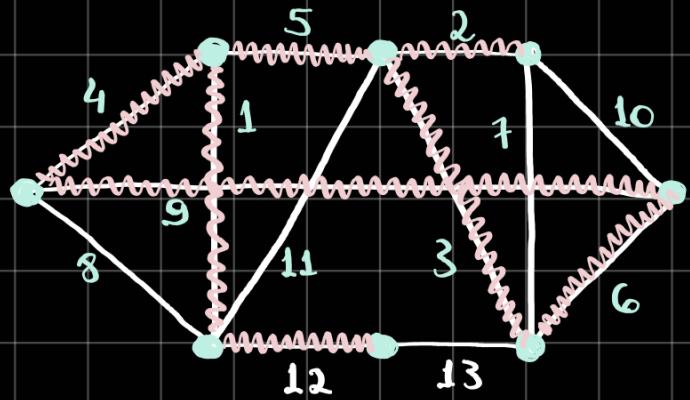


* Kruskal Algoritmo



- Implementação

Usa disjoint-set

Pega a de menor custo dentre todas do grafo.

1º pega a aresta mais leve no caso a de peso 1, depois procura a próxima mais leve que não forme um ciclo com ela.

Ou seja a proxima vai ser 2 se é a proxima aresta mais leve do grafo todo vai ser a aresta mais leve do corte determinado.

Conjunto variável de arestas

```
MST-KRUSKAL( $G, w$ )
1    $A = \emptyset$ 
2   for cada vértice  $v \in G.V$ 
3     do MAKE-SET( $v$ )
4   ordene as arestas de  $G.E$  em ordem não decrescente de peso  $w$ 
5   for cada aresta  $(u, v) \in G.E$ , tomada em ordem não decrescente de peso
6     do if FIND-SET( $u$ ) ≠ FIND-SET( $v$ )
7       then  $A = A \cup \{(u, v)\}$ 
8       UNION( $u, v$ )
9   return  $A$ 
```

Cria uma lista para cada v do grafo

SET = conjunto de

Vértices identificado por um deles, representante

↳ lista encadeada

↳ olha para as arestas e ordena elas

↳ para cada aresta que está aqui dentro fazemos

↳ olhar para as listas, operações FIND-SET que chega nas listas encadeadas e cospe de volta o numero do último vértice dela digamos que estamos examinando $\{u, v\} = \{1, 2\}$

de fato é a aresta mais barata do grafo, vamos olhar para o elemento 1 e vamos ver que ele é o ultimo elemento da

Vts numeros $1 \dots n$

$L_1 = \{(1, 1)\} \{ \dots$

$L_2 = \{(2, 1)\} \{ \dots$

$L_n = \{(n, 1)\} \{ \dots$

$\{u, v\} = \{1, 2\}$

lista dele, elemento 2 é o único da lista dele nesse momento conlcuo que essas listas são diferentes

→ e adiciono essa aresta do grafo no conjunto de arestas

A

→ realizo o procedimento que é a união desses dois "caras"

$$\left\{ \begin{array}{l} L_1 = \{(1, 1)\} \\ L_2 = \{(2, 1)\} \end{array} \right\} \rightarrow \text{e com a união deles passo a ter a lista } L = \{(1, 2), (2, 1)\}$$

nesse momento se perguntar quem é FIND-SET(1)

a resposta vai ser encontrar o 1 nessas lista no caso ele está na 1 encontrei o 1, do 1 vai para o 2, e o 2

é o último elemento da lista, então o FIND-SET(1) = 2

e se perguntar o FIND-SET(2) = 2 (também) por meio

dessa observação temos que 1 e 2 estão dentro da mesma componente conexa. E assim olho para uma próxima aresta, examino novamente os dois vértices da aresta

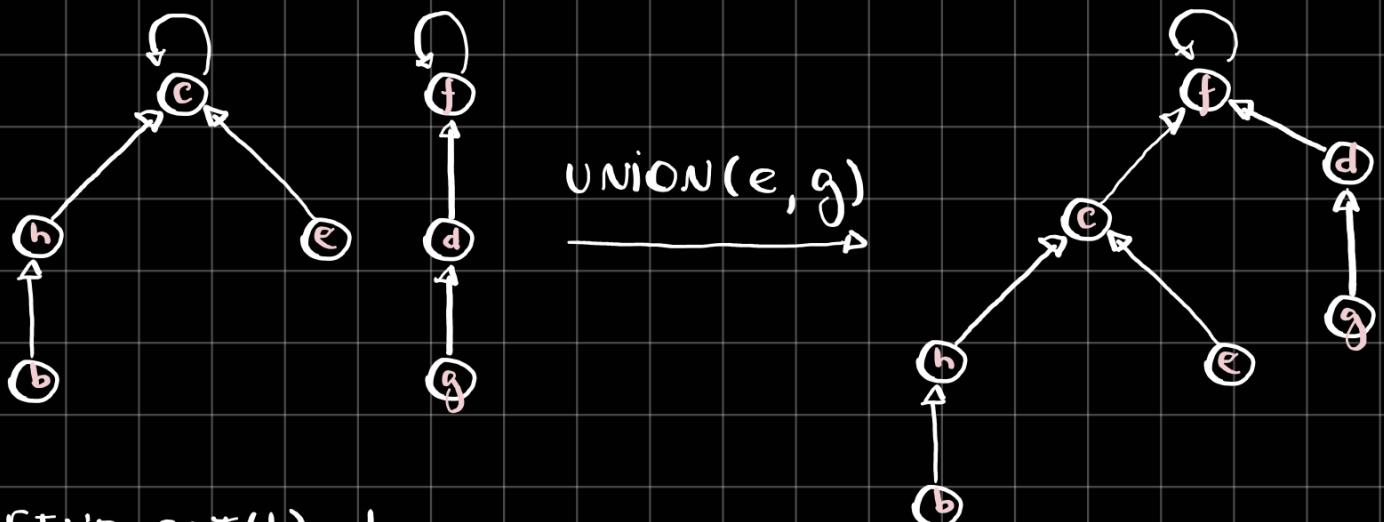
essa operação FINDSET vai responder/concluir se estão ou não na mesma componente se estão em componentes

diferentes adiciono a aresta no conjunto de arestas na árvore e faço a união dos componentes e assim em diante ...

Essa maneira como está construindo a estrutura de dados é um pouco mais complicado (ou seja não seria essa lista) a forma que sugere para o Kruskal seria encadeamento por árvore.

Permite que chega mais rápido no representante. (Cap. 21)
Ao invés da lista encadeada seria uma floresta.

Por exemplo:



$\text{FIND-SET}(b) = b$

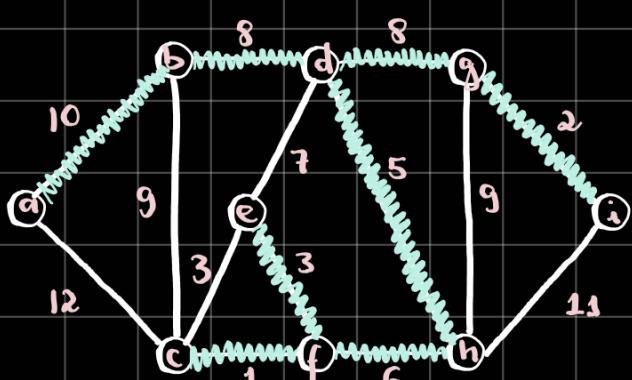
$\text{FIND-SET}(c) = c$ (ele mesmo)

Segue os ponteiros até a raiz e quando chega nela devolve a resposta

$\text{FIND-SET}(e) = \text{FIND-SET}(c) = c$

$\text{FIND-SET}(g) = \text{FIND-SET}(d) = \text{FIND-SET}(f) = f$

Por isso ligo $c \rightarrow f$ ($\text{Union}(f, c)$).



Rodando:

(c,f): safe

(b,c): reject

(g,i): safe

(a,b): safe

(e,f): safe

(h,i): reject

(c,e): reject

(a,c): reject

(d,h): safe

(f,h): safe

(d,e): reject

(b,d): safe

(d,g): safe

(g,h): reject

Análise

Iniciar conjunto arestas: $O(1)$

Primeiro loop for: $|V|$ Make-sets

Ordenar E por peso: $O(|E| \lg |E|)$

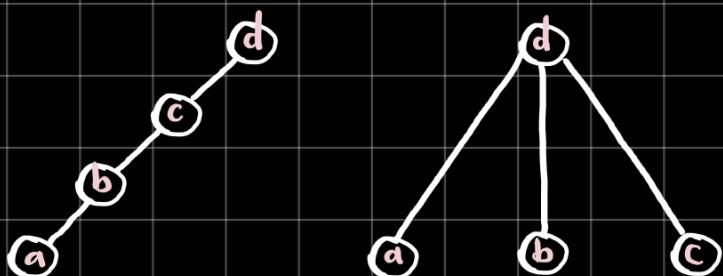
Segundo for: $O(|E|)$ FIND-SET and UNION

Utilizando path compression e union by rank

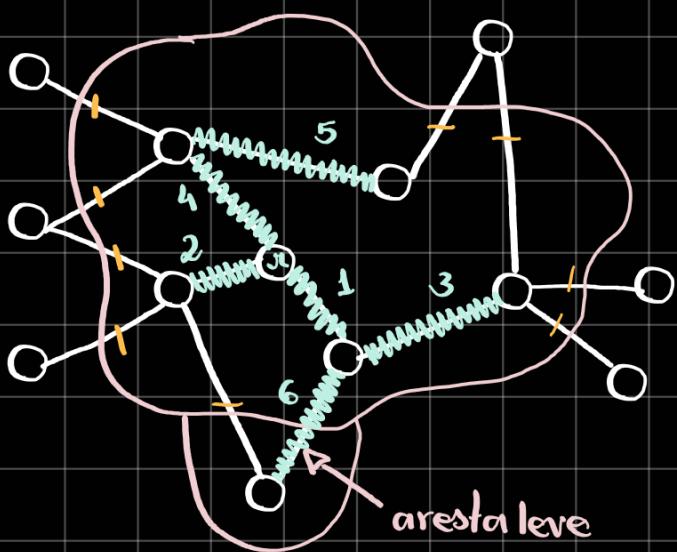
Desse G é conexo $|E| \geq |V|$

$O(\lg v)$

Path compression: quando tem o caminho troca ele por arestas que vão diretamente.



* Prim's Algoritmo



Começa com um vértice arbitrário, que é chamado de raiz e em cada momento do algoritmo, ele expande a partir desse vértice, escolhendo sempre a aresta leve dentre todas as candidatas que são as arestas que atravessam o corte

formado pelo conjunto de vértice já selecionado para estar na árvore.

Temos um conjunto de 5 vértices (verde) vamos examinar todas as arestas que cruzam o corte formado pelo conjunto então vimos que tem uma mais leve e ela passa a fazer parte do conjunto.

- Como encontrar uma aresta leve de forma mais rápida?

Usar uma fila de prioridade, temos uma fila só que quem determina o lugar da fila não é a ordem que o elemento entrou na fila mas sim a ordenação da chave que corresponde ao elemento.

Cada objeto na fila de prioridade é um vértice fora do conjunto $V - VA$.

E a chave do vértice é o menor peso de uma aresta que conecte ele lá dentro.

→ Inicia $V, E, peso, raiz$

```
PRIM(V, E, w, r)
Q ← ∅ → Fila vazia
for each  $u \in V$ 
  do  $key[u] \leftarrow \infty$ 
   $\pi[u] \leftarrow \text{NIL}$ 
  INSERT(Q, u) } Para todos os  $v$ 
                do grafo coloca
                a chave 0 e o pai
                Sendo vazio.

DECREASE-KEY( $Q, r, 0$ )   →  $key[r] \leftarrow 0$ 
while  $Q \neq \emptyset$  → Fila não vazia
  do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
    for each  $v \in Adj[u]$ 
      do if  $v \in Q$  and  $w(u, v) < key[v]$ 
        then  $\pi[v] \leftarrow u$ 
              DECREASE-KEY( $Q, v, w(u, v)$ )
```

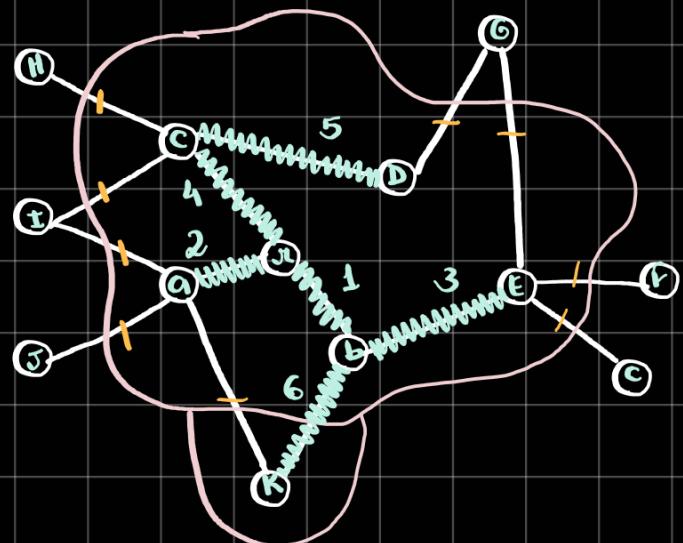
→ Pega a raiz que foi dada como entrada realiza a operação de diminuir a chave da raiz para zero.

→ [0] elemento r
[∞] [∞] todos os outros
que aponta para ele.

→ Remove o de menor chave que na primeira chamada é o elemento r
→ Examina os vizinhos, se eles estão na fila, se as arestas que conectam r na primeira chamada ou aos vértices que já pertence a árvore forem arestas que podem ser adicionadas a

árvore e se o peso da aresta que conecta os vizinhos ao vértice u que está examinando for menor do que a chave do vértice v .

→ Fazemos que o pai de v é u e diminui a chave do vértice v para ser esse peso, dai é concluído e assim em diante...



$Q = \text{todos}$

$\text{Key}(\text{vértices}) = \infty$

$\text{Key}(v) = 0$

$\text{Extract-Min}(Q) = r$

$\text{Adj}(v) = a, b, c$

$\text{Key}(a) = 2, \text{Key}(b) = 1, \text{Key}(c) = 11$
pai deles é r

voltar no while far Extract-Min

$\text{Extract-Min}(Q) = a$ vai sair

da fila e examina

$\text{Adj}(a) = \dots$

Análise

- Suponha que seja implementado com uma binary heap.

Inicializa Q $O(V \lg V)$

Decrease Key de r $O(\lg v)$

While loop $|V| \text{ Extract-Min } O(V \lg V)$

e no máximo $|E| \text{ Decrease-Key } O(E \lg V)$

$O(E \lg V)$