Algoritmos em grafos: árvore de expansão mínima (*minimum spanning tree*)

R. Rossetti, A.P. Rocha, R. Camacho

FEUP, MIEIC, CAL

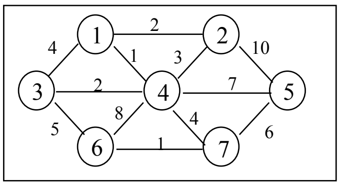
Árvore de expansão mínima

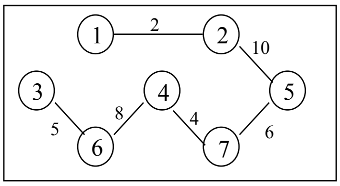
Árvore que liga todos os vértices do grafo usando arestas com um custo total mínimo

- caso do grafo não dirigido
- grafo tem que ser conexo
- árvore ⇒ grafo conexo acíclico
- número de arestas = |V| - 1

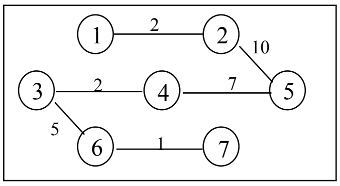
- exemplo de aplicação: cablagem, e.g. casa, ou avião (*fly-by-wire*)
  - vértices são as tomadas
  - arestas são os comprimentos dos troços

Exemplos de Árvores de expansão

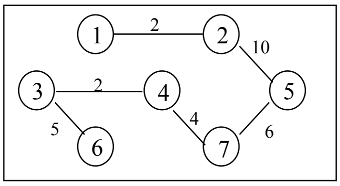




35



27



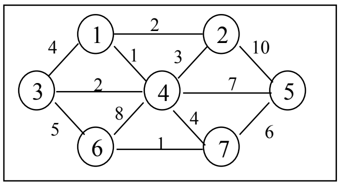
29

Algoritmo de Prim

- expandir a árvore por adição sucessiva de arestas e respectivos vértices
  - critério de selecção: ***escolher a aresta (u,v) de menor custo tal que u já pertence à árvore e v não*** (ganancioso)
  - início: um vértice qualquer

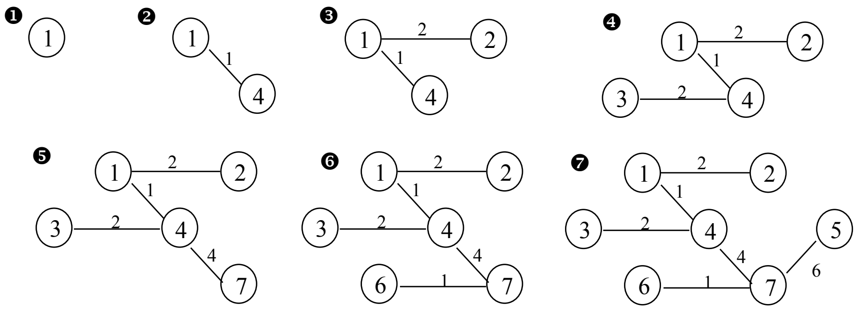
- idêntico ao algoritmo de Dijkstra para o caminho mais curto
  - informação para cada vértice
    - dist(v) é o custo mínimo das arestas que ligam a um vértice já na árvore
    - path(v) é o último vértice a alterar dist(v)
    - known(v) indica se o vértice já foi processado (i.e., já pertence à árvore)
  - diferença na regra de actualização: *após a selecção do vértice v, para cada w não processado, adjacente a v, dist(w) = min{ dist(w), cost(v,w) }*
  - tempo de execução
    - O($|V|^2$) sem fila de prioridade
    - **O(|E| log |V|)** com fila de prioridade

Evolução do algoritmo de Prim



última tabela

v	known	dist	path
1	1	0	0
2	1	2	1
3	1	2	4
4	1	1	1
5	1	6	7
6	1	1	7
7	1	4	4



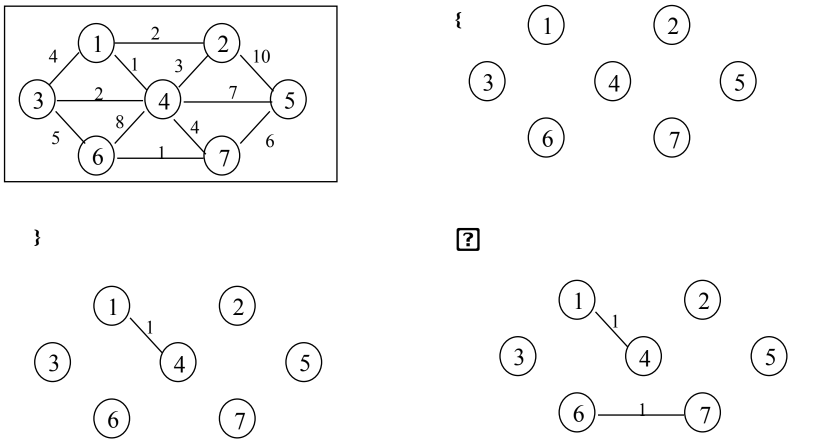
16

Algoritmo de Kruskal

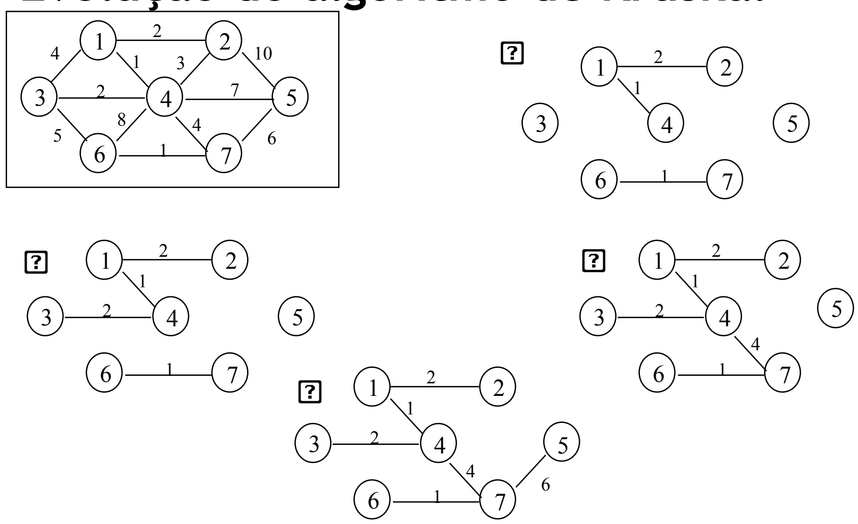
- ❑ **analisar as arestas por ordem crescente de peso e aceitar as que não provocarem ciclos** (ganancioso)
- ❑ método
  - manter uma floresta, inicialmente com um vértice em cada árvore (há |V|)
  - adicionar uma aresta é fundir duas árvores
  - quando o algoritmo termina há só uma árvore (de expansão mínima)

- ❑ aceitação de arestas — algoritmo de Busca/União em conjuntos disjuntos
  - representados como árvores
  - se dois vértices pertencem à mesma árvore/conjunto, mais uma aresta entre eles provoca um ciclo (2 Buscas)
  - se são de conjuntos disjuntos, aceitar a aresta é aplicar-lhes uma União
- ❑ selecção de arestas: ordenar por peso ou, melhor, construir fila de prioridade em tempo linear e usar deleteMin (*heapsort*)
  - tempo no pior caso **O(|E| log |E|),** dominado pelas operações na fila
  - como $|E| \le |V|^2$, $\log |E| \le 2 \log |V|$, logo eficiência é também **O(|E| log |V|)**

Evolução do algoritmo de Kruskal



Evolução do algoritmo de Kruskal



Pseudocódigo (Kruskal)

```
void kruskal() {
      int edgesAccepted = 0;

      PriorityQueue<Edge> h = readGraphIntoHeapArray();
      h.buildHeap();
      DisjSet<Vertex> s = new DisjSet(NUM_VERTICES);

      while(edgesAccepted < NUM_VERTICES -1 ) {
            Edge e = h.deleteMin();     // e = (u,v)
            SetType uset = s.find(u);
            SetType vset = s.find(v);
            if (uset != vset) {
                  edgesAccepted++;
                  s.union(uset, vset);
            }
      }
}
```


Referência e informação adicional

- "Data Structures and Algorithm Analysis in Java", Second Edition, Mark Allen Weiss, Addison Wesley, 2006