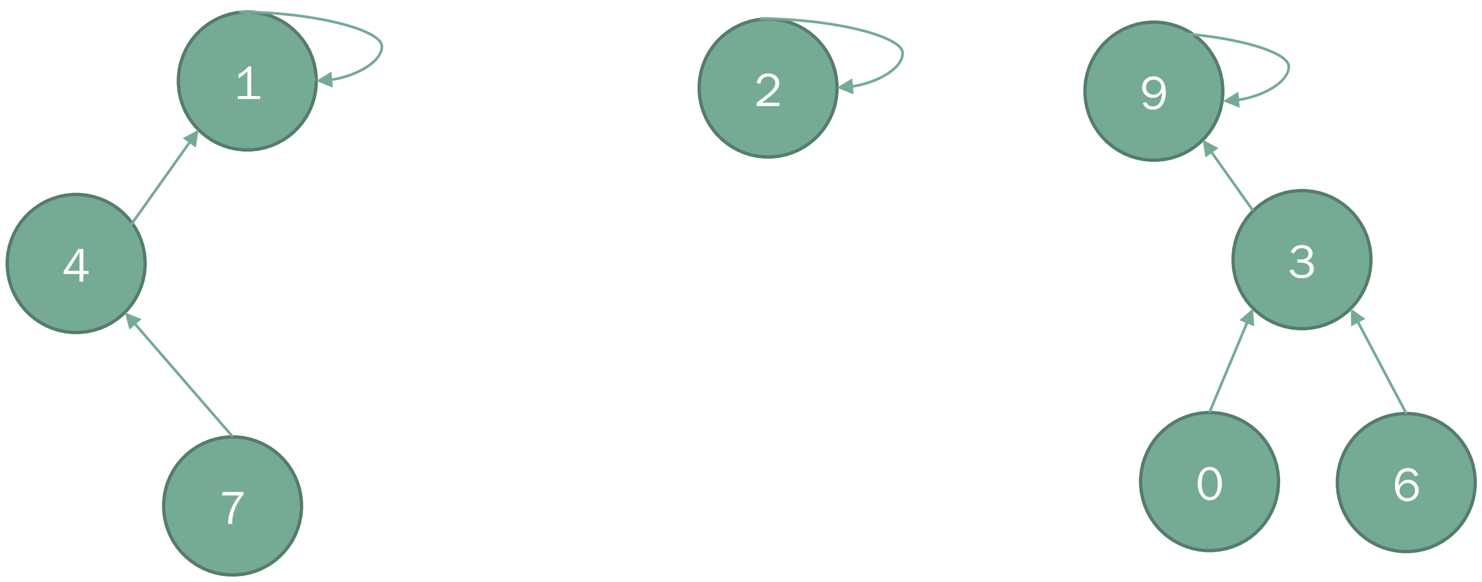
- Um id
- Uma referência para seu pai (ou raiz)

A partir da disposição das raízes dos elementos temos a formação de uma ou mais árvores. Cada árvore representa um conjunto.

Se a referência para o pai de um elemento for nula (ou ele mesmo), então esse elemento é a **raiz** da árvore e representa o seu conjunto.

Cada conjunto pode ser unitário (apenas um elemento), ou é formado por uma cadeia de referências para seus pais, até que se encontre o elemento representativo na raiz da árvore.

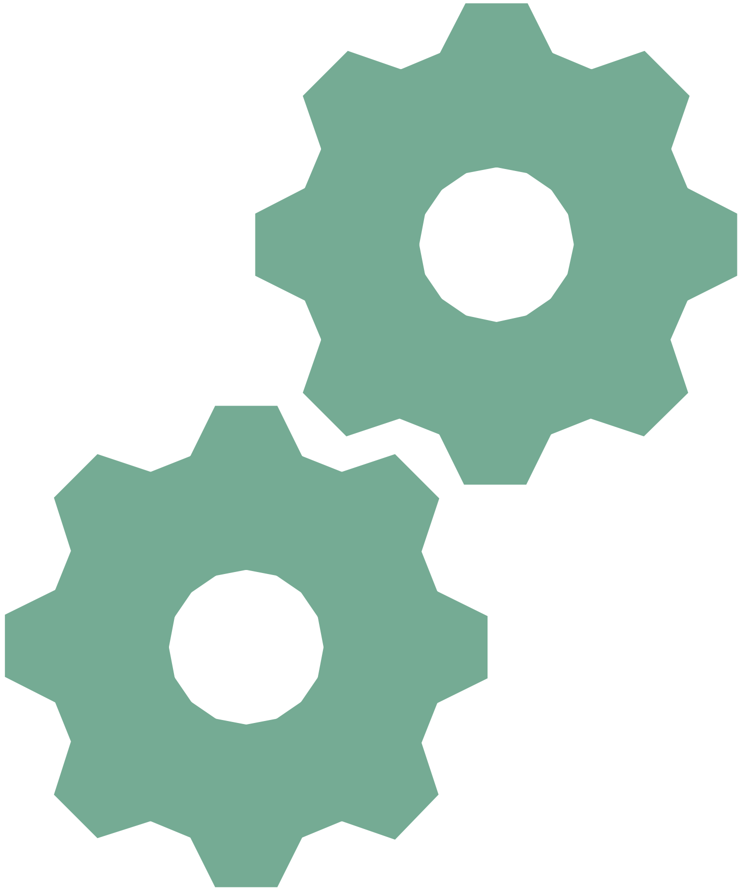
Exemplo Union-Find



Cada elemento possui seu id (valor) e um ponteiro para o ‘pai’.

A raiz de cada conjunto é aquele cujo pai é ele mesmo.

Nesse exemplo, temos 3 conjuntos disjuntos: (1, 4, 7), (2), (9, 3, 6, 0)



Operações

Nessa estrutura, 3 operações básicas podem ser implementadas

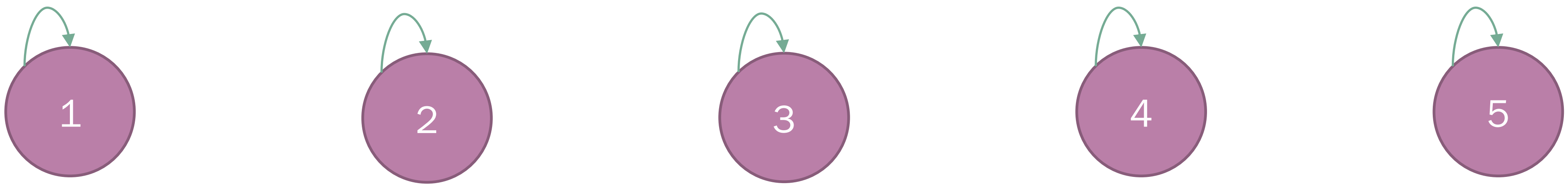
- Make set – inicializar todos os elementos, ou seja, tornar cada elemento ‘raiz’ de seu próprio conjunto
- Union(x, y) – unificar dois conjuntos, ou seja atribuir a mesma raiz aos elementos x e y
- Find(x)– descobrir quem é o ‘pai’ do elemento x, ou seja descobrir a qual conjunto x pertence seguindo a cadeia de referências

Exemplo de Operações

Para ilustrar o funcionamento dessa operação, considere os elementos 1, 2, 3, 4 e 5.

1º. Passo: **Make Set**

Nesse momento, Find(i) = i, ou seja Find(1) = 1, Find (2) = 2, e assim por diante



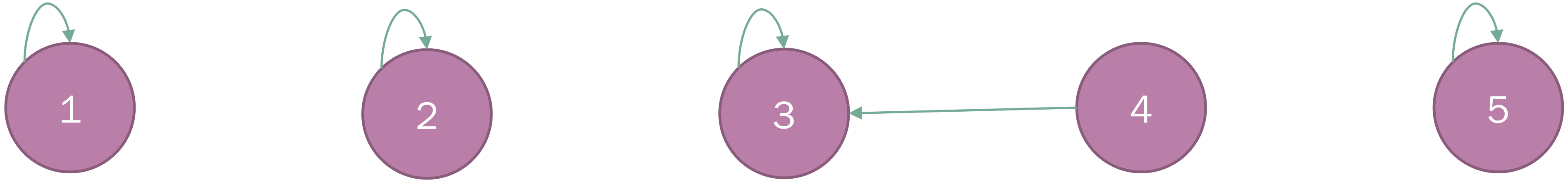
Conjunto 1: {1}   Conjunto 2: {2}   Conjunto 3: {3}   Conjunto 4: {4}   Conjunto 5: {5}

Exemplo de Operações

Se fizermos union(3, 4) os conjuntos 3 e 4 serão unificados no conjunto 3. Teremos:

Union(3,4)

Find(4) = Find(3) = 3



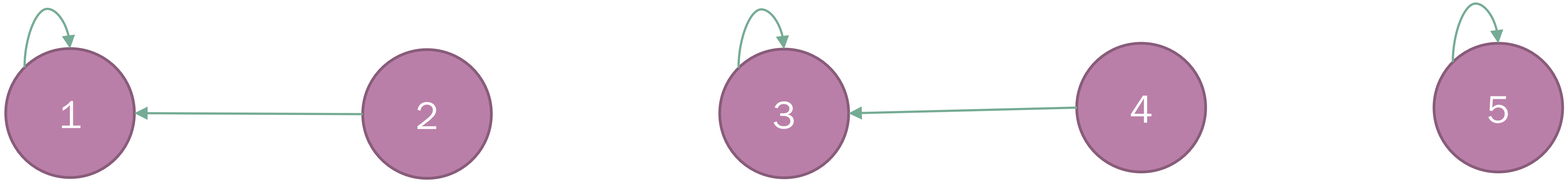
Conjunto 1: {1} Conjunto 2: {2} Conjunto 3: {3, 4} Conjunto 5: {5}

Exemplo de Operações

Se fizermos agora union(1, 2) os conjuntos 1 e 2 serão unificados no conjunto 1. Teremos:

Union(1, 2)

Find(1) = Find(2) = 2



Conjunto 1: {1, 2}

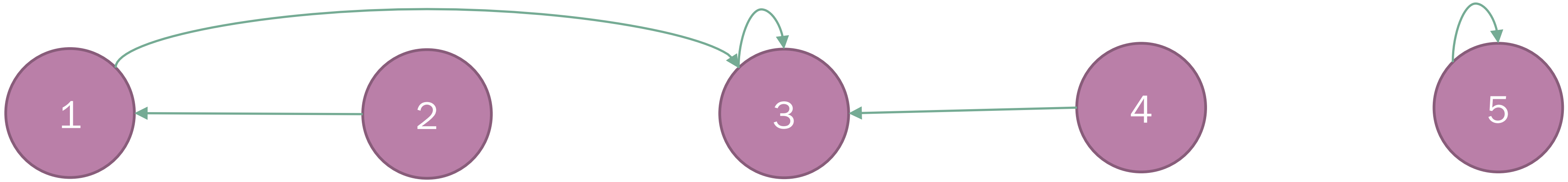
Conjunto 3: {3, 4}

Conjunto 5: {5}

Exemplo de Operações

Se fizermos agora union(3, 1) os conjuntos 3 e 1 serão unificados no conjunto 3. Teremos:

Union(3, 1)



Conjunto 3: {3, 4, 1, 2}

Conjunto 5: {5}

Union-Find

```
#Exemplo de Código

def Find(X):

    if X.parent == X              #Se o X for raiz, retorna X

        return X

    return Find(X.parent)         #Senão, retorna a raiz do pai de X
```

Union-Find

```
def union (X, Y):
    Xroot = Find(X)       #encontra o raiz de X
    Yroot = Find(Y)       #encontra o raiz de Y
    if (Xroot==Yroot):    #se já pertencem à mesma árvore, sai
        return
    Y.parent=Xroot        #raiz de Y passa a ser a raiz de X
```

Union-Find

Note que essa abordagem introduz um problema de desempenho:

Dependendo da sequência de operações Union, a operação Find pode ter o mesmo desempenho de Busca em uma lista ligada, i.e. O(n) – onde n é o comprimento da lista

Isso pode ser contornado pela introdução de duas melhorias.

Melhoria #1

Union by Rank

Consiste de:

- Manter uma informação adicional em cada elemento (rank)
- Sempre conectar a árvore menor à raiz da árvore maior.

Por que melhora?

- Uma vez que a altura da árvore afeta o tempo de execução, inserir a árvore de menor altura só vai aumentar a profundidade quando unirmos duas árvores de altura igual.
- A raiz do conjunto é definido com rank igual a 0 (zero), e toda vez que duas árvores de mesma altura são unidas, o rank resultante é r+1.
- Com isso, o tempo de execução para a operação melhora para O (log N)

Melhoria #2

Path Compression

Consiste de:

- Toda vez que a função Find for invocada, ‘achatar’ a estrutura
- Todos os elementos da árvore ficam a uma distância = 1 do nó raiz.

Por que melhora?

- Uma vez que a altura da árvore afeta o tempo de execução, diminui drasticamente essa altura para as chamadas futuras não só para o próprio elemento, mas para todos os que o referenciam, a um custo de implementação baixo

Pseudocódigos

```
def MakeSet(X):
    X.parent = X  #O elemento é o raiz de seu próprio conjunto
    X.rank = 0    #O rank é 0
```

Pseudocódigos

```
def UnionByRank(X, Y):
    xRoot = Find(X)
    yRoot = Find(Y)
    if (xRoot == yRoot):
        return
    if (xRoot.rank < yRoot.rank)
        xRoot.parent = yRoot
elif (xRoot.rank>yRoot.rank):
    yRoot.parent = xRoot
else:
    yRoot.parent = xRoot
    xRoot.rank +=1
```

Pseudocódigos

```
def Find(X):
    if (X.parent != X):
        X.parent = Find(X.parent) #path compression
    return X.parent
```

Aplicações

- Além de ser usado no algoritmo de Kruskal, Conjuntos Disjuntos podem ser usados para detectar ciclos em um grafo
- Outra aplicação é dividir uma imagem em regiões de cor igual por meio de segmentação e aglomeração. Cada pixel inicialmente é sua própria região, e sucessivamente mescla regiões adjacentes até atingir a um critério de parada
- Separar background/foreground em uma imagem
- Um problema da OBI (Olimpíada Brasileira de Informática): Fusões

  - a descrição está no arquivo pdf disponível no moodle

  - um link para tentar submeter sua resposta no SphereOnline Judge:
    - https://br.spoj.com/problems/FUSOES1/

Referências usadas nesse material

Ian’s Website. 2019. *Union Find*. [online] Disponível em: <https://ianding.io/2019/03/26/union-find/> [Accessed 25 June 2020].

Disjoint Sets Tutorial. Disponível em: <https://helloacm.com/disjoint-sets/>. Acesso em: 26 jun. 2020.