

Países em guerra

por Alessandro Luna Brasil

Timelimit: 2

No ano 2050, após diversas tentativas da **ONU** de manter a paz no mundo, explode a terceira guerra mundial. Segredos industriais, comerciais e militares obrigaram todos os países a utilizar serviços de espionagem extremamente sofisticados, de forma que em cada cidade do mundo há ao menos um espião de cada país. Esses espiões precisam se comunicar com outros espiões, com informantes e mesmo com as suas centrais durante as suas ações. Infelizmente não existe uma forma segura de um espião se comunicar em um período de guerra, então as mensagens são sempre enviadas em código para que somente o destinatário consiga ler a mensagem e entender o seu significado.

Os espiões utilizam o unico serviço que funciona no período de guerra, os correios. Cada cidade possui uma agência postal onde as cartas são enviadas. As cartas podem ser enviadas diretamente ao seu destino ou a outras agências postais, até que a carta chegue à agência postal da cidade de destino, se isso for possível.

Uma agência postal na cidade **A** pode enviar diretamente uma carta impressa para a agência postal da cidade **B** se houver um acordo de envio de cartas, que determina o tempo, em horas, que uma carta leva para chegar da cidade **A** à cidade **B** (e não necessariamente o contrário). Se não houver um acordo entre as agências **A** e **B**, a agência **A** pode tentar enviar a carta a quantas agências for necessário para que a carta chegue ao seu destino, se isso for possível.

Algumas agências são interligadas por meios eletrônicos de comunicação, como satélites e fibras ópticas. Antes da guerra, essas ligações atingiam todas as agências, fazendo com que uma carta fosse enviada de forma instantânea, mas durante o período de hostilidades cada país passou a controlar a comunicação eletrônica e uma agência somente pode enviar uma carta a outra agência por meio eletrônico (ou seja, instantaneamente) se ela estiver no mesmo país. Duas agências, **A** e **B**, estão no mesmo país se houver uma forma de uma carta impressa enviada de uma das agências ser entregue na outra agência.

O serviço de espionagem do seu país conseguiu obter o conteúdo de todos os acordos de envios de mensagens existentes no mundo e deseja descobrir o tempo mínimo para se enviar uma carta entre diversos pares de cidades. Você seria capaz de ajudá-lo?

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois inteiros separados por um espaço, **N** ($1 \leq N \leq 500$) e **E** ($0 \leq E \leq N^2$), indicando o número de cidades (numeradas de 1 a **N**) e de acordos de envio de mensagens, respectivamente. Seguem-se, então, **E** linhas, cada uma com três inteiros separados por espaços, **X**, **Y** e **H** ($1 \leq X, Y \leq N$, $1 \leq H \leq 1000$), indicando que existe um acordo para enviar uma carta impressa da cidade **X** à cidade **Y**, e que tal carta será entregue em **H** horas.

Em seguida, haverá uma linha com um inteiro **K** ($0 \leq K \leq 100$), o número de consultas. Finalmente, virão **K** linhas, cada uma representando uma consulta e contendo dois inteiros separados por um espaço, **O** e **D** ($1 \leq O, D \leq N$). Você deve determinar o tempo mínimo para se enviar uma carta da cidade **O** à cidade **D**.

Saída

Para cada caso de teste da entrada seu programa deve produzir **K** linhas na saída. A **I**-ésima linha deve conter um inteiro **M** , o tempo mínimo, em horas, para se enviar uma carta na **I**-ésima consulta. Se não houver meio de comunicação entre as cidades da consulta, você deve imprimir "Nao e possivel entregar a carta"(sem acentos).

Imprima uma linha em branco após cada caso de teste.

Exemplo de Entrada

```
4 5
1 2 5
2 1 10
3 4 8
4 3 7
2 3 6
5
1 2
1 3
1 4
4 3
4 1
3 3
1 2 10
2 3 1
3 2 1
3
1 3
3 1
3 2
0 0
```

Exemplo de Saída

```
0
6
6
0
Nao e possivel entregar a carta
10
Nao e possivel entregar a carta
0
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  int destino, origem, vertices = 0;
6  int custo, *custos = NULL;
7  int e = 0;
8
9  void dijkstra(int vertices,int origem,int destino,int *custos)
10 {
11     int i,v, cont = 0;
12     int *ant, *tmp;
13     int *z; /* vertices para os quais se conhece o caminho minimo */
14     double min;
15     double dist[vertices]; /* vetor com os custos dos caminhos */
16
17     /* aloca as linhas da matriz */
18     ant = (int*) calloc (vertices, sizeof(int *));
19     tmp = (int*) calloc (vertices, sizeof(int *));
20     z = (int*) calloc (vertices, sizeof(int *));
21     for (i = 0; i < vertices; i++) {
22         if (custos[(origem - 1) * vertices + i] != -1) {
23             ant[i] = origem - 1;
24             dist[i] = custos[(origem-1)*vertices+i];
25         }
26         else {
27             ant[i] = -1;
28             dist[i] = HUGE_VAL;
29         }
30         z[i] = 0;
31     }
32     z[origem-1] = 1;
33     dist[origem-1] = 0;
34
35     /* Laco principal */
36     do {
37
38         /* Encontrando o vertice que deve entrar em z */
39         min = HUGE_VAL;
40         for (i=0;i<vertices;i++)
41             if (!z[i])
42                 if (dist[i]>=0 && dist[i]<min) {
43                     min=dist[i];v=i;
44                 }
45
46         /* Calculando as distancias dos novos vizinhos de z */
47         if (min != HUGE_VAL && v != destino - 1) {
48             z[v] = 1;
49             for (i = 0; i < vertices; i++)
50                 if (!z[i]) {
51                     if (custos[v*vertices+i] != -1 && dist[v] + custos[v*vertices+i] < dist[i]) {
52                         dist[i] = dist[v] + custos[v*vertices+i];
53                         ant[i] = v;
54                     }
55                 }
56             }
57         } while (v != destino - 1 && min != HUGE_VAL);
58
59     if (min == HUGE_VAL) {
60         printf("Nao e possivel entregar a carta\n");
61     }
62     else {
63         i = destino;
64         i = ant[i-1];
65         while (i != -1) {
66             tmp[cont] = i+1;
67             cont++;
68             i = ant[i];
69         }

```

```

70     }
71
72     printf("%d\n", (int) dist[destino-1]);
73 }
74 }
75
76 void add()
77 {
78     int i, j;
79
80     if (!custos)
81         free(custos);
82     custos = (int *) malloc(sizeof(int)*vertices*vertices);
83     for (i = 0; i <= vertices * vertices; i++)
84         custos[i] = -1;
85
86     for(i=0;i<e;i++){
87         scanf("%d %d %d",&origem,&destino,&custo);
88         custos[(origem-1) * vertices + destino - 1] = custo;
89         if(custos[(origem-1) * vertices + destino - 1] != -1 && custos[(destino-1) *
90             vertices + origem - 1] != -1){
91             custos[(origem-1) * vertices + destino - 1] = 0;
92             custos[(destino-1) * vertices + origem - 1] = 0;
93         }
94     }
95
96
97 void procurar(int orig, int dest)
98 {
99     dijkstra(vertices, orig, dest, custos);
100 }
101
102
103
104 int main(int argc, char **argv) {
105     int i, j, k, orig, dest;
106     char opcao[3], l[50];
107     scanf("%d %d",&vertices,&e);
108     while(1){
109         if(vertices==0)
110             break;
111         add();
112         scanf("%d",&k);
113         for(i=0;i<k;i++){
114             scanf("%d %d",&orig,&dest);
115             procurar(orig, dest);
116         }
117         scanf("%d %d",&vertices,&e);
118         if(vertices!=0)
119             printf("\n");
120     }
121     printf("\n");
122     return 0;
123 }
124

```