Árvores de expansão mínimas

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 ⇒ acíclico
- número de arestas = |V| - 1

☐ exemplo de aplicação: cablamento de uma casa
- vértices são as tomadas
- arestas são os comprimentos dos troços

Grafos - 45

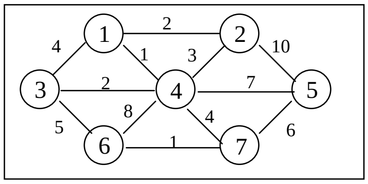
Algoritmo de Prim

☐ 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 jé foi processado (isto é, se 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(w,v) }*
- tempo de execução
  - O($|V|^2$) sem fila de prioridade
  - O(|E| log |V|) com fila de prioridade

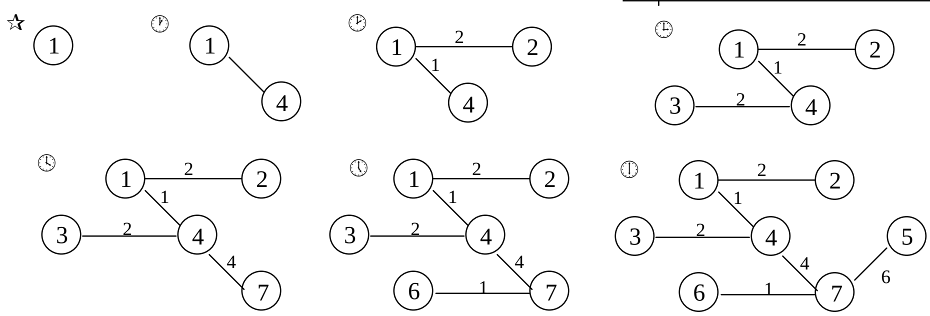
Grafos - 46

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



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
  - 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 Apaga_Min
  - tempo no pior caso O(|E| log |E|), dominado pelas operações na fila

Pseudocódigo (Kruskal)

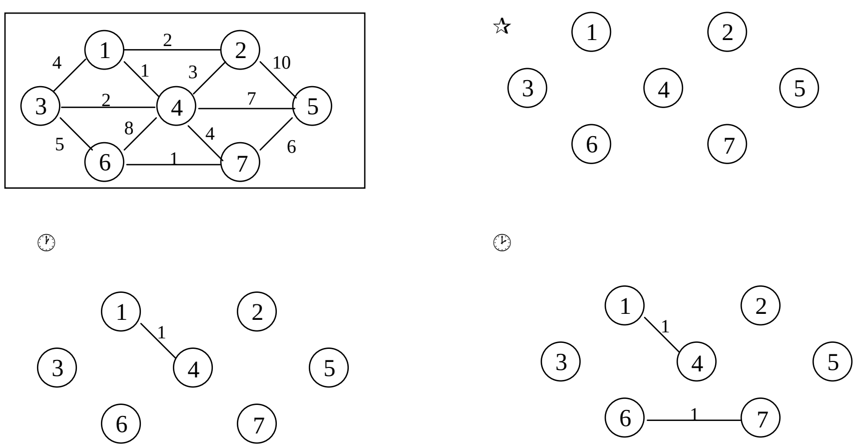
```
void kruskal()
{
        DisjSet s;
        PriorityQueue h;
        Vertex u, v;
        SetType uset, vset;
        Edge e,
        int edgesAccepted = 0;

        h = readGraphIntoHeapArray();
        h.buildHeap();
        s = new DisjSet(NUM_VERTICES);

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

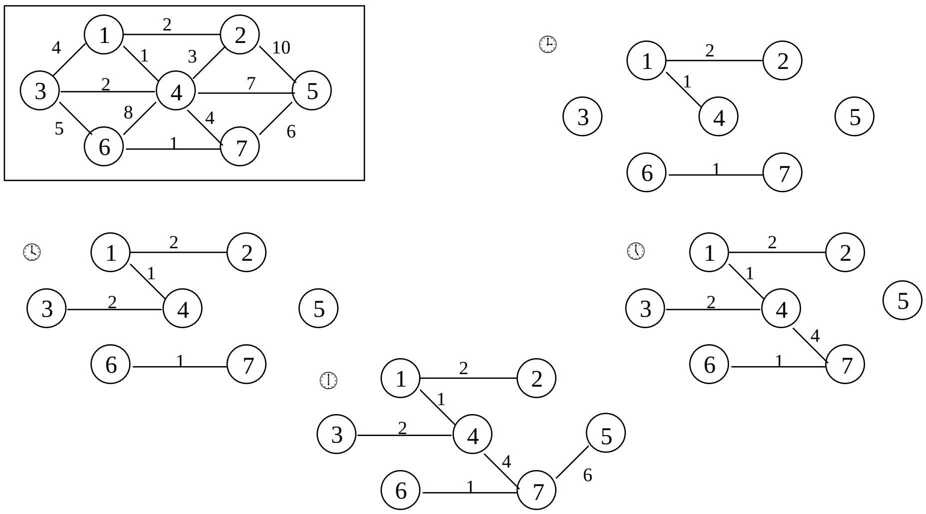
Grafos - 49

Evolução do algoritmo de Kruskal



Grafos - 50

Evolução do algoritmo de Kruskal



Grafos - 51