SPOJ COURIER

The Courier

Prof. Edson Alves – UnB/FGA

Problema

Byteland is a scarcely populated country, and residents of different cities seldom communicate with each other. There is no regular postal service and throughout most of the year a one-man courier establishment suffices to transport all freight. However, on Christmas Day there is somewhat more work for the courier than usual, and since he can only transport one parcel at a time on his bicycle, he finds himself riding back and forth among the cities of Byteland.

The courier needs to schedule a route which would allow him to leave his home city, perform the individual orders in arbitrary order (i.e. travel to the city of the sender and transport the parcel to the city of the recipient, carrying no more than one parcel at a time), and finally return home. All roads are bi-directional, but not all cities are connected by roads directly; some pairs of cities may be connected by more than one road. Knowing the lengths of all the roads and the errands to be performed, determine the length of the shortest possible cycling route for the courier.

Entrada e saída

Input

The input begins with the integer t, the number of test cases. Then t test cases follow.

Each test case begins with a line containing three integers: $n\ m\ b$, denoting the number of cities in Byteland, the number of roads, and the number of the courier's home city, respectively ($1 \leq n \leq 100$, $1 \leq b \leq n$, $1 \leq m \leq 10000$). The next m lines contain three integers each, the i-th being $u_i\ v_i\ d_i$, which means that cities u_i and v_i are connected by a road of length d_i ($1 \leq u_i, v_i \leq 100$, $1 \leq d_i \leq 10000$). The following line contains integer z – the number of transport requests the courier has received ($1 \leq z \leq 5$). After that, z lines with the description of the orders follow. Each consists of three integers, the j-th being $u_j\ v_j\ b_j$, which signifies that b_j parcels should be transported (individually) from city u_j to city v_j. The sum of all b_j does not exceed 12.

Entrada e saída

Output

For each test case output a line with a single integer – the length of the shortest possible bicycle route for the courier.

Exemplo de entradas e saídas

Sample Input

```
1
5 7 2
1 2 7
1 3 5
1 5 2
2 4 10
2 5 1
3 4 3
3 5 4
3
1 4 2
5 3 1
5 1 1
```

Sample Output

```
43
```

Solução

- Este é um problema bastante interessante, que pode ser modelado como um TSP, porém de forma não óbvia
- A travessia será composta pelas B entregas, sendo que cada pacote consiste em uma entrega individual
- Seja $dp(i, mask)$ a distância mínima para fazer as entregas partindo do vértice i e já tendo entregue os pacotes sinalizados pela máscara binária $mask$
- O custo das "arestas" entre as entregas é dado pela soma da distância entre i e o vértice u onde está o pacote mais a distância entre u e o destino do pacote v
- As distâncias mínimas entre quaisquer dois vértices pode ser computada por meio do algoritmo de Floyd-Warshall
- Serão $O(N \times 2^K)$ estados distintos, onde $K = \sum_i b_i$

Solução $O(N^3 + K^2 2^K)$

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

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

const int MAXN { 110 }, MAXB { 13 }, oo { 1000000010 };

int dist[MAXN][MAXN];
int st[MAXN][1 << MAXB];

int tsp(int i, int mask, int N, int B, const vector<ii>& xs)
{
    if (mask == (1 << N) - 1)
        return dist[i][B];

    if (st[i][mask] != -1)
        return st[i][mask];

    int res = oo;
```

Solução $O(N^3 + K^2 2^K)$

```
21      for (int j = 0; j < N; ++j)
22      {
23          if (mask & (1 << j))
24              continue;
25
26          auto u = xs[j].first, v = xs[j].second;
27          auto cost = dist[i][u] + dist[u][v];
28
29          res = min(res, tsp(v, mask | (1 << j), N, B, xs) + cost);
30      }
31
32      st[i][mask] = res;
33      return res;
34  }
35
36  void floyd_warshall(int N, const vector<vector<ii>>& adj)
37  {
38      for (int u = 1; u <= N; ++u)
39          for (int v = 1; v <= N; ++v)
40              dist[u][v] = oo;
```

Solução $O(N^3 + K^2 2^K)$

```
42      for (int u = 1; u <= N; ++u)
43          dist[u][u] = 0;
44
45      for (int u = 1; u <= N; ++u)
46          for (const auto& [v, w] : adj[u])
47              dist[u][v] = w;
48
49      for (int k = 1; k <= N; ++k)
50          for (int u = 1; u <= N; ++u)
51              for (int v = 1; v <= N; ++v)
52                  dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);
53  }
54
55  int solve(int N, int B, const vector<vector<ii>>& adj, const vector<ii>& xs)
56  {
57      floyd_warshall(N, adj);
58
59      memset(st, -1, sizeof st);
60      return tsp(B, 0, (int) xs.size(), B, xs);
61  }
```

Solução $O(N^3 + K^2 2^K)$

```
63 int main()
64 {
65     ios::sync_with_stdio(false);
66
67     int T;
68     cin >> T;
69
70     while (T--) {
71         int N, M, B;
72         cin >> N >> M >> B;
73
74         vector<vector<ii>> adj(N + 1);
75
76         while (M--) {
77             int u, v, d;
78             cin >> u >> v >> d;
79
80             adj[u].push_back(ii(v, d));
81             adj[v].push_back(ii(u, d));
82         }
```

Solução $O(N^3 + K^2 2^K)$

```
84          int Z;
85          cin >> Z;
86
87          vector<ii> xs;
88
89          for (int i = 0; i < Z; ++i)
90          {
91              int u, v, b;
92              cin >> u >> v >> b;
93
94              while (b--)
95                  xs.push_back(ii(u, v));
96          }
97
98          cout << solve(N, B, adj, xs) << '\n';
99      }
100
101     return 0;
102 }
```