Caminho mais curto
Algoritmo de Dijkstra

Fernando Lobo

Algoritmos e Estrutura de Dados

Caminho mais curto a partir de um nó

- Input: Um grafo com pesos nos arcos $G = (V, E)$, $w : E \to \mathbb{R}$, e um nó de partida s.
- Output: Caminho mais curto de s para todos os outros nós do grafo.
- O peso (ou distância ou custo) de um caminho é igual ao somatório dos pesos dos arcos que constituem o caminho, ou ∞ se não existir caminho.
- Formalmente, seja p o caminho $v_0 \rightsquigarrow v_k = \langle v_0, v_1, \ldots, v_k \rangle$.

$$\text{peso}(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Algoritmo de Dijkstra

- Restrição: Os pesos são ≥ 0.
- Parecido com BFS.
  - Usa uma fila com prioridade em vez de uma fila FIFO.
  - Cada nó v do grafo tem um atributo $v.d$ que nos dá um limite superior para o peso de um caminho mais curto de s para v.
  - Esse limite superior é uma estimativa do custo para o caminho mais curto.
  - A dada altura essa estimativa deixa de ser estimativa e passa a ser uma certeza.

Algoritmo de Dijkstra

- O algoritmo mantém dois conjuntos de nós:
  - $S =$ nós para os quais já determinamos o caminho mais curto.
  - $Q = V - S =$ nós que ainda estão na fila.
- No pseudocódigo que vamos ver, S aparece de forma explícita, mas numa implementação concreta não é necessário mantê-lo.
- Ao início, o atributo $d = \infty$ para todos os nós excepto para o nó de partida s, que tem $s.d = 0$.
- Todos os nós do grafo vão para uma fila com prioridade Q. A chave (prioridade) de um nó é o atributo d.

Pseudocódigo

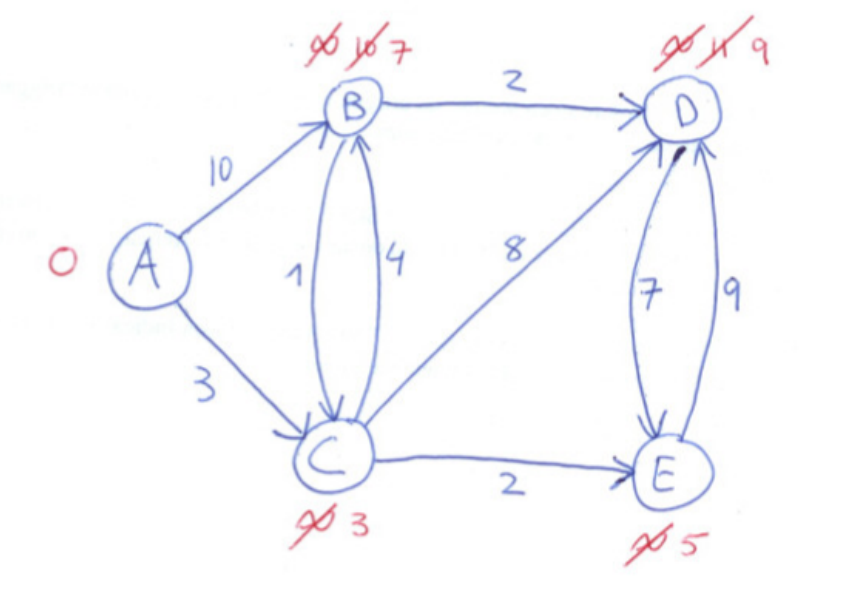
```
DIJKSTRA(G, w, s)
 1  for each v ∈ G.V
 2        v.d = ∞
 3        v.π = NIL
 4  s.d = 0
 5  S = ∅
 6  Q = G.V
 7  while Q ≠ ∅
 8        u = EXTRACT-MIN(Q)
 9        S = S ∪ {u}
10        for each v ∈ G.Adj[u]
11              if v.d > u.d + w(u, v)
12                    v.d = u.d + w(u, v)
13                    v.π = u
```


Ao final

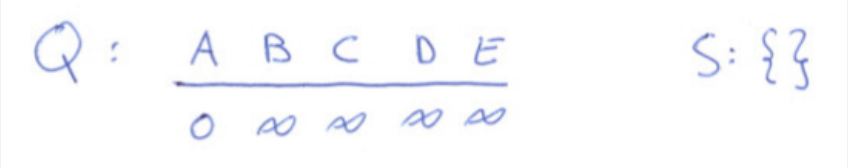
- Quando o algoritmo termina $v.d$ tem o peso (custo) do caminho mais curto do nó de origem s até v.
- Tal como no BFS, o atributo π permite-nos obter o caminho mais curto.

Exemplo

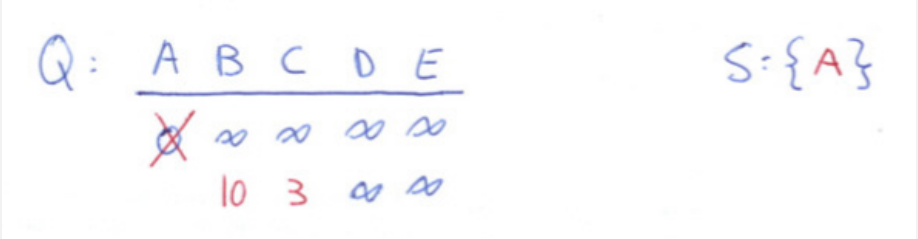
Nó de origem é A



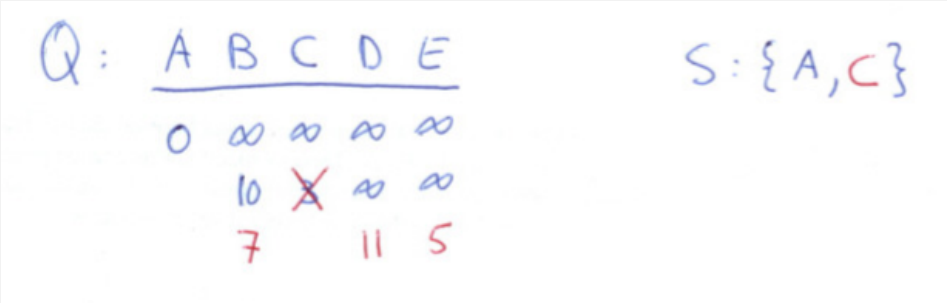
Inicialização



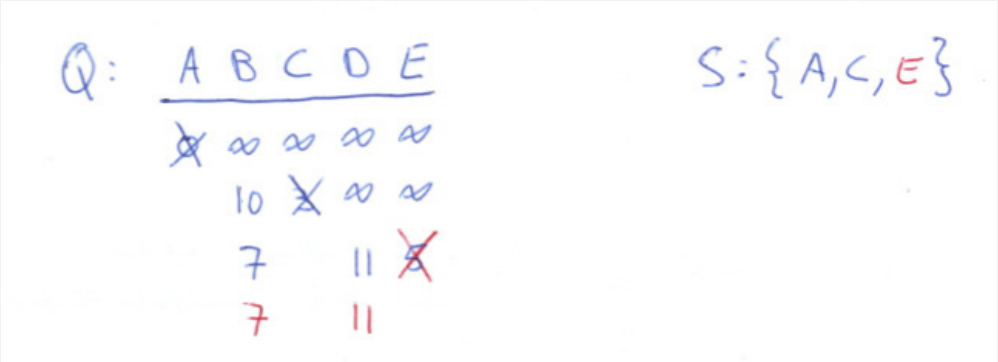
1ª iteração: nó A sai da fila



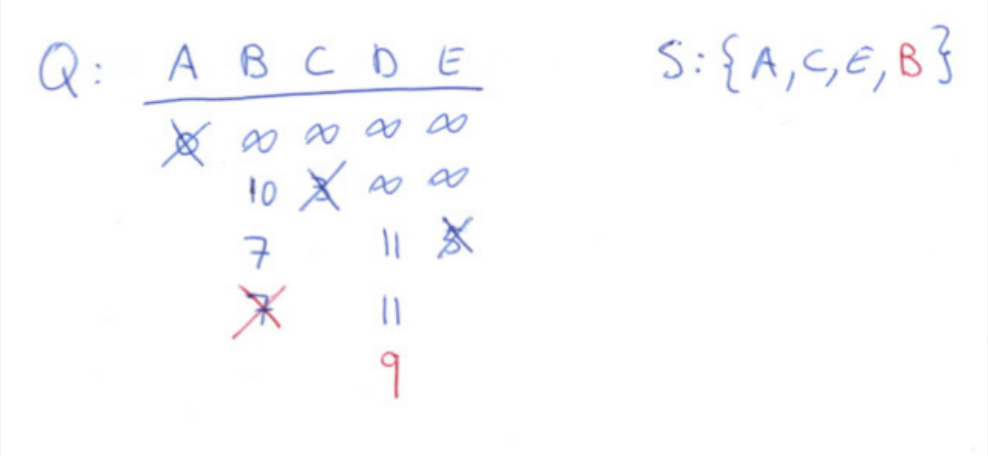
2ª iteração: nó C sai da fila



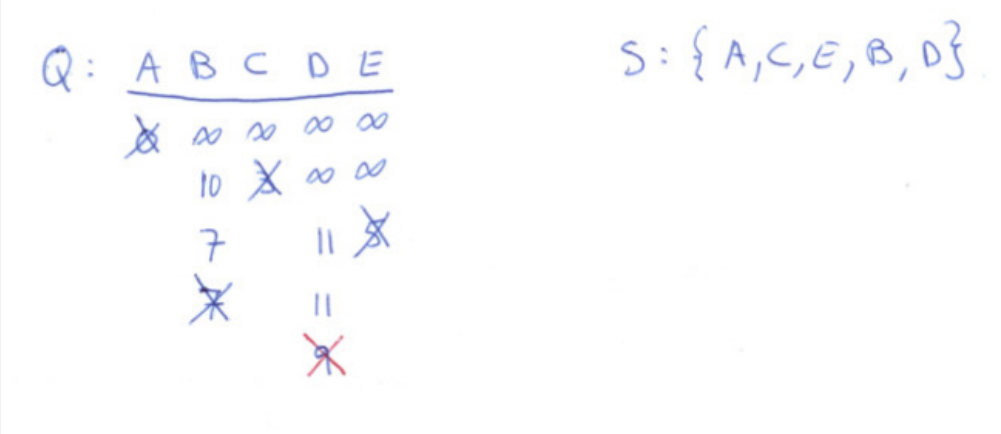
3ª iteração: nó E sai da fila



4ª iteração: nó B sai da fila



5ª iteração: nó D sai da fila



Correção do algoritmo

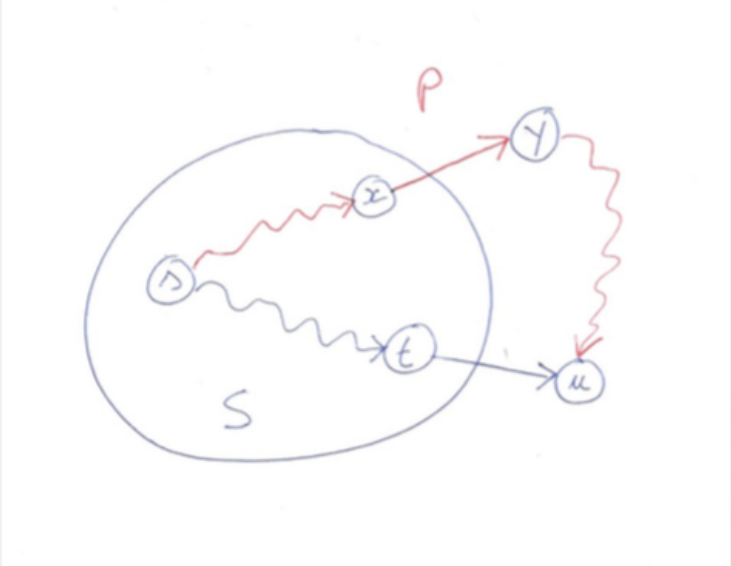
- Seja $\delta(u, v)$ o peso do caminho mais curto de u para v.
- Vamos demonstrar que em qualquer instante da execução do algoritmo, todo o nó $x \in S$ tem $x.d = \delta(s, x)$.
- Demonstração: por indução no tamanho de S.
- Caso base: $|S| = 1$.
  - Ocorre quando $S = \{s\}$, $s.d = 0 = \delta(s, s)$. Está correcto.
- Hipótese de indução: assumir que é verdade para $|S| = k$, com $k \geq 1$.

Correção do algoritmo: passo indutivo

- Na iteração $k+1$ do ciclo while o conjunto S vai passar a ter $k+1$ elementos pelo facto de lhe ser acrescentado um nó u.
- Vamos mostrar que, imediatamente antes de u ser acrescentado a S, $u.d = \delta(s, u)$.
- Para tal, vamos mostrar que qualquer outro caminho P de s para u tem um peso $\geq u.d$.

Correção do algoritmo: passo indutivo

- Qualquer outro caminho P de s para u terá forçosamente de ter uma aresta que liga um nó $x \in S$ com um nó $y \notin S$.



Correção do algoritmo: passo indutivo

$$\begin{aligned}\text{peso}(P) &= \text{peso}(s \rightsquigarrow x) + w(x,y) + \text{peso}(y \rightsquigarrow u) \\ &\geq \text{peso}(s \rightsquigarrow x) + w(x,y) \quad // \text{ porque os pesos são } \geq 0 \\ &\geq \delta(s,x) + w(x,y) \qquad // \text{ por hipótese de indução}\end{aligned}$$

- Acontece que logo após x ter sido acrescentado a S, as estimativas d para os vizinhos de x são actualizadas. Logo, temos a garantia que $y.d \leq \delta(s,x) + w(x,y)$.
- Se $P = s \rightsquigarrow x \rightarrow y \rightsquigarrow u$ for um caminho mais curto de s para u, então $s \rightsquigarrow x \rightarrow y$ é forçosamente um caminho mais curto de s para y, e nesse caso $y.d = \delta(s,y) = \delta(s,x) + w(x,y)$.
- Mas $u.d \leq y.d$. (Caso contrário u não seria o próximo nó a sair da fila.)
- Ou seja, $\text{peso}(P) \geq y.d \geq u.d$ □

Complexidade do Algoritmo de Dijkstra

- Inicialização: $\Theta(V)$
- Ciclo **while** é executado $|V|$ vezes.
  - $|V|$ Extract-Mins
  - Todos os arcos do grafo são visitados. Para cada um, há potencialmente um Decrease-Key.
  - $\implies O(V \cdot T_{\text{Extract-Min}} + E \cdot T_{\text{Decrease-Key}})$

Complexidade do Algoritmo de Dijkstra

- Se a fila com prioridade for implementada com um heap binário obtemos:
  - $T_{\text{EXTRACT-MIN}} = O(\lg V)$
  - $T_{\text{DECREASE-KEY}} = O(\lg V)$
  - Complexidade: $O(V \lg V + E \lg V) = O((V + E) \lg V)$

Outros algoritmos

- O Algoritmo de Dijkstra só funciona se todas as arestas do grafo tiverem pesos ≥ 0.
- No caso de poder haver arestas com pelo < 0, pode-se aplicar o algoritmo de Bellman-Ford. (Não vamos dar nesta disciplina mas podem ver detalhes no livro, caso tenham curiosidade.)