Caminhos mínimos

Algoritmo de Floyd-Warshall: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

UVA 10171 – Meeting Prof. Miguel...

Problema

I have always thought that someday I will meet Professor Miguel, who has allowed me to arrange so many contests. But I have managed to miss all the opportunities in reality. At last with the help of a magician I have managed to meet him in the magical **City of Hope**. The city of hope has many roads. Some of them are bi-directional and others are unidirectional. Another important property of these streets are that some of the streets are for people whose age is less than thirty and rest are for the others. This is to give the minors freedom in their activities. Each street has a certain length. Given the description of such a city and our initial positions, you will have to find the most suitable place where we can meet. The most suitable place is the place where our combined effort of reaching is minimum. You can assume that I am 25 years old and Prof. Miguel is 40+.

Problema

Figura 1: Shahriar Manzoor and Miguel A. Revilla (Shanghai, 2005). First meeting after five years of collaboration

Entrada e saída

Input

The input contains several descriptions of cities. Each description of city is started by a integer N, which indicates how many streets are there. The next N lines contain the description of N streets.

The description of each street consists of four uppercase alphabets and an integer. The first alphabet is either 'Y' (indicates that the street is for young) or 'M' (indicates that the road is for people aged 30 or more) and the second character is either 'U' (indicates that the street is unidirectional) or 'B' (indicates that the street is bi-directional). The third and fourth characters, X and Y can be any uppercase alphabet and they indicate that place named X and Y of the city are connected (in case of unidirectional it means that there is a one-way street from X to Y) and the last non-negative integer C indicates the energy required to walk through the street. If we are in the same place we can meet each other in zero cost anyhow. Every energy value is less than 500.

Entrada e saída

After the description of the city the last line of each input contains two place names, which are the initial position of me and Prof. Miguel respectively.

A value zero for N indicates end of input.

Output

For each set of input, print the minimum energy cost and the place, which is most suitable for us to meet. If there is more than one place to meet print all of them in lexicographical order in the same line, separated by a single space. If there is no such places where we can meet then print the line '`You will never meet.`'

Exemplo de entradas e saídas

Sample Input

```
4
Y U A B 4
Y U C A 1
M U D B 6
M B C D 2
A D
2
Y U A B 10
M U C D 20
A D
0
```

Sample Output

```
10 B
You will never meet.
```

Solução com complexidade $O(TV^3)$

- Observe que o valor de N da entrada corresponde ao número de arestas, e não de vértices
- Como cada vértice corresponde a uma letra maiúscula do alfabeto, vale que $V \leq 26$
- Para a solução do problema é necessário computar as distâncias mínimas entre todos os vértices dos dois grafos: o grafo associado ao professor Manzoor, cujas arestas correspondem às pessoas com menos de 30 anos, e o grafo associado ao professor Revilla, para pessoas com mais de 30 anos
- Em seguida, basta avaliar todos os 26 possíveis pontos de encontro, e escolher os que levam ao custo mínimo, que é a soma das distâncias mínimas do ponto de partida de ambos até o destino
- Deve-se ter cuidado em avaliar se ambos podem chegar ao destino, e também com a possibilidade de *autoloops*, que podem levar a respostas erradas, uma vez que o texto atribui custo zero quando ambos estão na mesma posição

Solução com complexidade $O(TV^3)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;
using edge = tuple<int, int, int>;

const int MAX { 26 }, oo { 1000000010 };
int distM[MAX][MAX], distR[MAX][MAX];

void floyd_warshall(int N, int dist[][MAX], const vector<edge>& edges)
{
    for (int u = 0; u < N; ++u)
        for (int v = 0; v < N; ++v)
            dist[u][v] = oo;

    for (const auto& [u, v, w] : edges)
        dist[u][v] = w;

    for (int u = 0; u < N; ++u)
        dist[u][u] = 0;

```

Solução com complexidade $O(TV^3)$

```
22      for (int k = 0; k < N; ++k)
23          for (int u = 0; u < N; ++u)
24              for (int v = 0; v < N; ++v)
25                  dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);
26  }
27
28  vector<int>
29  solve(int m, int r, const vector<edge>& ys, const vector<edge>& ms)
30  {
31      floyd_warshall(MAX, distM, ys);
32      floyd_warshall(MAX, distR, ms);
33
34      int min_cost = oo;
35      vector<int> ans;
36
37      for (int u = 0; u < MAX; ++u)
38      {
39          if (distR[r][u] == oo or distM[m][u] == oo)
40              continue;
41
42          auto cost = distR[r][u] + distM[m][u];
```

Solução com complexidade $O(TV^3)$

```
43
44          if (cost > min_cost)
45              continue;
46
47          if (cost == min_cost)
48              ans.push_back(u);
49          else
50          {
51              min_cost = cost;
52              ans = vector<int> { min_cost, u };
53          }
54      }
55
56      return ans;
57  }
58
59  int main()
60  {
61      ios::sync_with_stdio(false);
62
63      int N;
```

Solução com complexidade $O(TV^3)$

```
65      while (cin >> N, N)
66      {
67          vector<edge> ms, ys;
68
69          while (N--)
70          {
71              string type, dir, x, y;
72              int w;
73              cin >> type >> dir >> x >> y >> w;
74
75              auto u = x[0] - 'A', v = y[0] - 'A';
76
77              if (type == "Y")
78              {
79                  ys.push_back(edge(u, v, w));
80
81                  if (dir == "B")
82                      ys.push_back(edge(v, u, w));
83              } else
84              {
85                  ms.push_back(edge(u, v, w));
```

Solução com complexidade $O(TV^3)$

```
87                  if (dir == "B")
88                      ms.push_back(edge(v, u, w));
89              }
90          }
91
92          string M, R;
93          cin >> M >> R;
94
95          auto m = M[0] - 'A', r = R[0] - 'A';
96          auto ans = solve(m, r, ys, ms);
97
98          if (ans.empty())
99              cout << "You will never meet.\n";
100         else
101         {
102             cout << ans.front() << " ";
103
104             for (size_t i = 1; i < ans.size(); ++i)
105                 cout << (char) (ans[i] + 'A')
106                     << (i + 1 == ans.size() ? '\n' : ' ');
107         }
```

Solução com complexidade $O(TV^3)$

```
108         }
109
110         return 0;
111 }
```

URI 1454 – O País das Bicicletas

Problema

Como você já deve saber, a bicicleta é um dos meios de transportes mais populares da China. Quase todos os chineses possuem a sua, e utilizam-na para trabalho, recreação, e outras atividades.

Após muitos anos pedalando, Mr. Lee já não têm a mesma disposição para encarar as diversas subidas da cidade onde mora. E a cidade em que Mr. Lee vive é extremamente montanhosa. Por razões sentimentais, ele não quer mudar para uma cidade mais plana. Resolveu, então, que tentaria evitar grandes altitudes em seus caminhos mesmo que, para isso, tivesse que pedalar um pouco mais.

Mr. Lee obteve com o serviço topográfico chinês uma coleção de mapas de sua cidade, em que cada rua desses mapas possui a informação da maior altitude encontrada quando trafegada. Tudo que ele precisa fazer agora é determinar rotas que minimizem a altura percorrida entre pares (origem, destino).

Problema

Sabendo que você planeja visitar a cidade em que ele mora no próximo ano, Mr. Lee pediu sua ajuda. Em uma primeira etapa, ele deseja que você implemente um programa que receba mapas topográficos da cidade e uma coleção de pares (origem, destino). Para cada par, seu programa deve imprimir a maior altura encontrada em uma rota entre a origem e o destino. Lembre-se que as rotas devem minimizar tais alturas. As rotas propriamente ditas, serão determinadas em uma segunda etapa (quando você chegar à China para visitá-lo).

Como o transporte utilizado é uma bicicleta, você pode considerar que todas as ruas da cidade são de mão dupla. Não demore, pois Mr. Lee conta com você. :-)

Entrada e saída

Entrada

Seu programa deve estar preparado para trabalhar com diversos mapas, doravante denominados instâncias. Cada instância tem a estrutura que segue.

Na primeira linha são fornecidos dois inteiros n $(0 \leq n \leq 100)$ e m $(0 \leq m \leq 4950)$ que representam, respectivamente, os números de interseções e de ruas. Por razões de clareza, as interseções são numeradas de 1 a n, inclusive; toda rua começa e termina em uma interseção; e não existem interseções fora das extremidades de uma rua.

Nas próximas m linhas são fornecidos três inteiros: i e j $(1 \leq i, j \leq n)$ que indicam a existência de uma rua entre as interseções i e j; e h que representa a maior altitude encontrada quando a rua é trafegada. Esses inteiros estão separados por espaços em branco.

Entrada e saída

Na linha seguinte, é dado um inteiro k $(1 \leq k \leq 50)$ que representa o número de pares (origem, destino) que serão especificados nas próximas k linhas. Cada par é formado por dois inteiros i e j como acima. Isto é, origem e destino são interseções de ruas, e também estão separados por espaços em branco.

Valores $n = m = 0$ indicam o final das instâncias e não devem ser processados.

Saída

Para cada instância solucionada, você deverá imprimir um identificador `Instancia` h em que h é um número inteiro, sequencial e crescente a partir de 1. Nas próximas k linhas, você deve imprimir as maiores alturas encontradas nas rotas entre os k pares (origem, destino) fornecidos, um valor por linha, na ordem da entrada.

Uma linha em branco deve ser impressa após cada instância.

Exemplo de entradas e saídas

Exemplo de Entrada

```
12 17
1 4 4
4 7 6
7 10 6
2 5 4
5 8 5
8 11 2
3 6 5
6 9 3
9 12 1
1 2 1
2 3 9
4 5 3
5 6 7
7 8 7
8 9 2
10 11 1
11 12 2
4
1 5
6 8
6 7
11 10
0 0
```

Exemplo de Saída

```
Instancia 1
4
3
6
1
```

Solução com complexidade $O(TV^3)$

- A maior altura observada no caminho de u a v que minimiza as alturas pode ser obtida através de uma variação do algoritmo de Floyd-Warshall
- A modificação a ser feita é no cálculo do custo do caminho mínimo: ao invés de acumular os custos das arestas, deve ser registrada a menor entre as alturas já observadas, e as alturas do caminho de u a k e do caminho de k a v
- A entrada não especifica dois pontos importantes, cuja interpretação errônea pode levar ao WA
- Em primeiro lugar, não são informados os limites da variável h, a qual pode ser negativa
- Também não é especificado o comportamento do caminho de u a u: como é possível ter alturas negativas, seria possível construir um caminho de u a u passando por v com altura mínima menor do que zero, mas o problema espera zero como resposta nestes casos

Solução com complexidade $O(TV^3)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;
using edge = tuple<int, int, int>;

const int MAX { 110 }, oo { 2000000010 };
int dist[MAX][MAX];

void floyd_warshall(int N, const vector<edge>& edges)
{
    for (int u = 1; u <= N; ++u)
        for (int v = 1; v <= N; ++v)
            dist[u][v] = oo;

    for (int u = 1; u <= N; ++u)
        dist[u][u] = 0;

    for (const auto& [u, v, w] : edges)
        dist[u][v] = w;

```

Solução com complexidade $O(TV^3)$

```
22      for (int k = 1; k <= N; ++k)
23          for (int u = 1; u <= N; ++u)
24              for (int v = 1; v <= N; ++v)
25                  dist[u][v] = min(dist[u][v], max(dist[u][k], dist[k][v]));
26  }
27
28  vector<int>
29  solve(int N, const vector<edge>& es, const vector<ii>& qs)
30  {
31      floyd_warshall(N, es);
32      vector<int> ans;
33
34      for (const auto& [u, v] : qs)
35          ans.push_back(u == v ? 0 : dist[u][v]);
36
37      return ans;
38  }
39
40  int main()
41  {
42      ios::sync_with_stdio(false);
```

Solução com complexidade $O(TV^3)$

```
43
44      int N, M, test = 0;
45
46      while (cin >> N >> M, N | M)
47      {
48          vector<edge> es;
49
50          while (M--)
51          {
52              int u, v, w;
53              cin >> u >> v >> w;
54
55              es.push_back(edge(u, v, w));
56              es.push_back(edge(v, u, w));
57          }
58
59          int K;
60          cin >> K;
61
62          vector<ii> qs;
63
```

Solução com complexidade $O(TV^3)$

```
64          while (K--)
65          {
66              int u, v;
67              cin >> u >> v;
68
69              qs.push_back(ii(u, v));
70          }
71
72          auto ans = solve(N, es, qs);
73
74          cout << "Instancia " << ++test << '\n';
75
76          for (const auto& x : ans)
77              cout << x << '\n';
78
79          cout << '\n';
80      }
81
82      return 0;
83  }
```

Referências

1. UVA 10171 – Meeting Prof. Miguel...
2. URI 1454 – O País das Bicicletas