Algoritmo em Grafos: Caminho mais curto (Parte II)

Caminhos mais curto entre dois pontos numa rede viária

Problema muito importante na prática

- Exemplo: caminho mais curto entre dois pontos num mapa de estradas Não se conhece o algoritmo mais eficiente a resolver este problema do que a resolver o mais geral (de um vértice para todos os outros) Portanto, encontra-se o caminho mais curto da origem para todos os outros, e seleciona-se depois o caminho da origem para o destino pretendido

Otimização: parar assim que chegar a vez de processar o vértice de destino

- Num mapa de estradas, ajuda para distâncias curtas, mas não para distâncias longas

Método base (rede viária)

- Rede viária pode ser representada por um grafo dirigido em que
  - os vértices representam interseções
  - as arestas representam vias (possivelmente de sentido único)
  - os pesos representam distâncias, tempos, custos, etc
- O algoritmo de Dijkstra é a base para encontrar o caminho mais curto entre dois pontos s e t, parando-se a pesquisa quando o próximo nó a processar é o nó t
- Uma vez que o algoritmo processa os vértices por distâncias crescentes ao vértice de partida, é inspecionado um círculo em torno de s de raio igual à distância entre s e t.

Otimizações

- Mas os mapas de estradas são enormes
- Algoritmo de Dijkstra pode demorar muitos segundos ou minutos a encontrar o caminho mais curto em trajetos de longa distância
- Otimizações que não exigem pré-processamento conseguem ganhos de desempenho modestos (até 10x)
- Com pré-processamento, conseguem-se ganhos da ordem de 10^3 ou mesmo 10^6
  - Compromisso entre tempo de pré-processamento, tempo de pesquisa, espaço de armazenamento, facilidade de atualização c/info.dinâmica

Pesquisa bidirecional

- Executar o algoritmo de Dijkstra no sentido de s para t e em sentido inverso de t para s (no grafo invertido), alternando entre um e outro
- Terminar quando se vai processar um vértice x já processado na outra direção (podendo o caminho mais curto passar pelo contacto entre os dois círculos ou não)
- Manter a distância do caminho entre s e t: ao processar uma aresta (v, w) tal que w já foi processada na outra direção, verificar se o correspondente caminho caminho s-t melhora essa distância
- Retornar a distância mínima e o caminho correspondente

Área processada $2 * pi * r^2$ em vez de ~ $4 * pi * r^2$ na pesquisa unidirecional
Speedup - 2x

Pesquisa orientada

- Algoritmo A* : escolher para processar o vértice v com valor mínimo de dsv + pivt, para quando se vai processar o vértice t
  - dsv - distância mínima conhecida de s a v (como no algoritmo de Dijkstra)
  - pivt - estimativa por baixo da dist. mínima de v a t (função potencial)
- Em geral, não garante o ótimo
- Em certos casos, garante o ótimo, por exemplo:
  - Pesos das arestas são distâncias em km
  - pivt é a distâncias Euclidiana (em linha reta) de v a t
  - Equivale então a aplicar o algoritmo de Dijkstra com peso das arestas modificados w'uv = wuv + piut + pivt, somando-se no final pist à distância mínima obtida de s para t
  - Pode ser combinado com pesquisa bidirecional
  - Ganho (speedup) na prática é moderado

Redes hierárquicas (highway networks)

- Pré-processamento decompõe a rede em vários níveis hierárquicos
  - Analogia com mapa de estradas nacional e mapas de ruas locais
  - Uma aresta (u, v) é classificada automaticamente como **highway edge** se existe pelo menos um par de nós s e t da rede tal que:
    - o caminho mais curto de s a t passa em (u, v)
    - u está a mais de H nós de distância de s
    - v está a mais de H nós de distância de t
  - H é um parâmetro configurável
  - Aplicável a mais níveis (local, highway, super-highway, ...)
  - Pré-processamento de mapa de USA ou Europa Ocidental pode ser efetuado em tempo da ordem de 15 minutos
- Pesquisa é bidirecional e usa rede mais densa próximo de s e t e mais

esparsa longe de s e t
- Exige pouco espaço adicional: um campo por aresta

Nós de trânsito

- Pré-processamento determina:
  - **nós de trânsito** - nós tal que o caminho mais curto entre quaisquer 2 nós da rede que não estão "muito perto" entre si passa por pelo menos um dos nós de trânsito
    - Exemplo: acessos de auto-estradas
    - Armazenam-se numa tabelas as distâncias entre todos os pares de nós de trânsito
  - **nós de acesso** - para cada nó da rede, são os nós de trânsito mais próximos
    - Tipicamente, há 10 nós de acesso por nó da rede
    - Armazenam-se numa tabela, para cada nó da rede, os nós de acesso e distâncias
    - Na verdade, determinam-se dois conjuntos de nós de acesso: de saída e de entrada
- A pesquisa do caminho mais curto entre dois pontos afastados é reduzida a poucos table lookups, mas exige espaço de armazenamento adicional significativo
  - Obter nós de acesso dos nós de partida (s) e chegada (t)
  - Para cada par, obter distância de s a t em 3 table lookups

Caminho mais curto entre todos os pares de vértices

- Relevante por exemplo para pré-processamentos de mapa de estradas
- Execução **repetida do algoritmo de Dijkstra** (greedy) $O(|V| (|V| + |E|) \log |V|)$:
  - Bom se o grafo for esparso, como é o caso das redes viárias
- **Algoritmo de Floyd-Warshall**, programação dinâmica
  - Melhor que o anterior se o grafo for denso
  - Mesmo em grafos pouco densos pode ser melhor porque o código é mais simples
  - Baseia-se em matriz de adjacências com pesos
  - Calcula matriz de distâncias mínimas e outra de predecessor no caminho mais curto de i para j

Algoritmo de Floyd-Warshall

- Invariante do ciclo principal: em cada iteração k (de 0 a $|V|$), D[i, j] tem a distância mínima do vertice i a j, usando apenas vértices intermédios do conjunto [1, ..., k]

- Inicialização (k = 0):
    - $D[i, j]^{(0)} = W[i, j]$ $P[i, j]^{(0)} = nil$
- Recorrência (k=1, ..., |V|):
    - D[i, j] (k) = min(D[i, j] (k - 1), D[i, k] (k - 1) + D[k, j] (k - 1))
    - Valor de P[i, j] (k) é atualizado conforme o termo mínimo escolhido
- Para minimizar memória, pode-se atualizar a matriz em cada iteração k, em vez de criar uma nova