

O estudo utilizando apenas este material **não é suficiente** para o entendimento do conteúdo. Recomendamos a leitura das referências no final deste material e a resolução (por parte do aluno) de todos os exercícios indicados.

Grafos

Caminhos mínimos de única origem

Conteúdo

Introdução

- Tipos de problemas

Caminhos mínimos

- Subestrutura ótima

- Ciclos e arestas de pesos negativos

- Representação

- Relaxamento

- Propriedades

Algoritmos

- Ideia

- Algoritmo de Bellman-Ford

- Algoritmo para gaos

- Algoritmo de Dijkstra

Referências

Introdução

- ▶ Como encontrar o caminho mínimo entre duas cidades?
- ▶ Vamos estudar este tipo de problema, que é conhecido como **problema de caminho mínimo**
- ▶ Entrada
 - ▶ Um grafo orientado $G = (V, E)$
 - ▶ Uma função peso $w : E \rightarrow \mathbb{R}$

Definições

- ▶ O **peso do caminho** $p = \langle v_0, v_1, \dots, v_k \rangle$ é

- ▶ a soma dos pesos das arestas no caminho

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

- ▶ Definimos o **peso do caminho mínimo** deste u até v como

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

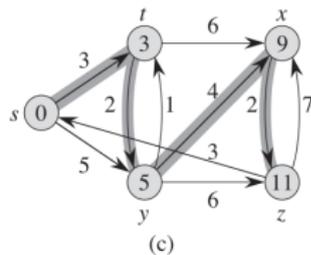
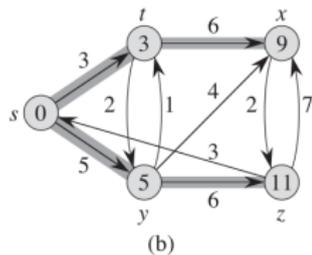
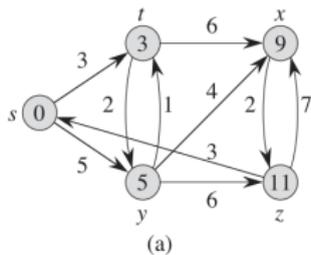
- ▶ Um **caminho mínimo** do vértice u até o vértice v é qualquer caminho p com peso $w(p) = \delta(u, v)$

Tipos de problemas de caminho mínimo

- ▶ **Origem única:** Encontrar um caminho mínimo a partir de uma dada origem $s \in V$ até todo vértice $v \in V$
- ▶ **Destino único:** Encontrar um caminho mínimo até um determinado vértice de destino t a partir de cada vértice v
- ▶ **Par único:** Encontrar o caminho mínimo de u até v
- ▶ **Todos os pares:** Encontrar um caminho mínimo deste u até v para todo par de vértices u e v

Exemplo

- ▶ Exemplo de caminhos mínimos de única origem (comentado em sala, veja o livro para a explicação do exemplo)



Caminhos mínimos

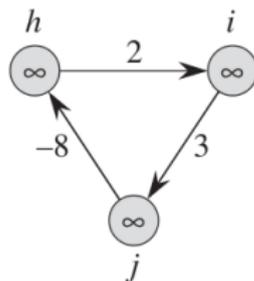
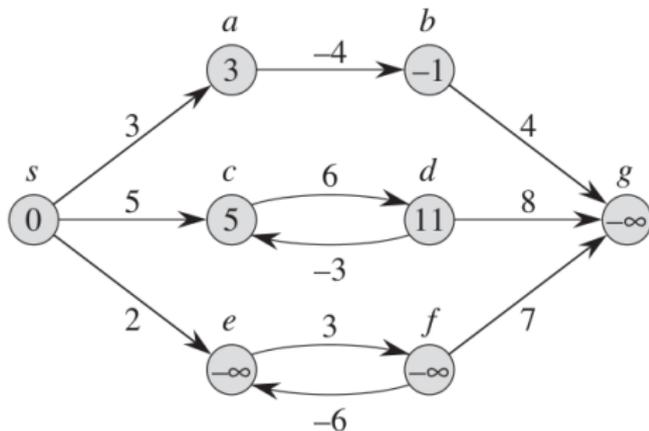
- ▶ Agora veremos algumas características dos caminho mínimos

Subestrutura ótima

- ▶ Em geral os algoritmos de caminhos mínimos se baseiam na seguinte propriedade
 - ▶ **Lema 24.1** Qualquer subcaminho de um caminho mínimo é um caminho mínimo
 - ▶ Como provar este lema? (Comentado em sala, veja o livro para a prova)

Arestas com pesos negativos

- ▶ Não apresentam problemas se nenhum ciclo com peso negativo é alcançável a partir da origem
- ▶ Nenhum caminho da origem até um vértice em um ciclo negativo pode ser mínimo
- ▶ Se existe um ciclo de peso negativo em algum caminho de s até v , definimos $\delta(s, v) = -\infty$



Ciclos

- ▶ Caminhos mínimos podem conter ciclos? Não
 - ▶ Peso negativo, acabamos de descartar
 - ▶ Peso positivo, podemos obter um caminho mínimo eliminando o ciclo
 - ▶ Peso nulo, não existe razão para usar tal ciclo
- ▶ Qualquer caminho acíclico em um grafo $G = (V, E)$ contém no máximo $|V|$ vértices distintos e no máximo $|V| - 1$ arestas

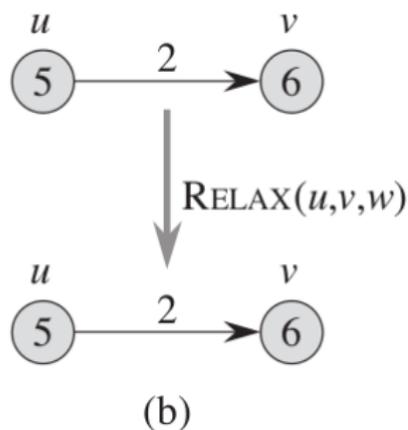
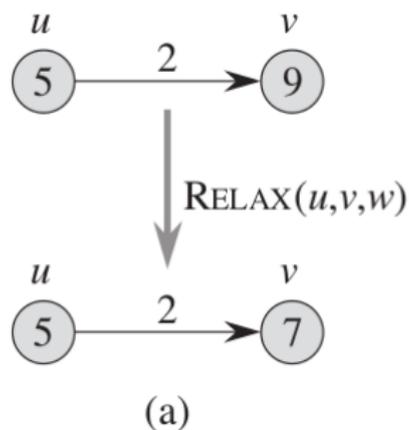
Representação

- ▶ Para cada vértice $v \in V$
 - ▶ $v.d = \delta(s, v)$
 - ▶ Inicialmente $v.d = \infty$
 - ▶ Diminui conforme o algoritmo progride, mas sempre mantém a propriedade $v.d \geq \delta(s, v)$
 - ▶ Vamos chamar $v.d$ de estimativa do caminho mínimo
 - ▶ $v.\pi =$ predecessor de v no caminho mínimo a partir de s
 - ▶ Se não existe predecessor, então $v.\pi = \text{nil}$
 - ▶ π induz uma árvore, a árvore de caminhos mínimos

Relaxamento

- ▶ Sendo os vértices inicializados com a função
`initialize-single-source(G, s)`
 - 1 for cada vértice v em $G.V$
 - 2 $v.d = \text{infinito}$
 - 3 $v.pai = \text{nil}$
 - 4 $s.d = 0$
- ▶ Podemos melhorar a estimativa do caminho mínimo para v , indo através de u e seguindo (u, v) ?

Relaxamento



`relax(u, v, w)`

1 if $v.d > u.d + w(u, v)$

2 $v.d = u.d + w(u, v)$

3 $v.pai = u$

Propriedades

- ▶ Desigualdade de triângulos (Lema 24.10)
 - ▶ Para toda $(u, v) \in E$, temos que $\delta(s, v) \leq \delta(s, u) + w(u, v)$
- ▶ Para as próximas propriedades supomos que
 - ▶ O grafo é inicializado com uma chamada a `initialize-single-source`
 - ▶ O único modo de modificar $v.d$ e $v.\pi$ (para qualquer vértice) e pela chamada de `relax`

Propriedades (continuação)

- ▶ Propriedade do limite superior (Lema 24.11)
 - ▶ Sempre temos $v.d \geq \delta(s, v)$ para todo v . Uma vez que $v.d = \delta(s, v)$, ele nunca muda
- ▶ Propriedade de nenhum caminho (Corolário 24.12)
 - ▶ Se $\delta(s, v) = \infty$, então sempre $v.d = \infty$
- ▶ Propriedade de convergência (Lema 24.14)
 - ▶ Se $s \rightsquigarrow u \rightarrow v$ é um caminho mínimo, $u.d = \delta(s, u)$ e $\text{relax}(u, v, w)$ é chamado, então, em todos os momentos após a chamada, temos $v.d = \delta(s, v)$
- ▶ Propriedade de relaxamento de caminho (Lema 24.15)
 - ▶ Seja $p = \langle v_0, v_1, \dots, v_k \rangle$ o caminho mínimo de $s = v_0$ até v_k , se a função relax for chamada na ordem $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$, mesmo que intercalada com outros relaxamentos, então $v_k.d = \delta(s, v_k)$

Algoritmos

- ▶ Agora veremos alguns algoritmos

Ideia dos algoritmos

- ▶ Os algoritmos que veremos usam a mesma ideia
 - ▶ inicializar os atributos $v.d$ e $v.\pi$
 - ▶ relaxar as arestas
- ▶ Eles diferem na ordem e na quantidade de vezes que cada aresta é relaxada

Algoritmo de Bellman-Ford

- ▶ Resolve o problema para o caso geral, as arestas podem ter pesos negativos
- ▶ Detecta ciclos negativos acessíveis a partir da origem e devolve `false`, caso contrário, devolve `true`
- ▶ Calcula $v.d$ e $v.\pi$ para todo $v \in V$
- ▶ Ideia
 - ▶ Relaxar todas as arestas, $|V| - 1$ vezes

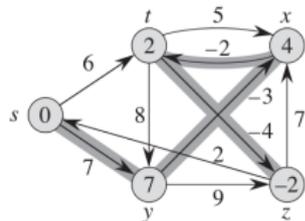
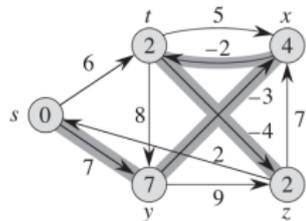
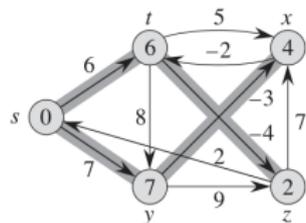
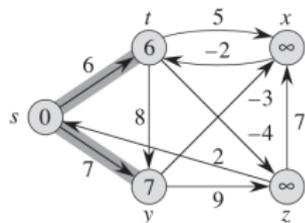
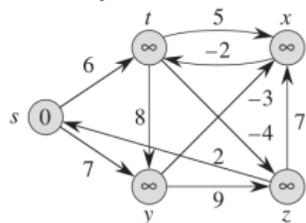
Algoritmo de Bellman-Ford

```
bellman-ford(G, w, s)
1 initialize-single-source(G, s)
2 for i = 1 to |G.V| - 1
3   for cada aresta (u, v) em G.E
4     relax(u, v, w)
5 for cada aresta (u, v) em G.E
6   if v.d > u.d + w(u, v)
7     return false
8 return true
```

- ▶ Análise do tempo de execução
 - ▶ A inicialização na linha 1 demora $\Theta(V)$
 - ▶ Cada uma das $|V| - 1$ passagens das linha 2 a 4 demora o tempo $\Theta(E)$, totalizando $O(V \cdot E)$
 - ▶ O laço das linha 5 a 7 demora $O(E)$
 - ▶ Tempo de execução do algoritmo $\Theta(V \cdot E)$

Algoritmo de Bellman-Ford

Exemplo



As arestas foram relaxadas na ordem

$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$

Algoritmo de Bellman-Ford

- ▶ Por que este algoritmo funciona?
 - ▶ Propriedade de relaxamento de caminho
 - ▶ Seja v acessível a partir de s , e seja $p = \langle v_0, v_1, \dots, v_k \rangle$ um caminho mínimo acíclico entre $s = v_0$ e $v = v_k$. p tem no máximo $|V| - 1$ arestas, e portanto $k \leq |V| - 1$
 - ▶ Cada iteração do laço da linha 2 relaxa todas as arestas
 - ▶ A primeira iteração relaxa (v_0, v_1)
 - ▶ A segunda iteração relaxa (v_1, v_2)
 - ▶ A k -ésima iteração relaxa (v_{k-1}, v_k)
 - ▶ Pela propriedade de relaxamento de caminho
 $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$

Exercícios

- ▶ 24.1-1, 24.1-3, 24.1-4

Exercícios

- 24.1-1 Execute o algoritmo de Bellman-Ford no grafo orientado da figura 24.4, usando o vértice z como origem. Em cada passagem, relaxe as arestas na mesma ordem da figura e mostre os valores de d e π após cada passagem. Agora, mude o peso da aresta (z, x) para 4 e execute o algoritmo novamente, usando s como origem.
- 24.1-3 Dado um grafo orientado ponderado $G = (V, E)$ sem ciclos de peso negativo, seja m o máximo sobre todos os vértices $v \in V$ do número mínimo de arestas em um caminho mínimo de s até v . (Aqui, o caminho mínimo é por peso, não pelo número de arestas.) Sugira uma mudança simples no algoritmo de Bellman-Ford que permita encerrá-lo em $m + 1$ passagens, mesmo se m não é conhecido antecipadamente.

Exercícios

- 24.1-4 Modifique o algoritmo de Bellman-Ford de modo que ele defina $v.d$ como $-\infty$ para todos os vértices v para os quais existe um ciclo de peso negativo em algum caminho da origem até v .

Algoritmo para gao

- ▶ Grafo acíclico orientado (gao) ponderado
- ▶ Caminhos mínimos são sempre bem definidos em um gao, pois não existem ciclos (de peso negativo)
- ▶ Ideia
 - ▶ Relaxar as arestas em uma ordem topológica de seus vértices

Caminhos mínimos de única origem em gaos

```
dag-shortest-paths(G, w, s)
```

```
1 ordenar topologicamente os vértices de G
```

```
2 initialize-single-source(G, s)
```

```
3 for cada vértice u tomado na ordem topológica
```

```
4   for cada vértice v em u.adj
```

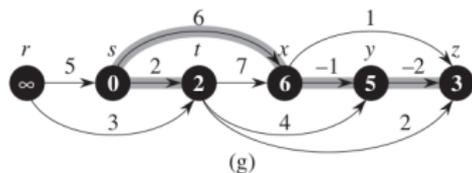
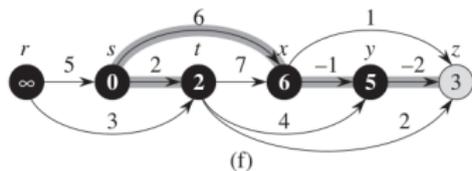
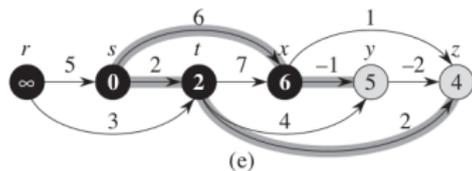
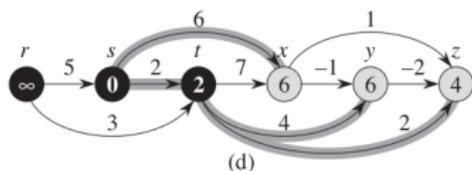
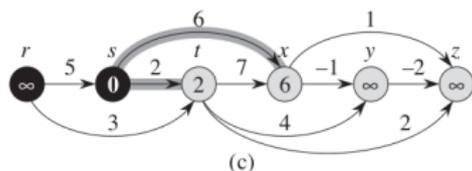
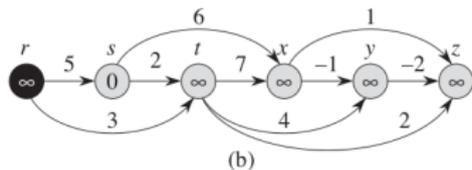
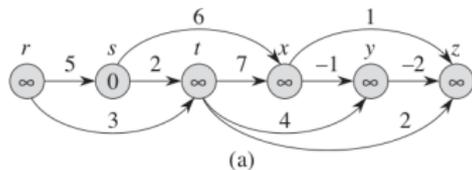
```
5     relax(u, v, w)
```

▶ Análise do tempo de execução

- ▶ A ordenação topológica da linha 1 demora $\Theta(V + E)$
- ▶ `initialize-single-source` na linha 2 demora $\Theta(V)$
- ▶ Nos laços das linhas 2 e 3 a lista de adjacências de cada vértices é visitada apenas uma, totalizando $V + E$ (análise agregada), como o relaxamento de cada aresta custa $O(1)$, o tempo total é $\Theta(E)$
- ▶ Portanto, o tempo de execução do algoritmo é $\Theta(V + E)$
 - ▶ Linear no tamanho de uma representação de lista de adjacências do grafo

Caminhos mínimos de única origem em grafos

Exemplo:



Caminhos mínimos de única origem em grafos

- ▶ Por que este algoritmo funciona?
 - ▶ Como os vértices são processados em ordem topológica, as arestas de qualquer caminho são relaxadas na ordem que aparecem no caminho
 - ▶ Pela propriedade de relaxamento de caminho, o algoritmo funciona corretamente

Aplicação

- ▶ Caminhos críticos na análise de diagramas PERT (*program evaluation and review technique*)
- ▶ As arestas representam serviços a serem executados
- ▶ Os pesos de arestas representam os tempos necessários para execução de determinados serviços
- ▶ (u, v) , v , (v, x) : serviço (u, v) deve ser executado antes do serviço (v, x)
- ▶ Um caminho através desse grafo: sequência de serviços
- ▶ Caminho crítico: é um caminho mais longo pelo grafo
 - ▶ Tempo mais longo para execução de uma sequência ordenada
- ▶ O peso de um caminho crítico é um limite inferior sobre o tempo total para execução de todos os serviços

Aplicação

- ▶ Podemos encontrar um caminho crítico de duas maneiras:
 - ▶ Tornando negativos os pesos das arestas e executando `dag-shortest-paths`; ou
 - ▶ Executando `dag-shortest-paths`, substituindo “ ∞ ” por “ $-\infty$ ” na linha 2 de `initialize-single-source` e “ $>$ ” por “ $<$ ” no procedimento `relax`

Exercícios

- ▶ 24.2-1 a 24.2-4

Exercícios

- 24.2-1 Execute `dag-shortest-paths` sobre o grafo orientado da figura 24.5(a), usando o vértice r como origem.
- 24.2-2 Suponha que mudamos a linha 3 de `dag-shortest-paths` para
3 for os primeiros $|V| - 1$ vértices, tomados em
sequencia ordenada topologicamente
Mostre que o procedimento continue funcionando.

Exercícios

- 24.2-3 A formulação de diagramas PERT dada anteriormente é um tanto antinatural. Seria mais natural que os vértices representassem serviços e as arestas, restrições de sequenciamento; isto é, a aresta (u, v) indicaria que o serviço u deve ser executado antes do serviço v . Então, seriam atribuídos pesos a vértices, e não a arestas. Modifique o procedimento `dag-shortest-paths` de forma que ele encontre um caminho mais longo em um grafo acíclico orientado com vértices ponderados em tempo linear.
- 24.2-4 **Desafio:** Forneça um algoritmo eficiente para contar o número total de caminhos em um grafo acíclico orientado. Analise seu algoritmo.

Algoritmo de Dijkstra

- ▶ Caminho mínimo de única origem em um grafo orientado ponderado
- ▶ Todos os pesos de arestas são não negativos, ou seja $w(u, v) \geq 0$ para cada aresta $(u, v) \in E$

Algoritmo de Dijkstra

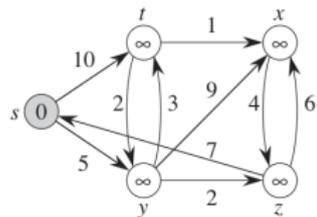
- ▶ Ideia
 - ▶ Essencialmente, uma versão ponderada da busca em largura
 - ▶ Ao invés de uma fila FIFO, usa uma fila de prioridades
 - ▶ As chaves são os pesos dos caminhos mínimos ($v.d$)
 - ▶ Mantém dois conjuntos de vértices
 - ▶ S : vértices cujo caminho mínimo desde a origem já foram determinados
 - ▶ $Q = V - S$: fila de prioridades
 - ▶ O algoritmo seleciona repetidamente o vértice $u \in Q$ com a estimativa mínima de caminhos mínimos, adiciona u a S e relaxa todas as arestas que saem de u

Algoritmo de Dijkstra

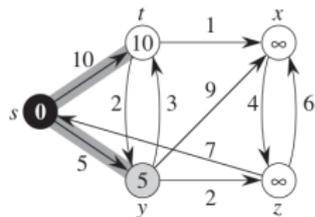
```
dijkstra(G, w, s)
1 initialize-single-source(G, s)
2 S = {}
3 Q = G.V
4 while Q != {}
5   u = extract-min(Q)
6   S = S U {u}
7   for cada vértice v em u.adj
8     relax(u, v, w)
```

Algoritmo de Dijkstra

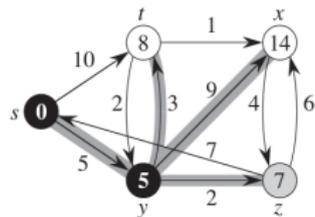
Exemplo:



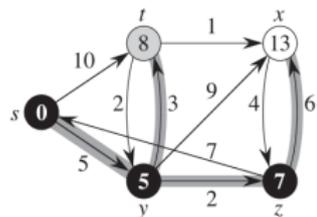
(a)



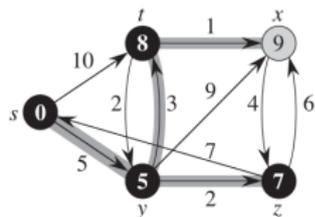
(b)



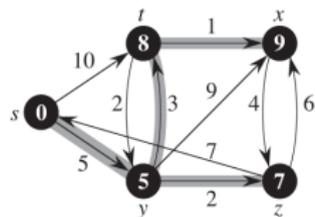
(c)



(d)



(e)



(f)

Algoritmo de Dijkstra

- ▶ Análise do tempo de execução
 - ▶ Operação de fila
 - ▶ insert implícita na linha 3 (executado uma vez para cada vértice)
 - ▶ extract-min na linha 5 (executado uma vez para cada vértice)
 - ▶ decrease-key implícita em relax (executado no máximo de $|E|$ vezes, uma vez para cada aresta relaxada)
 - ▶ Depende da implementação da fila de prioridade

Algoritmo de Dijkstra

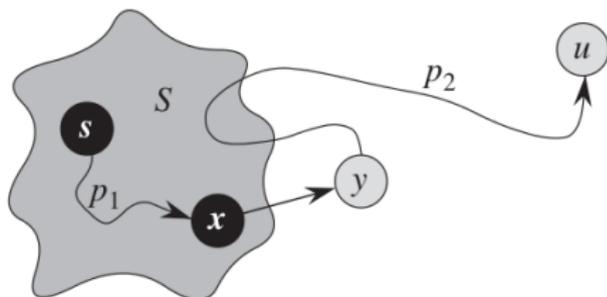
- ▶ Análise do tempo de execução
 - ▶ Arranjos simples
 - ▶ Como os vértices são enumerados de 1 a $|V|$, armazenamos o valor $v.d$ na v -ésima entrada de um arranjo
 - ▶ Cada operação `insert` e `decrease-key` demora $O(1)$
 - ▶ Cada operação `extract-min` demora $O(V)$ (pesquisa linear)
 - ▶ Tempo total de $O(V^2 + E) = O(V^2)$
 - ▶ Heap
 - ▶ Se o grafo é esparso, em particular, $E = o(V^2 / \lg V)$ é prático utilizar um heap binário
 - ▶ O tempo para construir um heap é $O(V)$
 - ▶ Cada operação de `extract-min` e `decrease-key` demora $O(\lg V)$
 - ▶ Tempo total de $O((V + E) \lg V + V)$, que é $O(E \lg V)$ se todos os vértices são acessíveis a partir da origem

Algoritmo de Dijkstra

- ▶ Análise do tempo de execução
 - ▶ Heap de Fibonacci
 - ▶ Cada operação `extract-min` demora $O(\lg V)$
 - ▶ Cada operação `decrease-key` demora o tempo amortizado de $O(1)$
 - ▶ Tempo total de $O(V \lg V + E)$

Algoritmo de Dijkstra

- ▶ Porque este algoritmo funciona?
 - ▶ Invariante de laço: no início de cada iteração do laço while, $v.d = \delta(s, v)$ para todos $v \in S$
 - ▶ Inicialização: $S = \emptyset$, então é verdadeiro
 - ▶ Término: No final, $Q = \emptyset \Rightarrow S = V \Rightarrow v.d = \delta(s, v)$, para todo $v \in V$
 - ▶ Manutenção: precisamos mostrar que $u.d = \delta(s, u)$ quando u é adicionado a S em cada iteração (Comentado em sala, veja o livro para a prova completa)



Exercícios

- ▶ Os números dos exercícios referem-se a 3^o edição
- ▶ Entre parênteses está o número do exercício correspondente na 2^o edição
- ▶ Se não existe informação entre parênteses, significa que o exercício é o mesmo nas duas edições
- ▶ 24.3-1 a 24.3-3, 24.3-4 (não presente), 24.3-6 (22.3-4), 24.3-8 (24.3-6) e 24.3-9 (24.3-7)

Exercícios

- 24.3-1 Execute o algoritmo de Dijkstra sobre o grafo orientado da figura 24.2, primeiro usando o vértice s como origem, e depois usando o vértice y como origem. No Estilo da figura 24.6, mostre os valores de d e π e os vértices no conjunto S após cada iteração do laço while.
- 24.3-2 Forneça um exemplo simples de um grafo orientado com arestas de peso negativo para o qual o algoritmo de Dijkstra produz respostas incorretas. Por que a prova do teorema 24.6 não é válida quando são permitidas arestas de peso negativo?

Exercícios

24.3-3 Suponha que mudamos a linha 4 do algoritmo de Dijkstra para o seguinte:

```
4 while |Q| > 1
```

Essa mudança faz o laço `while` ser executado $|V| - 1$ vezes em lugar de $|V|$ vezes. Esse algoritmo proposto é correto?

24.3-6 Temos um grafo orientado $G = (V, E)$ no qual cada aresta $(u, v) \in E$ tem um valor associado $r(u, v)$, o qual é um número real no intervalo $0 \leq r(u, v) \leq 1$ que representa a confiabilidade de um canal de comunicação do vértice u até o vértice v . Interpretamos $r(u, v)$ como a probabilidade de que o canal de u até v não venha a falhar, e supomos que essas probabilidades são independentes. Forneça um algoritmo eficiente para encontrar o caminho mais confiável entre dois vértices dados.

Referências

- ▶ Thomas H. Cormen et al. Introduction to Algorithms. 3rd edition. Capítulo 24.