TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Procura em largura primeiro (BFS). Caminhos mais curtos. Dijkstra.

CLRS Cap. 22 e 24

Instituto Superior Técnico

2022/2023

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos
  - Fluxos máximos
  - Árvores abrangentes
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Procura em Largura Primeiro (BFS)

Funcionamento do Algoritmo

Dado um grafo $G = (V, E)$ e um vértice fonte s, o algoritmo BFS explora sistematicamente os vértices de G para descobrir todos os vértices atingíveis a partir de s

Procura em Largura Primeiro (BFS)

Funcionamento do Algoritmo

Dado um grafo $G = (V, E)$ e um vértice fonte s, o algoritmo BFS explora sistematicamente os vértices de G para descobrir todos os vértices atingíveis a partir de s

Intuição

Fronteira entre vértices descobertos e não descobertos é expandida uniformemente

- Vértices à distância k descobertos antes de qualquer vértice à distância $k + 1$

Procura em Largura Primeiro (BFS)

Resultado

- Identificação de árvore Breadth-First (BF): caminho mais curto de s para cada vértice atingível v
- $\delta(u, v)$: distância do caminho mais curto (menor número de arcos) de u para v

Procura em Largura Primeiro (BFS)

Implementação

- $color[v]$: cor do vértice v
  - branco: não visitado
  - cinzento: já visitado, mas:
    - algum dos adjacentes não visitado
    - ou procura em algum dos adjacentes não terminada
  - preto: já visitado e procura nos adjacentes já terminada
- $\pi[v]$: predecessor de v na árvore BF
- $d[v]$: tempo de descoberta do vértice v

Procura em Largura Primeiro (BFS)

BFS(G, s)

for each $u \in G.V \setminus \{s\}$ **do**
  $color[u] \leftarrow white$; $d[u] \leftarrow \infty$; $\pi[u] \leftarrow NIL$
end for
$color[s] \leftarrow gray$; $d[s] \leftarrow 0$; $\pi[s] \leftarrow NIL$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
  $u \leftarrow$ Dequeue(Q)
  for each $v \in G.Adj[u]$ **do**
    if $color[v] == white$ **then**
      $color[v] \leftarrow gray$; $d[v] \leftarrow d[u] + 1$; $\pi[v] \leftarrow u$
      Enqueue(Q, v)
    end if
  end for
  $color[u] \leftarrow black$
end while

Procura em Largura Primeiro (BFS)

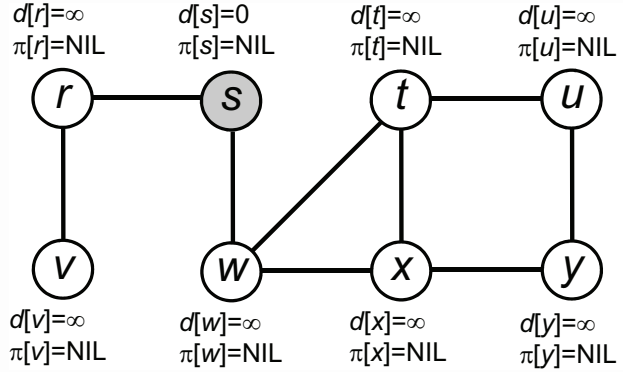
Complexidade

Tempo de execução: $O(V + E)$

- Inicialização: $O(V)$
- Para cada vértice
  - Colocado na fila apenas 1 vez: $O(V)$
  - Lista de adjacências visitada 1 vez: $O(E)$

Procura em Largura Primeiro (BFS)

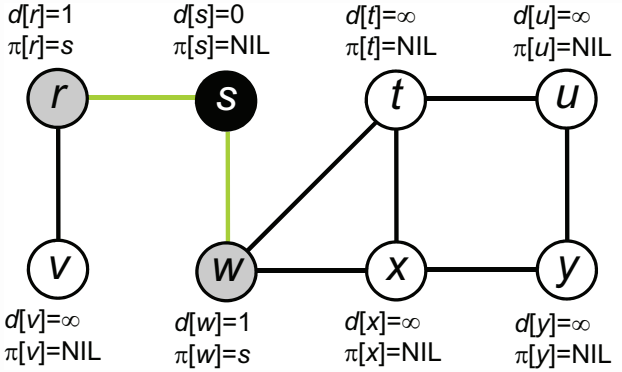
Exemplo



Q
s

Procura em Largura Primeiro (BFS)

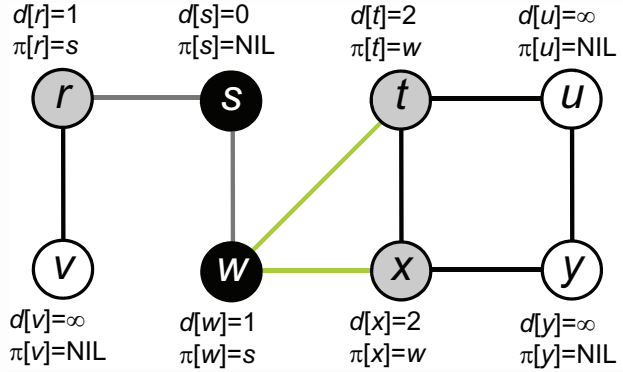
Exemplo



Q
w
r

Procura em Largura Primeiro (BFS)

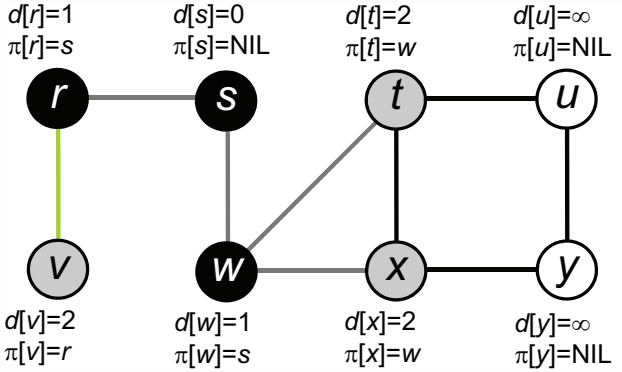
Exemplo



Q
r
t
x

Procura em Largura Primeiro (BFS)

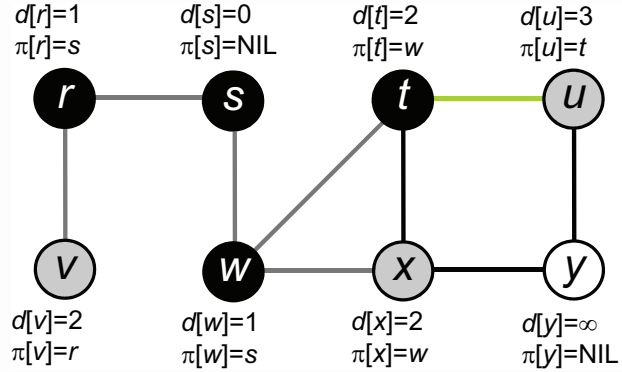
Exemplo



Q
t
x
v

Procura em Largura Primeiro (BFS)

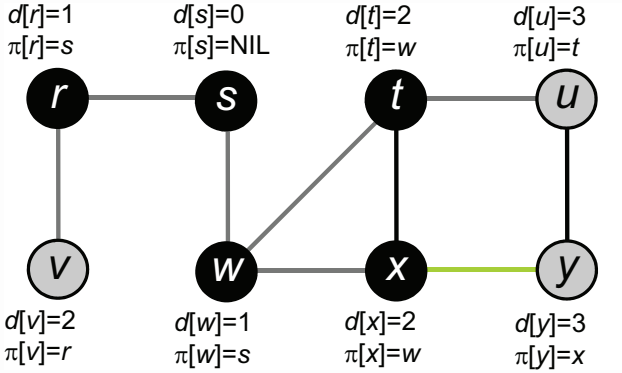
Exemplo



Q
x
v
u

Procura em Largura Primeiro (BFS)

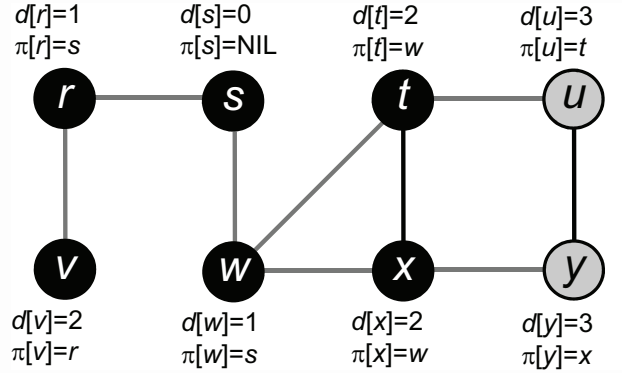
Exemplo



Q
v
u
y

Procura em Largura Primeiro (BFS)

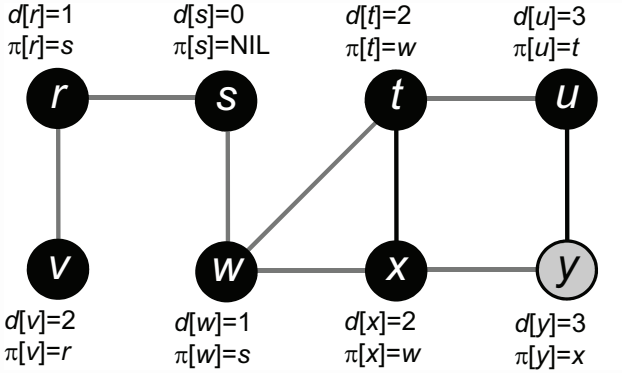
Exemplo



Q
u
y

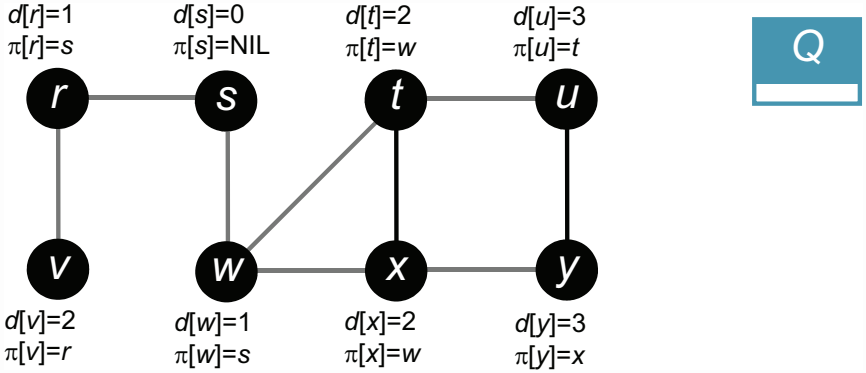
Procura em Largura Primeiro (BFS)

Exemplo



Q
y

Exemplo



Resultados

Para qualquer arco (u, v):

- Se u atingível, então v atingível
  - caminho mais curto para v não pode ser superior ao caminho mais curto para u mais o arco (u, v)
  - $\delta(s, v) \leq \delta(s, u) + 1$
- Se u não atingível, então resultado é válido (independentemente de v ser atingível)

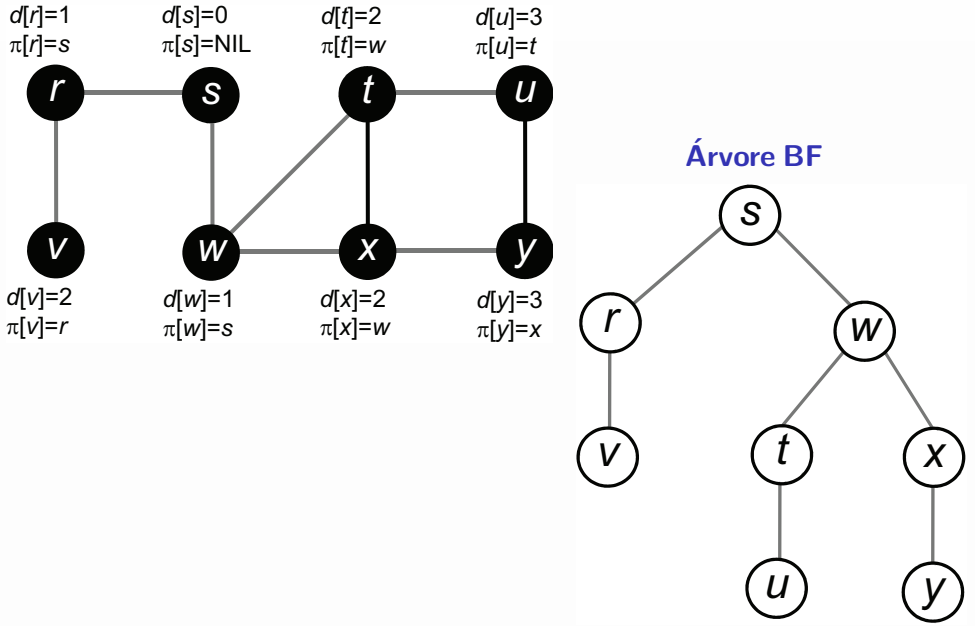
No final da BFS: $d[u] = \delta(s, u)$, para todo o vértice $u \in V$

Árvore Breadth-First

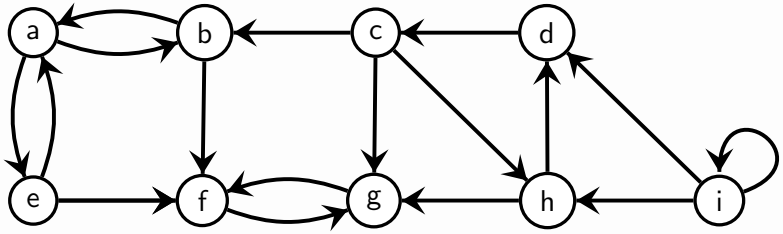
- $G_\pi = (V_\pi, E_\pi)$
- $V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$
- $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi \setminus \{s\}\}$

Características

- Árvore BF é sub-grafo de G
- Vértices atingíveis a partir de s
- Arcos que definem a relação predecessor da BFS



Exercício



Grafo Semi-Ligado
Um grafo dirigido G diz-se semi-ligado se para qualquer par de vértices (u, v), u é atingível a partir de v ou v é atingível a partir de u

Grafo Semi-Ligado
Um grafo dirigido G diz-se semi-ligado se para qualquer par de vértices (u, v), u é atingível a partir de v ou v é atingível a partir de u

Problema
Indique um algoritmo eficiente para determinar se um grafo $G = (V, E)$ é semi-ligado

Pontos de Articulação
Indique um algoritmo eficiente para determinar se um grafo $G = (V, E)$ não dirigido e ligado tem pontos de articulação

- Um grafo não dirigido diz-se ligado se para qualquer par de vértices $u, v \in V$, existe pelo menos um caminho entre u e v.
- Um vértice $u \in V$ diz-se um ponto de articulação de um grafo se a remoção do vértice u implicar que o grafo deixa de ser ligado.

Tipos de problemas

- Caminhos Mais Curtos com Fonte Única (SSSPs)
  - Identificar o caminho mais curto de um vértice fonte $s \in V$ para qualquer outro vértice $v \in V$
- Caminhos Mais Curtos com Destino Único
  - Identificar o caminho mais curto de qualquer vértice $v \in V$ para um vértice destino $t \in V$
- Caminho Mais Curto entre Par Único
  - Identificar caminho mais curto entre dois vértices u e v
- Caminhos Mais Curtos entre Todos os Pares (APSPs)
  - Identificar um caminho mais curto entre cada par de vértices de V

Definições

- Dado um grafo $G = (V, E)$, dirigido, com uma função de pesos $w : E \to IR$, define-se o peso de um caminho p, onde $p = \langle v_0, v_1, \ldots, v_k \rangle$, como a soma dos pesos dos arcos que compõem p:

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

- O peso do caminho mais curto de u para v é definido por:

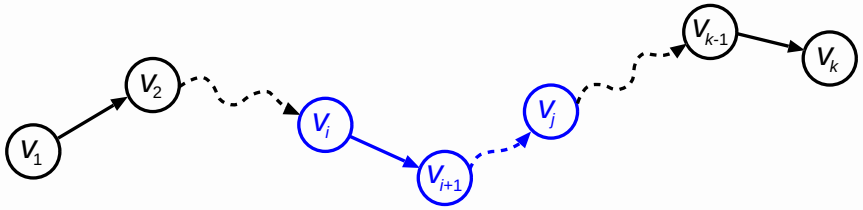
$$\delta(u, v) = \begin{cases} \min \{w(p) : u \to_p v\} & \text{se existe caminho de } u \text{ para } v \\ \infty & \text{caso contrário} \end{cases}$$

- Um caminho mais curto de u para v é qualquer caminho p tal que $w(p) = \delta(u, v)$

Sub-estrutura Ótima

Sub-caminhos de caminhos mais curtos são caminhos mais curtos

- Seja $p = \langle v_1, v_2, \ldots, v_k \rangle$ um caminho mais curto entre v_1 e v_k, e seja $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ um sub-caminho de p entre v_i e v_j
- Então p_{ij} é um caminho mais curto entre v_i e v_j
  - Porquê? Se existisse caminho mais curto entre v_i e v_j então seria possível construir caminho entre v_1 e v_k mais curto do que p
  - Contradição, dado que p é um caminho mais curto entre v_1 e v_k



Peso de um Caminho Mais Curto

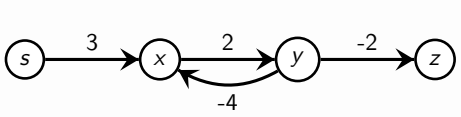
Seja $p = \langle s, \ldots, v \rangle$ um caminho mais curto entre s e v, que pode ser decomposto em $p_{su} = \langle s, \ldots, u \rangle$ e (u, v).

Então o peso do caminho mais curto será $\delta(s, v) = \delta(s, u) + w(u, v)$

- p_{su} é caminho mais curto entre s e u
- $\delta(s, v) = w(p) = w(p_{su}) + w(u, v) = \delta(s, u) + w(u, v)$

Ciclos Negativos

- Arcos podem ter pesos com valor negativo
- É possível a existência de ciclos com peso total negativo
  - Se ciclo negativo não atingível a partir da fonte s, então $\delta(s, v)$ bem definido
  - Se ciclo negativo atingível a partir da fonte s, então os pesos dos caminhos mais curtos não são bem definidos
  - Neste caso, é sempre possível encontrar um caminho mais curto de s para qualquer vértice incluído no ciclo e define-se $\delta(s, v) = -\infty$



$w(\langle s, x, y, z\rangle) = 3$
$w(\langle s, x, y, x, y, z\rangle) = 1$
$w(\langle s, x, y, x, y, x, y, z\rangle) = -1$
$\ldots$

Representação de Caminhos Mais Curtos

- Para cada vértice $v \in V$ associar predecessor $\pi[v]$
- Após identificação dos caminhos mais curtos, $\pi[v]$ indica qual o vértice anterior a v num caminho mais curto de s para v
- Sub-grafo de predecessores $G_\pi = (V_\pi, E_\pi)$:

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

Representação de Caminhos Mais Curtos

Uma árvore de caminhos mais curtos é um sub-grafo dirigido $G' = (V', E')$, $V' \subseteq V$ e $E' \subseteq E$, tal que:

- V' é o conjunto de vértices atingíveis a partir de s em G
- G' forma uma árvore com raiz s
- Para todo o $v \in V'$, o único caminho de s para v em G' é um caminho mais curto de s para v em G

Observações

- Após identificação dos caminhos mais curtos de G a partir de fonte s, G' é dado por $G_\pi = (V_\pi, E_\pi)$
- Dados os mesmos grafo G e vértice fonte s, G' não é necessariamente único

Relação caminho mais curto com arcos do grafo

Para todos os arcos $(u, v) \in E$ verifica-se $\delta(s, v) \leq \delta(s, u) + w(u, v)$

- Caminho mais curto de s para v não pode ter mais peso do que qualquer outro caminho de s para v
- Assim, peso do caminho mais curto de s para v não superior ao peso do caminho mais curto de s para u seguido do arco (u, v) (i.e. exemplo de um dos caminhos de s para v)

Propriedades de Caminhos Mais Curtos

Resumo

Sub-estrutura óptima: Sub-caminhos de caminhos mais curtos são caminhos mais curtos

- Seja $p = \langle v_1, v_2, \ldots, v_k \rangle$ um caminho mais curto entre v_1 e v_k, e seja $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ um sub-caminho de p entre v_i e v_j.
- Então p_{ij} é um caminho mais curto entre v_i e v_j
- Seja $p = \langle s, \ldots, v \rangle$ um caminho mais curto entre s e v, que pode ser decomposto em $p_{su} = \langle s, \ldots, u \rangle$ e (u, v). Então $\delta(s, v) = \delta(s, u) + w(u, v)$

Relação caminho mais curto com arcos do grafo: para todos os arcos $(u, v) \in E$ verifica-se $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Operação de Relaxação

Conceitos

- Operação básica dos algoritmos para cálculo dos caminhos mais curtos com fonte única
- $d[v]$: denota a estimativa do caminho mais curto de s para v limite superior no valor do peso do caminho mais curto;
- Algoritmos aplicam sequência de relaxações dos arcos de G após inicialização para actualizar a estimativa do caminho mais curto

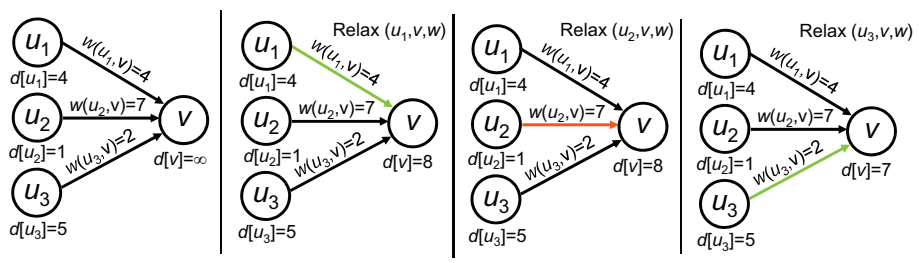
Relax(u, v, w)
  if $d[v] > d[u] + w(u, v)$ **then**
    $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$
  end if

Propriedades da Relaxação

Propriedades da Relaxação

Após relaxar arco (u, v), temos que $d[v] \leq d[u] + w(u, v)$

- Se $d[v] > d[u] + w(u, v)$ antes da relaxação, então $d[v] = d[u] + w(u, v)$ após relaxação
- Se $d[v] \leq d[u] + w(u, v)$ antes da relaxação, então $d[v] \leq d[u] + w(u, v)$ após relaxação
- Em qualquer caso, $d[v] \leq d[u] + w(u, v)$ após relaxação



Propriedades da Relaxação

Triangle inequality
Para qualquer arco $(u, v) \in E$, temos $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Upper-bound property
Para qualquer vértice $v \in V$, temos $d[v] \geq \delta(s, v)$ e uma vez que $d[v]$ atinja o valor $\delta(s, v)$, já não se altera mais.

No-path property
Se não existir um caminho de s para v, então $d[v] = \delta(s, v) = \infty$

Convergence property
Se $s \rightsquigarrow u \rightarrow v$ é um caminho mais curto em G para algum $u, v \in V$ e se $d[u] = \delta(s, u)$ antes de relaxar o arco (u, v), então $d[v] = \delta(s, v)$ depois de relaxar (u, v)

Propriedades da Relaxação

Path-relaxation property
Se $p = \langle v_0, v_1, \ldots, v_k \rangle$ é um caminho mais curto de $s = v_0$ até v_k, e relaxarmos os arcos de p por ordem $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, então $d[v_k] = \delta(s, v_k)$

Predecessor-subgraph property
Se $d[v] = \delta(s, v)$ para todo o $v \in V$, então o grafo de predecessores é uma árvore dos caminhos mais curtos com raíz s.

Algoritmo de Dijkstra

Intuição

- Algoritmo Greedy - generalização da BFS (pesos não unitários)
- Todos os arcos com pesos não negativos
- Mantém conjunto de vértices S com pesos dos caminhos mais curtos já calculados
- A cada passo seleciona vértice u em $V - S$ com menor estimativa do peso do caminho mais curto
  - vértice u é inserido em S
  - arcos que saem de u são relaxados
- No final, $d[u] = \delta(s, u)$ para cada $u \in V$

Algoritmo de Dijkstra

Dijkstra(G, w, s)
  Initialize-Single-Source(G, s)
  $S \leftarrow \emptyset$
  $Q \leftarrow G.V$
  while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Extract-Min(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$ **do**
      Relax(u, v, w)
    end for
  end while

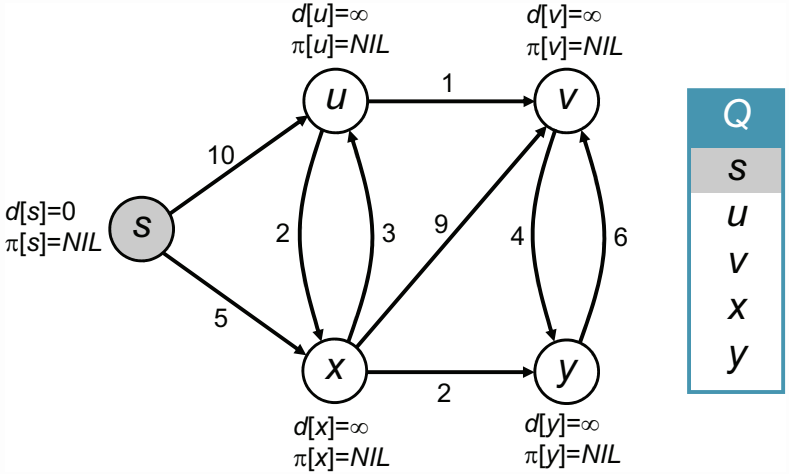
Initialize-Single-Source(G, s)
  for each vertex $v \in G.V$ **do**
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow NIL$
  end for
  $d[s] = 0$

Relax(u, v, w)
  if $d[v] > d[u] + w(u, v)$ **then**
    $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$
  end if

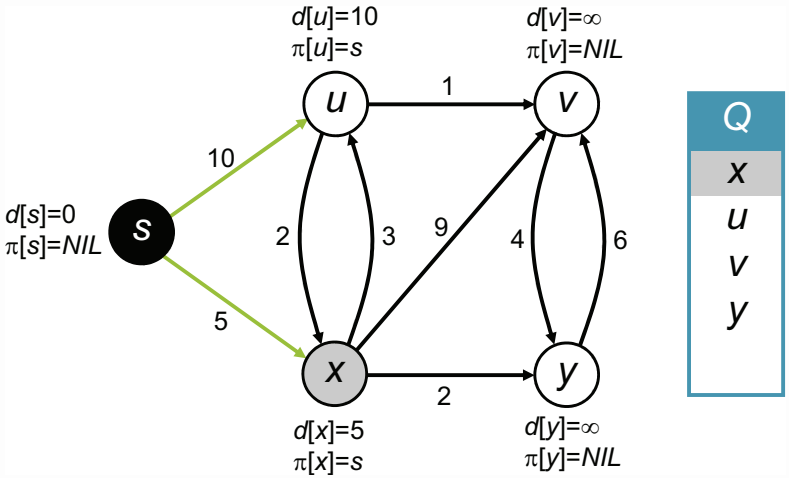
Invariantes

- $Q \leftarrow V - S$
- $d[u] \leftarrow \delta(s, u)$ quando u é adicionado a S

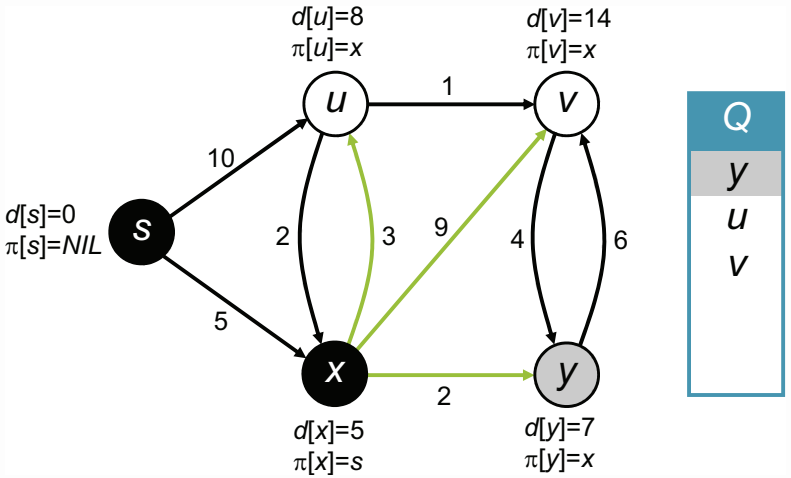
Algoritmo de Dijkstra: Exemplo



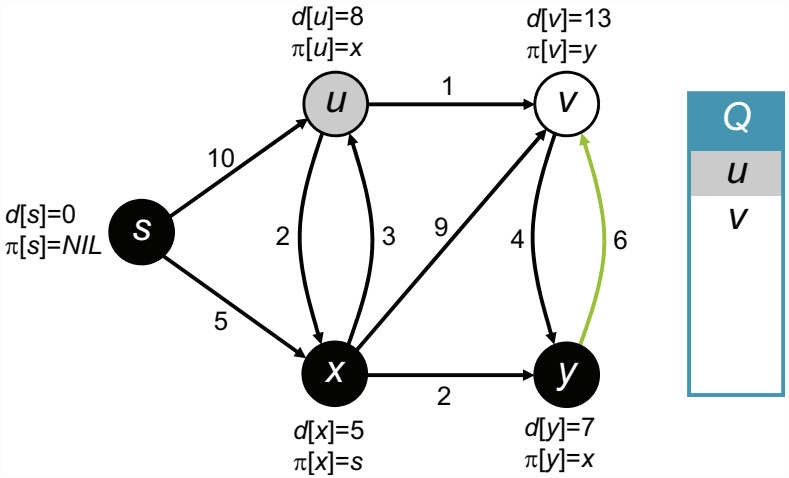
Algoritmo de Dijkstra: Exemplo



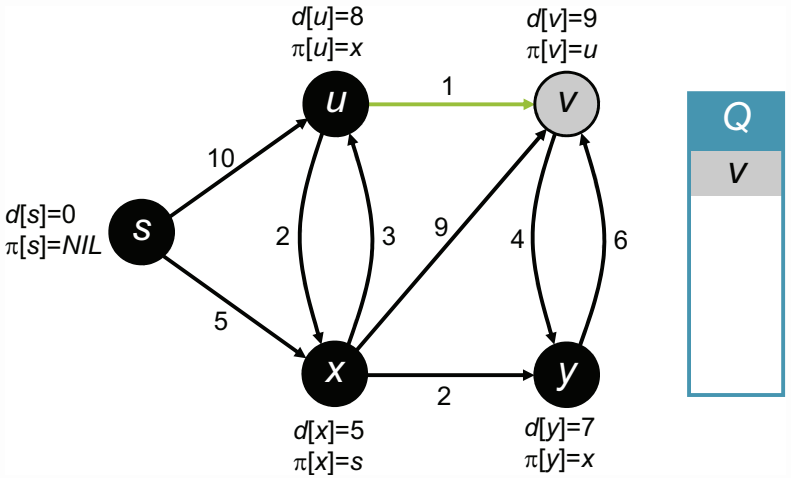
Algoritmo de Dijkstra: Exemplo



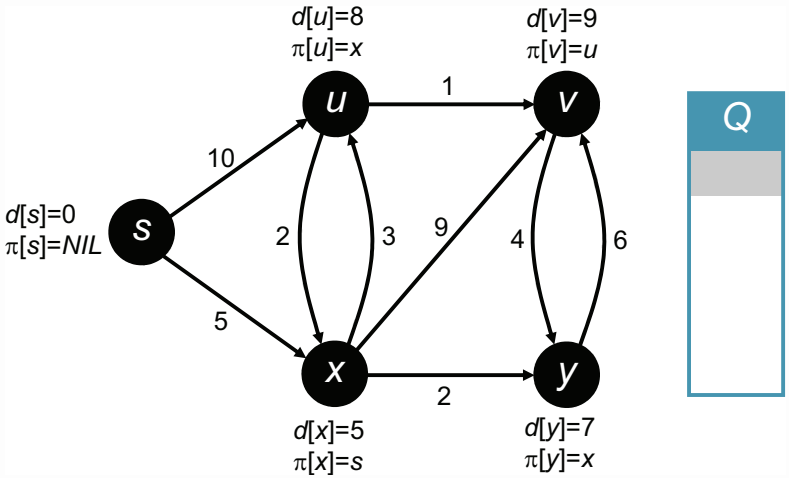
Algoritmo de Dijkstra: Exemplo



Algoritmo de Dijkstra: Exemplo



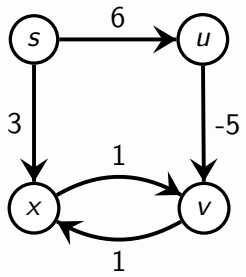
Algoritmo de Dijkstra: Exemplo



Algoritmo de Dijkstra

Pesos Negativos

Os pesos dos arcos têm que ser todos não negativos, para garantir a correção do algoritmo



- Analisar x com $d[x] = 3$
- Analisar v com $d[v] = 4$
- Analisar u com $d[u] = 6$
- No final temos $d[x] = 3 \neq \delta(s, x) = 2$

Algoritmo de Dijkstra

Complexidade

- Fila de prioridade baseada em amontoados (heap)
- Quando um vértice é extraído da fila Q, implica atualização de Q
  - Cada vértice é extraído apenas 1 vez: $|V|$ vértices
  - Actualização de Q: $O(\lg V)$
  - Então, $O(V \lg V)$
- Cada operação de relaxação pode implicar uma atualização de Q
  - Cada arco é relaxado apenas 1 vez: $|E|$ arcos
  - Atualização de Q: $O(\lg V)$
  - Então, $O(E \lg V)$
- Complexidade algoritmo Dijkstra: $O((V + E) \lg V)$

Algoritmo de Dijkstra

Correção do Algoritmo

Provar invariante do algoritmo: $d[u] = \delta(s, u)$ quando u é adicionado ao conjunto S, e que a igualdade é posteriormente mantida

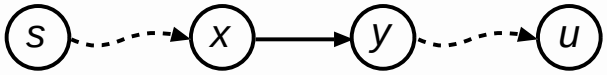
Prova por Contradição

- Assume-se que existe um primeiro vértice u tal que $d[u] \neq \delta(s, u)$ quando u é adicionado a S
- Necessariamente temos que:
  - $u \neq s$ porque $d[s] = \delta(s, s) = 0$
  - $S \neq \emptyset$ porque $s \in S$ quando u é adicionado a S
  - Tem que existir caminho mais curto de s para u, dado que caso contrário teriamos $d[u] = \delta(s, u) = \infty$ (no-path property)

Correcção do Algoritmo (cont.)

Pressuposto: u é o primeiro vértice tal que $d[u] \neq \delta(s,u)$ quando u é adicionado a S

- Seja $p = \langle s, \ldots, x, y, \ldots, u \rangle$ o caminho mais curto de s para u
- Tem que existir pelo menos um vértice do caminho p que ainda não esteja em S, caso contrário, $d[u] = \delta(s,u)$ devido à relaxação dos arcos que compõem o caminho p



Correcção do Algoritmo (cont.)

- ...
- Seja (x,y) um arco de p tal que $x \in S$ e $y \notin S$
  - Temos que $d[x] = \delta(s,x)$ porque $x \in S$ e u é o primeiro vértice em que isso não ocorre
  - Temos também que $d[y] = \delta(s,y)$ porque o arco (x,y) foi relaxado quando x foi adicionado a S (convergence property)
  - Como y é predecessor de u no caminho mais curto até u, então $\delta(s,y) \leq \delta(s,u)$, porque os pesos dos arcos são não-negativos
  - Mas se u é adicionado a S antes de y, temos que $d[u] \leq d[y]$. Logo, $d[u] \leq \delta(s,y) \leq \delta(s,u)$. O que contradiz o pressuposto de $d[u] \neq \delta(s,u)$.