

## Árvores geradoras mínimas

## Árvores geradoras mínimas

1. Grafos com pesos nas arestas:  $G = (V, E)$ ,  $c: E \rightarrow \mathbb{R}$  (usualmente,  $c(e) \geq 0$  para toda  $e \in E$ )

## Árvores geradoras mínimas

1. Grafos com pesos nas arestas:  $G = (V, E)$ ,  $c: E \rightarrow \mathbb{R}$  (usualmente,  $c(e) \geq 0$  para toda  $e \in E$ )
2. Neste capítulo:  $G$  será sempre conexo.

## Árvores geradoras mínimas

1. Grafos com pesos nas arestas:  $G = (V, E)$ ,  $c: E \rightarrow \mathbb{R}$  (usualmente,  $c(e) \geq 0$  para toda  $e \in E$ )
2. Neste capítulo:  $G$  será sempre conexo.
3. Queremos: *Árvore geradora*  $T$  de  $G$  com  $c(E(T))$  mínimo (MST: 'minimum spanning tree')

## Árvores geradoras mínimas

1. Grafos com pesos nas arestas:  $G = (V, E)$ ,  $c: E \rightarrow \mathbb{R}$  (usualmente,  $c(e) \geq 0$  para toda  $e \in E$ )
2. Neste capítulo:  $G$  será sempre conexo.
3. Queremos: *Árvore geradora*  $T$  de  $G$  com  $c(E(T))$  mínimo (MST: 'minimum spanning tree')
4. Algoritmos?

Algoritmos?

## Algoritmos?

▷ Pela definição?

## Algoritmos?

- ▷ Pela definição?
- ▷ Algoritmo de Prim



## Algoritmos?

- ▷ Pela definição?
- ▷ Algoritmo de Prim
- ▷ Algoritmo de Kruskal

## Algoritmos?

- ▷ Pela definição?
- ▷ Algoritmo de Prim
- ▷ Algoritmo de Kruskal
- ▷ Algoritmo de Boruvka

Representação de  $(G, c)$  [ $G$  com a função custo  $c$ ]

Representação de  $(G, c)$  [ $G$  com a função custo  $c$ ]

Fácil!

## Representação de $(G, c)$ [ $G$ com a função custo $c$ ]

Fácil!

```
typedef struct { int v; int w; double wt; } Edges;  
Edge EDGE(int, int, double);
```

## Representação de $(G, c)$ ; Programa 20.1

## Representação de $(G, c)$ ; Programa 20.1

```
#include "GRAPH.h"
struct graph { int V; int E; double **adj; };
Graph GRAPHinit(int V)
{ Graph G = malloc(sizeof *G);
  G->adj = MATRIXdouble(V, V, maxWT);
  G->V = V; G->E = 0;
  return G;
}
void GRAPHinsertE(Graph G, Edge e)
{
  if (G->adj[e.v][e.w] == maxWT) G->E++;
  G->adj[e.v][e.w] = e.wt;
  G->adj[e.w][e.v] = e.wt;
}
```

## Representação de $(G, c)$ ; Programa 20.2



## Representação de $(G, c)$ ; Programa 20.2

```
#include "GRAPH.h"
typedef struct node *link;
struct node { int v; double wt; link next; };
struct graph { int V; int E; link *adj; };
link NEW(int v, double wt, link next)
{ link x = malloc(sizeof *x);
  x->v = v; x->wt = wt; x->next = next;
  return x;
}
```

## Representação de $(G, c)$ ; Programa 20.2

## Representação de $(G, c)$ ; Programa 20.2

```
Graph GRAPHinit(int V)
{ Graph G = malloc(sizeof *G); int i;
  G->adj = malloc(V*sizeof(link));
  G->V = V; G->E = 0;
  for (i = 0; i < V; i++) G->adj[i] = NULL;
  return G;
}

void GRAPHinsertE(Graph G, Edge e)
{ int v = e.v, w = e.w;
  if (v == w) return;
  G->adj[v] = NEW(w, e.wt, G->adj[v]);
  G->adj[w] = NEW(v, e.wt, G->adj[w]);
  G->E++;
}
```

Antes dos algoritmos...

## Antes dos algoritmos...

Seja  $G = (V, E)$  um grafo. Um *corte (de arestas)* de  $G$  é um conjunto da forma

$$E(S, V \setminus S) = \left\{ \{s, t\} \in E : s \in S, t \in V \setminus S \right\}.$$

## Antes dos algoritmos...

Seja  $G = (V, E)$  um grafo. Um *corte (de arestas)* de  $G$  é um conjunto da forma

$$E(S, V \setminus S) = \{ \{s, t\} \in E : s \in S, t \in V \setminus S \}.$$

Às vezes,  $(S, V \setminus S)$  é chamado de 'corte'.

## Antes dos algoritmos...

Seja  $G = (V, E)$  um grafo. Um *corte (de arestas)* de  $G$  é um conjunto da forma

$$E(S, V \setminus S) = \left\{ \{s, t\} \in E : s \in S, t \in V \setminus S \right\}.$$

Às vezes,  $(S, V \setminus S)$  é chamado de 'corte'.

Seja  $T$  uma AG de  $G$ . Para cada  $e \in E(G) \setminus E(T)$ , definimos o *circuito fundamental de  $e$  em relação a  $T$*  como sendo o único circuito em  $T + e$ .

(Notação:  $C(e, T)$ )

Antes dos algoritmos...



## Antes dos algoritmos...

**Propriedade 1 (Propriedade 20.1, Prop. dos Cortes).** (i) *Qualquer aresta  $e \in E(S, V \setminus S)$  de peso mínimo pertence a alguma AGM de  $G$ .*

(ii) *Qualquer AGM de  $G$  contém uma aresta  $e \in E(S, V \setminus S)$  de peso mínimo.*

*Demonstração. (...)*



Antes dos algoritmos...

## Antes dos algoritmos...

Seja  $T$  uma AG de  $G$ . Para cada  $e \in E(T)$ , sejam  $T_1^e$  e  $T_2^e$  os componentes de  $T - e$ .

## Antes dos algoritmos...

Seja  $T$  uma AG de  $G$ . Para cada  $e \in E(T)$ , sejam  $T_1^e$  e  $T_2^e$  os componentes de  $T - e$ .

Se  $c$  é injetora, então a AGM de  $G$  é única. De fato, segue da Propriedade dos Cortes. (Prova: ...)

## Antes dos algoritmos...

Seja  $T$  uma AG de  $G$ . Para cada  $e \in E(T)$ , sejam  $T_1^e$  e  $T_2^e$  os componentes de  $T - e$ .

Se  $c$  é injetora, então a AGM de  $G$  é única. De fato, segue da Propriedade dos Cortes. (Prova: ...)

São equivalentes:

## Antes dos algoritmos...

Seja  $T$  uma AG de  $G$ . Para cada  $e \in E(T)$ , sejam  $T_1^e$  e  $T_2^e$  os componentes de  $T - e$ .

Se  $c$  é injetora, então a AGM de  $G$  é única. De fato, segue da Propriedade dos Cortes. (Prova: ...)

São equivalentes:

1.  $T$  é uma AGM.
2. Toda  $e \in E(T)$  tem peso mínimo dentre as arestas em  $E(V(T_1^e), V(T_2^e))$ .

## Propriedade dos Circuitos

## Propriedade dos Circuitos

**Propriedade 2 (Propriedade 20.2, Prop. dos Circuitos).** *Suponha que  $G' = G + e'$  (supomos que  $e'$  já ‘vem com um peso’). Seja  $T$  uma AGM de  $G$ , e seja  $e$  uma aresta de peso máximo em  $C(e', T)$ . Então  $T - e + e'$  é uma AGM de  $G'$ .*

*Demonstração. (...)*





## Outro critério de otimalidade

## Outro critério de otimalidade

São equivalentes:

## Outro critério de otimalidade

São equivalentes:

1.  $T$  é uma AGM.
2. Toda  $e \in E(G) \setminus E(T)$  tem peso máximo dentre as arestas em  $C(e, T)$ .

*Demonstração.* (...)



AGM-GENERIC( $G, c$ )

## AGM-GENERIC( $G, c$ )

1.  $A \leftarrow \emptyset$
2. **enquanto**  $A$  não gera uma AG
3.     **faça** encontre uma boa aresta  $e = \{u, w\}$  para  $A$
4.      $A \leftarrow A \cup \{e\}$
5. **devolva**  $A$

AGM-GENERIC( $G, c$ )

## AGM-GENERIC( $G, c$ )

- ▷ Suponha que  $A$  está contida em uma AGM de  $(G, c)$ . Uma aresta  $e \in E(G) \setminus A$  é *boa para*  $A$  se  $A \cup \{e\}$  também está contida em uma AGM de  $G$ .

## AGM-GENERIC( $G, c$ )

- ▷ Suponha que  $A$  está contida em uma AGM de  $(G, c)$ . Uma aresta  $e \in E(G) \setminus A$  é *boa para*  $A$  se  $A \cup \{e\}$  também está contida em uma AGM de  $G$ .
- ▷  $(S, V \setminus S)$  respeita  $A \subset E(G)$  se  $A \cap E(S, V \setminus S) = \emptyset$ .



## AGM-GENERIC( $G, c$ )

- ▷ Suponha que  $A$  está contida em uma AGM de  $(G, c)$ . Uma aresta  $e \in E(G) \setminus A$  é *boa para*  $A$  se  $A \cup \{e\}$  também está contida em uma AGM de  $G$ .
- ▷  $(S, V \setminus S)$  respeita  $A \subset E(G)$  se  $A \cap E(S, V \setminus S) = \emptyset$ .

**Propriedade 3.**  $G = (V, E)$  grafo conexo,  $c: E \rightarrow \mathbb{R}$ ,  $A \subset E$  contida em alguma AGM de  $(G, c)$ . Seja  $(S, V \setminus S)$  um corte que respeita  $A$ . Se  $e$  tem peso mínimo dentre todas as arestas em  $E(S, V \setminus S)$ , então  $e$  é boa para  $A$ .

*Demonstração.* (...)



## Prim e Kruskal

## Prim e Kruskal

- ▶ Prim e Kruskal são casos particulares de  $\text{AGM-GENERIC}(G, c)$ .

## Prim e Kruskal

- ▶ Prim e Kruskal são casos particulares de  $\text{AGM-GENERIC}(G, c)$ .

Correção decorre da Proposição 3.