Union-Find

Marlon Henry Schweigert
Matheus Vinícius Valenza
Bacharelado em Ciência da Computação
Complexidade de Algoritmos

Conjunto Disjunto

- É uma estrutura de dados!
- Esta estrutura é composta de um conjunto de elementos particionados em vários subconjuntos disjuntos
- Union-Find é o algoritmo que realiza duas operações importantes na estrutura Conjunto Disjunto

Union-find

- Union: agrupa dois conjuntos distintos em um
- Find: determina a qual conjunto determinado elemento pertence
- Make-set: faz o conjunto conter apenas um elemento. Geralmente é trivial

Conjuntos do que?

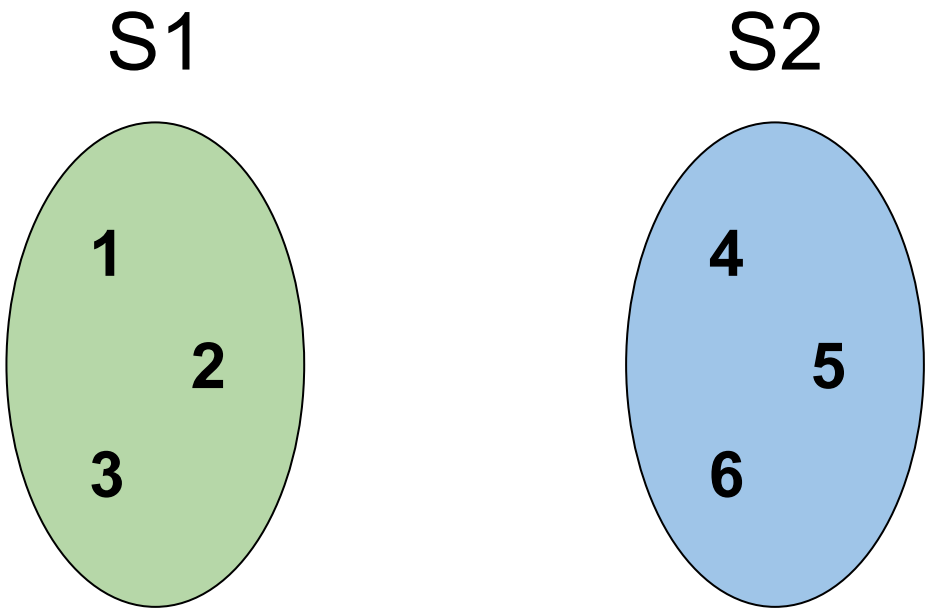
Os conjuntos em questão podem ser compostos por qualquer tipo de elementos comparável.

Intuitivamente, pensaremos nestes elementos como pertencentes aos naturais.

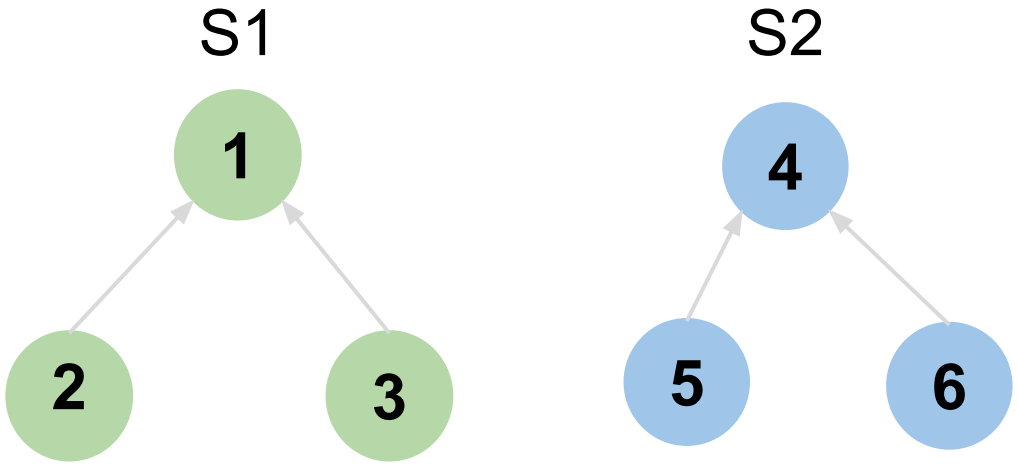
“para todo elemento, elemento = i tal que i pertence aos naturais != 0”
S1 = {1, 2, 3} e S2 = {4, 5, 6}

- Conjunto dos números ímpares, representado por 1:
  - Busca(3) = 1
- Conjunto dos números pares, representado por 2:
  - Busca(4) = 2
- União(s1,s2) = {1,2,3,4,5,6}

Por diagramas



Por árvores



i	[1]	[2]	[3]	[4]	[5]	[6]
p	1	1	1	4	4	4

Algoritmo Noviço

Busca

```ruby
class UniaoBusca
  def initialize elementos
    @el = []
    elementos.each { |n| @el[n] = n }
  end

  def busca a
    @el[a]
  end
end
```

Algoritmo Noviço

União

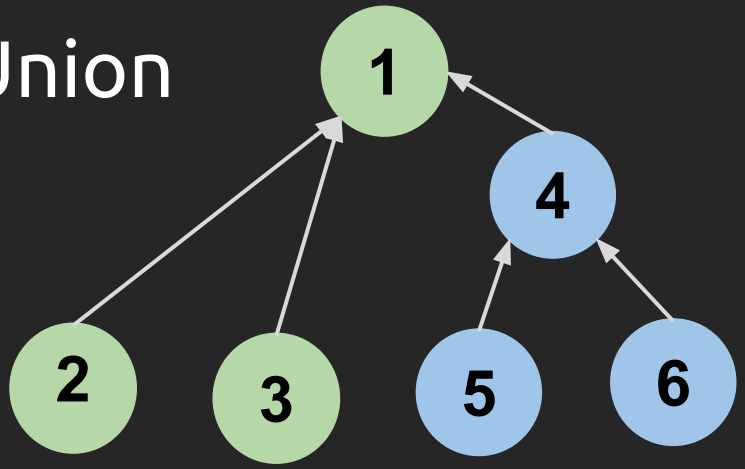
```ruby
class UniaoBusca
  def initialize elementos
    @el = []
    elementos.each { |n| @el[n] = n }
  end

  def busca a
    @el[a]
  end

  def uniao s1, s2
    s1, s2 = busca s1, busca s2
    for i in 0...@el.size
      @el[s1] = s2 if @el[i] == s1
    end
  end
end
```

Complexidade

Seja N o tamanho do conjunto universo (Lista de dados iniciais):

- Construtor:
  - Θ(N)
- Busca (q):
  - Θ(1)
- União (s1, s2):
  - Θ(N)

Quick Union



i	[1]	[2]	[3]	[4]	[5]	[6]
p	1	1	1	1	4	4

União Rápida

Busca

```ruby
class UniaoBusca
  def initialize elementos
    @el = []
    @sz = []
    elementos.each do |n|
      @el[n] = n
      @sz[n] = 1
    end
  end

  def busca a
    return a if @el[a] == a
    busca @el[a]
  end
end
```

União Rápida

União

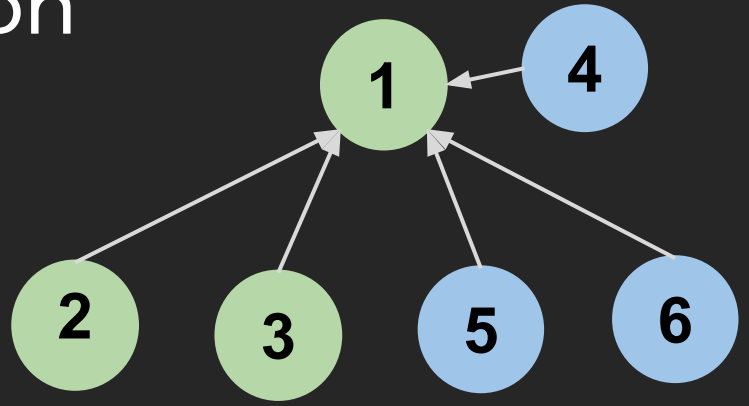
```ruby
class UniaoBusca
  def initialize elementos
    @el = []
    @sz = []
    elementos.each do |n|
      @el[n] = n
      @sz[n] = 1
    end
  end

  def busca a
    return a if @el[a] == a
    busca @el[a]
  end

  def uniao s1, s2
    s1, s2 = busca s1, busca s2
    return if s1 == s2
    s1, s2 = s2, s1 if @sz[s1] > @sz[s2]
    @el[s1] = s2
    @sz[s1] += @sz[s2]
    @sz[s2] += @sz[s1]
  end
end
```

Complexidade

Seja N o tamanho do conjunto universo (Lista de dados iniciais):

- Construtor:
  - Θ(N)
- Busca (q):
  - Θ(log(n))
- União (s1, s2):
  - Θ(log(n))

Path Compression



i	[1]	[2]	[3]	[4]	[5]	[6]
p	1	1	1	1	4	4

Compressão de Caminho

```
class UniaoBusca
  def initialize elementos
    @el = []
    @sz = []
    elementos.each do |n|
      @el[n] = n
      @sz[n] = 1
    end
  end

  def busca a
    return a if @el[a] == a
    @el[a] = busca @el[a]
  end

  def uniao s1, s2
    s1, s2 = busca s1, busca s2
    return if s1 == s2
    s1, s2 = s2, s1 if @sz[s1] > @sz[s2]
    @el[s1] = s2
    @sz[s1] += @sz[s2]
    @sz[s2] += @sz[s1]
  end
end
```

Complexidade

Seja N o tamanho do conjunto universo (Lista de dados iniciais):

- Construtor:
  - $\Theta(N)$
- Busca (q):
  - Inversa da Função de Ackermann
    - $\Theta(\alpha^{-1}(N)) \cong \Theta(1)$
- União (s1, s2):
  - $\Theta(1)$

Implementação do Trabalho

Feito em Ruby.
Publicado como uma biblioteca livre.
Sob licença MIT.

git clone https://github.com/schweigert/unionf

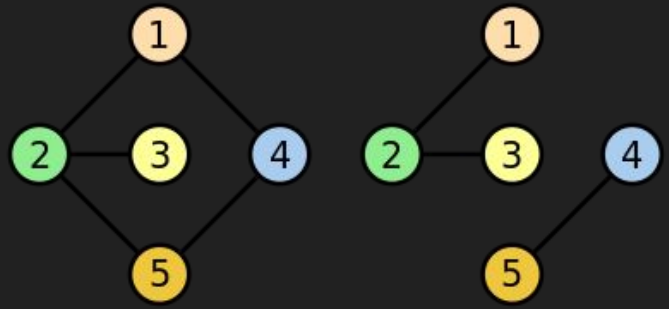
sudo apt-get install ruby
gem install unionf

Exemplo: Facebook


```ruby
require 'unionf'
include Unionf

nodos = [:udesc, :matheus, :marlon, :exemplo_man]
conjunto = UnionFind.new nodos

# Curtem a página UDESC
conjunto.union :udesc, :matheus
conjunto.union :udesc, :marlon
conjunto.union :udesc, :exemplo_man

puts "Quantos likes a UDESC tem?"
puts conjunto.size?(:udesc) - 1

puts "Devo sugerir amizade entre 'Exemplo Man' e 'Marlon'?"
puts conjunto.connected?(:marlon, :exemplo_man)
```

Existe caminho entre nodos?

a) b)

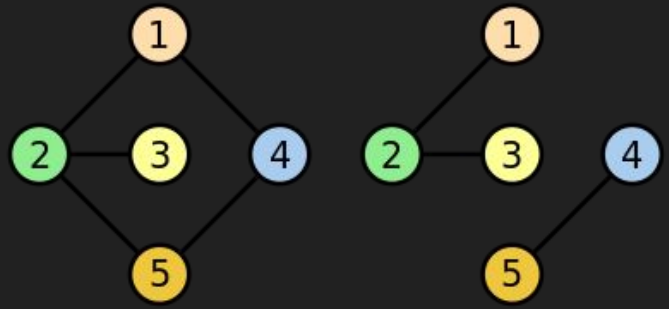


Descobrir se existe um caminho entre dois nós pode ser facilmente implementado utilizando chamadas recursivas em uma busca em profundidade. O seu grande problema é a complexidade de cada busca ser O(n), onde n é o número de vértices para cada chamada.

Caso várias buscas sejam necessárias nesse grafo, talvez seja uma boa alternativa utilizar o algoritmo de Union-Find.

Existe caminho entre nodos?

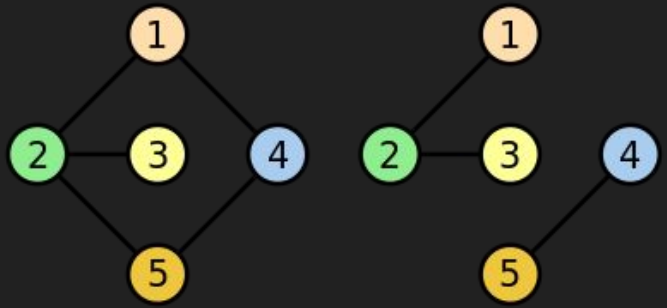
a) b)




```ruby
require 'unionf'
include Unionf

nodos = [1,2,3,4,5]
conjunto = UnionFind.new nodos
lista_adj = {
              1 => [2,4],
              2 => [1,3,5],
              3 => [2],
              4 => [1,5],
              5 => [2,4]
            }

lista_adj.each_key do |k|
  lista_adj[k].each { |n| conjunto.union k, n }
end

puts 'Existe caminho de 1 para 5?'
puts conjunto.connected? 1,5
```

Existe caminho entre nodos?

a) b)




```ruby
require 'unionf'
include Unionf

nodos = [1,2,3,4,5]
conjunto = UnionFind.new nodos
lista_adj = {
              1 => [2],
              2 => [1,3],
              3 => [2],
              4 => [5],
              5 => [4]
            }

lista_adj.each_key do |k|
  lista_adj[k].each { |n| conjunto.union k, n }
end

puts 'Existe caminho de 1 para 5?'
puts conjunto.connected? 1,5
```

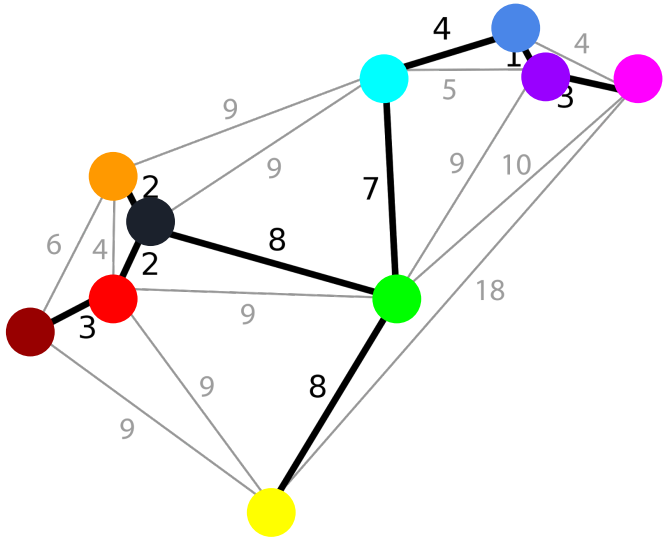
Árvore geradora mínima [Kruskal]

```ruby
require 'unionf'
include Unionf

nodos = [1,2,3,4,5]
conjunto = UnionFind.new nodos
lista_adj = {
              1 => [2],
              2 => [1,3],
              3 => [2],
              4 => [5],
              5 => [4]
            }

lista_adj.each_key do |k|
  lista_adj[k].each { |n| conjunto.union k, n }
end

puts 'Existe caminho de 1 para 5?'
puts conjunto.connected? 1,5
```

Árvore geradora mínima [Kruskal]

Algoritmo de Krushal:

Dado um grafo conexo, gerar uma árvore a qual a soma dos pesos da árvore é mínima.



Árvore geradora mínima [Kruskal]

```
g = [
  [:vinho, :vermelho, 3],
  [:vinho, :laranja, 6],
  [:vinho, :amarelo, 9],
  [:vermelho, :preto, 2],
  [:vermelho, :laranja, 4],
  [:vermelho, :verde, 9],
  [:vermelho, :amarelo, 9],
  [:preto, :laranja, 2],
  [:preto, :verde, 8],
  [:preto, :ciano, 9],
  [:laranja, :ciano, 9],
  [:verde, :amarelo, 8],
  [:verde, :azul, 7],
  [:verde, :roxo, 9],
  [:verde, :pink, 10],
  [:amarelo, :pink, 18],
  [:ciano, :azul, 4],
  [:ciano, :roxo, 5],
  [:azul, :roxo, 1],
  [:azul, :pink, 4],
  [:roxo, :pink, 3]
]
```

```
nodes = [
  :vinho,
  :vermelho,
  :preto,
  :laranja,
  :verde,
  :amarelo,
  :ciano,
  :azul,
  :roxo,
  :pink
]
```

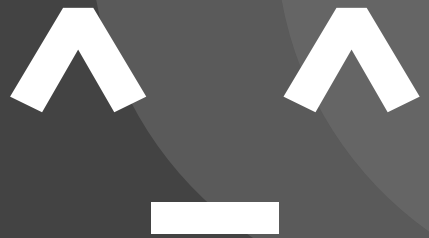
Árvore geradora mínima [Kruskal]

```
require 'unionf'
include Unionf

nodes = [⋯

floresta = UnionFind.new nodes

g = [⋯

arv_min = []

g.sort! do | a,b |
  a[2] <=> b[2]
end

for v in g
  next if floresta.connected? v[0], v[1]

  arv_min << v
  floresta.union v[0], v[1]
end

arv_min.each { |a| puts a.to_s }
```

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Perguntas?

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA



Marlon Henry Schweigert
marlon.henry@magrathealabs.com
Matheus Vinícius Valenza
matheusvvalenza@gmail.com