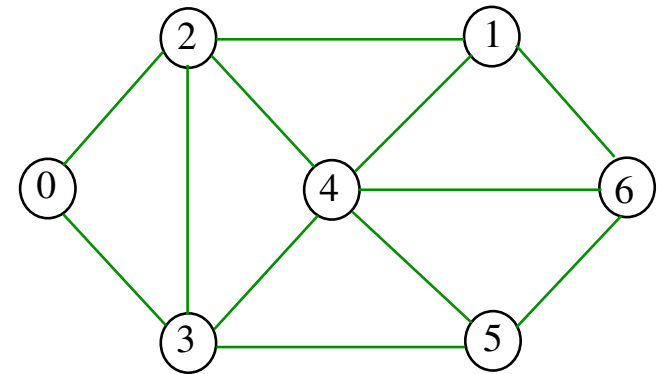


Árvores geradoras de grafos

Árvores geradoras

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

Exemplo:

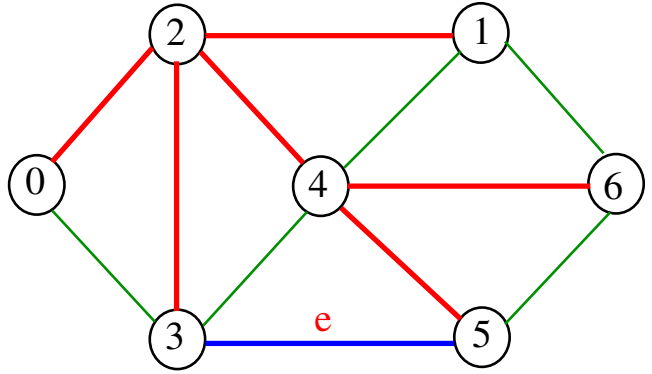


Exemplo: as arestas em **vermelho** formam uma árvore

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, o **ciclo fundamental** $C(T, e)$.

Exemplo: **T+e**

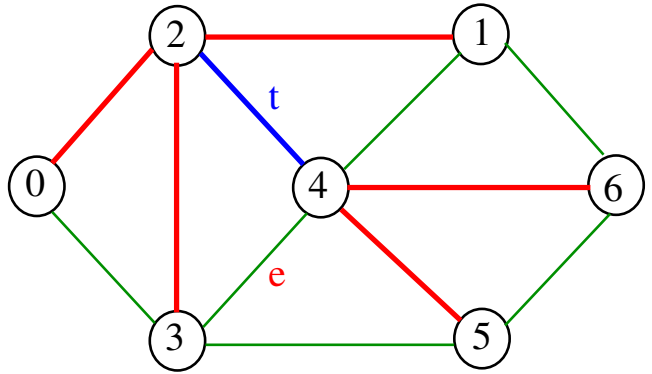


Segunda propriedade da troca de arestas

Seja **T** uma **árvore geradora** de um grafo **G**
 Para qualquer aresta **t** de **T**, **T-t** tem duas componentes.
 O corte em **G** que separa essas componentes é o **corte fundamental** $D(T, t)$.

Exemplo: **T-t**

$$\{0, 1, 2, 3\} \xrightarrow{D(T,t)} \{4, 5, 6\}$$



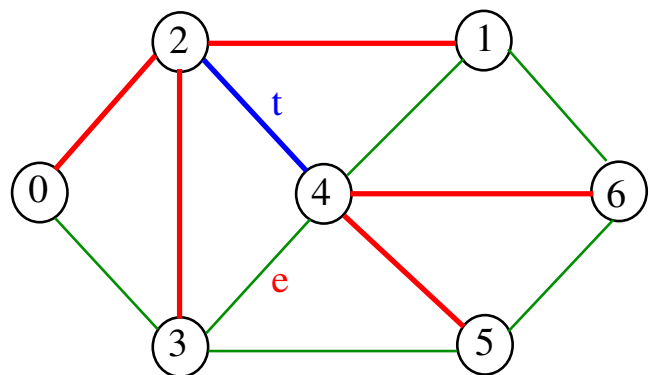
Segunda propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G

Para qualquer aresta t de T , se $e \in D(T, t)$ então

$T - t + e$ é uma árvore geradora.

Exemplo: $T-t$



Exemplo: $T-t+e$

Árvores geradoras de custo mínimo

S 20.1 e 20.2

Propriedade fundamental

Se T é árvore geradora de G , t é uma aresta de T e e é uma aresta fora de T , então

$$t \in C(T, e) \iff e \in D(T, t).$$

Dem: São equivalentes

- $t \in C(T, e)$
- t está no caminho em T que liga as pontas de e
- $T - t$ não tem caminho entre as pontas de e
- $e \in D(T, t)$.

5 / 1

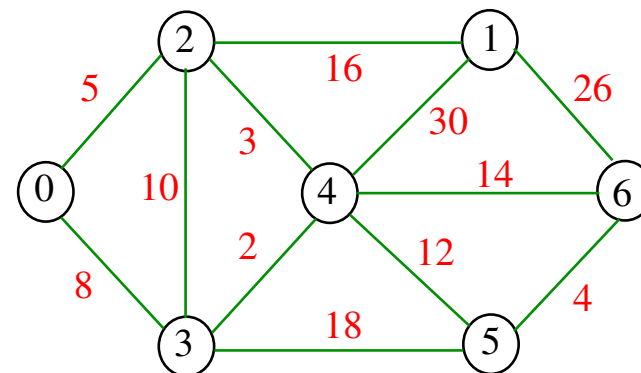
Algoritmos em Grafos — 1º sem 2014

6 / 1

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

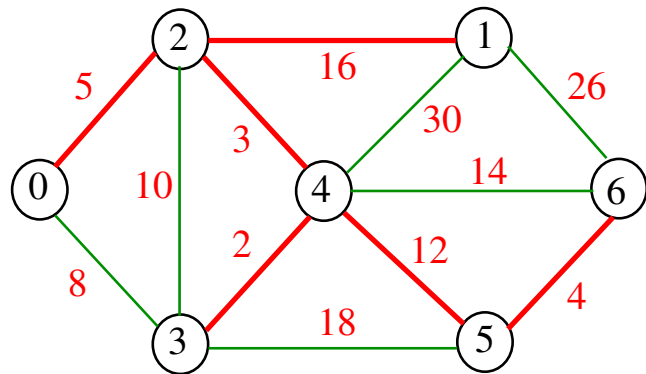


Problema MST

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

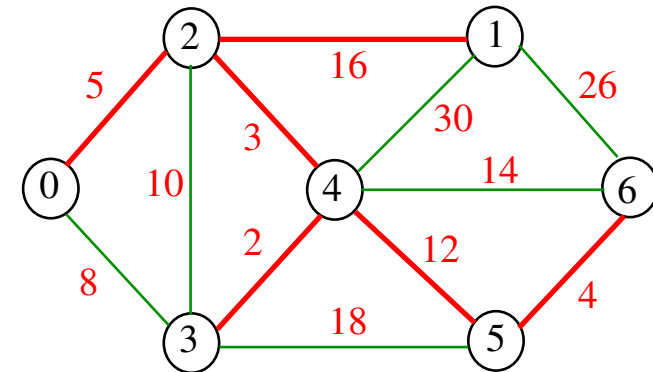
Exemplo: MST de custo 42



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 ciclo fundamental 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.

Seja M uma MST tal que o número de arestas comuns entre T e M seja **máximo**.

Se $T = M$ não há o que demonstrar.

Suponha que $T \neq M$ e seja e uma aresta de custo mínimo dentre as arestas que estão em M mas não estão em T .

Seja d uma aresta qualquer que **não está** em M mas **está** no ciclo fundamental $C(T, e)$.

Continuação

Logo, $\text{custo}(d) \leq \text{custo}(e)$ (1).

Seja f uma aresta qualquer em $C(M, d) - T$.

Como M é uma MST, $\text{custo}(f) \leq \text{custo}(d)$ (2).

Pela escolha de e , $\text{custo}(e) \leq \text{custo}(f)$ (3).

Juntando (1), (2) e (3), vem que

$$\text{custo}(d) = \text{custo}(f) = \text{custo}(e)$$

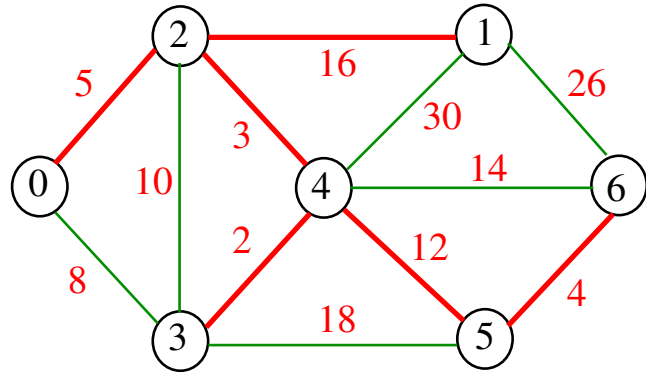
Mas então, $M - f + d$ é uma MST que tem o mesmo custo que M , logo é mínima. Por outro lado, tem uma aresta a mais em comum com T do que M . Isso contradiz a escolha de M .

Portanto, $T = M$, o que mostra que T é uma MST.

Propriedade dos cortes

Condição de Otimalidade: T é uma MST se e somente se cada aresta t de T é uma aresta mínima no corte fundamental T, t .

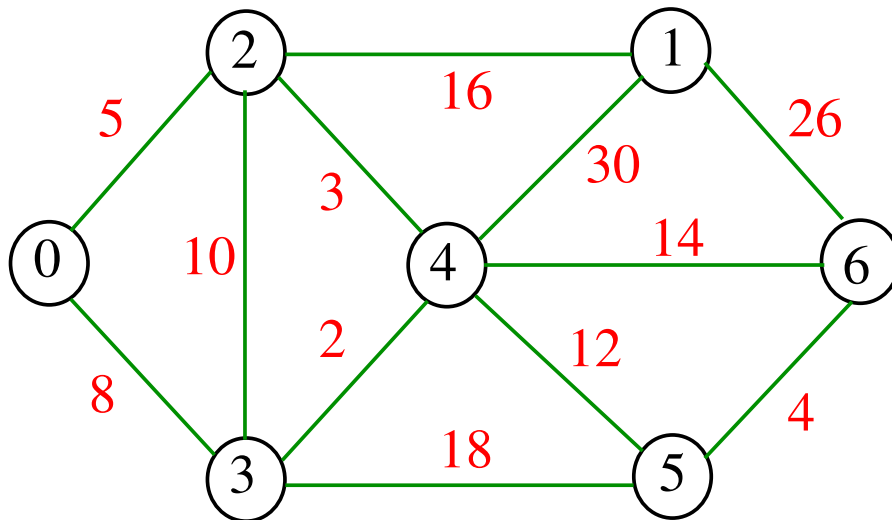
Exemplo: MST de custo 42



Algoritmo de Prim

S 20.3

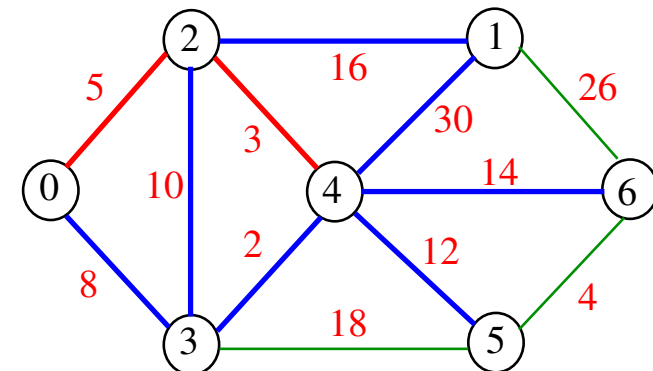
Simulação



Franja

A **franja** (= *fringe*) de uma subárvore 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 um árvore com apenas 1 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
Faça $T \leftarrow T + e$

Relação invariante chave

No início de cada iteração vale que

existe uma MST que contém as arestas em T .

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

Demonstração. 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. Pela relação invariante existe uma MST M que contém T .

Se e está em M , então não há o que demonstrar. Suponha, portanto, que e não está em M .

Seja t uma aresta que está $C(M, e)$ que está na franja de T . Pela escolha de e feita pelo algoritmo, $\text{custo}(e) \leq \text{custo}(t)$.

Portanto, $M - t + e$ é uma MST que contém $T+e$.

Implementações do algoritmo de Prim

S 20.3

Implementação grosseira

A função abaixo recebe um grafo G com custos nas arestas e calcula uma MST da componente que contém o vértice 0 .

```
void bruteforcePrim(Graph G, Vertex parnt[]){
0  Vertex v , w ;
1  for (v =0;v < G->V; v++) parnt[v] = -1;
3  parnt[0] = 0;
```

Implementação grosseira

```
4  while (1) {
5  double mincst = INFINITO;
6  Vertex v0, w0;
7  for (w = 0; w < G->V ; w++)
8      if (parnt[w] == -1)
9          for (v = 0; v < G->V ; v++)
10             if (parnt[v] != -1
11                 && mincst > G->adj[v][w])
12                 mincst = G->adj[v0 = v][w0 = w];
12  if (mincst == INFINITO) break;
13  parnt[w0] = v0;
14  }
}
```

Implementações eficientes

Uma idéia que leva a implementações mais eficientes do algoritmo de Prim é considerar:

Dada uma árvore não-geradora do grafo, o **custo de um vértice w** que está fora da árvore é o custo de uma aresta mínima dentre as que incidem em w e estão na franja da árvore.

Se nenhuma aresta da franja incide em w , o custo de w é INFINITO.

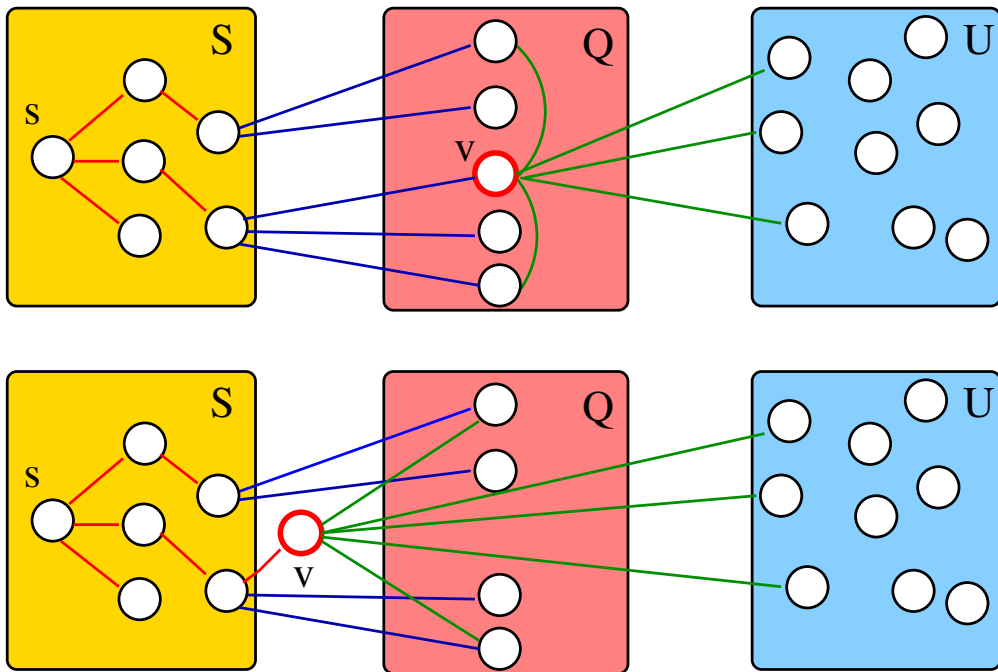
Implementações eficientes

Nas implementações que examinaremos, o custo do vértice w em relação à árvore é $cst[w]$.

Para cada vértice w fora da árvore, o vértice $fr[w]$ está na árvore e a aresta que liga w a $fr[w]$ tem custo $cst[w]$.

Cada iteração do algoritmo de Prim escolhe um vértice w fora da árvore e adota $fr[w]$ como valor de $parnt[w]$.

Iteração



Implementação eficiente para grafos densos

Recebe grafo G com custos nas arestas e calcula uma MST da componente de G que contém o vértice 0. A função armazena a MST no vetor `parnt`, tratando-a como uma arborescência com raiz 0. O grafo G é representado por sua matriz de adjacência.

```
void GRAPHmstP1 (Graph G, Vertex parnt[]) {
1  double cst[maxV]; Vertex v, w, fr[maxV];
2  for (v = 0; v < G->V; v++) {
3      parnt[v] = -1;
4      cst[v] = INFINITO;
5  }
6  v = 0; fr[v] = v; cst[v] = 0;
```

```
6  while (1) {
7  double mincst = INFINITO;
8  for (w = 0; w < G->V; w++)
9      if (parnt[w] == -1 && mincst > cst[w])
10         mincst = cst[v = w];
11  if (mincst == INFINITO) break;
12  parnt[v] = fr[v];
13  for (w = 0; w < G->V; w++)
14      if (parnt[w] == -1
15          && cst[w] > G->adj[v][w]) {
16          cst[w] = G->adj[v][w];
17          fr[w] = v;
18      }
19  }
```

Consumo de tempo

O consumo de tempo da função `GRAPHmstP1` é $O(V^2)$.

Este consumo de tempo é ótimo para **digrafos densos**.

Recordando Dijkstra para digrafos densos

```
#define INFINITO maxCST

void
DIGRAPHsptD1 (Digraph G, Vertex s,
              Vertex parnt[], double cst[]) {
1  Vertex w, w0, fr[maxV];
2  for (v = 0; v < G->V ; v++) {
3      parnt[v] = -1;
4      cst[v] = INFINITO;
5  }
6  fr[s] = s ;
7  cst[s] = 0;
```

```
8  while (1) {
9      double mincst = INFINITO;
10     for (w = 0; w < G->V ; w++)
11         if (parnt[w]==-1 && mincst>cst[w])
12             mincst = cst[v=w];
13     if (mincst == INFINITO) break;
14     parnt[v] = fr[v];
15     for (w = 0; w < G->V ; w++)
16         if (cst[w]>cst[v]+G->adj[v][w]){
17             cst[w] = cst[v]+G->adj[v][w];
18             fr[w] = v ;
19         }
20     }
21 }
```

Implementação para grafos esparsos

Recebe grafo G com custos nas arestas e calcula uma MST da componente de G que contém o vértice 0.

A função armazena a MST no vetor `parnt`, tratando-a como uma arborescência com raiz 0.

O grafo G é representado por *listas de adjacência*.

GRAPHmstP2

```
#define INFINITO maxCST
void GRAPHmstP2 (Graph G, Vertex parnt[]){
1  Vertex v , w , fr[maxV]; link p ;
2  for (v = 0; v < G->V ; v++) {
3      cst[v] = INFINITO;
4      parnt[v] = -1;
5  }
6  PQinit(G->V);
7  cst[0] = 0;
8  fr[0] = 0;
9  PQinsert(0);
```



```

9  while (!PQempty()) {
10 v = PQdelmin();
11 parnt[v] = fr[v];
12 for (p = G->adj[v]; p; p = p->next){
13     w = p->w ;
14     if (parnt[w] == -1){
15         if (cst[w] == INFINITO){
16             cst[w] = p->cst;
17             fr[w] = v ;
18             PQinsert(w);
19         }
20     }
21 }

```

```

19         else if (cst[w] > p->cst){
20             cst[w] = p->cst;
21             fr[w] = v ;
22             PQdec(w);
23         }
24     } /* if (parnt[w] ...*/
25 } /* for (p...*/
26 } /* while ...
27 }

```

Consumo de tempo

O consumo de tempo da função `GRAPHmstP2` implementada com um min-heap é $O(E \lg V)$.

Recordando Dijkstra para digrafos esparsos

```

#define INFINITO maxCST
void dijkstra(Digraph G, Vertex s ,
             Vertex parnt[], double cst[]);
{
1  Vertex v , w ; link p ;
2  for (v = 0; v < G->V ; v++) {
3      cst[v] = INFINITO;
4      parnt[v] = -1;
5  }
6  PQinit(G->V);
7  cst[s] = 0;
8  parnt[s] = s ;
9  PQinsert(s);

```

```

9  while (!PQempty()) {
10     v = PQdelmin();
11     for(p = G->adj[v]; p; p = p->next)
12         w = p ->w ;
12     if (cst[w]==INFINITO) {
13         cst[w]=cst[v]+p->cst;
14         parnt[w]=v ;
15         PQinsert(w);
    }

```

```

16     else
17         if(cst[w]>cst[v]+p->cst)
18             cst[w]=cst[v]+p->cst
19             parnt[w] = v ;
20             PQdec(w);
    }
21 PQfree();
    }

```