Árvores geradoras mínimas

1. Grafos com pesos nas arestas: $G = (V, E)$, $c\colon E \to \mathbb{R}$ (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')

Algoritmos

- Algoritmo de Prim
- Algoritmo de Kruskal
- Algoritmo de Boruvka (vamos omitir)

Antes dos algoritmos. . .

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

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

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

Definições para AGM-GENERICO(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 1. *$G = (V, E)$ grafo conexo, $c\colon E \to \mathbb{R}$, $A \subset E$ 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.*

AGM-GENERICO(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

Prim e Kruskal

- Prim e Kruskal são casos particulares de AGM-GENERICO(G, c).

Correção decorre da Proposição 1.

Algoritmo de Prim

Crescemos uma árvore T até que ela se torne geradora. Começamos com uma árvore com um único vértice e, genericamente, pomos $S = V(T)$ na Propriedade 1. Escolhemos e no AGM-GENERICO(G, c) como dado pela Propriedade 1.

Algoritmo de Prim, estrutura de dados

Mantemos

1. para cada vértice na árvore: pai (`st[]`)
2. para cada vértice fora da árvore: o vértice mais próximo da árvore (`fr[]`)
3. para cada vértice na árvore: comprimento da aresta ao pai (`wt[]`)
4. para cada vértice fora da árvore: o comprimento ao vértice mais próximo da árvore (`wt[]`)

Programa 20.3, Algoritmo de Prim (grafos densos)

```
static int fr[maxV];
#define P G->adj[v][w]
void GRAPHmstV(Graph G, int st[], double wt[])
{ int v, w, min;
  for (v = 0; v < G->V; v++) {st[v] = -1; fr[v] = v; wt[v] = maxWT;}
  st[0] = 0; wt[G->V] = maxWT;
  for (min = 0; min != G->V; ) {
      v = min; st[min] = fr[min];
      for (w = 0, min = G->V; w < G->V; w++)
        if (st[w] == -1) {
            if (P < wt[w]) { wt[w] = P; fr[w] = v; }
            if (wt[w] < wt[min]) min = w;
        }
      }
}
```

Programa 20.3, Algoritmo de Prim (grafos densos), cont.

Propriedade 2. *O Programa 20.3 tem complexidade* $O(n^2)$*.*

Prova. Por inspeção. □

Conclusão: Programa 20.3 é bom para grafos densos.

Algoritmo de Prim, grafos esparsos

Usamos o esquema de busca generalizada em grafos. Programa 18.10:

```
void pfs(Graph G, Edge e)
{ link t; int v, w;
  GQput(e); pre[e.w] = cnt++;
  while (!GQempty()) {
    e = GQget(); w = e.w; st[w] = e.v;
    for (t = G->adj[w]; t != NULL; t = t->next)
      if (pre[v = t->v] == -1) {GQput(EDGE(w, v)); pre[v] = cnt++;}
      else if (st[v] == -1) GQupdate(EDGE(w, v));
    }
}
```

Programa 20.4, Algoritmo de Prim (grafos esparsos)

```
static int fr[maxV];
static double *priority;
int less(int i, int j)
  { return priority[i] < priority[j]; }
#define P t->wt
```

Programa 20.4, Algoritmo de Prim (grafos esparsos)

```
void GRAPHmst(Graph G, int st[], double wt[])
{ link t; int v, w;
  PQinit(); priority = wt;
  for (v = 0; v < G->V; v++) { st[v] = -1; fr[v] = -1; }
  fr[0] = 0; PQinsert(0);
  while (!PQempty()) {
      v = PQdelmin(); st[v] = fr[v];
      for (t = G->adj[v]; t != NULL; t = t->next)
        if (fr[w = t->v] == -1)
          { wt[w] = P; PQinsert(w); fr[w] = v; }
        else if ((st[w] == -1) && (P < wt[w]))
          { wt[w] = P; PQdec(w); fr[w] = v; }
    }
}
```

Programa 20.3, Algoritmo de Prim (grafos esparsos), cont.

Propriedade 3. *O Programa 20.4, com a fila de prioridade implementada com um heap (binário), tem complexidade* $O((n+m)\log n) = O(m \log n)$.

Prova. Por inspeção. □

Conclusão: Programa 20.4 é bom para grafos esparsos.