

# Algoritmos em Grafos: Caminho mais curto (Parte II)

R. Rossetti, A. P. Rocha, L. Ferreira, J. P. Fernandes, F. Ramos, G. Leão  
FEUP, MIEIC, CAL

## Índice

- Caminho mais curto entre dois pontos numa rede viária
  - Método base
  - Pesquisa bidirecional
  - Pesquisa orientada
  - Redes hierárquicas (*highway networks*)
  - Nós de trânsito (*transit-node routing*)
- Caminho mais curto entre todos os pares de vértices
  - Algoritmo de Floyd-Warshall

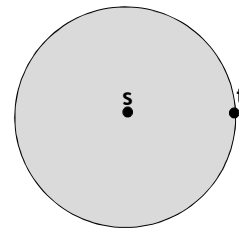
## Caminho mais curto entre dois pontos numa rede viária

## Caminho mais curto entre dois vértices

- Problema muito importante na prática
  - Exemplo: caminho mais curto entre dois pontos num mapa de estradas
- Não se conhece algoritmo mais eficiente a resolver este problema do que a resolver o mais geral (de um vértice para todos os outros)
- Portanto, acha-se o caminho mais curto da origem para todos os outros, e selecciona-se depois o caminho da origem para o destino pretendido
- Otimização: parar assim que chega 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  $t$  (na métrica escolhida)

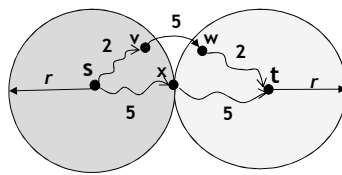


## Otimizações

- Mas os mapas de estradas são enormes
  - Mapa de 17 países da Europa ocidental, do 9º desafio DIMACS sobre caminhos mais curtos (2009), tem cerca de  $19 \times 10^6$  nós e  $23 \times 10^6$  arestas.
  - Em Fev. de 2018, open street maps (mapas disponíveis publicamente) tinha cerca de  $4.3 \times 10^9$  nós
- 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 (*speedup*) de desempenho modestos (até 10x)
- Com pré-processamento, conseguem-se ganhos da ordem de  $10^3$  ou mesmo  $10^6$ , reduzindo tempo de pesquisa para ms ou  $\mu$ s!
  - Compromisso entre tempo de pré-processamento, tempo de pesquisa, espaço de armazenamento, facilidade de atualização c/info. dinâmica

## Pesquisa bidirecional (1/2)

- 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 por  $x$  ou não)
- Manter a distância  $\mu$  do caminho mais curto conhecido entre  $s$  e  $t$ : ao processar uma aresta  $(v, w)$  tal que  $w$  já foi processado na outra direção, verificar se o correspondente caminho  $s$ - $t$  melhora  $\mu$
- Retornar a distância  $\mu$  e o caminho correspondente



Área processada  $\sim 2\pi r^2$ , em vez de  $\sim 4\pi r^2$  na pesquisa unidirecional.

Ganho (*speedup*)  $\sim 2x$

## Pesquisa bidirecional (2/2)

### Dijkstra vs Bi-directional Dijkstra

Comparison On  
Minimum Spanning Tree US Road Network

Dijkstra clássico: 493 passos; Dijkstra bidirecional: 285 passos  
<https://www.youtube.com/watch?v=1oVuQsxkhY0>

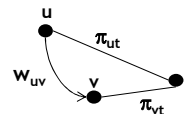
## Pesquisa orientada (1/3)

- **Algoritmo A\***: escolher para processar o vértice  $v$  com valor **mínimo** de  $d_{sv} + \pi_{vt}$ , parando quando se vai processar o vértice  $t$ 
  - $d_{sv}$  - distância mínima conhecida de  $s$  a  $v$  (como no algoritmo de Dijkstra)
  - $\pi_{vt}$  - **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
  - $\pi_{vt}$  é a distância Euclidiana (em linha reta) de  $v$  a  $t$
  - Equivale então a aplicar o algoritmo de Dijkstra com pesos das arestas modificados  $w'_{uv} = w_{uv} - \pi_{ut} + \pi_{vt}$ , somando-se no final  $\pi_{st}$  à distância mínima obtida de  $s$  para  $t$  (ver justificação a seguir).
  - Pode ser combinado com pesquisa bidirecional
  - Ganho (*speedup*) na prática é moderado

## Pesquisa orientada (2/3)

- Justificação:

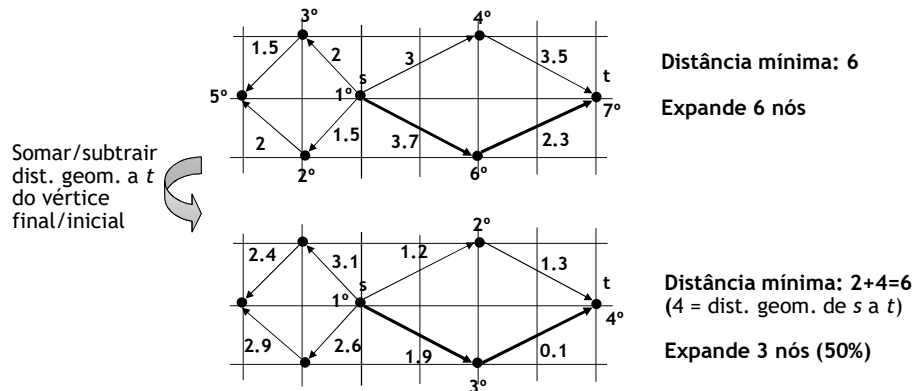
- Pela desigualdade triangular, garante-se  $\pi_{ut} \leq w_{uv} + \pi_{vt}$ , logo  $w'_{uv} = w_{uv} - \pi_{ut} + \pi_{vt} \geq 0$ .



- O peso ao longo de um caminho  $(s, v_1, v_2, \dots, v_k)$ , fica igual ao do grafo original, acrescido de  $\pi_{st} - \pi_{v_k t}$  (pois os potenciais intermédios cancelam-se)
- Logo, escolher o vértice  $v$  com menor  $d_{sv} + \pi_{vt}$  no grafo modificado (A\*), é o mesmo que escolher o vértice com menor  $d_{sv}$  no grafo original (Dijkstra)

## Pesquisa orientada (3/3)

### Exemplo

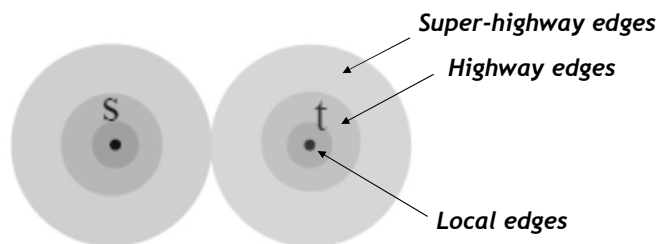


## Redes hierárquicas (*highway networks*) (1/2)

- 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 (por, exemplo, 40)
  - Aplicável a mais níveis (local, *highway*, *super-highway*, etc.)
  - Pré-processamento de mapa de USA ou Europa Ocidental pode ser efetuado em tempo da ordem de 15 minutos.

## Redes hierárquicas (*highway networks*) (2/2)

- Pesquisa é bidirecional e usa rede mais densa próximo de  $s$  e  $t$  e mais esparsa longe de  $s$  e  $t$
- A pesquisa realiza-se em tempo da ordem de 1ms
- Exige pouco espaço adicional: um campo por aresta

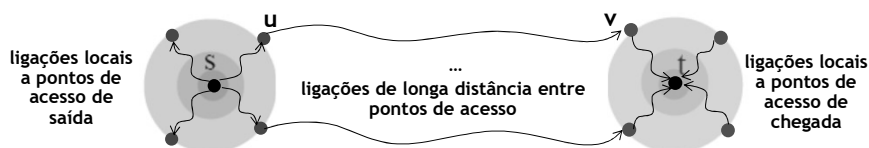


## Nós de trânsito (*transit-node routing*)

- 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
    - Há cerca de  $10^4$  nós de trânsito na Europa Ocidental e USA
    - Armazenam-se numa tabela 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: nós de saída (*forward*,  $A_F$ ) e nós de entrada (*backward*,  $A_B$ )

## Nós de trânsito (*transit-node routing*)

- A pesquisa do caminho mais curto entre dois pontos afastados é reduzida a poucos *table lookups*, e realizada em tempo da ordem de  $10\mu s$  (mas exige espaço de armazenamento adicional significativo)
  - Obter nós de acesso dos nós de partida ( $s$ ) e chegada ( $t$ )
  - Para cada par (nó de acesso inicial ( $u$ ), nó de acesso final ( $v$ )), obter distância de  $s$  a  $t$  em 3 *table lookups*



$$d_{min}(s,t) = \min \{d(s,u) + d(u,v) + d(v,t) \mid u \in A_F(s), v \in A_B(t)\}$$

com  $A_F(s)$ ,  $A_B(t)$ ,  $d(x,y)$  pré-processados

## Caminho mais curto entre todos os pares de vértices



## Caminho mais curto entre todos os pares de vértices

- Relevante por exemplo para pré-processamento de mapa de estradas
- Execução repetida do algoritmo de Dijkstra (ganancioso):  
 $O(|V| (|V| + |E|) \log |V|)$ 
  - Bom se o grafo for esparso ( $|E| \sim |V|$ ), como é o caso das redes viárias
- Algoritmo de Floyd-Warshall, programação dinâmica:  $\Theta(|V|^3)$ 
  - Melhor que o anterior se o grafo for denso ( $|E| \sim |V|^2$ )
  - Mesmo em grafos pouco densos pode ser melhor porque o código é mais simples
  - Baseia-se em matriz de adjacências  $W[i,j]$  com pesos ( $\infty$  quando não há aresta; 0 quando  $i = j$ )
  - Calcula matriz de distâncias mínimas  $D[i,j]$  e matriz  $P[i,j]$  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 vértice  $i$  a  $j$ , usando apenas vértices intermédios do conjunto  $\{1, \dots, k\}$
- Inicialização ( $k=0$ ):  

$$D[i,j]^{(0)} = W[i,j] \quad P[i,j](0) = \text{nil}$$
- Recorrência ( $k=1, \dots, |V|$ ):  

$$D[i,j]^{(k)} = \min( D[i,j]^{(k-1)}, D[i,k]^{(k-1)} + D[k,j]^{(k-1)} )$$
  - 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

## Referências e mais informação

- “Introduction to Algorithms”, 3rd Edition, T.H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein., MIT Press, 2009
  - Chapter 25 (All-Pairs Shortest Paths)
- Engineering Fast Route Planning Algorithms. P. Sanders, D. Schultes. In: Demetrescu C. (eds) Experimental Algorithms. WEA 2007. Lecture Notes in Computer Science, vol. 4525. Springer, Berlin, Heidelberg, 2007
- Efficient Point-to-Point Shortest Path Algorithms. Andrew V. Goldberg (Microsoft Research), Chris Harrelson (Google), Haim Kaplan (Tel Aviv University), Renato F. Werneck (Princeton University), 2006.
- Root Planning in Road Networks. Dominik Schultes, PhD Dissertation, Karlsruhe Institute of Technology, 2008