Desafios de Programação

Prof. Eduardo Theodoro

Universidade Federal de Mato Grosso do Sul (UFMS)

Problema do Caminho Mínimo

Seja G um grafo ponderado.

Objetivo

Consiste em encontrar um caminho de menor custo entre dois vértices dados, digamos u e v, pertencente a G.

Para resolver este problema, vamos estudar o algoritmo de Dijkstra (1959). Como veremos, este algoritmo não só encontra o caminho mínimo de u a v, mas de u a qualquer outro vértice do grafo. A complexidade do algoritmo é de $O(|E| + |V| log |V|)$.
Observação: o algoritmo de Dijkstra só funciona com arestas de custo positivo.

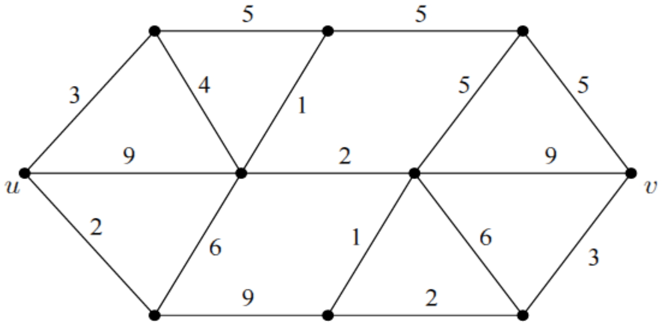
Algoritmo de Dijkstra

Vamos assumir que $c(x, y) = \infty$ se $(x, y) \notin E(G)$.

1. $d(u) \leftarrow 0$; $S \leftarrow \{u\}$;
2. Para cada $v \in (V(G) - \{u\})$ faça $d(v) \leftarrow c(u, v)$;
3. Enquanto $S \neq V(G)$ faça
   - "escolha $v \in V(G) - S$ tal que $d(v)$ seja mínimo";
   - $S \leftarrow S \cup \{v\}$;
   - Para cada $w \in V(G) - S$ faça $d(w) \leftarrow min\{d(w), d(v) + c(v, w)\}$;

Exemplo

Encontre o caminho mínimo entre os vértices u e v.



Dijkstra Código em C++

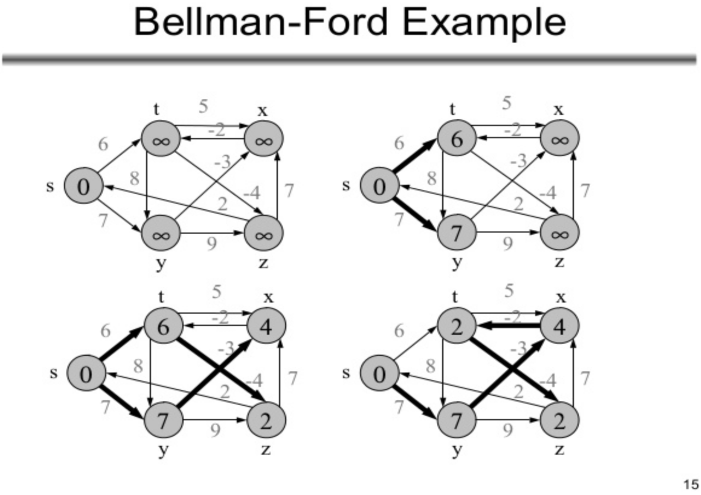
```
vi dist(V, INF); dist[s] = 0;                          // INF = 1B to avoid overflow
priority_queue< ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s));
                           // ^to sort the pairs by increasing distance from s
while (!pq.empty()) {                                               // main loop
  ii front = pq.top(); pq.pop();     // greedy: pick shortest unvisited vertex
  int d = front.first, u = front.second;
  if (d == dist[u])             // this check is important, see the explanation
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];                        // all outgoing edges from u
      if (dist[u] + v.second < dist[v.first]) {
        dist[v.first] = dist[u] + v.second;                 // relax operation
        pq.push(ii(dist[v.first], v.first));
}   } }  // note: this variant can cause duplicate items in the priority queue
```

Algoritmo de Dijkstra

- Para resolver o problema com arestas de pesos negativos, podemos usar o algoritmo de Bellman-Ford.
- A sua ideia é simples: relaxar todas as $|E|$ arestas $|V|$?1 vezes!

Complexidade: $O(|V| \times |E|)$.
Observação: Não aceita grafos com ciclos negativos!

Exemplo



Bellman-Ford Código

```
vi dist(V, INF); dist[s] = 0;
for (int i = 0; i < V - 1; i++)  // relax all E edges V-1 times, overall O(VE)
  for (int u = 0; u < V; u++)                        // these two loops = O(E)
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];       // we can record SP spanning here if needed
      dist[v.first] = min(dist[v.first], dist[u] + v.second);         // relax
    }
```

Example codes: ch4_06_bellman_ford.cpp; ch4_06_bellman_ford.java

```
bool hasNegativeCycle = false;
for (int u = 0; u < V; u++)                            // one more pass to check
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dist[v.first] > dist[u] + v.second)        // if this is still possible
      hasNegativeCycle = true;                     // then negative cycle exists!
  }
printf("Negative Cycle Exist? %s\n", hasNegativeCycle ? "Yes" : "No");
```