Caminhos simples

FATO: Num digrafo G, se existe caminho de u a v, existe *caminho simples* de u a v.

Caminhos simples

FATO: Num digrafo G, se existe caminho de u a v, existe *caminho simples* de u a v.

Logo, todo caminho mínimo é simples.

Caminhos simples

FATO: Num digrafo G, se existe caminho de u a v, existe *caminho simples* de u a v.

Logo, todo caminho mínimo é simples.

FATO: se $P : u \rightsquigarrow v$ é um caminho mínimo, então, qualquer trecho dele é caminho mínimo entre suas pontas.

Arestas com custos

Arestas com custos

EXEMPLO: Vamos associar um custo real a cada arco, e definir, para cada sequência de arcos, seu custo como sendo a soma dos custos dos arcos.

Problema

Dados vértices s e t, encontrar um caminho de custo mínimo de s a t.

Arestas com custos

EXEMPLO: Vamos associar um custo real a cada arco, e definir, para cada sequência de arcos, seu custo como sendo a soma dos custos dos arcos.

Problema

Dados vértices s e t, encontrar um caminho de custo mínimo de s a t.

Arestas com custos

EXEMPLO: Vamos associar um custo real a cada arco, e definir, para cada sequência de arcos, seu custo como sendo a soma dos custos dos arcos.

Problema

Dados vértices s e t, encontrar um caminho de custo mínimo de s a t.

Caminho de comprimento mínimo: custos 1.

Arestas com custos

EXEMPLO: Vamos associar um custo real a cada arco, e definir, para cada sequência de arcos, seu custo como sendo a soma dos custos dos arcos.

Problema

Dados vértices s e t, encontrar um caminho de custo mínimo de s a t.

Caminho de comprimento mínimo: custos 1.

O custo do arco $e = ij$ será denotado $c(e)$ ou $c(i, j)$ conforme conveniente.

Como provar que um caminho é mínimo?

Como provar que um caminho é mínimo?

Um c-potencial é um vetor real y indexado pelos vértices, tal que para todo arco $u \to v$ vale:

$$y_v - y_u \leq c(u, v).$$

Como provar que um caminho é mínimo?

Um c-potencial é um vetor real y indexado pelos vértices, tal que para todo arco $u \to v$ vale:

$$y_v - y_u \leq c(u, v).$$

Daí segue que se $P : u \rightsquigarrow v$, então $y_v - y_u \leq c(P)$.

Como provar que um caminho é mínimo?

Um c-potencial é um vetor real y indexado pelos vértices, tal que para todo arco $u \to v$ vale:

$$y_v - y_u \leq c(u, v).$$

Daí segue que se $P : u \rightsquigarrow v$, então $y_v - y_u \leq c(P)$.
Um arco $u \to v$ é justo para y se $y_v - y_u = c(u, v)$.

Como provar que um caminho é mínimo?

Um c-potencial é um vetor real y indexado pelos vértices, tal que para todo arco $u \to v$ vale:

$$y_v - y_u \leq c(u, v).$$

Daí segue que se $P : u \rightsquigarrow v$, então $y_v - y_u \leq c(P)$.
Um arco $u \to v$ é justo para y se $y_v - y_u = c(u, v)$.

Teorema

Seja G um digrafo com custos c. Suponha que y é um c-potencial P é um caminho de arcos justos para y. Então P é um caminho de custo mínimo do seu início ao seu fim.

Ciclos negativos

Qual a distância de u a u se esse vértice está num ciclo de custo negativo?

Ciclos negativos

Qual a distância de u a u se esse vértice está num ciclo de custo negativo?

Como seria um c-potencial nesse caso?

Ciclos negativos

Qual a distância de u a u se esse vértice está num ciclo de custo negativo?

Como seria um c-potencial nesse caso?

Ciclos negativos atrapalham completamente a busca de caminho mínimo.

Ciclos negativos

Qual a distância de u a u se esse vértice está num ciclo de custo negativo?

Como seria um c-potencial nesse caso?

Ciclos negativos atrapalham completamente a busca de caminho mínimo.

E se exigirmos caminho simples mínimo?

Ciclos negativos

Qual a distância de u a u se esse vértice está num ciclo de custo negativo?

Como seria um c-potencial nesse caso?

Ciclos negativos atrapalham completamente a busca de caminho mínimo.

E se exigirmos caminho simples mínimo?

Fica NP-difícil!

(Pelo menos tão difícil quanto NP-completo)

E se não tem ciclos negativos?

E se não tem ciclos negativos?

FATO: Se existe caminho, existe caminho mínimo que é simples (a menos de ciclos nulos, todos).

E se não tem ciclos negativos?

FATO: Se existe caminho, existe caminho mínimo que é simples (a menos de ciclos nulos, todos).

FATO: Se os custos são positivos e existe caminho, então todo caminho mínimo é simples.

Como lidar com isso?

Como lidar com isso?

Estratégia 1: Algoritmos que ou acham caminhos mínimos ou detectam ciclos negativos.

Como lidar com isso?

Estratégia 1: Algoritmos que ou acham caminhos mínimos ou detectam ciclos negativos.

Algoritmo de Bellman-Ford.

Como lidar com isso?

Estratégia 1: Algoritmos que ou acham caminhos mínimos ou detectam ciclos negativos.

Algoritmo de Bellman-Ford.

Estratégia 2: Impor condições nos dados que garantam a inexistência de ciclos negativos

- Grafos acíclicos.

Como lidar com isso?

Estratégia 1: Algoritmos que ou acham caminhos mínimos ou detectam ciclos negativos.

Algoritmo de Bellman-Ford.

Estratégia 2: Impor condições nos dados que garantam a inexistência de ciclos negativos

- Grafos acíclicos.
- Custos positivos (ou não-negativos).

Como lidar com isso?

Estratégia 1: Algoritmos que ou acham caminhos mínimos ou detectam ciclos negativos.

Algoritmo de Bellman-Ford.

Estratégia 2: Impor condições nos dados que garantam a inexistência de ciclos negativos

- Grafos acíclicos.
- Custos positivos (ou não-negativos).

Como lidar com isso?

Estratégia 1: Algoritmos que ou acham caminhos mínimos ou detectam ciclos negativos.

Algoritmo de Bellman-Ford.

Estratégia 2: Impor condições nos dados que garantam a inexistência de ciclos negativos

- Grafos acíclicos.
- Custos positivos (ou não-negativos).
  Algoritmo de Dijkstra.

Árvore de caminhos mínimos

Árvore de caminhos mínimos

Dado um vértice s, uma árvore de caminhos mínimos a partir de s é uma arborescência orientada a partir de s, em que para todo vértice v o caminho na árvore de s a v é mínimo em G.

Árvore de caminhos mínimos

Dado um vértice s, uma árvore de caminhos mínimos a partir de s é uma arborescência orientada a partir de s, em que para todo vértice v o caminho na árvore de s a v é mínimo em G.

Vai ser descrita usando $v.sob$, como antes.

Árvore de caminhos mínimos

Dado um vértice s, uma árvore de caminhos mínimos a partir de s é uma arborescência orientada a partir de s, em que para todo vértice v o caminho na árvore de s a v é mínimo em G.

Vai ser descrita usando $v.sob$, como antes.

Algoritmos construirão também um potencial tal que todos arcos da árvore são justos.

Árvore de caminhos mínimos

Dado um vértice s, uma árvore de caminhos mínimos a partir de s é uma arborescência orientada a partir de s, em que para todo vértice v o caminho na árvore de s a v é mínimo em G.

Vai ser descrita usando $v.sob$, como antes.

Algoritmos construirão também um potencial tal que todos arcos da árvore são justos.

Como antes, os valores temporários de y_v serão armazenados em $v.d$.

Ajustamento

Um truque a ser muito usado é a seguinte macro (CLRS chama de RELAX), aplicável quando $u \to v$ é um arco:

Ajustamento

Um truque a ser muito usado é a seguinte macro (CLRS chama de RELAX), aplicável quando $u \to v$ é um arco:

AJUSTA(u, v, c)

if $v.d - u.d > c(u,v)$

$v.d = u.d + c(u,v)$

$v.sob = u$

Ajustamento

Um truque a ser muito usado é a seguinte macro (CLRS chama de RELAX), aplicável quando $u \to v$ é um arco:

AJUSTA(u, v, c)

if $v.d - u.d > c(u,v)$

$v.d = u.d + c(u,v)$

$v.sob = u$

- Se as mudanças dos atributos d e sob são feitas só com essa macro, fica garantido que o arco $v \to v.sob$ é justo.

Ajustamento

Um truque a ser muito usado é a seguinte macro (CLRS chama de RELAX), aplicável quando $u \to v$ é um arco:

AJUSTA(u, v, c)

    if $v.d - u.d > c(u,v)$
        $v.d = u.d + c(u,v)$
        $v.sob = u$

- Se as mudanças dos atributos d e sob são feitas só com essa macro, fica garantido que o arco $v \to v.sob$ é justo.
- Nesse ajuste, $v.d$ nunca cresce.

Inicialização

INICIALIZA-FONTE-ÚNICA(G,s)

    for $v \in G.V$
        $v.d = \infty$
        $v.sob = \text{NIL}$
    $s.d = 0$

Digrafos acíclicos

Digrafos acíclicos

Dados G, c, s, construir a árvore de caminhos mínimos a partir de s

Digrafos acíclicos

Dados G, c, s, construir a árvore de caminhos mínimos a partir de s

ARVORE(G, c, s)

Digrafos acíclicos

Dados G, c, s, construir a árvore de caminhos mínimos a partir de s

ARVORE(G, c, s)

Construa uma ordenação topológica para G

Digrafos acíclicos

Dados G, c, s, construir a árvore de caminhos mínimos a partir de s

ARVORE(G, c, s)

Construa o subgrafo H acessado a partir de s junto com uma ordenação topológica

Digrafos acíclicos

Dados G, c, s, construir a árvore de caminhos mínimos a partir de s

ARVORE(G, c, s)

Construa o subgrafo H acessado a partir de s
junto com uma ordenação topológica
INICIALIZA-FONTE-ÚNICA(H,s)
for $u \in H.V$ em ordem topológica:
    for $v \in u.out$:
        AJUSTA(u,v,c)
return H

Funciona porque...

1. Depois que u foi processado, nenhuma aresta apontando u aparece; assim, $u.d$ fica fixo a partir daí.

Funciona porque...

1. Depois que u foi processado, nenhuma aresta apontando u aparece; assim, $u.d$ fica fixo a partir daí.
2. Depois que processamos u, vale $v.d - u.d \leq c(u, v)$ para todo arco $u \to v$, e isso continua valendo até o fim.

Funciona porque...

1. Depois que u foi processado, nenhuma aresta apontando u aparece; assim, $u.d$ fica fixo a partir daí.
2. Depois que processamos u, vale $v.d - u.d \leq c(u, v)$ para todo arco $u \to v$, e isso continua valendo até o fim.
3. Para cada $v \in H.V - \{s\}$, o arco $v \to v.sob$ é justo.

Custos positivos

```
DIJKSTRA(G, c, s)
 1  INICIALIZA-FONTE-ÚNICA(G,s)
 2  S = ∅
 3  Q = {s}
 4  while Q ≠ ∅
 5        u = elemento de Q que minimiza u.d
 6        Q = Q − {u}
 7        S = S ∪ {u}
 8        for v ∈ u.out:
 9              AJUSTA(u,v,c)
10              enfileira ou reajusta v em Q
```

Funciona porque...

1. Depois que u entra em S:

Funciona porque...

1. Depois que u entra em S:
   1. $u.d \leq v.d$ para todo $v \in Q$ até o fim.

Funciona porque...

1. Depois que u entra em S:
   1. $u.d \leq v.d$ para todo $v \in Q$ até o fim.
   2. $u.d$ fica fixo a partir daí.

Funciona porque...

1. Depois que u entra em S:
   1. $u.d \leq v.d$ para todo $v \in Q$ até o fim.
   2. $u.d$ fica fixo a partir daí.
   3. $v.d - u.d \leq c(u, v)$ para todo arco $u \to v$, e isso continua valendo até o fim.

Funciona porque...

1. Depois que u entra em S:
   1. $u.d \leq v.d$ para todo $v \in Q$ até o fim.
   2. $u.d$ fica fixo a partir daí.
   3. $v.d - u.d \leq c(u, v)$ para todo arco $u \to v$, e isso continua valendo até o fim.
2. Para cada $v \in H.V - \{s\}$, o arco $v \to v.sob$ é justo.

Complexidade

Complexidade

Examina cada aresta uma vez: $\mathcal{O}(n + m)$, exceto pela linha 5

Complexidade

Examina cada aresta uma vez: $\mathcal{O}(n + m)$, exceto pela linha 5

Melhor: $\mathcal{O}(m + nL)$ onde L estima o tempo para a linha 5

A linha 5

A linha 5

Q deve ser uma fila de prioridade mínima, comparada pela propriedade .d .

A linha 5

Q deve ser uma fila de prioridade mínima, comparada pela propriedade .d .

Linhas 4 e 6 combinadas: ExtractMin.

A linha 5

Q deve ser uma fila de prioridade mínima, comparada pela propriedade .d .

Linhas 4 e 6 combinadas: ExtractMin.

Ajusta mexe com as prioridades, gasta tempo para aualizar a fila.

A linha 5

Q deve ser uma fila de prioridade mínima, comparada pela propriedade .d .

Linhas 4 e 6 combinadas: ExtractMin.

Ajusta mexe com as prioridades, gasta tempo para aualizar a fila.

Ex: usa uma lista (vetor) para Q. Atualizações são $\mathcal{O}(1)$, linha 5 é $\mathcal{O}(n)$. Complexidade final: $\mathcal{O}(m + n^2) = \mathcal{O}(n^2)$. (bom para grafos densos)

A linha 5

Q deve ser uma fila de prioridade mínima, comparada pela propriedade .d .

Linhas 4 e 6 combinadas: ExtractMin.

Ajusta mexe com as prioridades, gasta tempo para aualizar a fila.

Ex: usa uma lista (vetor) para Q. Atualizações são $\mathcal{O}(1)$, linha 5 é $\mathcal{O}(n)$. Complexidade final: $\mathcal{O}(m + n^2) = \mathcal{O}(n^2)$. (bom para grafos densos)

Ex: usando um heap. Atualizações e linha 5 são $\mathcal{O}(\lg n)$. Complexidade final: $\mathcal{O}(m \lg n)$.