Arestas com custos

Arestas com custos

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

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

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

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.

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.

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 $u \longrightarrow v$ é um arco; modificado com a booleana ajustou:

Ajustamento

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

AJUSTA(u, v, c)

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

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

$v.sob = u$

ajustou = TRUE

Inicialização

INICIALIZA-FONTE-ÚNICA(G,s)

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

Algoritmo de Bellman-Ford

```
BELLMAN-FORD(G,c,s)
1   INICIALIZA-FONTE-ÚNICA(G,s)
2   for i = 1 to n
3         ajustou = FALSE
4         for todos os arcos u → v de G
5               AJUSTA(u, v, c)
6         if not ajustou
7               return Achou árvore
8   return Tem ciclo negativo
```

Por que funciona?

1. Se numa rodada não ajustou,

$$y_u = u.d$$

é um potencial, e para cada u, o arco $u.sob \to u$ é justo.

Por que funciona?

1. Se numa rodada não ajustou,

$$y_u = u.d$$

é um potencial, e para cada u, o arco $u.sob \to u$ é justo.

2. Se não tem ciclo negativo, então,

Por que funciona?

1. Se numa rodada não ajustou,

$$y_u = u.d$$

é um potencial, e para cada u, o arco $u.sob \to u$ é justo.

2. Se não tem ciclo negativo, então,
   1. Para cada u, se existe caminho mínimo $s \rightsquigarrow u$ de comprimento k, então após a rodada k, $u.d$ é o custo de um caminho mínimo.

Por que funciona?

1. Se numa rodada não ajustou,

$$y_u = u.d$$

é um potencial, e para cada u, o arco $u.sob \to u$ é justo.

2. Se não tem ciclo negativo, então,
   1. Para cada u, se existe caminho mínimo $s \rightsquigarrow u$ de comprimento k, então após a rodada k, $u.d$ é o custo de um caminho mínimo.
   2. Se k é o comprimento máximo de um caminho mínimo, o algoritmo para após a rodada $k+1$, com ajustou valendo FALSE.

Complexidade

Complexidade

$$\mathcal{O}(nm)$$

Distâncias entre todos os pares

Distâncias entre todos os pares

É dado um digrafo com custos. Queremos $d(u, v)$ para todos os pares de vértices.

Distâncias entre todos os pares

É dado um digrafo com custos. Queremos $d(u, v)$ para todos os pares de vértices.

Com custos não negativos DIJKSTRA n vezes resolve. Complexidade $\mathcal{O}(n^3)$ ou $\mathcal{O}(nm \lg n)$ conforme a implementação da fila de prioridades.

Distâncias entre todos os pares

É dado um digrafo com custos. Queremos $d(u, v)$ para todos os pares de vértices.

Com custos não negativos DIJKSTRA n vezes resolve. Complexidade $\mathcal{O}(n^3)$ ou $\mathcal{O}(nm \lg n)$ conforme a implementação da fila de prioridades.

Com custos negativos, $n\times$ BELLMAN-FORD, $\mathcal{O}(n^2 m)$.

Distâncias entre todos os pares

É dado um digrafo com custos. Queremos $d(u, v)$ para todos os pares de vértices.

Com custos não negativos DIJKSTRA n vezes resolve. Complexidade $\mathcal{O}(n^3)$ ou $\mathcal{O}(nm \lg n)$ conforme a implementação da fila de prioridades.

Com custos negativos, $n\times$ BELLMAN-FORD, $\mathcal{O}(n^2 m)$.

Precisa ser melhor que isso!

ED

ED

O resultado naturalmente cabe numa tabela $V \times V$.

Assim, é natural representar G por sua matriz de adjacência A com custos:

$$a_{u,v} = \begin{cases} c_{u,v} & \text{se } u \to v \\ 0 & \text{se } u = v \\ \infty & \text{caso contrário} \end{cases}$$

Algoritmo de Floyd-Warshall

1. Numere os vértices $1, 2, \ldots, n$.

Algoritmo de Floyd-Warshall

1. Numere os vértices $1, 2, \ldots, n$.
2. Para $k = 1, 2, \ldots, n$

Algoritmo de Floyd-Warshall

1. Numere os vértices $1, 2, \ldots, n$.
2. Para $k = 1, 2, \ldots, n$
3. Determine matriz D onde D_{ij} é o custo mínimo de um caminho $i \rightsquigarrow j$ cujos vértices internos são $\leq k$

Algoritmo de Floyd-Warshall

1. Numere os vértices $1, 2, \ldots, n$.
2. Para $k = 1, 2, \ldots, n$
3. Determine matriz D onde D_{ij} é o custo mínimo de um caminho $i \rightsquigarrow j$ cujos vértices internos são $\leq k$
4. Devolva D

Vértices internos

Vértices internos

São vértices do caminho, excluindo as pontas.

Vértices internos

São vértices do caminho, excluindo as pontas.

Antes do for, podemos fazer $D = A$, interpretando como $k = 0$.

Vértices internos

São vértices do caminho, excluindo as pontas.

Antes do for, podemos fazer $D = A$, interpretando como $k = 0$.

Para o ítem 3, é só notar que um caminho mínimo como desejado

> Ou um caminho mínimo cujos vértices internos são $\leq k - 1$

Vértices internos

São vértices do caminho, excluindo as pontas.

Antes do for, podemos fazer $D = A$, interpretando como $k = 0$.

Para o ítem 3, é só notar que um caminho mínimo como desejado

> Ou um caminho mínimo cujos vértices internos são $\leq k - 1$
>
> ou a concatenação de caminhos mínimos $i \rightsquigarrow k$, $k \rightsquigarrow j$, que não usam k internamente.

Vértices internos

São vértices do caminho, excluindo as pontas.

Antes do for, podemos fazer $D = A$, interpretando como $k = 0$.

Para o ítem 3, é só notar que um caminho mínimo como desejado

> Ou um caminho mínimo cujos vértices internos são $\leq k - 1$
>
> ou a concatenação de caminhos mínimos $i \rightsquigarrow k$, $k \rightsquigarrow j$, que não usam k internamente.
>
> novo $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$

Floyd-Warshall em sage

Indexação a partir de 0

```
def floyd_warshall(A):
    n = len(A)
    for k in range(n):
        A = [[min(A[i][j], A[i][k]+A[k][j])
              for j in range(n)]
              for i in range(n)]
    return A
```

Floyd-Warshall em sage

Indexação a partir de 0

```
def floyd_warshall(A):
  n = len(A)
  for k in range(n):
    for i in range(n)
      for j in range(n)
        A[i][k] = min(A[i][j],A[i][k]+A[k][j])
  return A
```