DCC206 – Algoritmos 1

Aula 17 – Programação Dinâmica – Parte 3

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Vimos anteriormente os algoritmos de Dijkstra e Bellman-Ford para computação de menor caminho a partir de uma única origem
- O custo de Dijkstra é menor que de Bellman-Ford, porém o segundo admite arestas negativas
- Esses algoritmos podem ser usados para resolver o problema do menor caminho entre todos os pares de vértices
  - Rodar o algoritmo para cada vértice do grafo
  - Dijkstra: O(VE lg V)
  - Bellman-Ford: O(V^4)

Introdução

- Felizmente, podemos fazer melhor do que isso
- Os algoritmos que veremos assumem, em sua maioria, que o grafo é representado por uma matriz de adjacência
- A matriz W_{nxn} contém o peso das arestas do grafo direcionado $G=(V,E)$, onde $|V|= n$
- $w[u,v] = \infty$, se (u,v) não é uma aresta do grafo
- O objetivo é encontrar o menor caminho entre quaisquer vértices u e v
  - Encontrar a matriz D, onde $D[u,v]$ contenha a distância entre u e v
  - Determinar o caminho de custo $D[u,v]$
- Vamos supor que o grafo não tenha ciclos negativos

Caminhos mais curtos e multiplicação de matrizes

- O primeiro algoritmo que discutiremos é um algoritmo de programação dinâmica que se assemelha à multiplicação de matrizes
- Como todo algoritmo de programação dinâmica, temos que:
  1. Caracterizar a estrutura de uma solução ótima
  2. Definir recursivamente o valor de uma solução ótima
  3. Desenhar uma abordagem bottom-up para calcular o valor da solução ótima
  4. Construir a solução ótima usando as informações calculadas

Caminhos mais curtos e multiplicação de matrizes

- Como vimos anteriormente, todos os subcaminhos de um caminho mais curto são caminhos mais curtos
- Suponha que *p* seja o caminho mais curto entre *u* e *v*
- Suponha que p tenha no máximo m arestas
- Se u = v, então p tem zero arestas e custo zero
- Se u ≠ v, então podemos decompor p em um caminho p’ entre u e k mais a aresta (k,v)
  - p’ tem no máximo m-1 arestas e é o menor caminho entre u e k
  - Logo, d[u,v] = d[u,k] + w[k,v]

Caminhos mais curtos e multiplicação de matrizes

- Seja $l_{uv}^{(m)}$ o custo do caminho entre u e v com, no máximo, m arestas
- Para m = 0, $l_{uv}^{(m)} = \begin{cases} 0, se\ u = v \\ \infty, se\ u \neq v \end{cases}$
- Para m > 0, o caminho mais curto é o menor entre o caminho entre u e v usando no máximo m-1 arestas, e a alternativa indo até um vértice k, depois seguindo pela aresta (k,v) com custo w[k,v]
- $l_{uv}^{(m)}$= min($l_{uv}^{(m-1)}, \min\{l_{uk}^{(m-1)} + w[k,v]\}) = \min\{l_{uk}^{(m-1)} + w[k,v]\}$

Caminhos mais curtos e multiplicação de matrizes

- Como, por hipótese, o grafo não tem ciclos negativos, o caminho mais curto deve ser simples, ou seja, ter n-1 arestas (caso exista)
- Portanto, distancia(u,v) = $l_{uv}^{(m)}$= $l_{uv}^{(m+1)} = l_{uv}^{(m+2)}$ ...
- Dessa forma, temos de preencher a matriz $L^{(m)}$ para m=0 até n-1

Caminhos mais curtos e multiplicação de matrizes

- É fácil perceber que $L^{(1)} = W$
- A função a seguir estende os caminhos mais curtos já conhecidos com mais uma aresta, preservando os de menor custo

```
Extend-shortest-path(L,W)
    para i = 1 até n
        para j = 1 até n
            L'[i,j] = ∞
            para k = 1 até n
                L'[i,j] = min(L'[i,j], L[i,k] + W[k,j])
    retornar L'
```

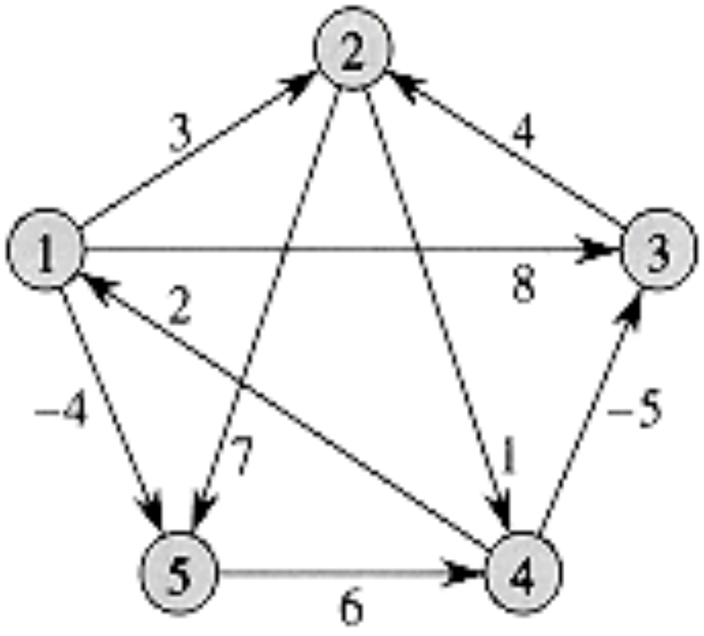
Caminhos mais curtos e multiplicação de matrizes

- O custo do algoritmo anterior é $O(n^3)$
- Se trocarmos a função min por soma e soma por multiplicação, percebemos que o algoritmo é muito similar ao algoritmo de multiplicação de matrizes
  - Em C++ podemos sobrecarregar os operadores e implementá-lo diretamente como multiplicação de matrizes
- Assim, podemos expressar as matrizes L através de multiplicações de matrizes
  - $L^{(i)} = L^{(i-1)}W = W^i$
- Como devemos aplicar o algoritmo n vezes, o custo total é $O(n^4)$

Caminhos mais curtos e multiplicação de matrizes

- Nosso interesse não é computar todas as matrizes L
- Podemos reduzir o custo reorganizando as multiplicações
  - Podemos fazer teto(lg n-1) multiplicações
  - Calculando, W^2, W^4, W^8, …, W^{2^k} até que $2^k > n-1$
- Essa versão otimizada terá custo $O(n^3 \lg n-1)$
- De fato, é possível reduzir ainda mais o custo do algoritmo para $O(n^3)$ explorando outras propriedades do problema para desenhar uma abordagem dividir-e-conquistar (ver Goodrich e Tamassia)

Caminhos mais curtos e multiplicação de matrizes



Algoritmo de Warshall (Fecho transitivo)

- O fecho transitivo de um grafo direcionado G=(V,E) é um grafo G*=(V,E*) em que (u,v) pertence a E* se existe caminho entre u e v em G
- Existem diversas aplicações que fazem uso do fecho transitivo de um grafo
  - Um processador de planilhas deve atualizar todas as células que referenciam uma dada célula quando seu valor é alterado
- Embora o problema possa ser resolvido com n aplicações da busca em largura (profundidade), estudaremos o algoritmo proposto por Warshall em 1962, pois, além de ser mais eficiente, tem grande relação com a busca por caminhos mais curtos

Algoritmo de Warshall (Fecho transitivo)

- O algoritmo avalia os caminhos entre pares de vértices restringindo os vértices intermediários que podem ser incluídos no caminho
- Durante a iteração k, o algoritmo avalia caminhos entre vértices u e v cujos vértices intermediários têm, no máximo, índice k
- Apesar de Warshall não ter explicitamente mencionado programação dinâmica, seu algoritmo se tornou um exemplo de aplicação da técnica

Algoritmo de Warshall (Fecho transitivo)

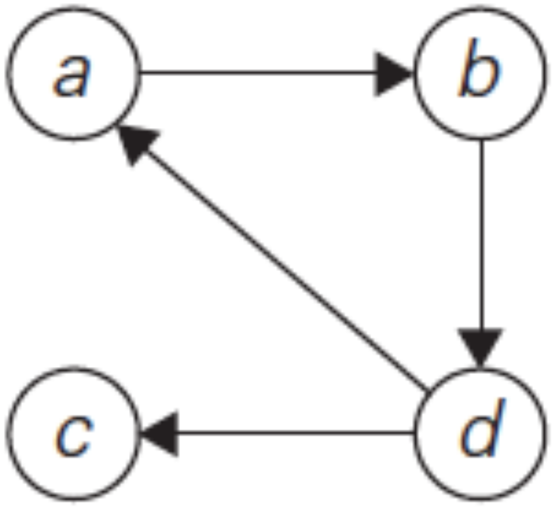
- O algoritmo inicia com k=0 e incrementa o valor até que todos os vértices possam ser intermediários em algum caminho
- Considere um caminho p entre os vértices u e v com vértices intermediários no máximo igual a k
  - p = u, vértices com índices menores ou iguais a k, v
- Existem duas possibilidades para p, incluir ou não k
  - Se p não inclui k, então o caminho entre u e v tem vértices no máximo iguais a k-1
  - Se $k \in p$, então p = u, vértices menores que k, k, vértices menores que k, v

Algoritmo de Warshall (Fecho transitivo)

- Se $r_{uv}^{(k)}$ indicar se há caminho entre u e v com vértices intermediários menores ou iguais a k, pelo dedução anterior temos
  - $r_{uv}^{(k)} = r_{uv}^{(k-1)} \vee \left(r_{uk}^{(k-1)} \wedge r_{kv}^{(k-1)}\right)$
- A relação de recorrência permite a implementação direta do algoritmo de programação dinâmica

```
Warshall(A)
    R = A
    para k = 1 até n
        para i = 1 até n
            para j = 1 até n
                R[i,j] = R[i,j] || (R[i,k] && R[k,j])
    retornar R
```

Algoritmo de Warshall (Fecho transitivo)



Algoritmo de Floyd

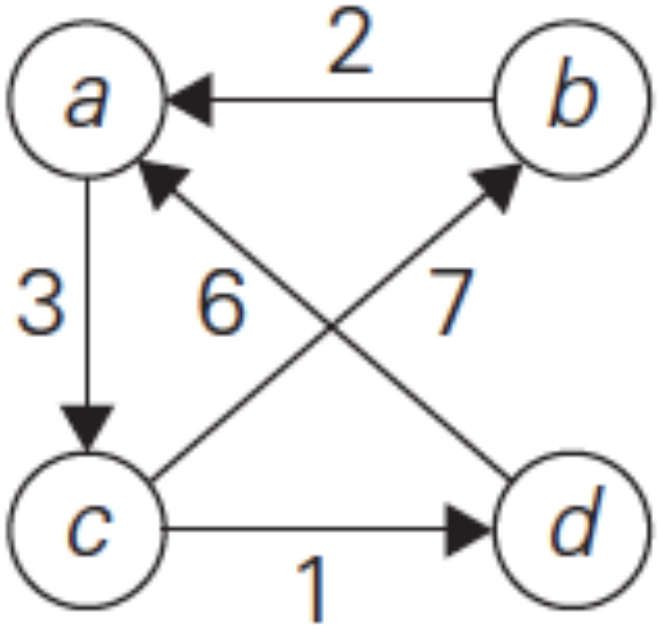
- O algoritmo de Floyd serve para encontrar o menor caminho entre todos os pares de vértices de um grafo direcionado cujas arestas possuem pesos
- Floyd propôs o algoritmo também 1962 e teve como inspiração o algoritmo de Warshall (cujo artigo, de acordo com Levitin, foi referenciado por Floyd)
- A adaptação sugerida por Floyd é, ao invés de simplesmente avaliar a existência do caminho passando por k, verificar qual deles possui o menor custo

Algoritmo de Floyd

- Assim, a relação de recorrência é:
  - $d_{uv}^{(k)} = \min(d_{uv}^{(k-1)}, d_{uk}^{(k-1)} + d_{kv}^{(k-1)})$

```
Floyd(W)
    D = W
    para k = 1 até n
        para i = 1 até n
            para j = 1 até n
                D[i,j] = min(D[i,j], D[i,k] + D[k,j])
    retornar D
```

Algoritmo de Floyd



Leitura

- Seção 8.4 (Levitin)
- Capítulo 25 (CLRS)
- Seção 14.5 (Goodrich e Tamassia)