DCC206 – Algoritmos 1

Aula 07 – Caminhos mais curtos de origem única

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- O problema de encontrar o caminho mais curto (com o menor custo) ocorre frequentemente na prática
  - Determinar a melhor rota entre dois pontos da cidade
  - Determinar a rota que pacotes de dados têm de seguir em uma rede de comunicações
- Esse problema pode ser resolvido com grafos:
  - Vértices: interseções de vias da cidade; nós da rede
  - Arestas: direções das vias ou conexões da rede
  - Peso: custo associado às vias ou conexões (distância, trânsito, …)

Definições

- O peso de um caminho em um grafo direcionado G=(V,E) é a soma dos pesos de suas arestas
  - $p(C) = \sum_{i=1}^{k} p(v_{i-1}, v_i)$
- A distância entre dois vértices *u* e *v* é:
  - $d(u,v) = \begin{cases} \min p(C): se\ existe\ caminho\ entre\ u\ e\ v \\ \infty: se\ não\ existe\ caminho\ entre\ u\ e\ v \end{cases}$
- Os algoritmos que veremos consideram uma única origem
  - Queremos determinar o caminho mais curto entre um vértice *s* e todos os demais

Variações do problema

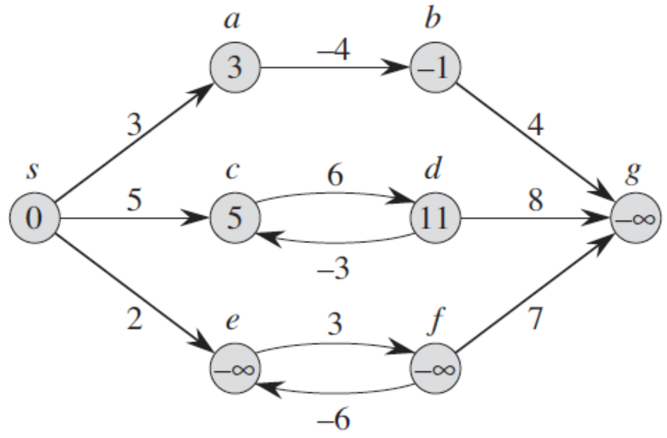
- Apesar do foco ser em caminhos mais curtos em uma única origem, os algoritmos podem ser usados para:
  - Encontrar o caminho mais curto para um único destino
  - Encontrar o menor caminho entre um par de vértices
  - Encontrar o menor caminho entre todos os pares de vértices

Propriedades

- **Lema (subestrutura ótima de um caminho mais curto)**: Dado um grafo direcionado $G=(V,E)$ cujas arestas possuem pesos, seja $C=(v_0,v_2,v_3,...,v_k)$ um caminho mais curto entre os vértices v_0 e v_k. O caminho $(v_i,v_{i+1},…,v_j)$ entre os vértices v_i e v_j para $0 \leq i \leq j \leq k$ é um caminho mais curto entre v_i e v_j. Em outras palavras, *todo subcaminho de um caminho mais curto é um caminho mais curto*.
- **Prova**: O caminho mais curto entre v_0 e v_k, *C*, pode ser decomposto em $C_1 = (v_0,v_2,...v_i)$, $C_2=(v_i,v_{i+1},...,v_j)$, e $C_3=(v_j,v_{j+1},...,v_k)$. Logo, $p(C) = p(C_1) + p(C_2) + p(C_3)$. Suponha que exista um caminho C_2' com custo menor que C_2. Portanto, existe outro caminho entre v_0 e v_k $(C_1C_2'C_3)$ com custo menor que C, contradizendo a hipótese de que C é um menor caminho.

Propriedades

- As arestas com pesos negativos podem representar problemas para os algoritmos
- Se o caminho mais curto entre s e um vértice v passa por um ciclo com custo negativo, então dizemos que $d(s, v) = -\infty$
- Alguns algoritmos admitem arestas negativas nos grafos, outros não.
    - Bellman-Ford x Dijkstra
- Caminhos mais curtos não possuem ciclos



Propriedades

- Associaremos, ao conjunto de vértices, um vetor *p* que armazenará uma estimativa (limite superior) de menor caminho entre *s* e os demais
- Os algoritmos avaliarão a existência de caminhos mais curtos que a estimativa atual para um vértice v
  - Caso exista um caminho até v passando por u, então o antecessor de v é atualizado, assim como seu custo
  - Essa técnica é chamada de **relaxamento**
- **relaxar**(u,v,p)
  - se p[v] > p[u] + p(u,v)
    - antecessor[v] = u
    - p[v] = p[u] + p(u,v)

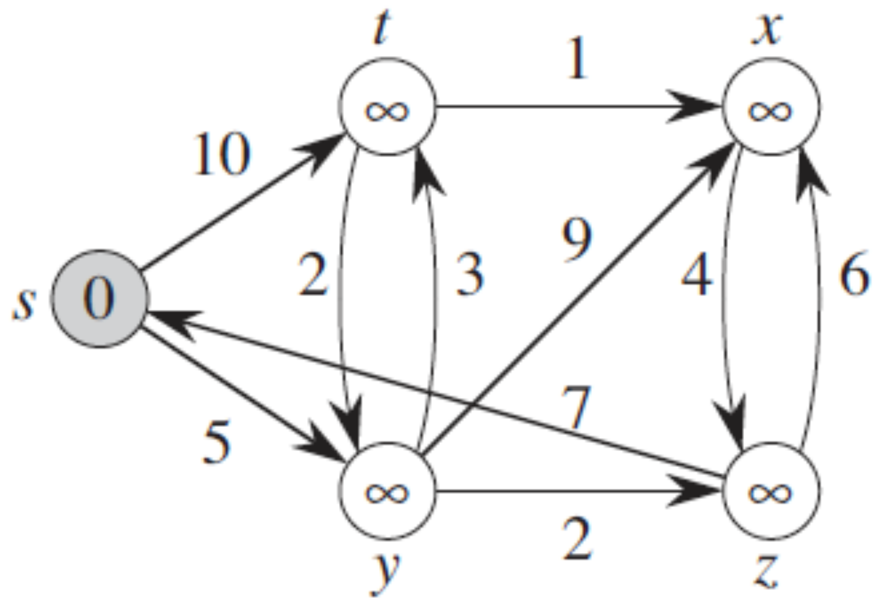
Algoritmo de Dijkstra

- O algoritmo de Dijkstra não admite arestas negativas
- Dijkstra se assemelha ao algoritmo de Prim e também usa a estratégia gulosa.
    - Também é demonstrado que ele encontra a solução ótima
- O algoritmo mantém um vetor de antecessores e de pesos estimados os quais são inicializados com -1 e infinito (exceto p[0] = 0)
- O algoritmo tenta adicionar vértices ao conjunto solução escolhendo aqueles com o menor custo
    - Inicialmente o conjunto solução é vazio

Algoritmo de Dijkstra

- Cria-se um heap mínimo dos vértices com o peso como chave
- Enquanto heap não estiver vazio
  - Remover vértice com menor custo (min)
  - Para cada vértice v adjacente a min
    - Relaxar(min,v,p)

Algoritmo de Dijkstra



Algoritmo de Dijkstra

- A inicialização do vetor de antecessores e estimativa de custo dos vértices tem custo O(|V|)
- O laço é executado para todos os vértices, logo tem custo O(|V|)
- Cada iteração faz uma chamada remover o menor elemento do heap, gerando uma atualização. Isso impõe custo O(lg |V|).
- Assim como na DFS, os adjacentes a um vértice v são avaliados. Ou seja, no total, todas as arestas do grafo são avaliadas
- Dessa forma, serão feitos, no pior caso, O(|E|) relaxamentos. Cada relaxamento incorre custo O(lg |V|)
- Portanto, o custo total do laço é O((|V|+|E|) lg |V|)
  - Ou seja, para grafos conexos, O(|E| lg |V|)

Algoritmo de Dijkstra

- Para demonstrar a corretude do algoritmo, vamos considerar, assim como no algoritmo de Prim, uma partição dos vértices em explorados e a explorar (S, V-S).
- Vamos demonstrar por indução no número de vértices que o menor caminho entre os vértices de S é computado corretamente.
- Para $|S|=1$, sabemos que somente a origem s pertencerá ao conjunto. Logo o algoritmo computa corretamente o menor caminho entre todos os vértices de S.
- Vamos supor, como hipótese de indução, que o algoritmo tenha computado corretamente os menores caminhos para $k \geq 1$ vértices

Algoritmo de Dijkstra

- Seja v o vértice explorado na k+1-ésima iteração
- Suponha que o antecessor de v seja u
- Como u pertence a S, sabemos que p[u] armazena o custo do menor caminho de s para u
- Vamos demonstrar que qualquer outro caminho C' entre s e v possui custo maior que o caminho C = $s \rightsquigarrow u,v$
- O caminho C' deve percorrer uma sequência de vértices em S até encontrar um vértice em V-S para depois alcançar v
- Suponha que esse vértice seja y e que seu antecessor seja x
- Como x está em S, p[x] é o custo do menor caminho s-x.
- Na iteração k+1, y foi considerado entre os vértices a serem explorados mas v foi escolhido
  - Ou seja, $p[x] + p(x,y) \geq p[y] \geq p[v]$
- Como não existem arestas com peso negativo, o caminho C já tem custo menor que o subcaminho s-x,y. Pela subestrutura ótima do problema, ele não pode ser menor.
- Assim, o caminho de s-u,v é o menor caminho de s para v

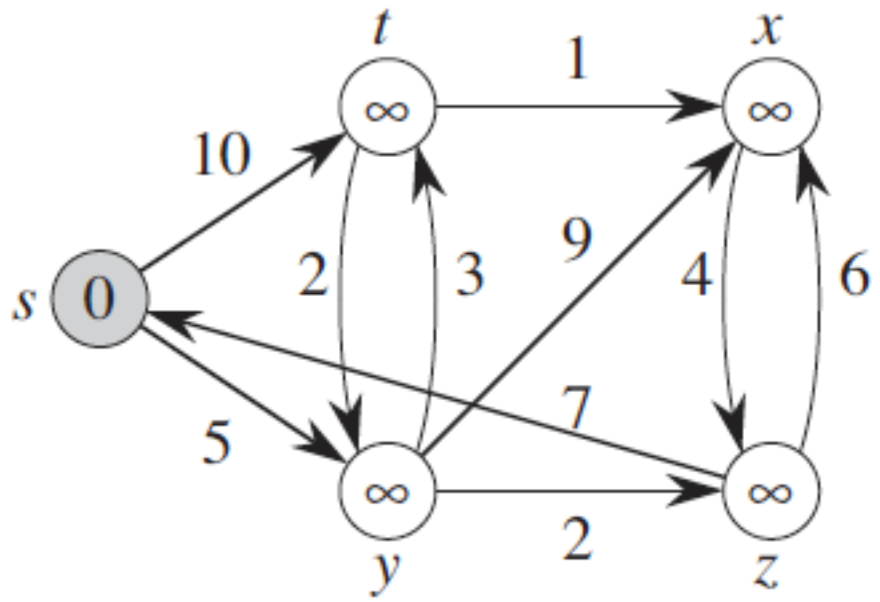
Algoritmo de Bellman-Ford

- O algoritmo de Bellman-Ford (Bellman-Ford-Moore) aceita arestas negativas
- Além de computar o caminho mais curto, ele também detecta ciclos negativos
- O algoritmo retorna falso, caso haja ciclos negativos; verdadeiro caso contrário
- O algoritmo relaxa as arestas do grafo |V|-1 vezes
- Se após todas as iterações ainda existirem vértices com potencial para relaxamento, então existe ciclo negativo no grafo

Algoritmo Bellman-Ford

- O algoritmo inicializa os vetores de antecessores e peso com -1 e infinito (exceto p[0] = 0) tal como em Dijkstra
- Para i=0 até |V|-1 //relaxamento
  - Para cada aresta (u,v) em E
    - Relaxar(u,v,p)
- Para cada aresta (u,v) em E //verifica ciclos negativos
  - Se p[v] > p[u] + p(u,v)
    - Retorna falso
- Se não existem ciclos negativos, retornar verdadeiro

Algoritmo de Bellman-Ford



Algoritmo de Bellman-Ford

- A intuição da demonstração da corretude está na seguinte ideia
- Todo caminho mínimo terá no máximo |V|-1 arestas
- A cada iteração, obtemos o menor caminho considerando, no máximo, i arestas
- Portanto, se após |V|-1 iterações ainda for possível reduzir o custo de algum caminho, existe algum ciclo de custo negativo nesse grafo
- Em relação ao custo computacional, executa O(|V|) relaxações em todas as arestas. Logo, o custo é O(|V||E|)
  - O algoritmo é mais caro que o de Dijkstra mas não assume que os pesos são positivos

Leitura

- Capítulo 24 (CLRS)
- Seções 14.2 e 14.3 (Goodrich e Tamassia)
- Seção 4.4 (SW) (contém otimizações que podem ser feitas no algoritmo Bellman-Ford para diminuir seu custo)