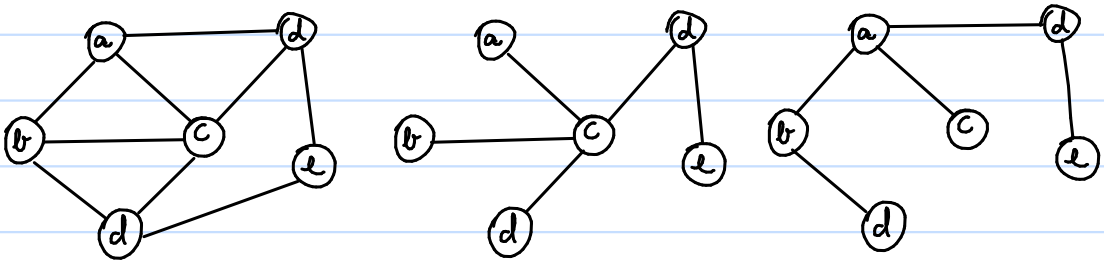
Árvore geradora mínima

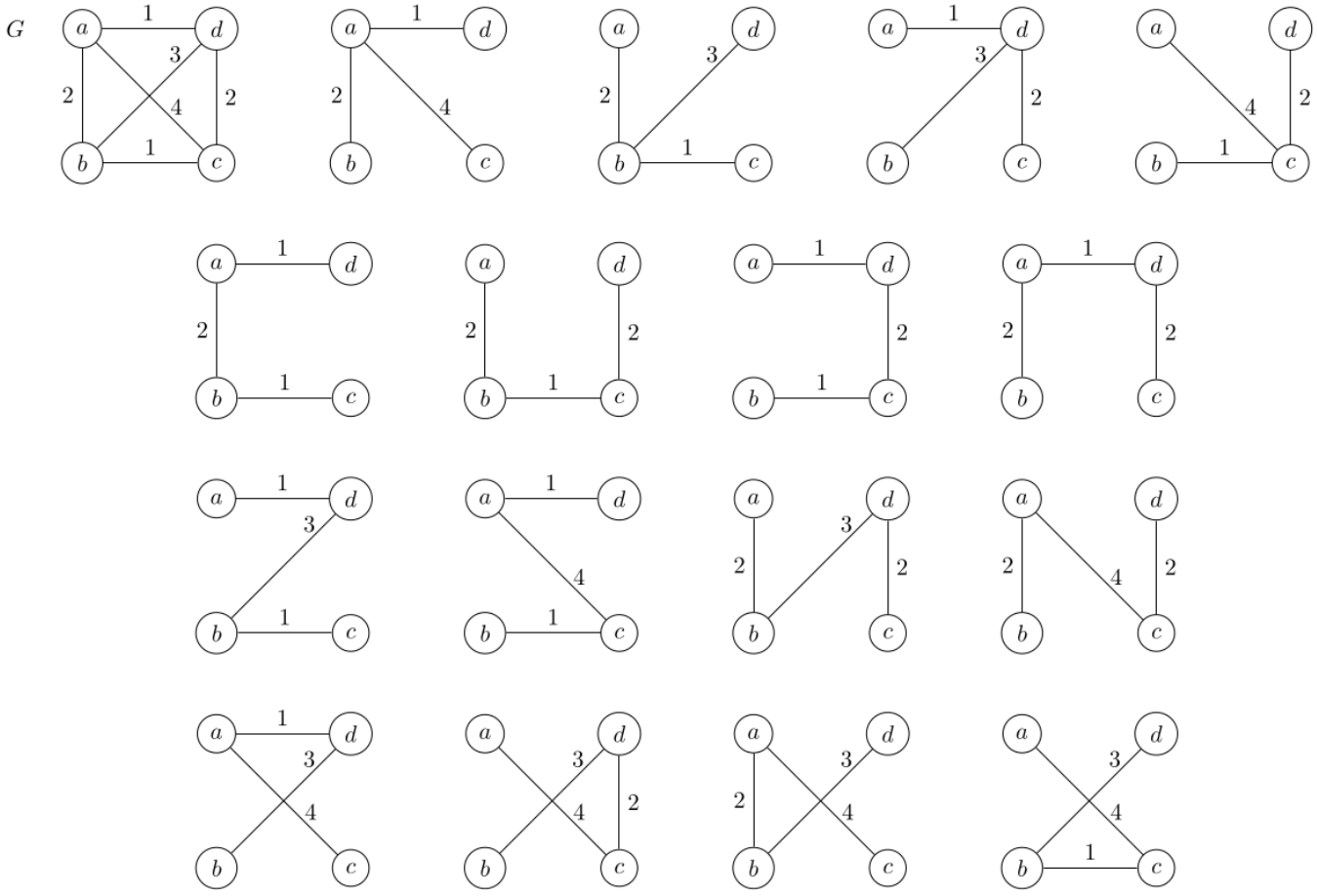
→ Seja G um grafo e $T \subseteq G$.
Se T é árvore e $V(T) = V(G)$, então T é árvore geradora.



→ Já vimos algoritmos que encontram árvores geradoras.

Problema *Árvore Geradora Mínima*

Entrada: $\langle G, w \rangle$, onde G é um grafo conexo e $w\colon E(G) \to \mathbb{R}$.
Saída: Árvore geradora T de G cujo peso $w(T) = \sum_{e \in E(T)} w(e)$ é mínimo.



Alguns conceitos importantes

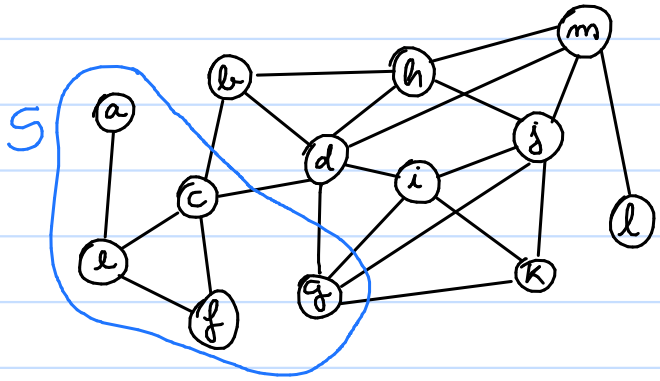
→ Se T é árvore, $|E(T)| = |V(T)| - 1$

Se $u, v \in V(T)$ e $uv \notin E(T)$, $T + uv$ tem um ciclo

→ Dado um grafo G e $S \subseteq V(G)$:

↳ $(S, V(G) \setminus S)$ é um corte

↳ $\partial_G(S) = \{uv \in E(G) : u \in S \text{ e } v \notin S\}$ são arestas que cruzam o corte.



↳ $e \in \partial_G(S)$ é mínima para esse corte se

$$w(e) = \min\{w(f) : f \in \partial_G(S)\}.$$

↳ o corte respeita $F \subseteq E(G)$ se $F \cap \partial_G(S) = \emptyset$ (nenhuma aresta de F cruza o corte).

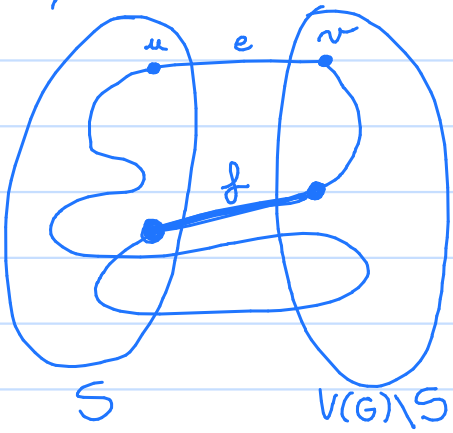
Resultados importantes

LEMA: Sejam H um grafo e C um ciclo de H. Se $e \in E(H)$ pertence a C e $e \in \partial_H(S)$ para algum $S \subseteq V(H)$, então existe $f \in E(H)$, com $f \neq e$, que pertence a C tal que $f \in \partial_H(S)$.

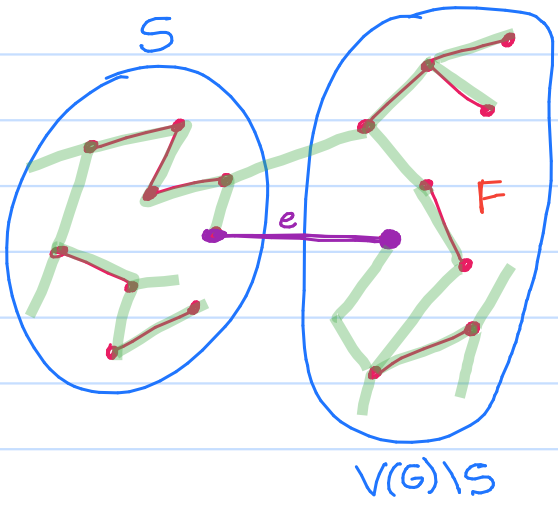
→ Uma aresta sozinha em um corte não pode participar de ciclos.

TEOREMA: Sejam G um grafo conexo e $w: E(G) \to \mathbb{R}$. Seja F um conj. de arestas que está contido em uma árv. geradora mínima de G. Sejam $(S, V(G)\setminus S)$ um corte que respeita F e $e \in \partial_G(S)$ uma aresta mínima desse corte. Então $F \cup \{e\}$ está contido em uma árv. geradora mínima de G.

Intuição do lema:



Intuição do teorema:



Como criar árvores geradoras mínimas?

→ Comece com F contido em uma AGM.
→ Repetidamente adicione uma aresta como a do teorema.
  ↳ mantém invariante de que F está em AGM.
→ Faça isso até $G[F]$ ser conexo: $G[F]$ é AGM!

Como obter árvores geradoras mínimas?

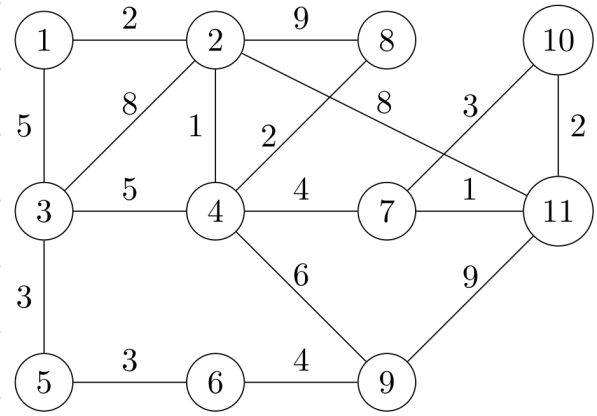
→ Problema: quem é o F inicial?

→ Os dois algoritmos clássicos usam essa abordagem:

① Começa com $H \subseteq G$ tal que $V(H) = V(G)$ e $E(H) = \emptyset$.
Enquanto H tem mais de uma componente conexa:
Adicione a $E(H)$ uma aresta de menor peso que
↳ conecta duas componentes distintas

② Começa com $H \subseteq G$ tal que $V(H) = \{s\}$, $s \in V(G)$, e $E(H) = \emptyset$.
Enquanto $V(H) \neq V(G)$:
Adicione a $E(H)$ uma aresta de menor peso que
↳ conecta $V(H)$ a $V(G) \setminus V(H)$.

Algoritmo de Kruskal



C:

2 4	7 11	1 2	4 8	10 11	3 5	5 6	7 10	4 7	6 9	1 3	3 4	4 9	2 3	2 11	2 8	9 11
1	1	2	2	2	3	3	3	4	4	5	5	6	8	8	9	9

KRUSKAL(G, w)

1 Crie um vetor $C[1..e(G)]$ e copie as arestas de G para C
2 Ordene C de modo não-decrescente de pesos das arestas
3 Seja $F = \emptyset$
4 **para** $i = 1$ *até* $e(G)$, *incrementando* **faça**
5 **se** $G[F \cup \{C[i]\}]$ *não contém ciclos* **então**
6 $F = F \cup \{C[i]\}$
7 **devolve** F

Ele mantém a invariante P(t) = "antes da t-ésima iteração começar, F está contido em uma árvore geradora de G".
Então ao final, ainda é necessário mostrar que G[F] é conexo e gerador.

Kruskal sempre devolve uma árvore? Ótima?

LEMA: Seja G um grafo conexo e $w: E(G) \to \mathbb{R}$. O conjunto F de arestas devolvido por KRUSKAL (G, w) é tal que $G[F]$ é árvore geradora mínima de G.

PROVA: É possível demonstrar que $P(t)$ = "antes da t-ésima iteração começar, F está contido em uma árvore geradora de G" é uma invariante do laço para do algoritmo.
Com isso, ao fim do laço, temos que F está contido em uma AGM.
Resta mostrar que $G[F]$ é conexo e gerador.
Para isso, basta mostrar que para qualquer corte não-vazio do grafo existe uma aresta de F que o cruza.
Seja $S \subseteq V(G)$ qualquer. Seja $e \in \partial_G(S)$ a primeira aresta de $\partial_G(S)$ que é considerada por Kruskal e suponha que isso acontece na i-ésima iteração ($C[i] = e$).
Por ser a primeira desse corte que é considerada, $\partial_{G[F_i \cup \{e\}]}(S)$ só contém essa aresta. Mas então, por ser sozinha, $G[F_i \cup \{e\}]$ não tem ciclos.
Logo, e é adicionada a F. CQD

Qual o tempo de execução?

```
KRUSKAL(G, w)
1 Crie um vetor C[1..e(G)] e copie as arestas de G para C
2 Ordene C de modo não-decrescente de pesos das arestas
3 Seja F = ∅
4 para i = 1 até e(G), incrementando faça
5     se G[F ∪ {C[i]}] não contém ciclos então
6         F = F ∪ {C[i]}
7 devolve F
```

- linha 1: $\Theta(n^2)$ matriz, $\Theta(n+m)$ listas
- linha 2: $\Theta(m \log m)$
- linha 4: $\Theta(m)$
- linha 5: $O(m \cdot n^2)$ matriz, $O(mn)$ listas
- linha 6: $\Theta(n)$

TOTAL LISTAS: $\Theta(n+m) + \Theta(m \log m) + \Theta(m) + O(mn) + \Theta(n)$

$= \Theta(m \log m) + O(mn) = O(mn)$

pois como $m \leq n^2$, então $\log m \leq \log n^2 = 2 \log n$

e assim $\Theta(m \log m) = O(m \log n)$

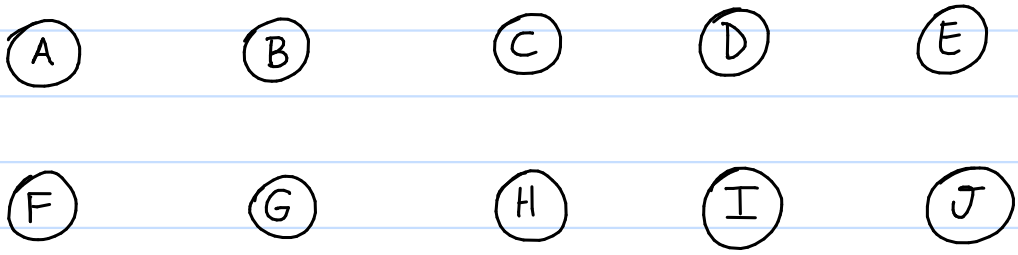
Dá para melhorar?

→ Vamos usar a estrutura de dados UNION-FIND para manter quais vértices estão em quais componentes conexas.

→ Com isso, se uma aresta uv tem $\text{FIND}(u) \neq \text{FIND}(v)$, então ela conecta duas componentes distintas.
Adicionar essa aresta à árvore é equivalente a fazer $\text{UNION}(u, v)$.

Disjoint set

→ Tipo abstrato de dados que mantém um conjunto de elementos particionados em grupos / conjuntos.
  ↳ $A_1, \ldots, A_k$ é partição de B se $A_i \cap A_j = \emptyset$ e $\bigcup_{i=1}^{k} A_i = B$.



→ Cada grupo tem um representante, que é algum elem. do grupo.

→ Operações suportadas:
  ↳ Criar grupo novo
  ↳ Unir dois grupos
  ↳ Descobrir qual o grupo de um elemento

→ Implementações possíveis
  ↳ Árvores
  ↳ Union - Find ($O(\alpha(m))$)
    ↳ $\alpha(n) \leq 5$ para valores práticos de n

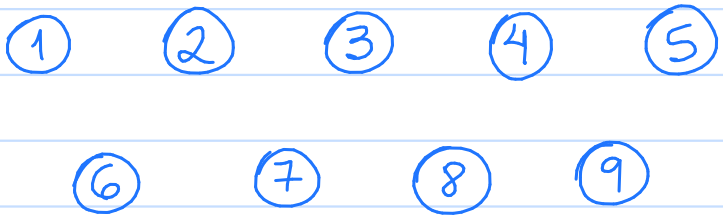
Union - Find (versão simples)

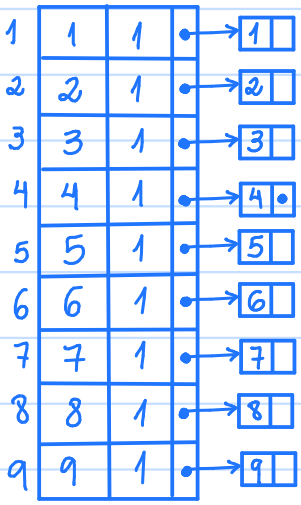
→ Cada elemento x terá
  ↳ x. representante
  ↳ x. tamanho
  ↳ lista $L[x]$ com os elementos de seu grupo, se x for o representante de algum grupo.

Exemplo

→ Elementos a serem mantidos : 1, 2, 3, 4, 5, 6, 7, 8, 9

→ Operações : UNION(2,3), UNION(5,6), UNION(3,9), UNION(1,7), UNION(3,4), UNION(3,8), UNION(2,4), UNION(6,8), UNION(6,7).





Ao unir grupos de tamanhos diferentes, faremos com que o representante do grupo maior passe a ser representante dos elementos do grupo menor.

Implementação

MAKESET(x)

```
1 x.representante = x
2 x.tamanho = 1
3 L[x].cabeca = x
4 L[x].cauda = x
```

FindSet(x)

```
1 devolve x.representante
```

$\Theta(1)$

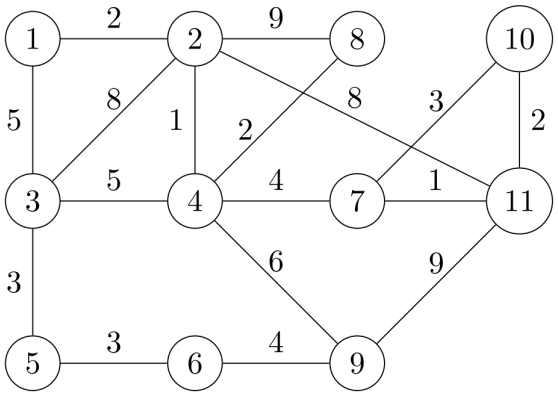
UNION(x, y)

```
1  X = FINDSET(x)
2  Y = FINDSET(y)
3  se X.tamanho < Y.tamanho então
4      para todo v em L[X] faça
5          v.representante = Y
6      Y.tamanho = X.tamanho + Y.tamanho
7      X.tamanho = 0
8      L[Y].cauda.prox = L[X].cabeca
9      L[Y].cauda = L[X].cauda
10     L[X].cabeca = NULO
11 senão
12     para todo v em L[Y] faça
13         v.representante = X
14     X.tamanho = X.tamanho + Y.tamanho
15     Y.tamanho = 0
16     L[X].cauda.prox = L[Y].cabeca
17     L[X].cauda = L[Y].cauda
18     L[Y].cabeca = NULO
```

$\Theta(\min\{x.repr.tam, y.repr.tam\})$

Implementação do Kruskal com Union-Find

```
KruskalUnionFind(G, w)
1  Crie um vetor C[1..e(G)] e copie as arestas de G para C
2  Ordene C de modo não-decrescente de pesos das arestas
3  Seja F = ∅
4  para todo vértice v ∈ V(G) faça
5      MakeSet(v)
6  para i = 1 até e(G), incrementando faça
7      Seja uv a aresta em C[i]
8      se FindSet(u) ≠ FindSet(v) então
9          F = F ∪ {C[i]}
10         Union(u, v)
11 devolve F
```



C:

2 4	7 11	1 2	4 8	10 11	3 5	5 6	7 10	4 7	6 9	1 3	3 4	4 9	2 3	2 11	2 8	9 11
1	1	2	2	2	3	3	3	4	4	5	5	6	8	8	9	9

UNION (2,4) → 4. representante = 2

UNION (7,11) → 11. representante = 7

UNION (1,2) → 1. representante = 2

UNION (4,8) → 8. representante = 2

UNION (10,11) → 10. representante = 7

UNION (3,5) → 5. representante = 3

UNION (5,6) → 6. representante = 5

UNION (4,7) → 7. representante = 2, 10. representante = 2, 11. representante = 2

UNION (6,9) → 9. representante = 3

UNION (1,3) → 3. representante = 2, 5. representante = 2, 6. representante = 2
9. representante = 2

Qual o tempo de execução?

```
KruskalUnionFind(G, w)
1  Crie um vetor C[1..e(G)] e copie as arestas de G para C
2  Ordene C de modo não-decrescente de pesos das arestas
3  Seja F = ∅
4  para todo vértice v ∈ V(G) faça
5      MakeSet(v)
6  para i = 1 até e(G), incrementando faça
7      Seja uv a aresta em C[i]
8      se FindSet(u) ≠ FindSet(v) então
9          F = F ∪ {C[i]}
10         Union(u, v)
11 devolve F
```

linha 1: $\Theta(n+m)$

linha 2: $\Theta(m \lg m)$

linha 4: $\Theta(n)$

linha 5: $\Theta(n)$

linha 6: $\Theta(m)$

linha 7: $\Theta(m)$

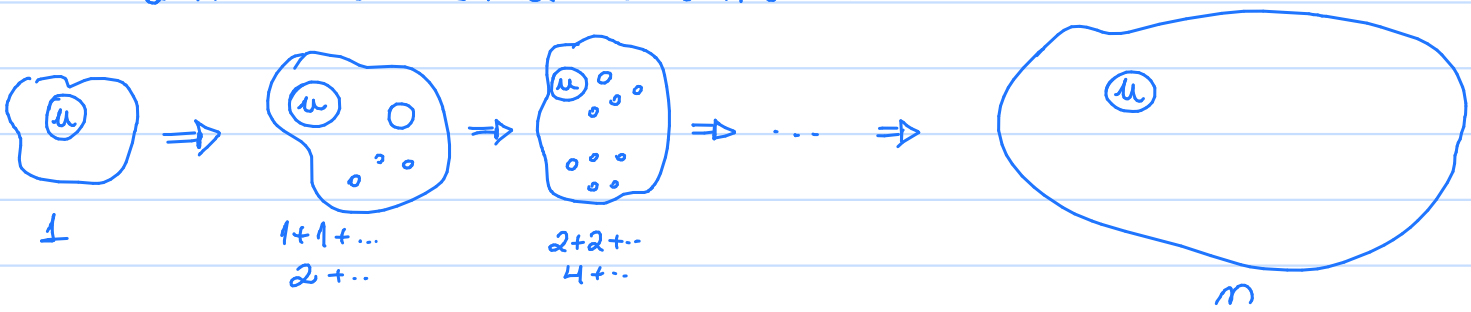
linha 8: $\Theta(m)$

linha 9: $\Theta(n)$

linha 10: Rápida: $O(n^2)$ — n execuções, cada união sendo $O(n)$

Mas cada vértice participa "pouco" do union: do ponto de vista do vértice, seu representante é atualizado poucas vezes.

Todo vértice u começa em uma componente conexa de tamanho 1 e termina em uma de tamanho n.



Quando o representante de u é atualizado, a componente de u pelo menos dobra de tamanho.

$\therefore$ são $O(\lg n)$ atualizações para u.

$\therefore$ $O(n \lg n)$ é o tempo total da linha 10.

TOTAL: $\Theta(m) + \Theta(n) + \Theta(m \lg m) + O(n \lg n)$

$= \Theta(m \lg m) + O(n \lg n)$

$= O(m \lg n)$

pois $n - 1 \leq m$ (grafo é conexo)

e $\lg m \leq 2 \lg n$ (pois $m \leq n^2$).