## CAMINHOS MÍNIMOS ENTRE TODOS OS PARES DE VÉRTICES (Cormen, §25.2)

Dado um grafo simples e não-direcionado, cujas arestas possuem pesos reais não-negativos, desejamos computar as distâncias entre todos os pares de vértices do grafo, sendo a distância entre "u" e "v" a menor soma de todos os pesos das arestas de um caminho entre "u" e "v". A solução deve ser fornecida por meio de uma matriz D[1..n][1..n], sendo D[u][v] a distância entre "u" e "v". O grafo de entrada é também fornecido por meio de uma matriz A[1..n][1..n], tal que:

* Se {u,v} é uma aresta do grafo, então A[u][v] é o peso da aresta.
* Se u = v, então A[u][v] = 0.
* Em caso contrário, A[u][v] = infinito.

É imediato observar que o problema acima pode ser resolvido aplicando-se o algoritmo de caminhos mínimos de Dijkstra (Cormen, §24.3) "n" vezes, uma para cada vértice do grafo. Nesse caso, como o algoritmo de Dijkstra pode ser implementado de forma a executar em tempo $O(n^2)$ (basta usar um vetor no lugar de uma lista de prioridades), então o algoritmo resultante pode ser implementado de forma a executar em tempo $O(n^3)$. Abaixo, nós mostramos que o problema pode ser resolvido via programação dinâmica em tempo $\theta(n^3)$. O tempo do algoritmo de programação dinâmica não é menor na versão acima do problema; entretanto, ele também se aplica ao caso em que o grafo é direcionado e no qual as arestas podem ter pesos negativos (exceto que não pode haver circuito de peso negativo); nesse caso a estratégia anterior via Dijkstra não se aplica, devido às arestas poderem ter pesos negativos. Ainda assim, é possível utilizar o algoritmo de Bellman-Ford (Cormen, §24.1) ao invés do algoritmo de Dijkstra; entretanto, no pior caso, o algoritmo de Bellman-Ford executa em tempo $\theta(n^3)$, e portanto o algoritmo resultante executaria em tempo $\theta(n^4)$. Verificamos, portanto, que as soluções por programação dinâmica levam a algoritmos significativos para o problema.

## SOLUÇÕES POR PROGRAMAÇÃO DINÂMICA

Um ponto de partida interessante para a solução do problema é a observação de que, para quaisquer vértices distintos "u" e "v", temos:

$$D[u][v] = \text{mín}\{ D[u][j] + A[j][v] : j \neq v \} .$$

Entretanto, um algoritmo recursivo diretamente baseado na equação acima é circular, pois, para j != u,v, calcular D[u][j] por meio da equação acima implica utilizar D[u][v] para obter D[u][j], ao passo que já estamos utilizando D[u][j] para obter D[u][v].

Essa circularidade pode ser contornada de mais de uma maneira. Uma delas é começar calculando as distâncias num subgrafo induzido por um pequeno conjunto de vértices, e então progressivamente incluir vértices no subgrafo em questão até se chegar ao grafo inteiro. Uma opção conveniente é fazer o subgrafo inicial conter apenas um vértice, por exemplo o vértice 1; nesse caso, o único par de vértices desse subgrafo é o par {1,1}, e nós já sabemos que D[1][1] = 0. O algoritmo abaixo materializa essa estratégia:

```
=============================================================================
Algoritmo: cam_mín_todos_pares_PD
Entrada:   matriz A[1..n][1..n] de reais não-negativos/infinito.
Saída:     matriz de reais não-negativos/infinito.
-----------------------------------------------------------------------------
01. PARA u DE 1 A n
02. | PARA v DE 1 A n
03. | | D[u][v] := A[u][v]  // 0 se u = v, infinito se u ∉ N[v].
04. PARA v DE 2 A n
05. | PARA a DE 1 A v-1
06. | | PARA b DE 1 A v-1
07. | | | D[v][a] := mín{ D[v][a], A[v][b] + D[b][a] }
08. | | D[a][v] := D[v][a]
09. | PARA a DE 1 A v-1
10. | | PARA b DE a+1 A v-1
11. | | | D[a][b] := mín{ D[a][b], D[a][v] + D[v][b] }
12. | | | D[b][a] := D[a][b]
13. RETORNE D.
=============================================================================
```

1. EXERCÍCIOS:

   a) O algoritmo acima não calcula os caminhos mínimos em si, mas apenas os pesos desses caminhos. Estenda-o de forma que ele passe a informar também os caminhos propriamente ditos. A complexidade do algoritmo não deve mudar.

   b) Outra estratégia de programação dinâmica famosa para resolver o problema é a do algoritmo de Floyd-Warshall (Cormen, §25.2). Nela, começa-se por calcular, para cada par de vértices u != v, o melhor caminho de "u" a "v" cujo conjunto de vértices intermediários é vazio. Naturalmente, tal caminho pode consistir apenas numa aresta de "u" a "v", e portanto o peso de tal caminho é A[u][v]. A partir daí, em cada iteração, no início da qual se tem, para cada par {u,v}, o peso do "melhor" caminho de "u" a "v" cujos vértices intermediários estão todos no conjunto {1..k}, computa-se o peso do melhor caminho de "u" a "v" cujos vértices intermediários estão todos no conjunto {1..k+1}. Escreva um algoritmo $O(n^3)$ que materialize essa estratégia.

   c) Modifique o algoritmo acima para o caso de grafos direcionados. Ele funciona caso o grafo possua arestas de peso negativo mas não possua circuitos de peso negativo?

---
SUBESTRUTURA ÓTIMA
---

Uma propriedade essencial para os vários algoritmos programação dinâmica que vimos até aqui é a subestrutura ótima, que é essencialmente o fato de que, nos problemas em questão, toda solução ótima necessariamente possui em si soluções ótimas para instâncias menores do mesmo problema. De fato, observe que os vários algoritmos que consideramos procedem analisando todas as opções para uma certa escolha:

* No problema de caminhos mínimos, um caminho mínimo de "u" até "v" certamente termina com uma aresta de um vértice "j" a "v" (dado que u != v); assim, o algoritmo considera todos os vértices "j" passíveis de precederem "v" num caminho a partir de "u".

* No problema da árvore de busca ótima, toda árvore possui uma raiz; assim, o algoritmo considera, para cada nó, a melhor árvore de busca que é enraizada no nó em questão.

* Etc.

Além disso, observe que, para cada opção considerada para a escolha em questão, os algoritmos de programação dinâmica constroem uma solução para a instância original a partir de uma solução para uma instância menor. Essa estratégia é baseada no fato de que, nos problemas em questão, a solução para a instância original necessariamente é composta por uma solução para a instância menor. Exemplo:

2. LEMA: no problema de caminhos mínimos ponderados, se c = <u, ..., j, v> é um
   ----  caminho mínimo de "u" a "v", então o caminho c' obtido a partir de "c"
   pela mera remoção de "v" é também um caminho mínimo de "u" a "j".

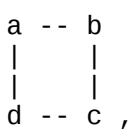
==========
PROVA:

Suponha, por absurdo, que, para um certo "c", o enunciado é falso. Nesse caso, existe um caminho c'' de "u" a "j" tal que peso(c'') < peso(c'). Dado um tal c'', se v ∈ c'', então existe um trecho inicial c''' de c'' que é um caminho de "u" a "v" e é tal que peso(c''') <= peso(c'') < peso(c') <= peso(c) (já que o grafo não possui arestas de peso negativo), logo "c" não é um caminho mínimo de "u" a "v", um absurdo. Finalmente, se v ∉ c'', então o que se obtém pelo acréscimo de "v" ao fim de c'' é um caminho de "u" a "v", e temos

  peso(c''') = peso(c'') + peso({j,v}) < peso(c') + peso({j,v}) = peso(c),

ou seja, "c" não é um caminho mínimo de "u" a "v", um absurdo. Logo, nós concluímos que o enunciado é verdadeiro para todo caminho "c", CQD.
==========

A propriedade ilustrada acima é conhecida como subestrutura ótima. Para os vários problemas que nós resolvemos via programação dinâmica, é fácil mostrar que o problema possui essa propriedade. Entretanto, NEM TODO PROBLEMA POSSUI ESSA PROPRIEDADE! Considere, por exemplo, o problema de caminhos MÁXIMOS não-ponderados. Nesta instância do problema




p = < a, b, c > é um caminho máximo de "a" a "c", mas o trecho inicial <a, b> de "p" não é um caminho máximo de "a" a "b", pois <a, d, c, b> é um caminho maior.