CAMINHO MÍNIMO

PROBLEMA DO CAMINHO MÍNIMO

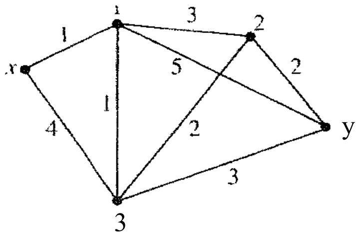
Suponha que temos um grafo simples conexo e com peso, onde os pesos são positivos. Então existe um caminho entre dois nós quaisquer x e y. De fato podem existir muitos desses caminhos. A pergunta é: como encontrar um caminho com peso mínimo? Como peso, representa, muitas vezes, a distância, esse problema ficou conhecido como o problema do "caminho mínimo" (no sentido de "mais curto"). É um problema importante para uma rede de computadores ou de comunicação, onde a informação em um nó tem que ser enviada a outro nó do modo mais eficiente possível, ou para uma rede de transporte, onde os produtos de uma cidade têm que ser enviados a outra.
O problema do caixeiro-viajante é um problema de caminho de peso mínimo com restrições tão severas sobre a natureza do caminho que um tal caminho pode não existir. No problema de caminho mínimo, não há restrições (fora o peso mínimo) sobre a natureza do caminho e, como o grafo é conexo, sabemos que existe um tal caminho. Por essa razão podemos esperar encontrar um algoritmo eficiente para resolver o problema, embora não se conheça um tal algoritmo para o problema do caixeiro-viajante. Existe, de fato, um tal algoritmo.
O algoritmo para o caminho mínimo, conhecido como algoritmo de Dijkstra, funciona da seguinte maneira: Queremos encontrar o caminho de distância mínima de um nó x dado a outro nó y dado. Vamos construir um conjunto (que chamaremos de IN) que contém apenas x inicialmente mas que aumenta durante a execução do algoritmo. Em qualquer instante dado, IN contém todos os nós, cujos caminhos mínimos a partir de x, usando apenas nós em IN, já foram determinados. Para todo nó z fora de IN, guardamos a menor distância d[z] de x àquele nó usando um caminho cujo único nó não pertencente a IN é z. Guardamos, também, o nó adjacente a z nesse caminho, s[z].
Como aumentamos IN, isto é, qual o próximo nó a ser incluído em IN? Escolhemos o nó não pertencente a IN que tem a menor distância d. Uma vez incluído esse nó, que chamaremos de p, em IN, teremos que recalcular d para todos os outros nós restantes fora de IN, já que pode existir um caminho menor (mais curto) a partir de x contendo p do que antes de p pertencer a IN. Se existir um caminho menor, precisaremos atualizar também s[z] de modo que p apareça como o nó adjacente a z no caminho mínimo atual. Assim que y for incluído em IN,IN pára de aumentar. O valor atual de d[y] é a distância correspondente ao menor caminho, cujos nós podem ser encontrados procurando-se y, s[y], s[s[y]], e assim por diante, até percorrer todo o caminho de volta e chegar a x.
É dada, a seguir, urna forma em pseudocódigo desse algoritmo (algoritmo CaminhoMínimo).os dados de entrada correspondem à matriz de adjacência de um grafo G simples e conexo com pesos positivos e nós x e y; o algoritmo descreve o caminho mais curto entre x e y e a distância correspondente. Aqui, caminho mínimo significa caminho de peso mínimo. De fato, supomos que a matriz A é uma matriz de adjacência modificada, onde A[i,j] é o peso do arco entre i e j, se existir, e A [i, j] tem o valor ∞ se não existir um arco, de i para j (o símbolo ∞ denota um número maior do que todos os pesos no grafo).

ALGORITMO DO CAMINHO MÍNIMO

```
CaminhoMínimo (matriz n X n A; nós x, y)
// Algoritmo de Dijkstra. A é uma matriz de adjacência modificada de um grafo simples
// e conexo com pesos positivos; x e y são nós no grafo; o algoritmo escreve os nós do
// caminho mínimo de x para y e a distância correspondente.
Variáveis locais:
conjunto de nós IN             // nós cujo caminho mínimo de x é conhecido
nós z,p                        // nós temporários
vetor de inteiros d            // para cada nó, distância de x usando nós em IN
vetor de nós s                 // para cada nó, nó anterior no caminho mínimo
inteiro DistânciaAnterior      // distância para comparar


  //inicializa o conjunto IN e os vetores d e s
  IN = {x}
  d[x] = 0
  para todos os nós z não pertencentes a IN faça
       d[z] = A[x, z]
       s[z] = x
  fim do para

  //coloca nós em IN
  enquanto y não pertence a IN faça
       //adiciona o nó de distância mínima não pertencente a IN
       p = nó z não pertencente a IN com d[ z] mínimo
       IN = IN ∪ {p}

       //recalcula d para os nós não pertencentes a IN, ajusta s se necessário
       para todos os nós não pertencentes a IN faça
           DistânciaAnterior = d[z]
           d[z] = min(d[z], d[p] + A[p, z])
           se d[z] ≠ DistânciaAnterior então
                     s[z] = p
           fim do se
       fim do para
  fim do enquanto

  //escreve os nós do caminho
  escreva("Em ordem inversa, os nós do caminho são")
  escreva(y)
  z = y
  repita
       escreva(s[z])
       z = s[z]
  até z=x

  //escreve a distância correspondente
  escreva("A distância percorrida é", d[y])
fim de CaminhoMínimo
```

Exercício

Siga o algoritmo CaminhoMínimo para o grafo ilustrado na Figura abaixo. Mostre os valores de p, o conjunto IN, e os valores dos vetores de s em cada passagem do laço de enquanto. Escreva os nós do caminho mínimo e a distância percorrida.



Ao procurar o próximo nó para inclusão em IN no algoritmo CamínhoMínimo, mais de um nó p pode ter um valor mínimo em d, caso em que p pode ser selecionado arbitrariamente. Pode existir, também, mais de um caminho mínimo entre x e y em um grafo.
O algoritmo CamínhoMínimo também funciona para grafos direcionados se a matriz de adjacência estiver na forma apropriada. Também funciona para grafos não conexos; se x e y não estiverem na mesma componente conexa, então d[y] vai permanecer igual a x durante todo o tempo. Depois da inclusão de y em IN, o algoritmo termina e esse valor ∞ para d[y] indica que não existe caminho entre x e y.
Podemos pensar no algoritmo CaminhoMínimo como sendo um algoritmo "míope". Ele não pode ver todo o grafo ao mesmo tempo para escolher os caminhos mínimos; escolhe apenas os caminhos mínimos em relação a IN em cada etapa. Um tal algoritmo é chamado de algoritmo **guloso** - faz o que parece ser melhor baseado em seu conhecimento imediato limitado. Nesse caso, o que parece melhor em determinado instante é, de fato, o melhor ao final.

ALGORITMO DE FLOYD

Inicialmente, esse algoritmo faz uma matriz de custo do grafo. Ou seja, ele verifica a distância entre cada par de vértices. Se existir uma aresta, o valor que ele coloca naquela posição da matriz é o custo da aresta. Se não existir uma aresta entre o par de vértice, ele coloca o valor ∞.
Em seguida, ele verifica se existe um caminho de menor custo entre cada par de vértices, ao passar por um vértice intermediário. Ou seja, suponha um grafo com 5 vértices. De um modo geral, após montar a matriz de distâncias, ele fará 5 iterações:
1ª. Iteração: descobrir se há caminhos que ficam menores ao passar pelo vértice 1
2ª. Iteração: descobrir se há caminhos que ficam menores ao passar pelo vértice 2
3ª. Iteração: descobrir se há caminhos que ficam menores ao passar pelo vértice 3
4ª. Iteração: descobrir se há caminhos que ficam menores ao passar pelo vértice 4
5ª. Iteração: descobrir se há caminhos que ficam menores ao passar pelo vértice 5

```
CaminhoMínimoEntreTodosOsPares (matriz n X n A)
//Algoritmo de Floyd - calcula o caminho mínimo entre dois nós
//quaisquer; inicialmente, A é a matriz de adjacência; ao final,
//A vai conter todas as distâncias dos caminhos mínimos

  para k = 1 até n faça
        para i = 1 até n faça
              para j = 1 até n faça
                    se A[i, k] + A[k, j] < A[i,j] então
                          A[i, j] = A[i, k] + A[k, j]
                    fim do se
              fim do para
        fim do para
  fim do para
fim de CaminhoMinimoEntreTodosOsPares
```

Exercício

Utilizando o algoritmo de Floyd, encontre a matriz de custo mínimo para o grafo abaixo: