Melhores momentos

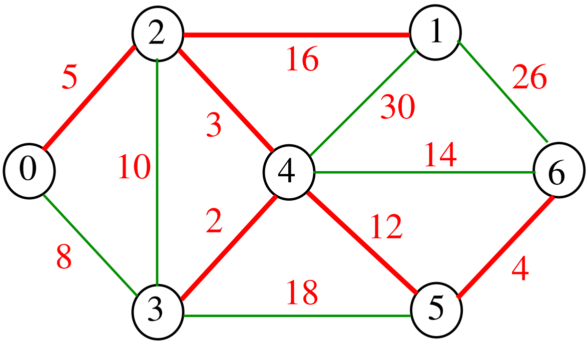
AULA 22

Problema MST

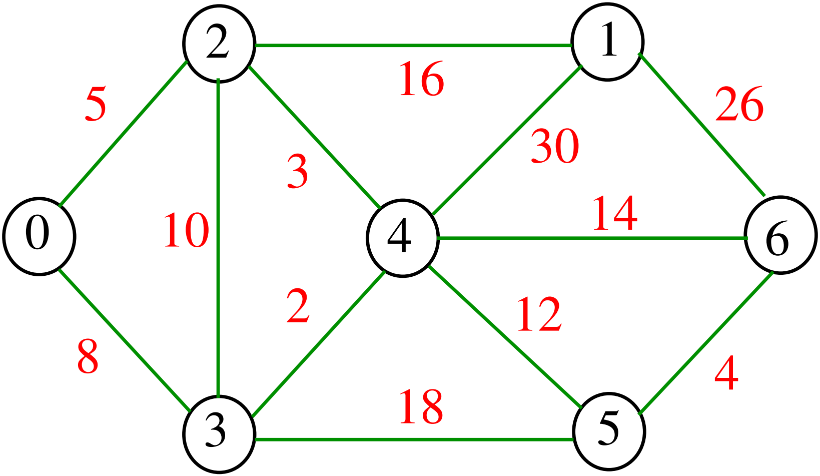
Problema: Encontrar uma MST de um grafo `G` com custos nas arestas

O problema tem solução se e somente se o grafo `G` é conexo

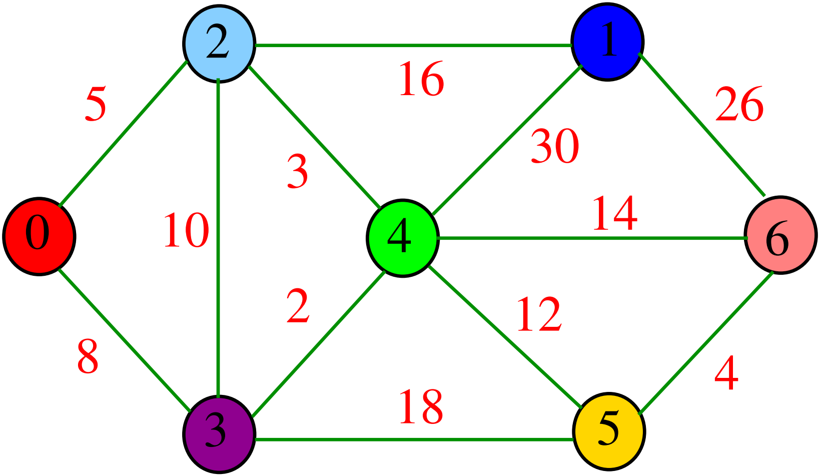
Exemplo: MST de custo 42



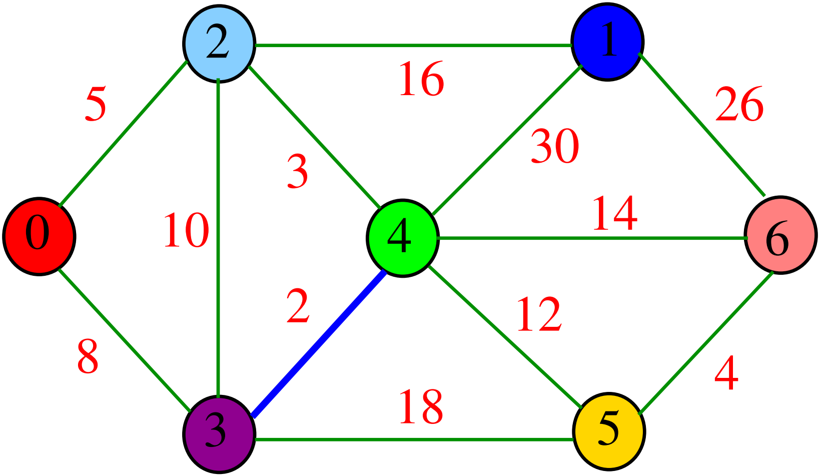
Algoritmo de Kruskal



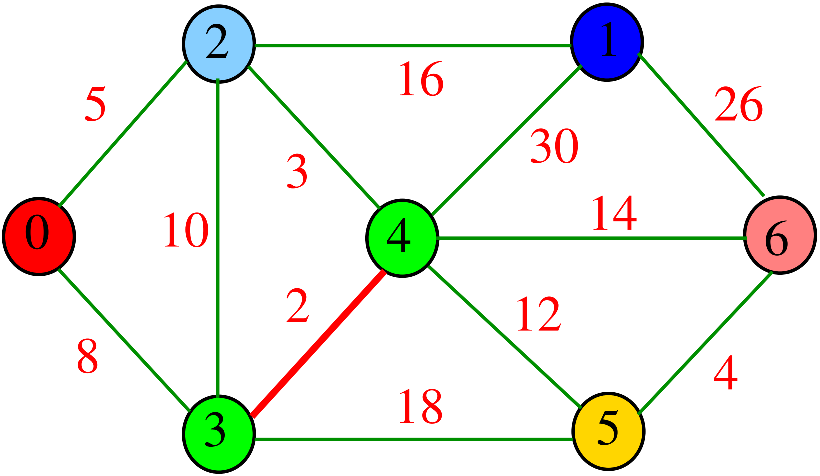
Algoritmo de Kruskal



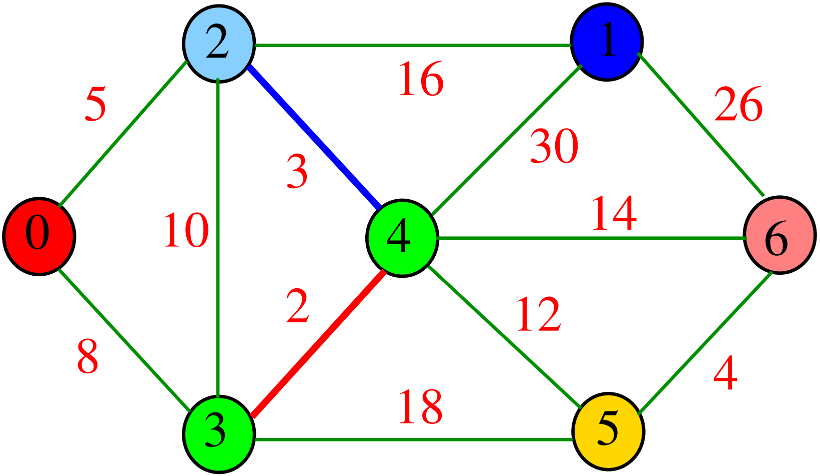
Algoritmo de Kruskal



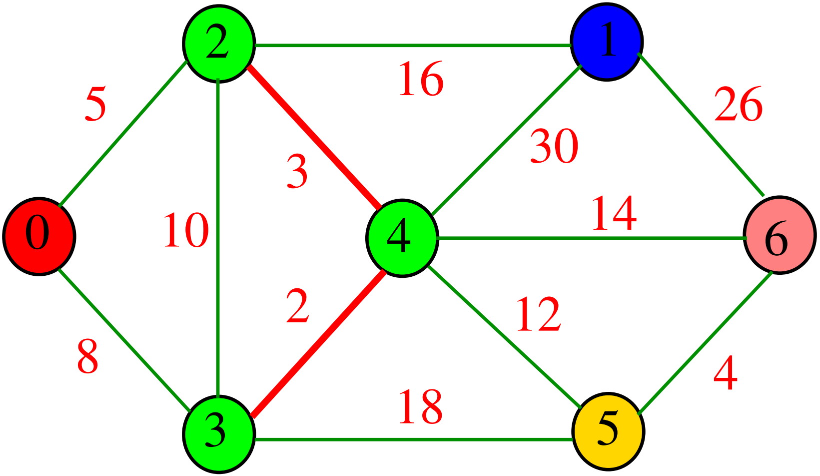
Algoritmo de Kruskal



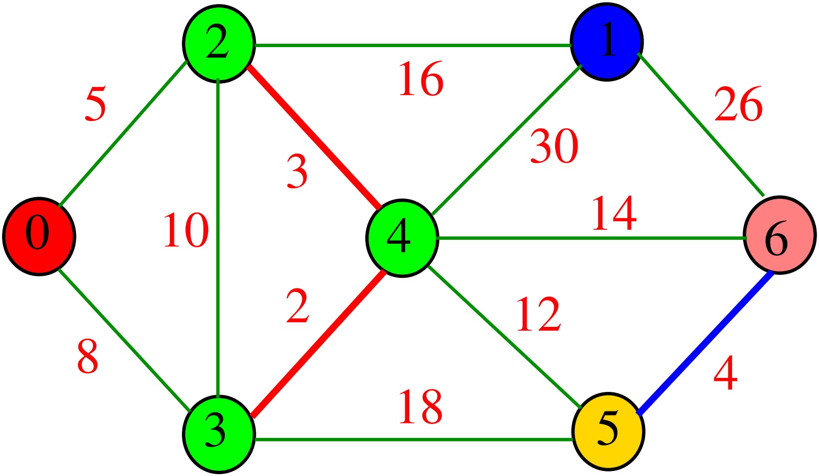
Algoritmo de Kruskal



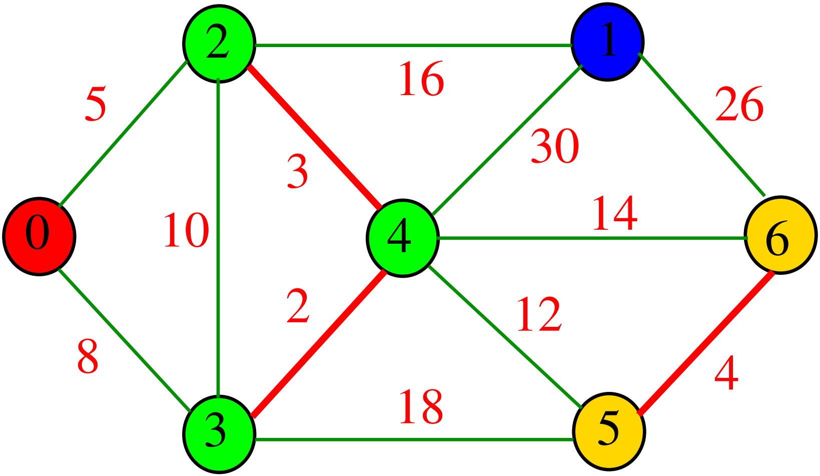
Algoritmo de Kruskal



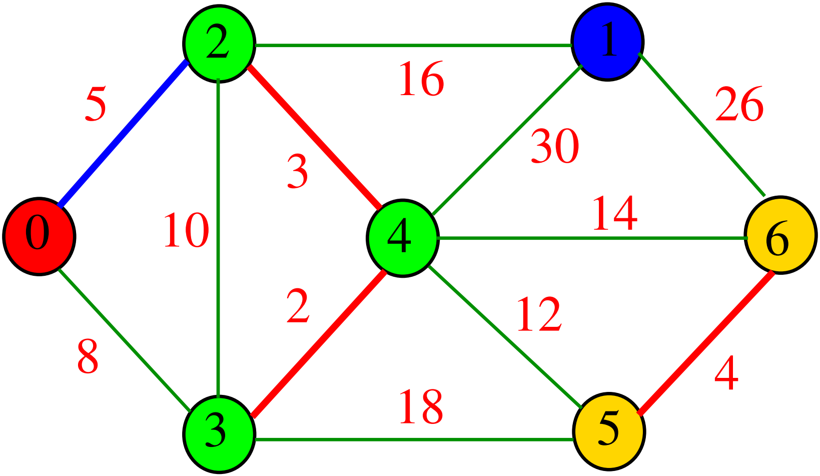
Algoritmo de Kruskal



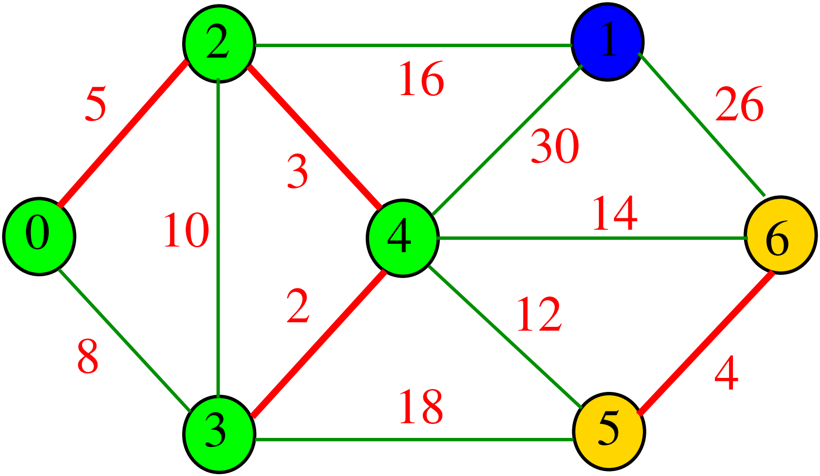
Algoritmo de Kruskal



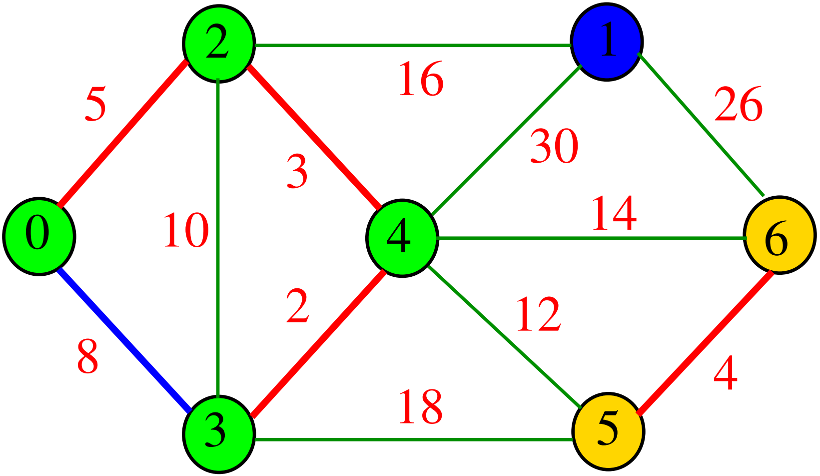
Algoritmo de Kruskal



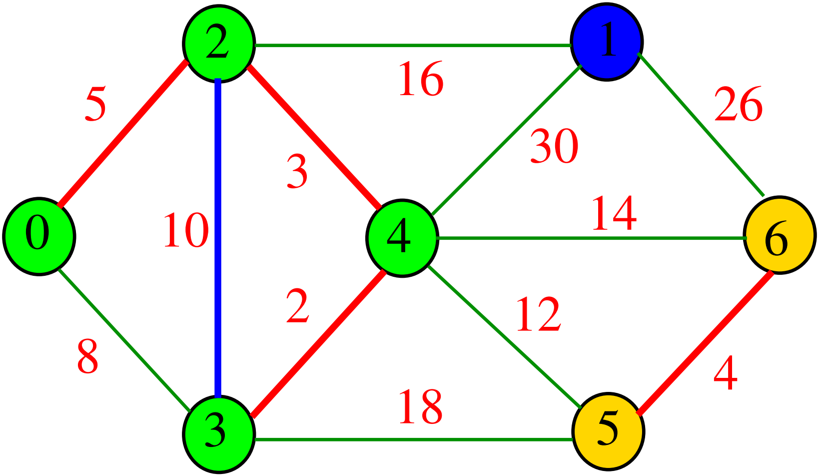
Algoritmo de Kruskal



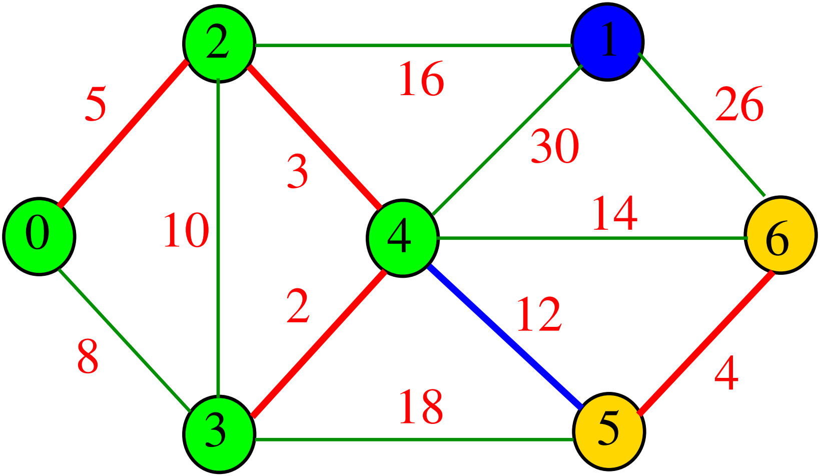
Algoritmo de Kruskal



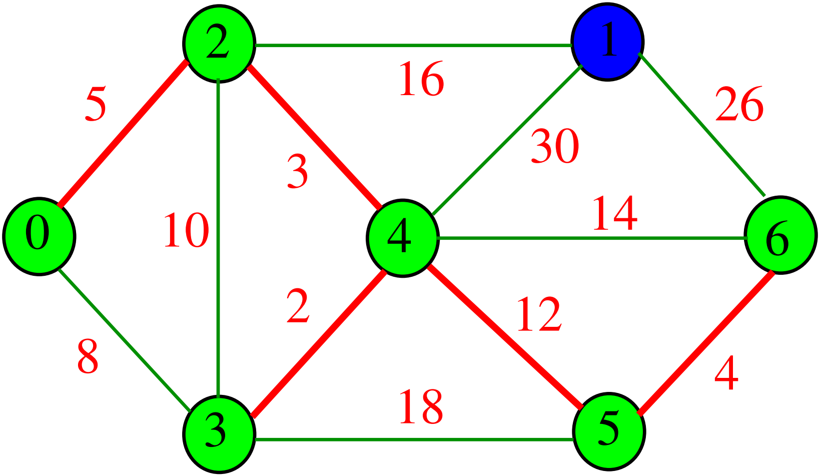
Algoritmo de Kruskal



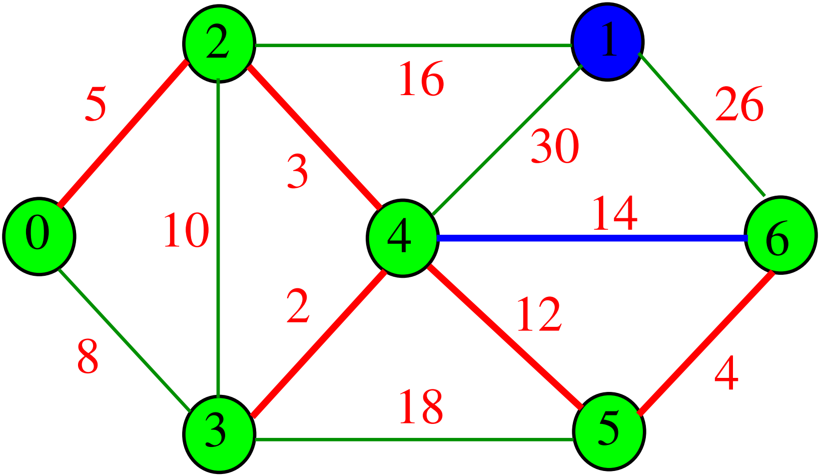
Algoritmo de Kruskal



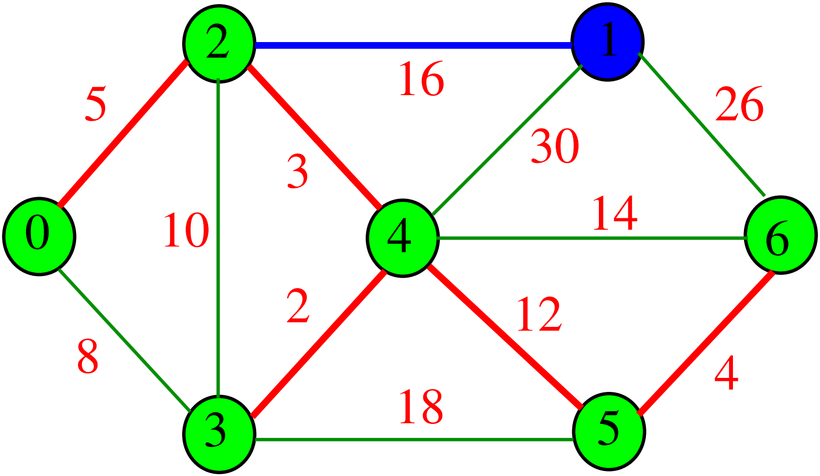
Algoritmo de Kruskal



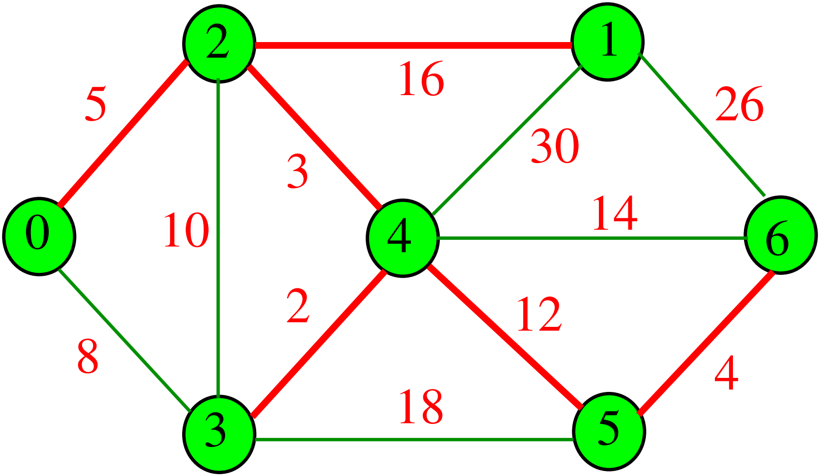
Algoritmo de Kruskal



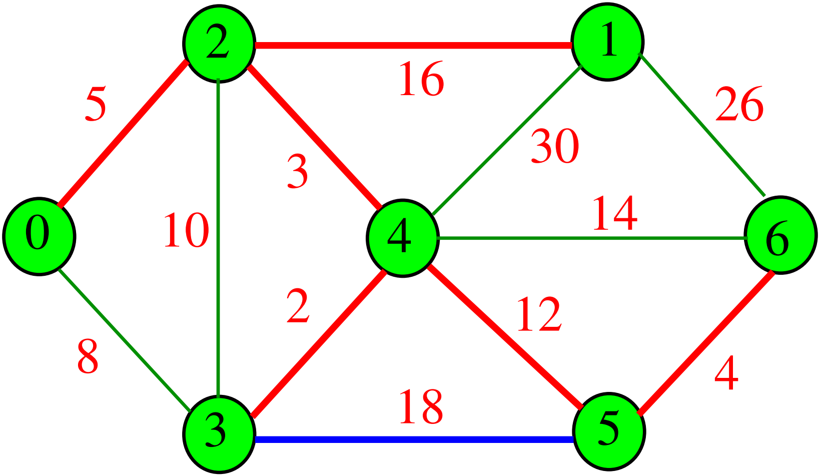
Algoritmo de Kruskal



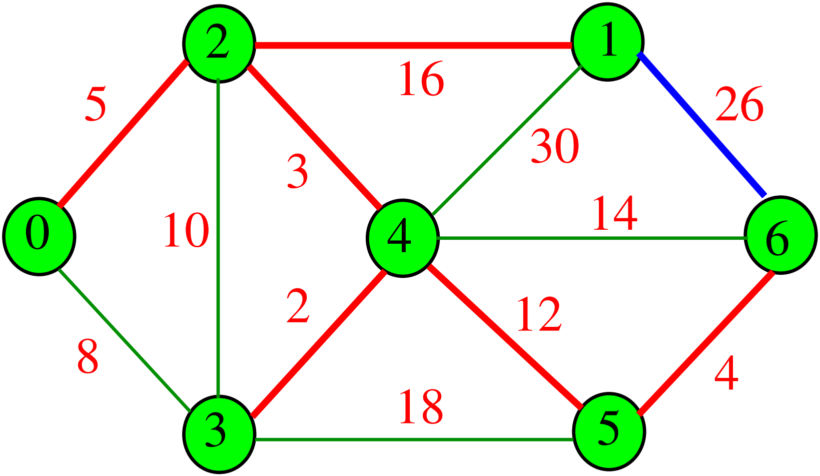
Algoritmo de Kruskal



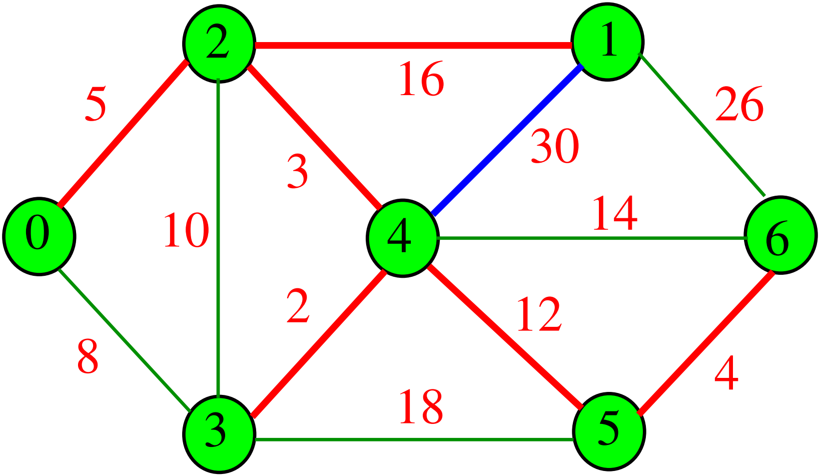
Algoritmo de Kruskal



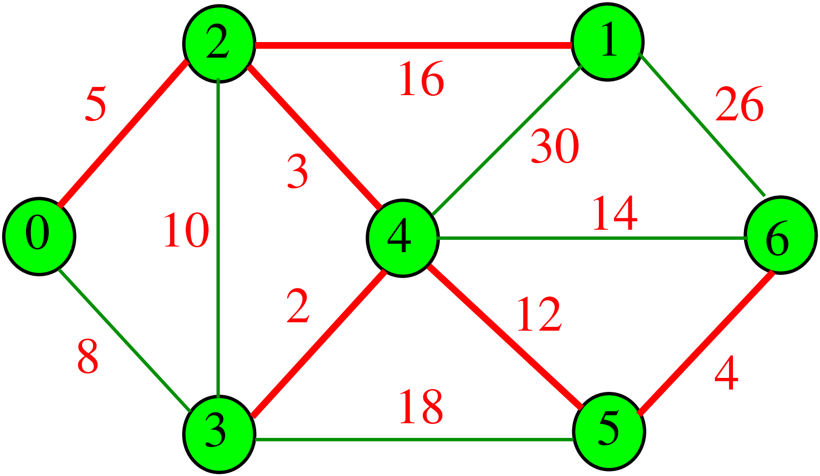
Algoritmo de Kruskal



Algoritmo de Kruskal



Algoritmo de Kruskal



Union-Find

As as funções `UFinit`, `UFfind` e `UFunion` têm o seguinte papel:

- `UFinit(V)` inicializa a floresta de conjuntos disjuntos com cada árvore contendo apenas 1 elemento;
- `UFfind(v, w)` tem valor 0 se e somente se `v` e `w` estão em componentes distintas da floresta;
- `UFunion(v, w)` promove a união das componentes que contêm `v` e `w` respectivamente.

Implementações eficientes

A função recebe um grafo `G` com custos nas arestas e calcula uma MST em cada componente de `G`.
A função armazena as arestas das MSTs no vetor mst[0..`k`-1] e devolve `k`.

```
#define maxE 10000
```

Kruskal

```
int GRAPHmstK (Graph G, Edge mst[]){
    int i, k, E = G->A/2;
    Edge a[maxE];
    GRAPHedges(a, G);
    sort(a, 0, E-1);
    UFinit(G->V);
    for (i= k= 0; i < E && k< G->V-1; i++)
        if (!UFfind(a[i].v, a[i].w)) {
            UFunion(a[i].v, a[i].w);
            mst[k++] = a[i];
        }
    return k;
}
```

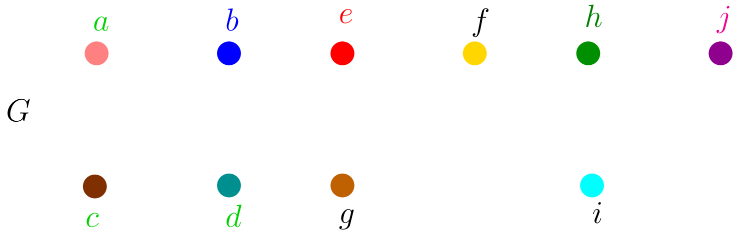
Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

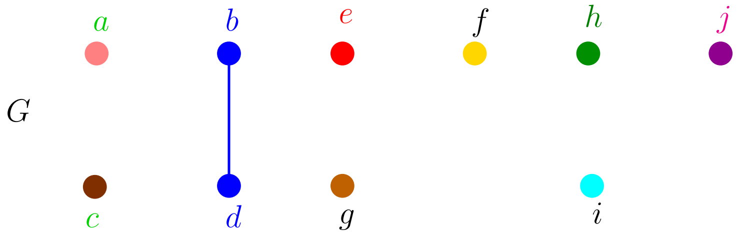


aresta	componentes
	$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

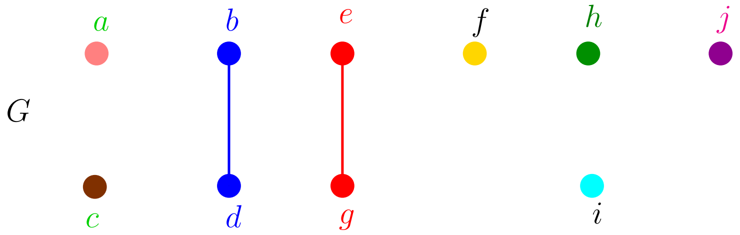


aresta	componentes
(b, d)	$\{a\}$ $\{b, d\}$ $\{c\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

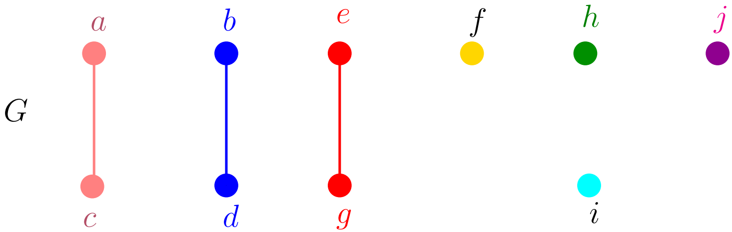


aresta	componentes
(e, g)	$\{a\}$ $\{b, d\}$ $\{c\}$ $\{e, g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

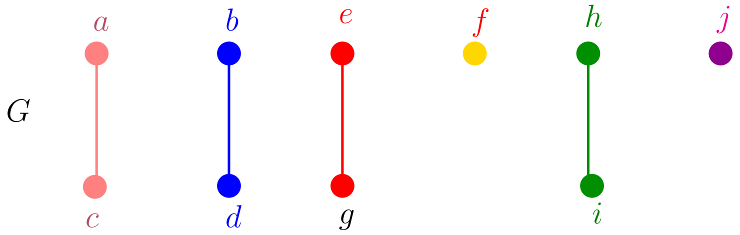


aresta	componentes
(a,c)	$\{a,c\}$ $\{b,d\}$ $\{e,g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

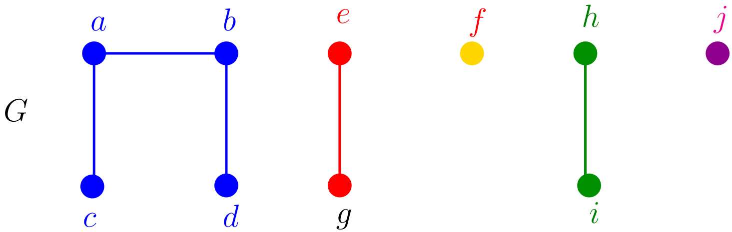


aresta	componentes
(h, i)	$\{a, c\}$ $\{b, d\}$ $\{e, g\}$ $\{f\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

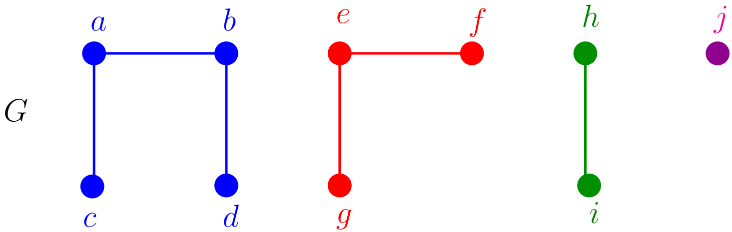


aresta	componentes
(a, b)	$\{a, b, c, d\}$ $\{e, g\}$ $\{f\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico

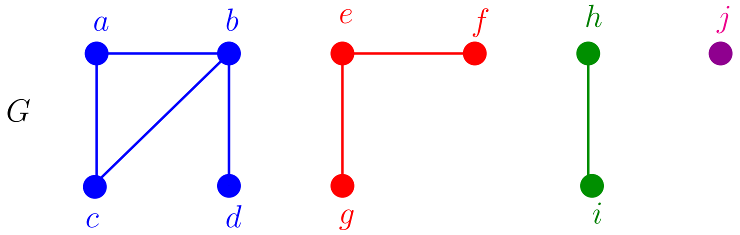


aresta	componentes
(e, f)	$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são **modificados ao longo do tempo**

Exemplo: grafo dinâmico



aresta	componentes
(b, c)	$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

Operações básicas

$\mathcal{S}$ coleção de conjuntos disjuntos.

Cada conjunto tem um **representante**.

MakeSet (x): x é elemento novo
$\mathcal{S} \leftarrow \mathcal{S} \cup \{\{x\}\}$

Union (x, y): x e y em conjuntos diferentes
$\mathcal{S} \leftarrow \mathcal{S} - \{S_x, S_y\} \cup \{S_x \cup S_y\}$
x está em S_x e y está em S_y

FindSet (x): devolve representante do conjunto que contém x

Conjuntos disjuntos dinâmicos

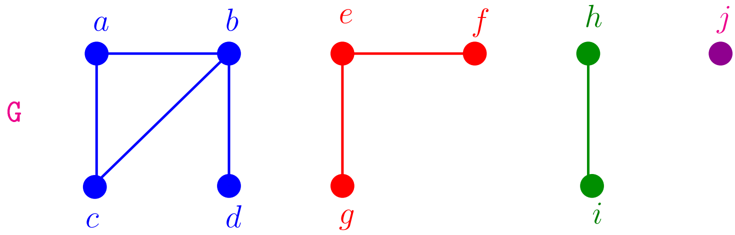
Seqüência de operações MakeSet, Union, FindSet

$$\underbrace{\underbrace{\text{M} \quad \text{M} \quad \text{M}}_{n} \quad \text{U} \quad \text{F} \quad \text{U} \quad \text{U} \quad \text{F} \quad \text{U} \quad \text{F} \quad \text{F} \quad \text{F} \quad \text{U} \quad \text{F}}_{m}$$

Que estrutura de dados usar?

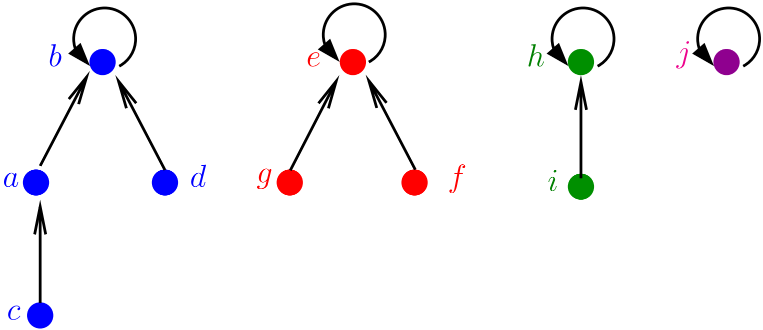
Compromissos (*trade-off*s).

Estrutura *disjoint-set forest*



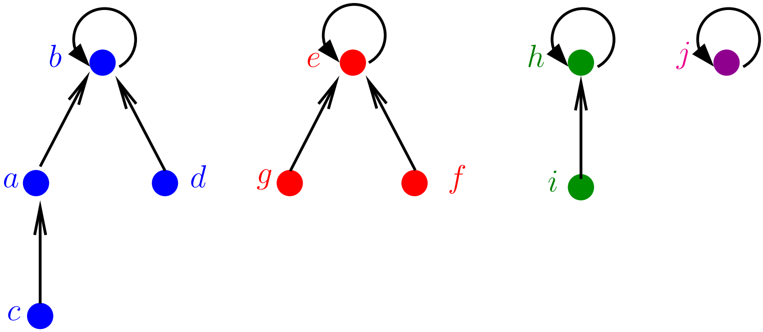
- cada conjunto tem uma *raiz*, que é o seu representate
- cada nó `x` tem uma `cor`
- $\texttt{cor}[\texttt{x}] = \texttt{x}$ se e só se `x` é uma raiz

Estrutura *disjoint-set forest*



- cada conjunto tem uma *raiz*
- cada nó `x` tem uma `cor`
- $\texttt{cor}[\texttt{x}] = \texttt{x}$ se e só se `x` é uma raiz

MakeSet$_0$ e FindSet$_0$



MAKESET$_0$ $(\mathtt{x})$

$\mathtt{cor}[\mathtt{x}] \leftarrow \mathtt{x}$

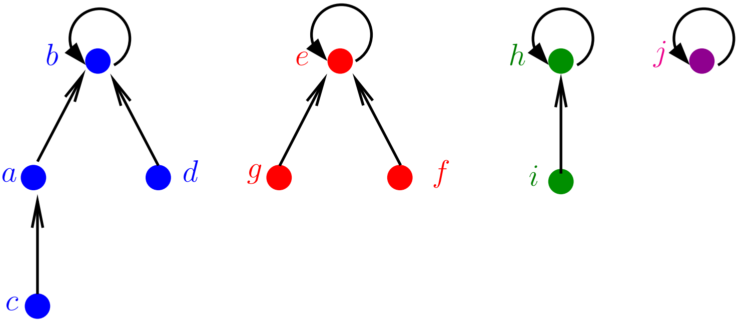
FINDSET$_0$ $(\mathtt{x})$

enquanto $\mathtt{cor}[\mathtt{x}] \neq \mathtt{x}$ **faça**

    $\mathtt{x} \leftarrow \mathtt{cor}[\mathtt{x}]$

devolva $\mathtt{x}$

Union$_0$



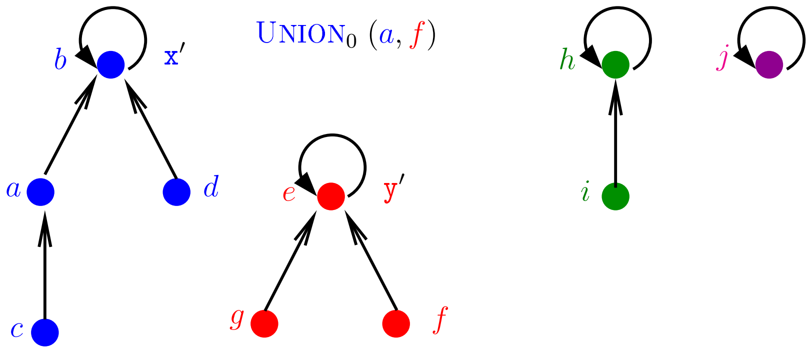
UNION$_0$ $(\mathtt{x}, \mathtt{y})$

$\mathtt{x}' \leftarrow$ FINDSET$_0$ $(\mathtt{x})$

$\mathtt{y}' \leftarrow$ FINDSET$_0$ $(\mathtt{y})$

$\mathtt{cor}[\mathtt{y}'] \leftarrow \mathtt{x}'$

Union_0



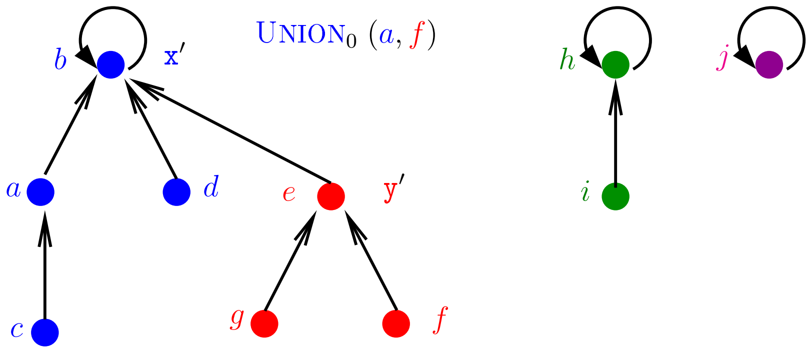
$\text{UNION}_0\ (\mathtt{x}, \mathtt{y})$

1 $\quad \mathtt{x}' \leftarrow \text{FINDSET}_0\,(\mathtt{x})$

2 $\quad \mathtt{y}' \leftarrow \text{FINDSET}_0\,(\mathtt{y})$

3 $\quad \mathtt{cor}[\mathtt{y}'] \leftarrow \mathtt{x}'$

Union$_0$



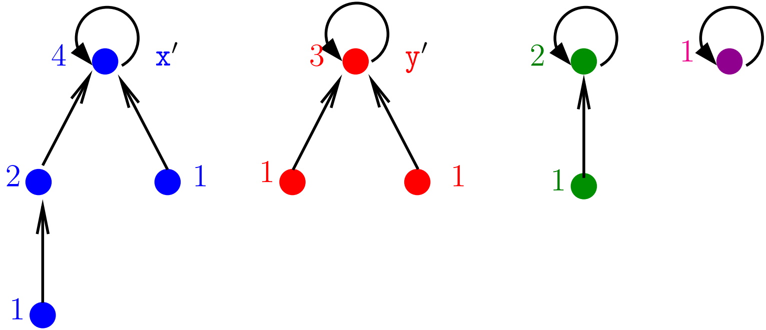
UNION$_0$ (x, y)

1 x′ ← FINDSET$_0$ (x)

2 y′ ← FINDSET$_0$ (y)

3 cor[y′] ← x′

Melhoramento 1: *union by rank*



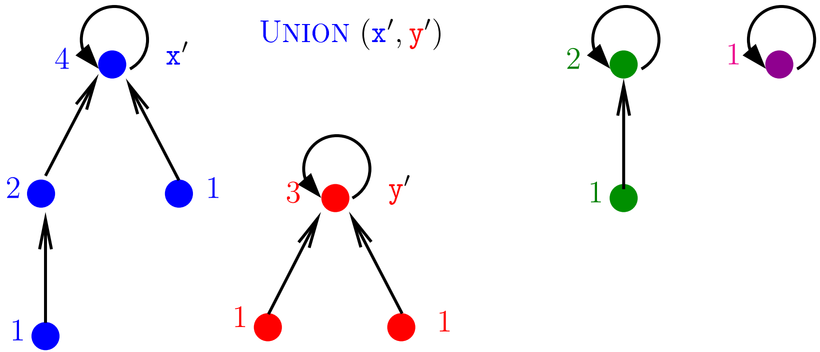
$\mathtt{sz}[\mathtt{x}]$ = número de nós na árvore com raiz $\mathtt{x}$

MAKESET $(\mathtt{x})$

1 $\quad \mathtt{cor}[\mathtt{x}] \leftarrow \mathtt{x}$

2 $\quad \mathtt{sz}[\mathtt{x}] \leftarrow 1$

Melhoramento 1: *union by rank*

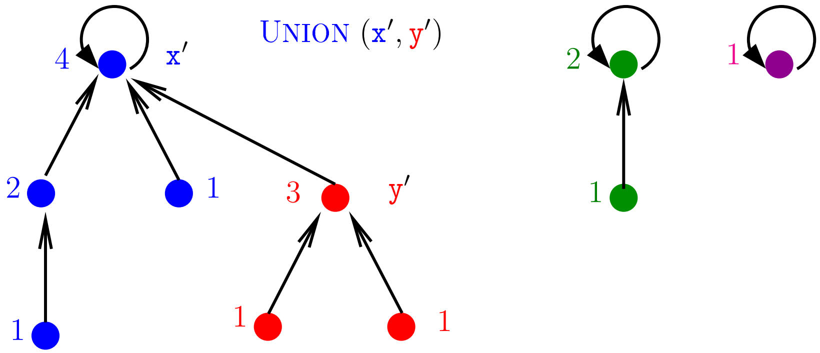


$sz[x]$ = número de nós na árvore com raiz x

MakeSet (x)

```
cor[x] ← x
sz[x] ← 1
```

Melhoramento 1: *union by rank*



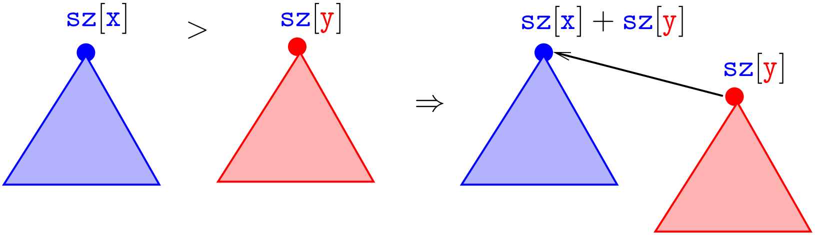
$sz[x]$ = número de nós na árvore com raiz x

MAKESET (x)

$cor[x] \leftarrow x$

$sz[x] \leftarrow 1$

Melhoramento 1: *union by rank*



Melhoramento 1: *union by rank*

UNION (x, y) ▷ com "union by rank"

$x' \leftarrow$ FINDSET (x)

$y' \leftarrow$ FINDSET (y) ▷ supõe que $x' \neq y'$

se $\mathtt{sz}[x'] > \mathtt{sz}[y']$

  então $\mathtt{cor}[y'] \leftarrow x'$

    $\mathtt{sz}[x'] = \mathtt{sz}[x'] + \mathtt{sz}[y']$

  senão $\mathtt{cor}[x'] \leftarrow y'$

    $\mathtt{sz}[y'] \leftarrow \mathtt{sz}[y'] + \mathtt{sz}[x']$

Consumo de tempo

Se conjuntos disjuntos são representados através de disjoint-set forest com *union by rank*, então uma seqüência de m operações MAKESET, UNION e FINDSET, sendo que n são MAKESET, consome tempo $O(m \lg n)$.

UFinit e UFfind

```
static Vertex cor[maxV];
static int sz[maxV];
void UFinit (int N) {
    Vertex v;
    for (v = 0; v < N; v++) {
        cor[v] = v;
        sz[v] = 1;
    }
}
int UFfind (Vertex v, Vertex w) {
    return (find(v) == find(w));
}
```

find

```
static Vertex find(Vertex v) {
    while (v != cor[v])
        v = cor[v];
    return v;
}
```

UFunion

```
void UFunion (Vertex v0, Vertex w0) {
    Vertex v = find(v0), w = find(w0);
    if (v == w) return;
    if (sz[v] < sz[w]) {
        cor[v] = w;
        sz[w] += sz[v];
    }
    else {
        cor[w] = v;
        sz[v] += sz[w];
    }
}
```

Consumo de tempo

Graças à maneira com duas *union-find trees* são unidas por `UFunion`, a altura de cada union-find tree é limitada por $\lg \texttt{V}$.

Assim, `UFfind` e `UFunion` consomem tempo $O(\lg \texttt{V})$.

Podemos supor que a função `sort` consome tempo proporcional a $\Theta(\texttt{E} \lg \texttt{E})$.

O restante do código de `GRAPHmstK` consome tempo proporcional a $O(\texttt{E} \lg \texttt{V})$.

Conclusão

O consumo de tempo da função `GRAPHmstK` é $O(E \lg V)$.

Algoritmos

função	consumo de tempo	observação
bruteforcePrim	$O(V^3)$	alg. de Prim
GRAPHmstP1	$O(V^2)$	grafos densos matriz adjacência
GRAPHmstP1	$O(E \lg V)$	grafos esparços listas de adjacência
bruteforceKruskal	$O(V^3)$	alg. de Kruskal
GRAPHmstK	$O(E \lg V)$	alg. de Kruskal *disjoint-set forest*

Os algoritmos funcionam para arestas com **custos quaisquer**.

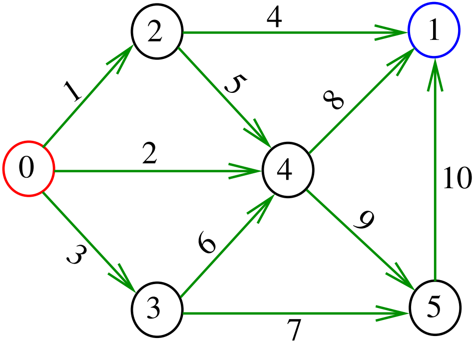
AULA 23

Fluxos em redes

S 22.1

Fluxos em arcos

Seja f uma função dos arcos de um digrafio G em $\mathbb{Z}_{\geq}$.
Diremos o valor de f num arco é o **fluxo no arco**.
Exemplo: o fluxo no arco 2-4 é 5

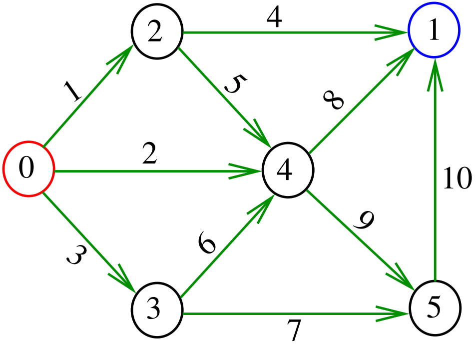


Influxos e efluxos

O **influxo** em v (= *inflow into* v) é a soma dos fluxos nos arcos que entram em v.

O **efluxo** de v (= *outflow from* v) é a soma dos fluxos nos arcos que saem de v.

Exemplo: em 4 o influxo é 13 e o efluxo é 17



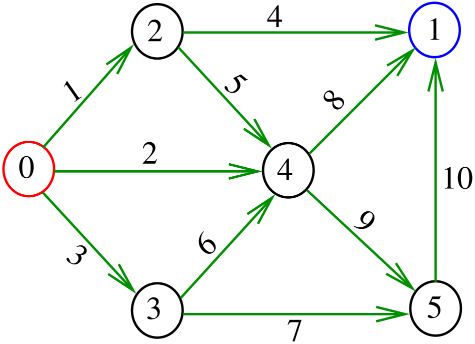
Saldos

O **saldo** em v é a diferença

$$\texttt{ef}(\texttt{v}) - \texttt{inf}(\texttt{v})$$

entre o efluxo de v e o influxo em v.

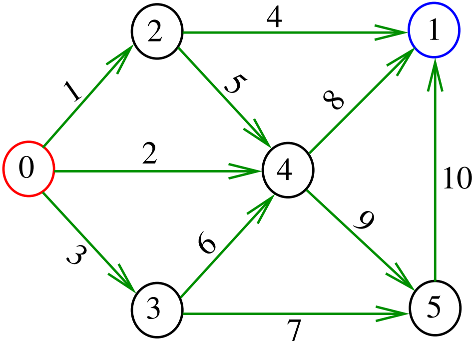
Exemplo: o saldo do vértice 4 é 17-13=4



Fluxos

Num digrafo com vértice inicial s e vértice final t, um **fluxo** (= *flow*) é uma função f que atribui valores em $\mathbb{Z}_{\geq}$ aos arcos de tal modo que o saldo em todo vértice distinto de s e t é nulo e em s é ≥ 0.

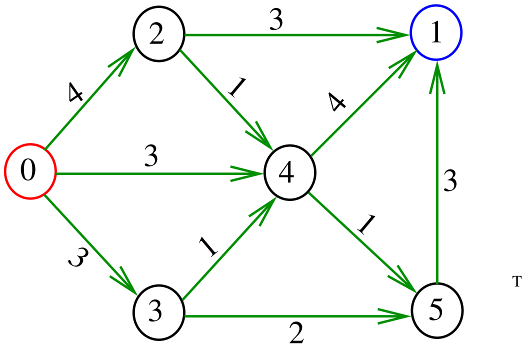
Exemplo: não é um fluxo



Fluxos

Num digrafo com vértice inicial `s` e vértice final `t`, um **fluxo** (= *flow*) é uma função `f` que atribui valores em $\mathbb{Z}_{\geq}$ aos arcos de tal modo que o saldo em todo vértice distinto de `s` e `t` é nulo e em `s` é ≥ 0.

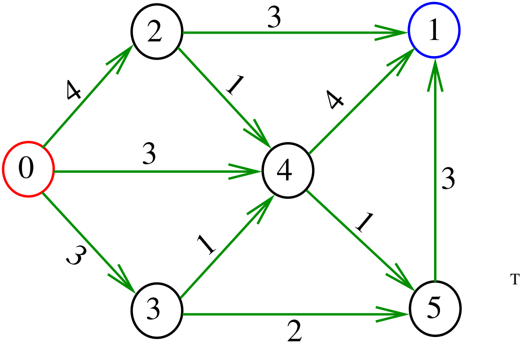
Exemplo: é um fluxo onde `s`=0 e `t`=1



Fontes e sorvedouros

Chamamos s de **fonte** e t de **sorvedouro**.

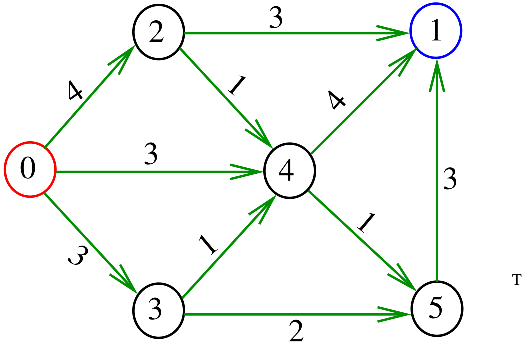
Exemplo: fluxo com fonte 0 e sorvedouro 1



Propriedade de Fluxos

Para qualquer fluxo num digrafo com fonte s e sorvedouro t, o saldo em t é igual ao saldo em s com sinal trocado.

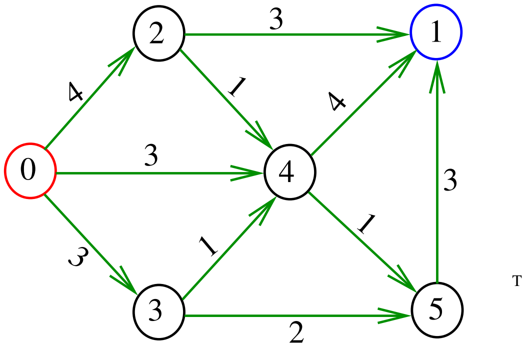
Exemplo: saldo em $0 = 10 = -10 =$ saldo em 1



Intensidade de fluxos

A **intensidade** de um fluxo `f` é o saldo de `f` em `s`.
Em geral (mas nem sempre) o influxo em `s` é nulo e o efluxo de `t` é nulo.

Exemplo: fluxo de intensidade 10



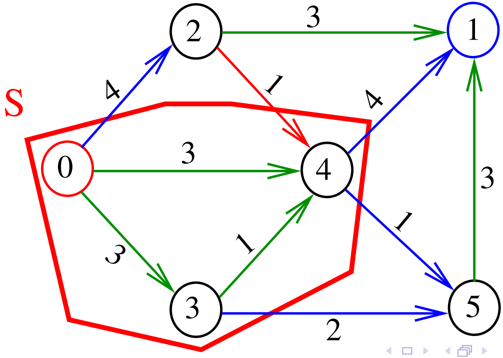
Saldo de fluxo num conjunto de vértices

Dado um conjunto S que contém s mas não contém t, o saldo em S é a diferença

$$\mathtt{ef}(\mathtt{S}) - \mathtt{inf}(\mathtt{S}),$$

entre o efluxo de S e o influxo em S

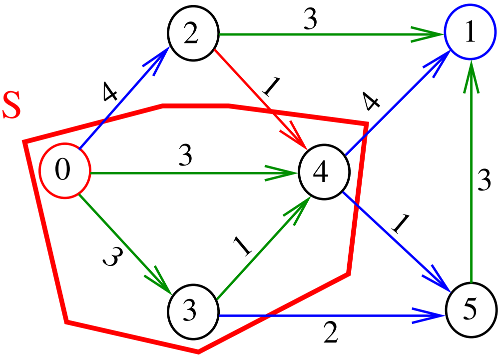
Exemplo: o saldo de S é $4 + 4 + 1 + 2 - 1 = 10$



Propriedade do Saldos

Para qualquer fluxo num digrafo com vértice inicial s e vértice final t e para qualquer conjunto S que contém s mas não contém t, o saldo em S é igual ao saldo em s.

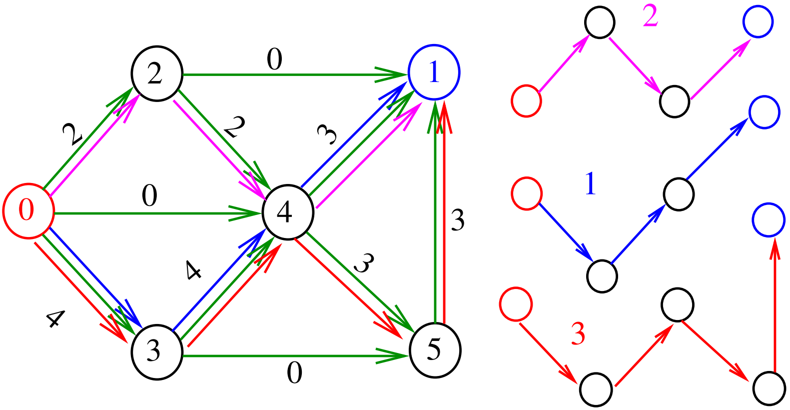
Exemplo: o saldo de S é $4 + 4 + 1 + 2 - 1 = 10$



Fluxos versus coleção de caminhos

Fluxos podem ser representados por caminhos de s a t. A soma das quantidades de fluxo conduzidas por cada caminho é igual à intensidade do fluxo.

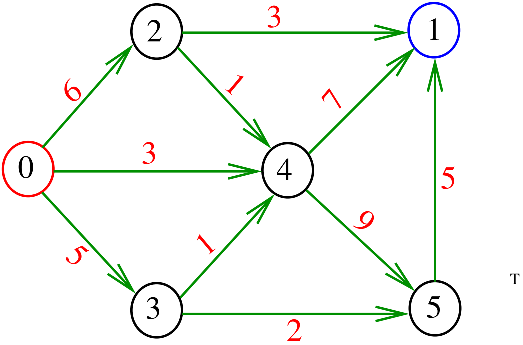
Exemplo:



Redes capacitadas

Uma **rede capacitada** é um digrafo com vértice inicial e vértice final em que a cada um arcos está associado um número em $\mathbb{Z}_{\geq}$ que chamaremos **capacidade do arco**.

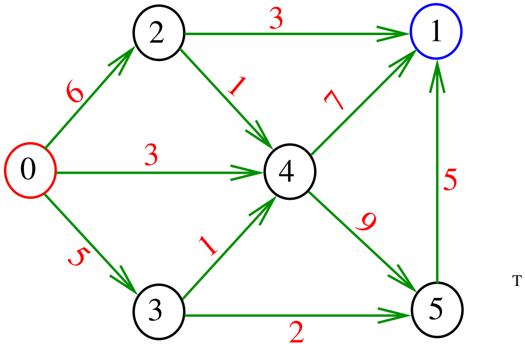
Exemplo:



Problema do fluxo máximo

Problema. Dada uma rede capacitada, encontrar um **fluxo de intensidade máxima** dentre os que respeitam as capacidades dos arcos.

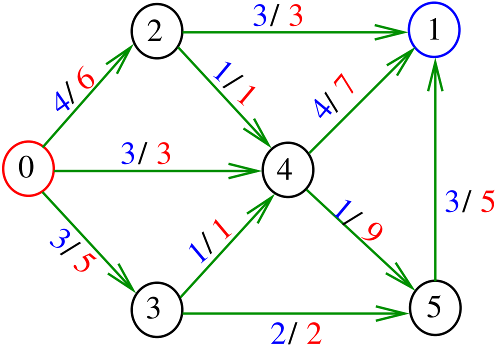
Exemplo: rede capacitada



Problema do fluxo máximo

Problema. Dada uma rede capacitada, encontrar um **fluxo de intensidade máxima** dentre os que respeitam as capacidades dos arcos.

Exemplo: fluxo que respeita as capacidades



Fluxo máximo (problema primal)

Podemos supor que a rede possui um arco $\mathtt{b}$ de $\mathtt{t}$ a $\mathtt{s}$ de capacidade $+\infty$.
O problema do fluxo máximo é equivalente ao seguinte programa linear, que chamamos de **primal**: encontrar um vetor $\mathtt{x}$ indexado por $\mathtt{A}$ que

$$
\begin{array}{rrcll}
\text{maximize} & \mathtt{x}(\mathtt{b}) & & & \\
\text{sob as restrições} & \mathtt{ef}(\mathtt{v}) - \mathtt{inf}(\mathtt{v}) & = & 0 & \forall\ \mathtt{v}, \\
 & \mathtt{x}(\mathtt{a}) & \leq & c(\mathtt{a}) & \forall\ \mathtt{a} \in \mathtt{A}, \\
 & \mathtt{x}(\mathtt{a}) & \geq & 0 & \forall\ \mathtt{a} \in \mathtt{A}.
\end{array}
$$