MAC0323 Algoritmos e Estruturas de Dados II

Edição 2020 – 2

AULA 24

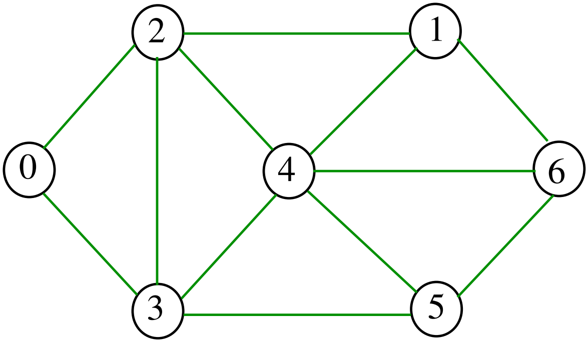
Árvores geradoras

Fonte: Pinterest

Subárvores

Uma **subárvore** de um grafo `G` é qualquer árvore `T` que seja subgrafo de `G`.

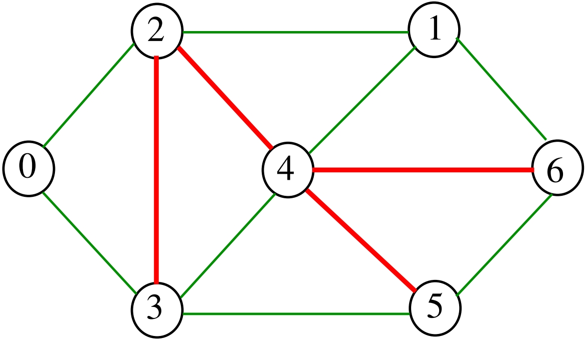
Exemplo:



Subárvores

Uma **subárvore** de um grafo G é qualquer árvore T que seja subgrafo de G.

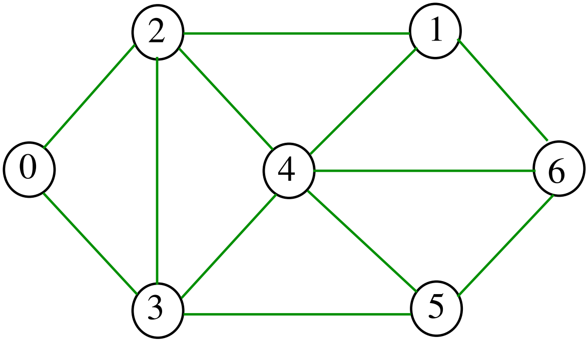
Exemplo: as arestas em vermelho formam uma subárvore



Árvores geradoras

Uma **árvore geradora** (= *spanning tree*) de um grafo é qualquer subárvore que contenha **todos** os vértices.

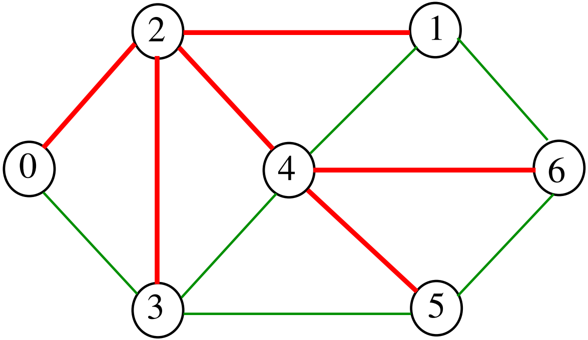
Exemplo:



Árvores geradoras

Uma **árvore geradora** (= *spanning tree*) de um grafo é qualquer subárvore que contenha **todos** os vértices.

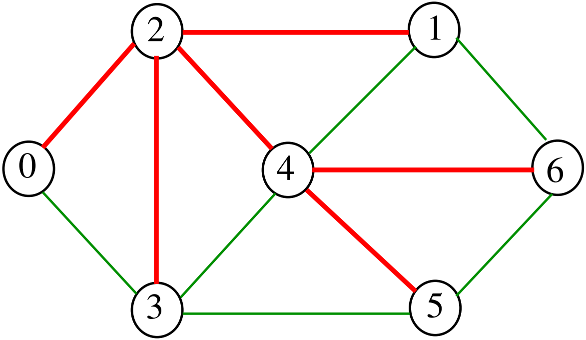
Exemplo: as arestas em vermelho formam uma árvore geradora



Árvores geradoras

Somente grafos conexos têm árvores geradoras.
Todo grafo conexo tem uma árvore geradora.

Exemplo:



Algoritmos que calculam árvores geradoras

É fácil calcular uma árvore geradora de um grafo conexo:

- a busca em profundidade (`DFS`) e
- a busca em largura (`BFS`)

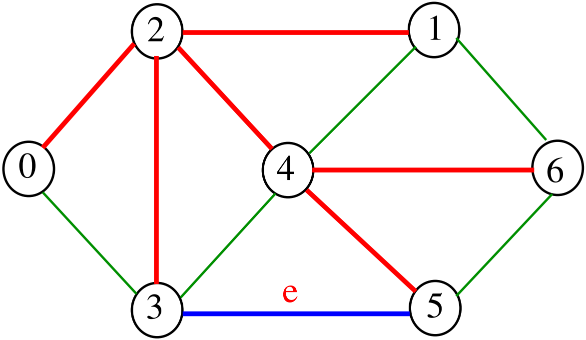
fazem isso.

Qualquer das duas buscas calcula uma arborescência que contém um dos arcos de cada aresta de uma árvore geradora do grafo.

Primeira propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G.
Para qualquer aresta e de G que não esteja em T,
T+e tem um **único ciclo** não-trivial.

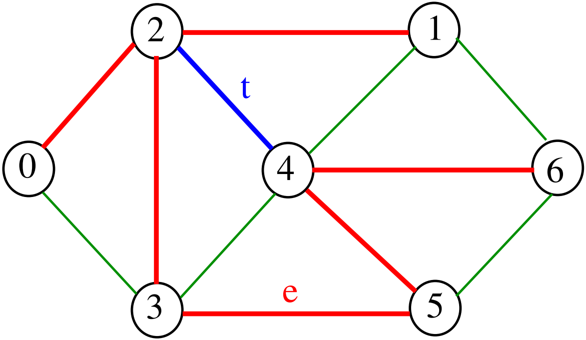
Exemplo: T+e



Primeira propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G.
Para qualquer aresta t desse ciclo,
T+e−t uma **árvore geradora**.

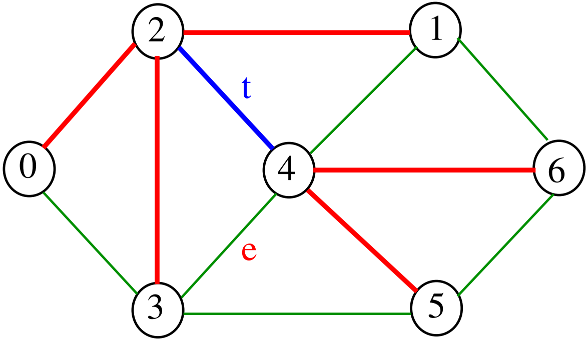
Exemplo: T+e−t



Segunda propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G.
Para qualquer aresta t de T e qualquer aresta e que atravesse o corte determinado por T−t, o grafo T−t+e é uma árvore geradora.

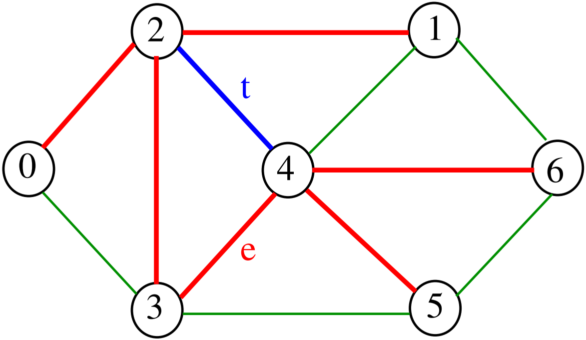
Exemplo: T−t



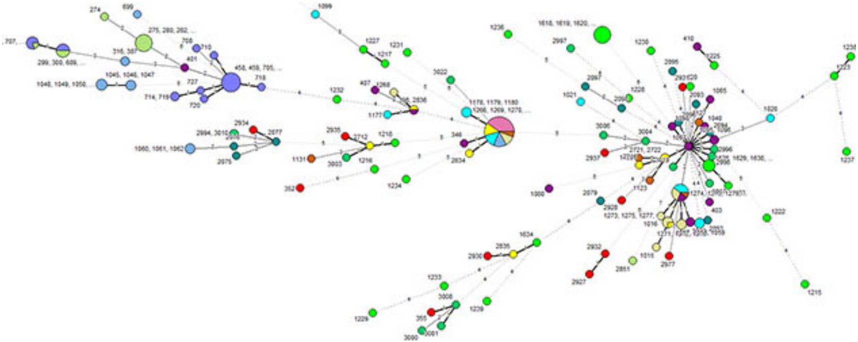
Segunda propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G.
Para qualquer aresta t de T e qualquer aresta e que atravesse o corte determinado por T−t, o grafo T−t+e é uma árvore geradora.

Exemplo: T−t+e



Árvores geradoras de custo mínimo

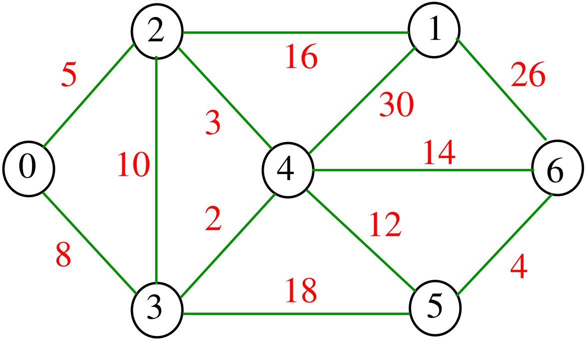


Fonte: Minimum Spanning Trees

Árvores geradoras mínimas

Uma **árvore geradora mínima** (= *minimum spanning tree*), ou MST, de um grafo com custos nas arestas é qualquer árvore geradora do grafo que tenha custo mínimo.

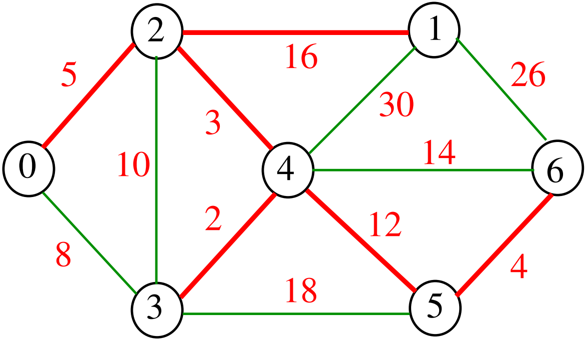
Exemplo: um grafo com custos nas arestas



Árvores geradoras mínimas

Uma **árvore geradora mínima** (= *minimum spanning tree*), ou `MST`, de um grafo com custos nas arestas é qualquer árvore geradora do grafo que tenha custo mínimo.

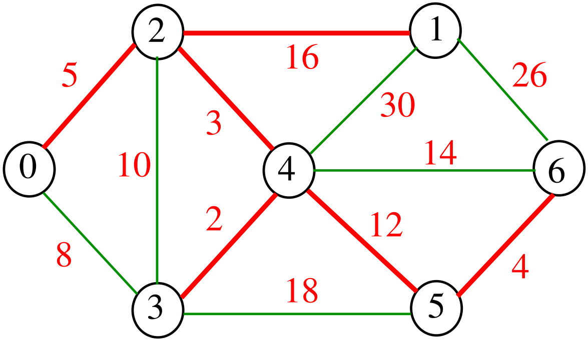
Exemplo: MST de custo 42



Problema MST

Problema: Encontrar uma `MST` de um grafo `G` com custos nas arestas.
O problema tem solução se e somente se `G` é conexo.

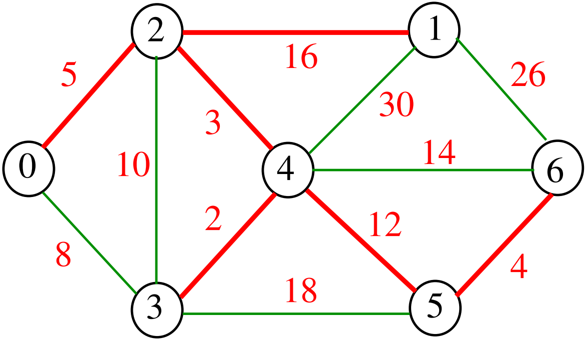
Exemplo: MST de custo 42



Propriedade dos ciclos

Condição de Otimalidade: Se `T` é uma MST então toda aresta `e` fora de `T` tem custo máximo dentre as arestas do único ciclo não-trivial em `T+e`.

Exemplo: MST de custo 42



Demonstração da recíproca

Seja `T` uma árvore geradora satisfazendo a condição de otimalidade.

Vamos mostrar que `T` é uma `MST`.

Demonstração da recíproca

Seja $\mathtt{T}$ uma árvore geradora satisfazendo a condição de otimalidade.

Vamos mostrar que $\mathtt{T}$ é uma $\mathtt{MST}$.

Seja $\mathtt{T}^*$ uma $\mathtt{MST}$ tal que o número de arestas comuns entre $\mathtt{T}$ e $\mathtt{T}^*$ seja máximo.

Se $\mathtt{T} = \mathtt{T}^*$, então não há o que demonstrar.

Demonstração da recíproca

Seja $\mathtt{T}$ uma árvore geradora satisfazendo a condição de otimalidade.

Vamos mostrar que $\mathtt{T}$ é uma $\mathtt{MST}$.

Seja $\mathtt{T}^*$ uma $\mathtt{MST}$ tal que o número de arestas comuns entre $\mathtt{T}$ e $\mathtt{T}^*$ seja máximo.

Se $\mathtt{T} = \mathtt{T}^*$, então não há o que demonstrar.

Suponha que $\mathtt{T} \neq \mathtt{T}^*$ e seja $\mathtt{e}^*$ uma aresta de custo mínimo dentre as arestas que estão em $\mathtt{T}^*$ mas não estão em $\mathtt{T}$.

Demonstração da recíproca

Seja $\mathtt{T}$ uma árvore geradora satisfazendo a condição de otimalidade.

Vamos mostrar que $\mathtt{T}$ é uma $\mathtt{MST}$.

Seja $\mathtt{T}^*$ uma $\mathtt{MST}$ tal que o número de arestas comuns entre $\mathtt{T}$ e $\mathtt{T}^*$ seja máximo.

Se $\mathtt{T} = \mathtt{T}^*$, então não há o que demonstrar.

Suponha que $\mathtt{T} \neq \mathtt{T}^*$ e seja $\mathtt{e}^*$ uma aresta de custo mínimo dentre as arestas que estão em $\mathtt{T}^*$ mas não estão em $\mathtt{T}$.

Seja $\mathtt{e}$ uma aresta qualquer que **não está** em $\mathtt{T}^*$ mas **está** no único ciclo em $\mathtt{T}+\mathtt{e}^*$.

Continuação

Logo, $\texttt{custo}(\texttt{e}) \leq \texttt{custo}(\texttt{e}^*)$.

Continuação

Logo, $\texttt{custo}(\texttt{e}) \leq \texttt{custo}(\texttt{e}^*)$.

Seja $\texttt{f}^*$ uma aresta qualquer que **não está** em $\texttt{T}$ mas **está** no único ciclo em $\texttt{T}^*+\texttt{e}$.

Como $\texttt{T}^*$ é uma $\texttt{MST}$ então $\texttt{custo}(\texttt{f}^*) \leq \texttt{custo}(\texttt{e})$.

Continuação

Logo, $\texttt{custo}(\texttt{e}) \leq \texttt{custo}(\texttt{e}^*)$.

Seja $\texttt{f}^*$ uma aresta qualquer que **não está** em $\texttt{T}$ mas **está** no único ciclo em $\texttt{T}^*+\texttt{e}$.

Como $\texttt{T}^*$ é uma $\texttt{MST}$ então $\texttt{custo}(\texttt{f}^*) \leq \texttt{custo}(\texttt{e})$.
Se $\texttt{custo}(\texttt{f}^*) = \texttt{custo}(\texttt{e})$, então
$\texttt{T}^*-\texttt{f}^*+\texttt{e}$ é uma $\texttt{MST}$ que contradiz a escolha de $\texttt{T}^*$.
Logo, $\texttt{custo}(\texttt{f}^*) < \texttt{custo}(\texttt{e})$.

Continuação

Logo, $\texttt{custo}(\texttt{e}) \leq \texttt{custo}(\texttt{e}^*)$.

Seja $\texttt{f}^*$ uma aresta qualquer que **não está** em $\texttt{T}$ mas **está** no único ciclo em $\texttt{T}^*+\texttt{e}$.

Como $\texttt{T}^*$ é uma $\texttt{MST}$ então $\texttt{custo}(\texttt{f}^*) \leq \texttt{custo}(\texttt{e})$.
Se $\texttt{custo}(\texttt{f}^*) = \texttt{custo}(\texttt{e})$, então
$\texttt{T}^*-\texttt{f}^*+\texttt{e}$ é uma $\texttt{MST}$ que contradiz a escolha de $\texttt{T}^*$.
Logo, $\texttt{custo}(\texttt{f}^*) < \texttt{custo}(\texttt{e})$.
Finalmente, concluímos que

$$\texttt{custo}(\texttt{f}^*) < \texttt{custo}(\texttt{e}) \leq \texttt{custo}(\texttt{e}^*)$$

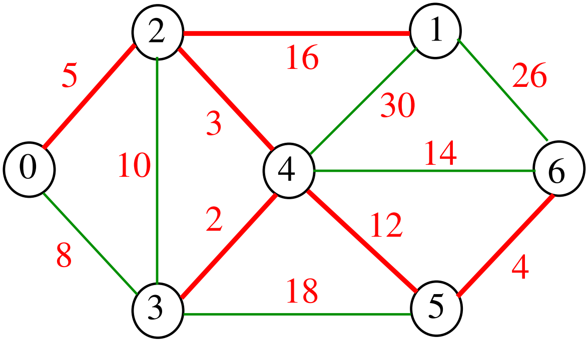
o que contradiz a nossa escolha de $\texttt{e}^*$.

Portanto, $\texttt{T} = \texttt{T}^*$ e $\texttt{T}$ é uma MST.

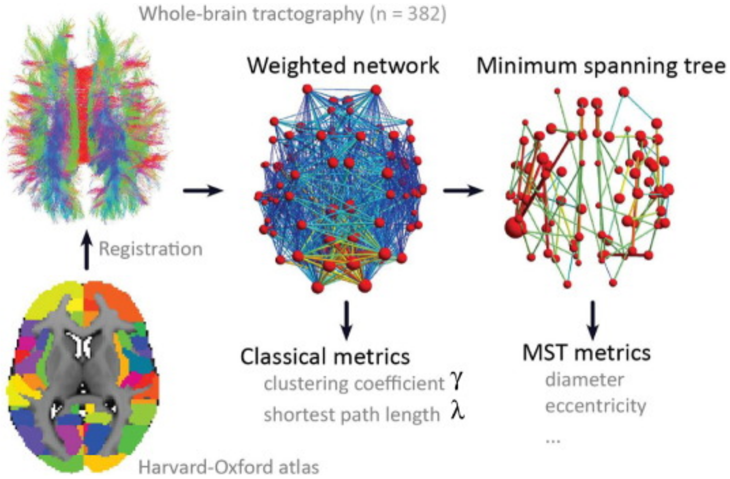
Propriedade dos cortes

Condição de Otimalidade: Se `T` é uma `MST` então cada aresta `t` de `T` é uma aresta mínima dentre as que atravessam o corte determinado por `T-t`.

Exemplo: `MST` de custo `42`



Algoritmo de Prim

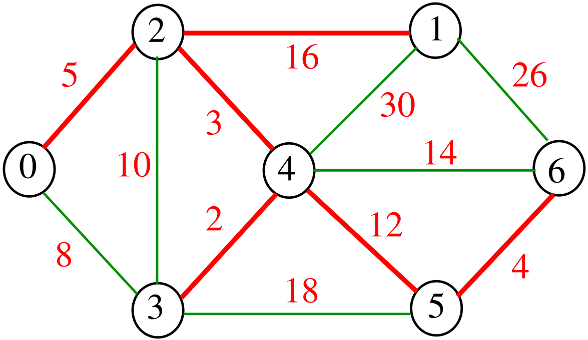


Fonte: Aging alterations in whole-brain networks ...mapped with the minimum spanning tree ...

Problema MST

Problema: Encontrar uma MST de um grafo G com custos nas arestas.
O problema tem solução se e somente se G é conexo.

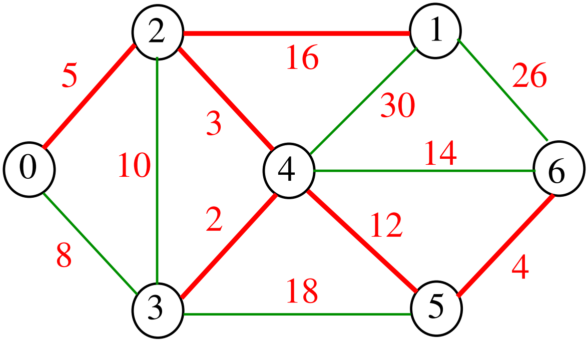
Exemplo: MST de custo 42



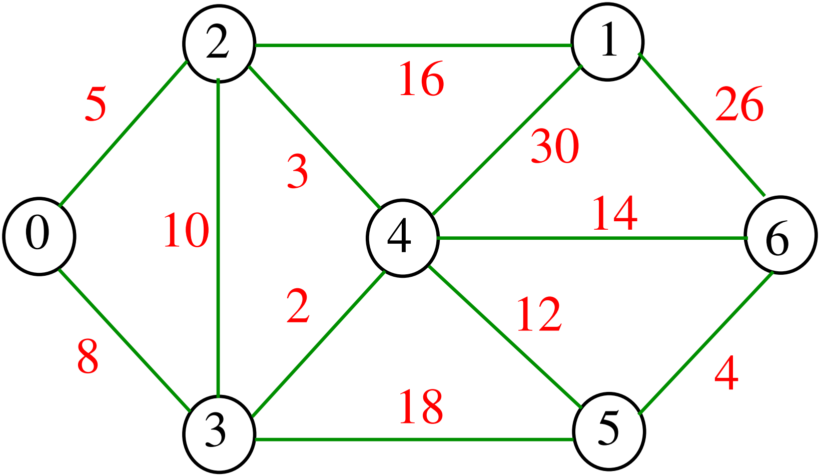
Propriedade dos cortes

Condição de Otimalidade: Se T é uma MST então cada aresta t de T é uma aresta mínima dentre as que atravessam o corte determinado por T−t.

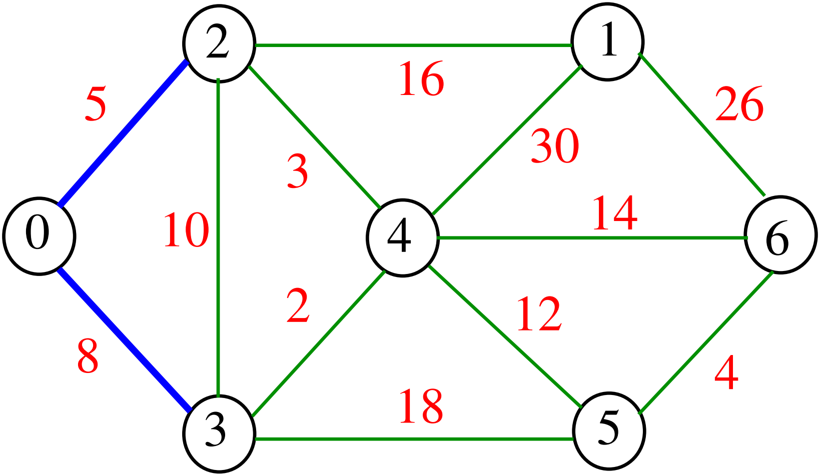
Exemplo: MST de custo 42



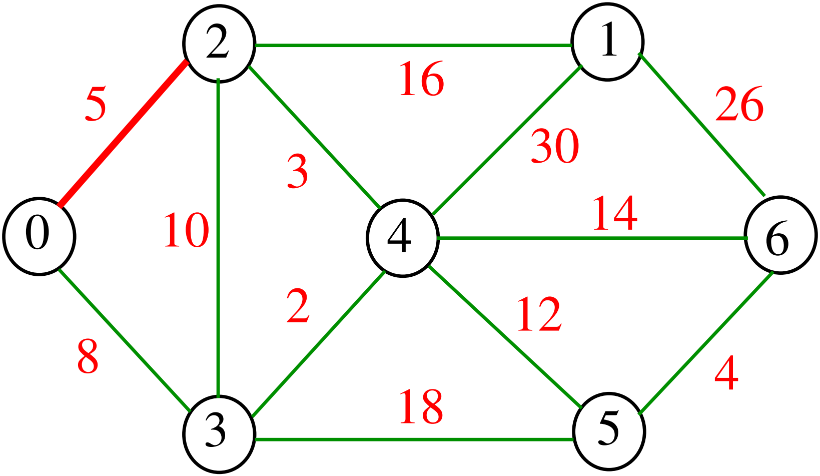
Simulação



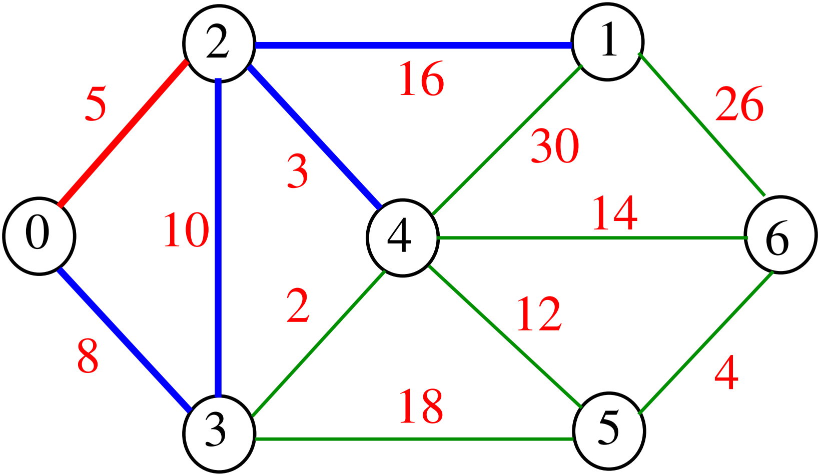
Simulação



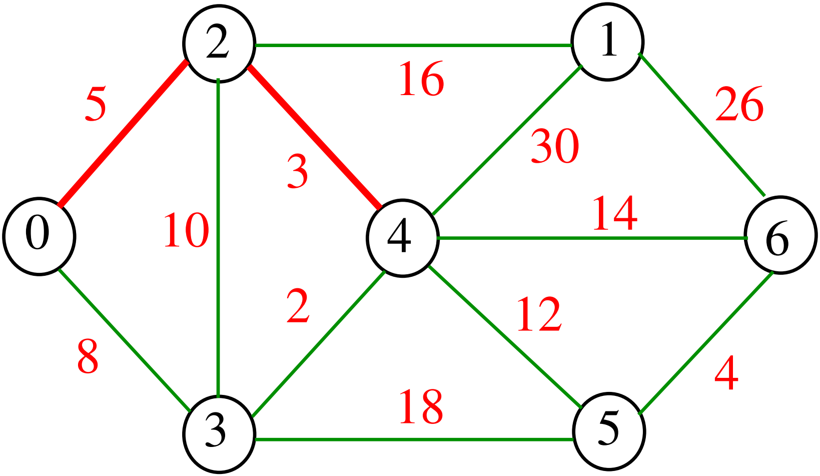
Simulação



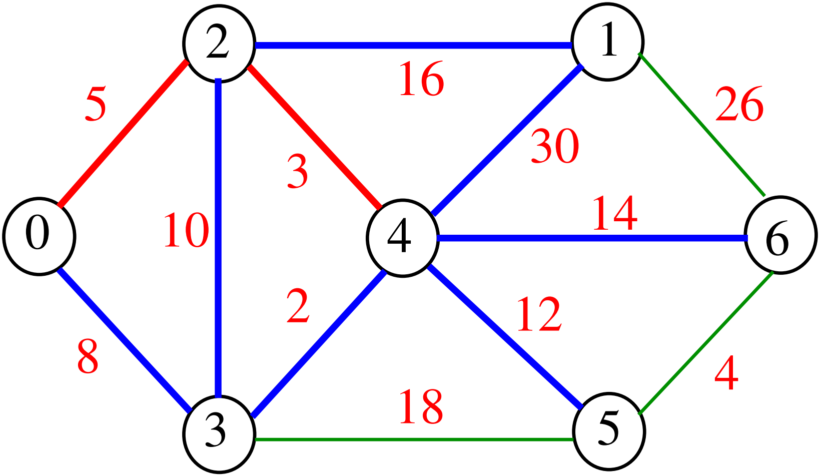
Simulação



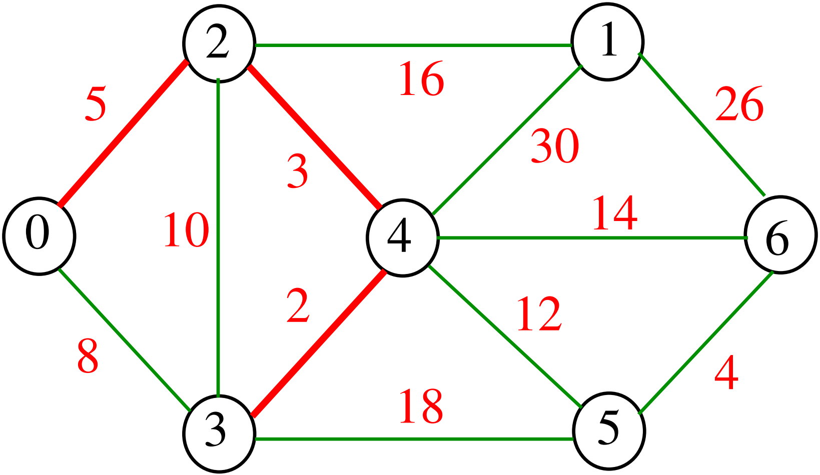
Simulação



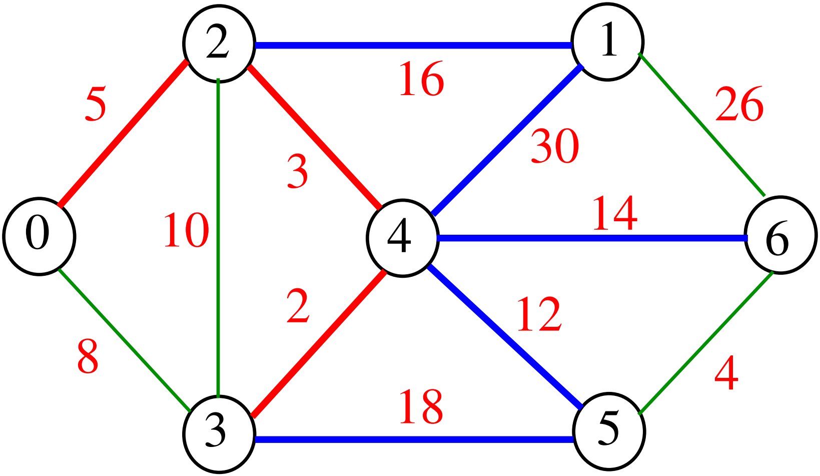
Simulação



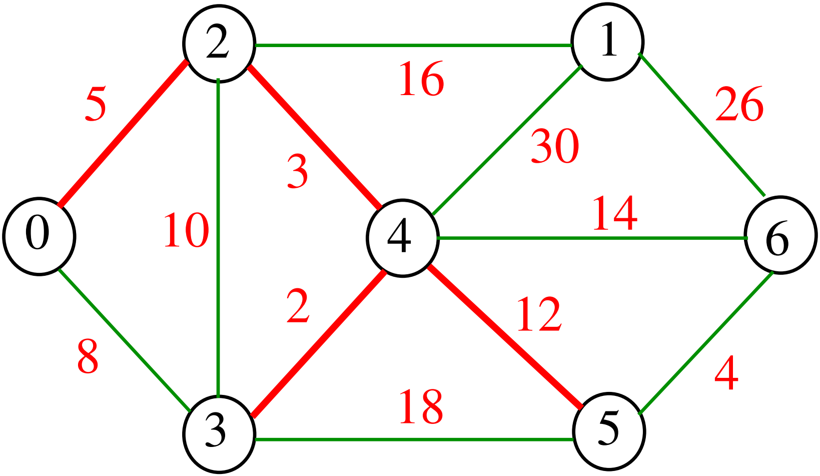
Simulação



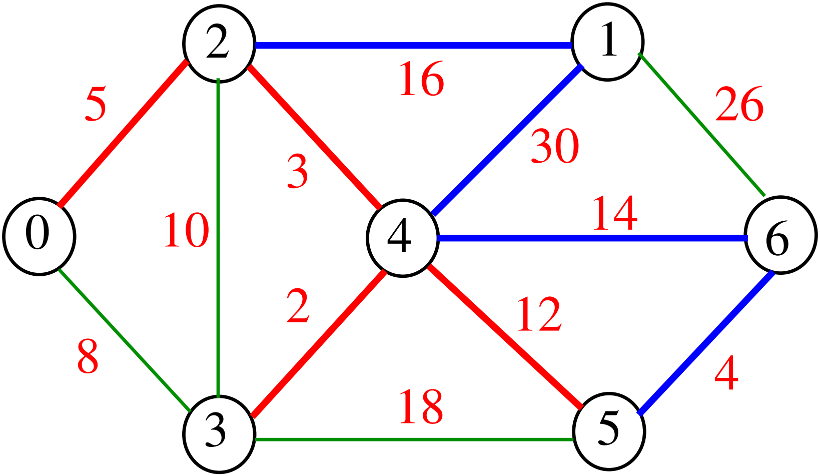
Simulação



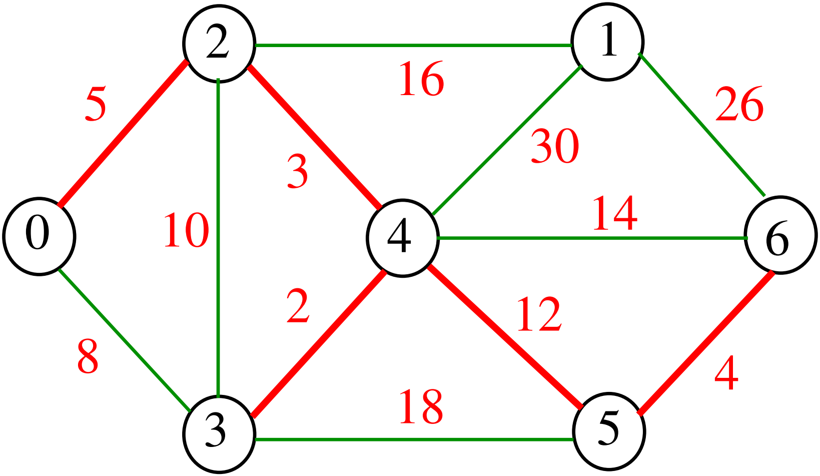
Simulação



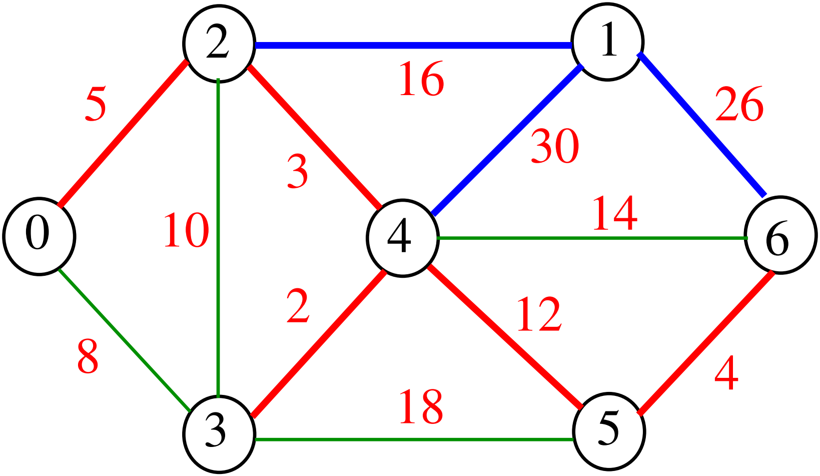
Simulação



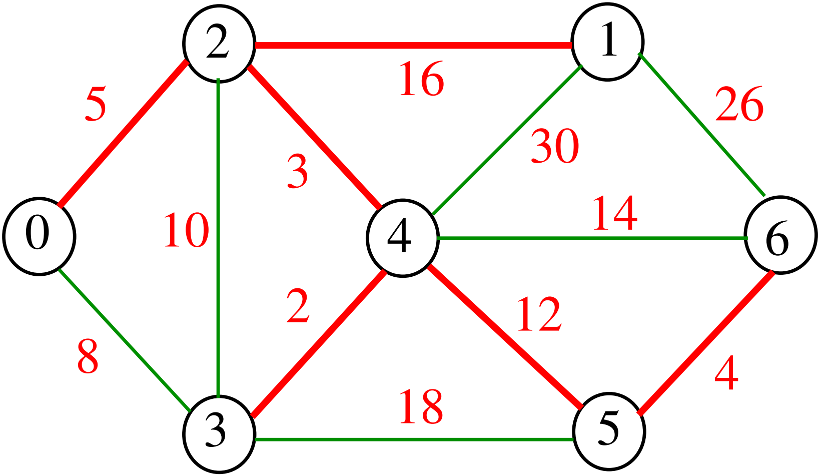
Simulação



Simulação



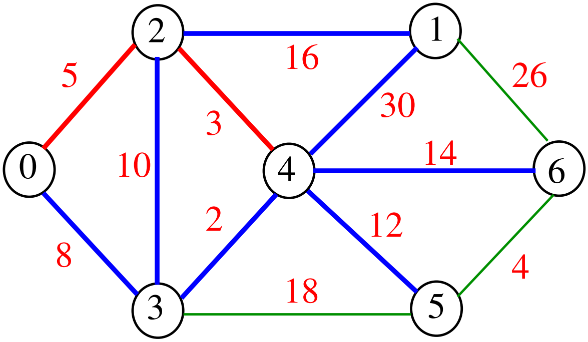
Simulação



Franja

A **franja** (= *fringe*) de uma subárvore não-geradora `T` é o conjunto de todas as arestas que têm uma ponta em `T` e outra ponta fora.

Exemplo: As arestas em azul formam a franja de `T`



Algoritmo de Prim

O algoritmo de Prim é iterativo.

Cada iteração começa com uma subárvore `T` de `G`.

No início da primeira iteração,
`T` é uma árvore com apenas um vértice.

Algoritmo de Prim

O algoritmo de Prim é iterativo.

Cada iteração começa com uma subárvore T de G.

No início da primeira iteração,
T é uma árvore com apenas um vértice.

Cada iteração consiste em:

Caso 1: franja de T é vazia
Devolva T e pare.

Algoritmo de Prim

O algoritmo de Prim é iterativo.

Cada iteração começa com uma subárvore `T` de `G`.

No início da primeira iteração,
`T` é uma árvore com apenas um vértice.

Cada iteração consiste em:

- Caso 1: franja de `T` é vazia
  Devolva `T` e pare.
- Caso 2: franja de `T` não é vazia
  Seja `e` uma aresta de custo mínimo
  na franja de `T`. Comece nova iteração
  com `T`+`e` no papel de `T`.

Relação invariante chave

No início de cada iteração vale que

existe uma MST *que contém as arestas em* T.

Demonstração. Se a relação vale no ínicio da última iteração então é evidente que, se o grafo é conexo, o algoritmo devolve uma MST.

Relação invariante chave

No início de cada iteração vale que

existe uma MST que contém as arestas em T.

Demonstração. Se a relação vale no ínicio da última iteração então é evidente que, se o grafo é conexo, o algoritmo devolve uma `MST`.

Vamos mostrar que se a relação vale no início de uma iteração que não seja a última, então ela vale no fim da iteração com `T+e` no papel de `T`.

A relação invariante certamente vale no início da primeira iteração.

Demonstração

Considere o início de uma iteração qualquer que não seja a última.

Seja `e` a aresta escolhida pela iteração no caso 2.

Pelo invariante, existe uma `MST` $\mathtt{T}^*$ que contém `T`.

Se `e` está em $\mathtt{T}^*$, então não há o que demonstrar.

Demonstração

Considere o início de uma iteração qualquer que não seja a última.

Seja $\mathtt{e}$ a aresta escolhida pela iteração no caso 2.

Pelo invariante, existe uma MST $\mathtt{T}^*$ que contém $\mathtt{T}$.

Se $\mathtt{e}$ está em $\mathtt{T}^*$, então não há o que demonstrar.

Suponha, portanto, que $\mathtt{e}$ não está em $\mathtt{T}^*$.

Seja $\mathtt{e}^*$ uma aresta que está no único ciclo em $\mathtt{T}^*+\mathtt{e}$ que está na franja de $\mathtt{T}$.

Pela escolha de $\mathtt{e}$, $\mathtt{custo}(\mathtt{e}) \leq \mathtt{custo}(\mathtt{e}^*)$.

Portanto, $\mathtt{T}^*-\mathtt{e}^*+\mathtt{e}$ é uma MST que contém $\mathtt{T}+\mathtt{e}$.

PrimMST: esqueleto

```
static double INFINITY;

/* marked[v] é true, v está na árvore */
static bool   *marked;
static double *distTo;
static int    *edgeTo;

double wmst;
Queue mst;

void PrimMST(EWGraph G) {}
static void prim(EWGraph G) {}
double weight() {}
```

PrimMST

Encontra árvore ou floresta geradora mínima de G.

```
void PrimMST(EWGraph G) {
   INFINITY = DBL_MAX;           /* double máximo */
   marked = mallocSafe(G->V*sizeof(bool));
   distTo = mallocSafe(G->V*sizeof(double));
   edgeTo = mallocSafe(G->V*sizeof(int));
   for (int v = 0; v < G->V; v++) {
      marked[s] = false;
      distTo[v] = INFINITY;
   for (int v = 0; v < G->V; v++)
      if (!marked[v]) prim(G, v);
}
```

`prim()`: inicializações

```
static void prim(EWGraph G, int s) {
   IndexMinPQ pq = IndexMinPQInit(G->V);
   int v;   Link p;
   distTo[s] = 0; /*s está na árvore */
   insert(pq, s, distTo[s]);

   /* aqui vem a iteração do próximo slide */
```

prim(): iteração

```
    while (!isEmpty(pq)) {
        v = delMin(pq);   marked[v] = true;
        for (p = G->adj[v]; p != NULL; p = p->next) {
            int w = p->vertex;
            if (marked[w]) continue;
            if (distTo[w] > p->weight) {
                edgeTo[w] = v;
                distTo[w] = p->weight;
                if (contains(pq, w))
                    decreaseKey(pq, w, p->weight);
                else insert(pq, w, p->weight);
            }
        }
    }
}
```

PrimMST: edges()

Para um grafo G conexo, retorna as arestas em uma árvore ou floresta geradora mínima.

```
Queue edges(EWGraph G) {
   if (mst != NULL) return mst;
   mst = queueInit();    /* fila de arestas */
   for (int v = 0; v < G->V; v++) {
      int w = edgeTo[v];
      if (w != NULL)
         enqueue(mst, v, w); /* aresta v-w */
   }
   return mst;
}
```

PrimMST: weight()

Considere que o vetor edgeTo armazena um apontador para a célula correspondente à aresta v-w e mst é uma fila destes apontadores.

Retorna o peso/custo de uma árvore ou floresta geradora mínima.

```
double weight() {
   double weight = 0.0;    Link p;
   while (!queueEmpty(mst)) {
      p = dequeue(mst)
      weight += p->weight;
   }
   return weight;
}
```

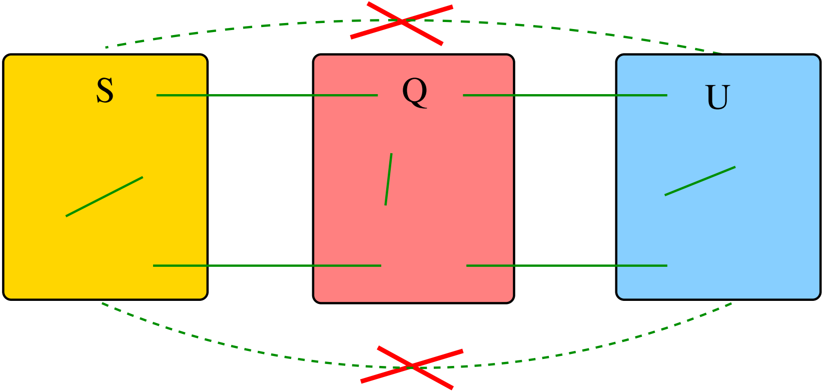
Relações invariantes

S = vértices examinados

Q = vértices visitados = vértices na fila

U = vértices ainda não visitados

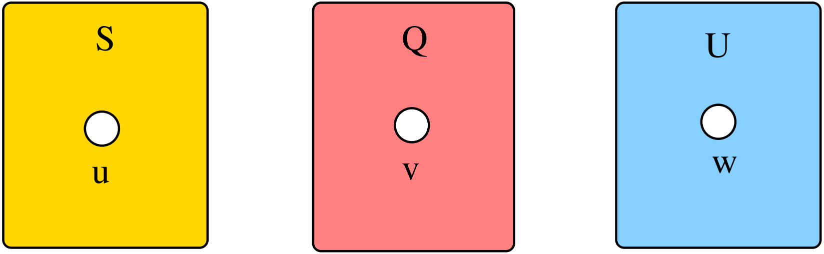
(i0) não existe **aresta** `v-w` com `v` em S e `w` em U.



Relações invariantes

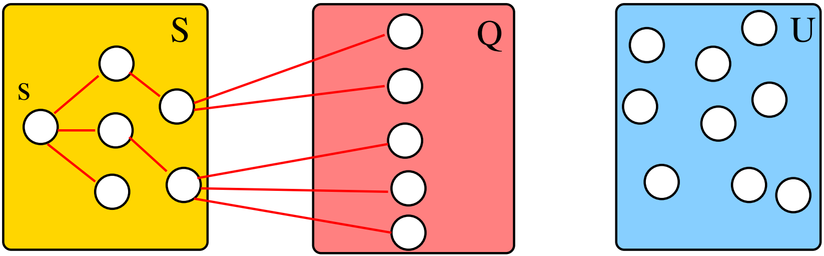
(i1) para cada `u` em `S`, `v` em `Q` e `w` em `U`

$$\texttt{marked}[\texttt{u}] == \texttt{true}$$
$$\texttt{distTo}[\texttt{v}] \leq \texttt{distTo}[\texttt{w}] == \texttt{INFINITY}.$$



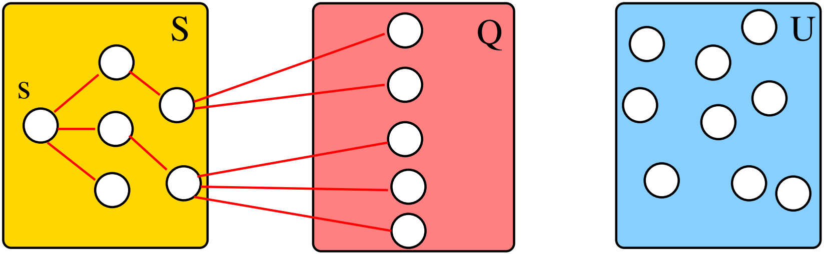
Relações invariantes

(i2) O vetor `edgeTo` restrito aos vértices de `S` determina uma `MST` "dos vértices em `S`".

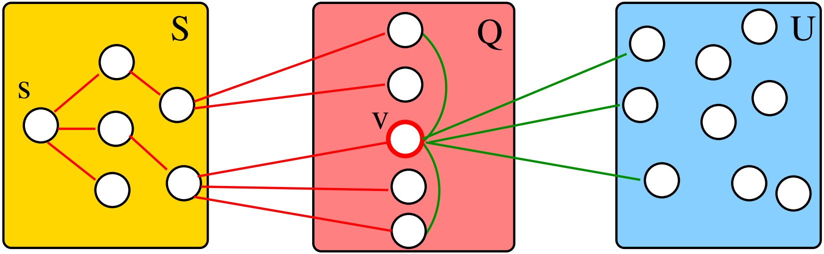


Relações invariantes

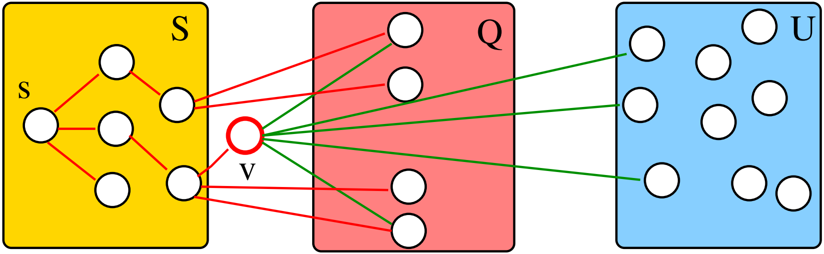
(i3) Para cada vértice `w` em `Q`, vale que `distTo[w]` é o menor custo de uma aresta com uma ponta em `S` e outra em `w`.



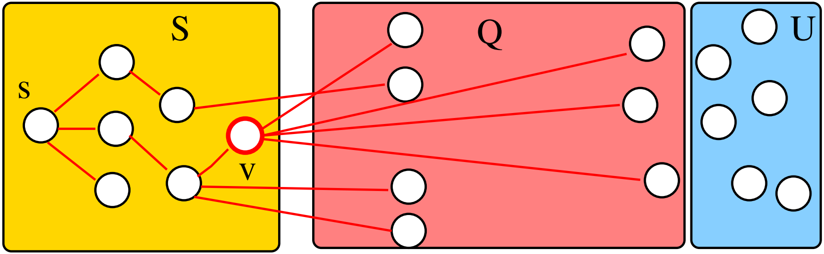
Iteração



Iteração



Iteração



Consumo de tempo

O consumo de tempo da função prim
é $O(V + E)$ mais o consumo de tempo de

$= 1$	execução de	IndexMinPQInit(),
$\leq$ V	execuções de	insert(),
$\leq$ V	execuções de	isEmpty(),
$\leq$ V	execuções de	delMin(), e
$\leq$ E	execuções de	contains(),
$\leq$ E	execuções de	decreaseKey().

Consumo de tempo MIN-HEAP

`IndexMinPQInit`	$\Theta(V)$
`isEmpty`	$\Theta(1)$
`insert`	$\Theta(\lg V)$
`delMin`	$O(\lg V)$
`decreaseKey`	$\Theta(\lg V)$
`contains`	$\Theta(1)$

Conclusão

O consumo de PrimMST é $O(E \lg V)$.