

AtCoder Beginner Contest 143

Problem E: Travel by Car

Prof. Edson Alves

Faculdade UnB Gama

There are N towns numbered 1 to N and M roads. The i -th road connects Town A_i and Town B_i bidirectionally and has a length of C_i .

Takahashi will travel between these towns by car, passing through these roads. The fuel tank of his car can contain at most L liters of fuel, and one liter of fuel is consumed for each unit distance traveled. When visiting a town while traveling, he can full the tank (or choose not to do so). Travel that results in the tank becoming empty halfway on the road cannot be done.

Process the following Q queries:

- The tank is now full. Find the minimum number of times he needs to full his tank while traveling from Town s_i to Town t_i . If Town t_i is unreachable, print -1 .

Há N cidades numeradas de 1 a N e M estradas. A i -ésima estrada conecta a Cidade A_i à Cidade B_i bidirecionalmente e tem comprimento C_i .

Takahashi irá viajar de carro entre estas cidades, passando por estas estradas. O tanque de gasolina de seu carro por conter, no máximo, L litros de gasolina, e um litro de gasolina é consumido para cada unidade de distância viajada. Quando ele visita uma cidade, ele pode encher seu tanque (ou escolher não o fazer). Viagens nas quais o tanque se esvazia ao longo de uma estrada não podem ser feitas.

Responda às seguintes Q consultas:

- O tanque está cheio. Encontre o número mínimo de vezes que ele precisará encher o tanque para viajar da Cidade s_i para a Cidade t_i . Se a Cidade t_i é inatingível, imprima -1 .

Constraints

- ▶ *All values in input are integers.*
- ▶ $2 \leq N \leq 300$
- ▶ $0 \leq M \leq \frac{N(N-1)}{2}$
- ▶ $1 \leq L \leq 10^9$
- ▶ $1 \leq A_i, B_i \leq N$
- ▶ $A_i \neq B_i$
- ▶ $(A_i, B_i) \neq (A_j, B_j)$ (if $i \neq j$)
- ▶ $(A_i, B_i) \neq (B_j, A_j)$ (if $i \neq j$)
- ▶ $1 \leq C_i \leq 10^9$
- ▶ $1 \leq Q \leq N(N-1)$
- ▶ $1 \leq s_i, t_i \leq N$
- ▶ $s_i \neq t_i$
- ▶ $(s_i, t_i) \neq (s_j, t_j)$ (if $i \neq j$)

Restrições

► Todos os valores da entrada são inteiros.

► $2 \leq N \leq 300$

► $0 \leq M \leq \frac{N(N-1)}{2}$

► $1 \leq L \leq 10^9$

► $1 \leq A_i, B_i \leq N$

► $A_i \neq B_i$

► $(A_i, B_i) \neq (A_j, B_j)$ (se $i \neq j$)

► $(A_i, B_i) \neq (B_j, A_j)$ (se $i \neq j$)

► $1 \leq C_i \leq 10^9$

► $1 \leq Q \leq N(N-1)$

► $1 \leq s_i, t_i \leq N$

► $s_i \neq t_i$

► $(s_i, t_i) \neq (s_j, t_j)$ (se $i \neq j$)

Input

Input is given from Standard Input in the following format:

$$\begin{array}{ccc} N & M & L \\ A_1 & B_1 & C_1 \\ \vdots & & \\ A_M & B_M & C_M \\ Q & & \\ s_1 & t_1 & \\ \vdots & & \\ s_Q & t_Q & \end{array}$$

Entrada

A entrada é dada na Entrada Padrão no seguinte formato:

$$\begin{array}{ccc} N & M & L \\ A_1 & B_1 & C_1 \\ \vdots & & \\ A_M & B_M & C_M \\ Q & & \\ s_1 & t_1 & \\ \vdots & & \\ s_Q & t_Q & \end{array}$$

Output

Print Q lines.

The i -th line should contain the minimum number of times the tank needs to be filled while traveling from Town s_i to Town t_i . If Town t_i is unreachable, the line should contain -1 instead.

Saída

Imprima Q linhas.

A i -ésima linha deve conter o número mínimo de vezes que o tanque tem que ser reabastecido para se viajar da Cidade s_i para a Cidade t_i . Se a Cidade t_i é inatingível, imprima o número -1 .

Exemplo de entrada e saída

Exemplo de entrada e saída

3 2 5

Exemplo de entrada e saída

3 2 5



de cidades

Exemplo de entrada e saída

3 2 5



de cidades

1

2

3

Exemplo de entrada e saída

3 2 5



de estradas

1

2

3

Exemplo de entrada e saída

3 2 5



capacidade do tanque

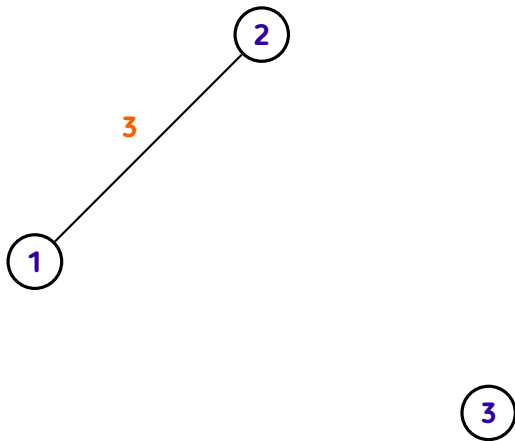
1

2

3

Exemplo de entrada e saída

3 2 5
1 2 3

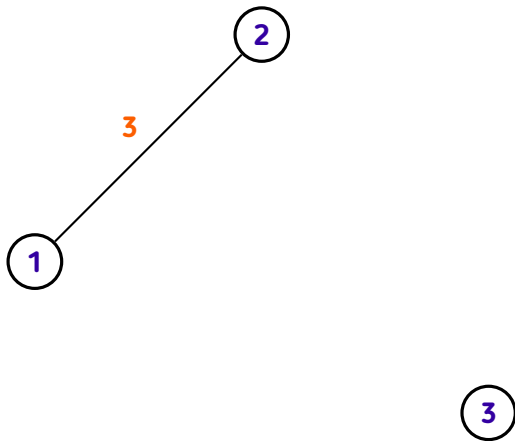


Exemplo de entrada e saída

3 2 5

1 2 3

2 3 3

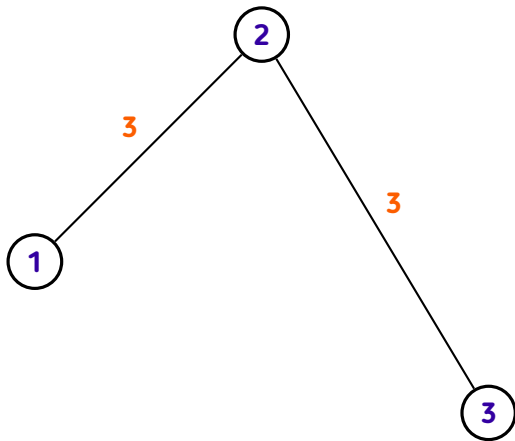


Exemplo de entrada e saída

3 2 5

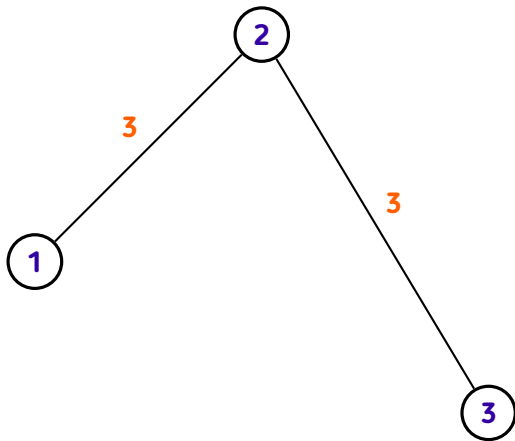
1 2 3

2 3 3



Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2

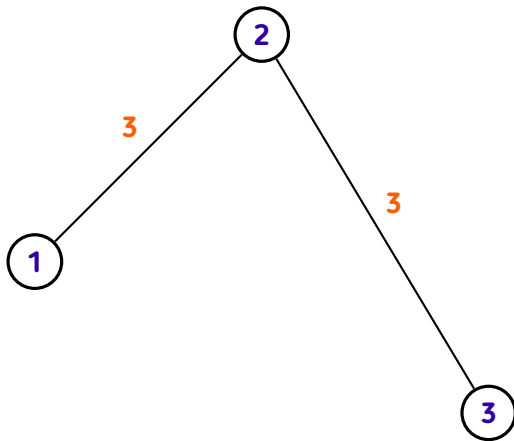


Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2

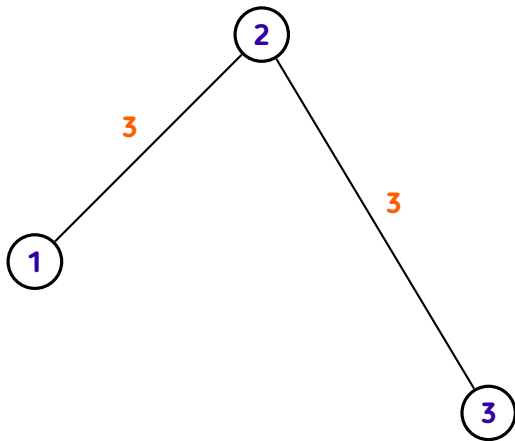


de consultas



Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2



Exemplo de entrada e saída

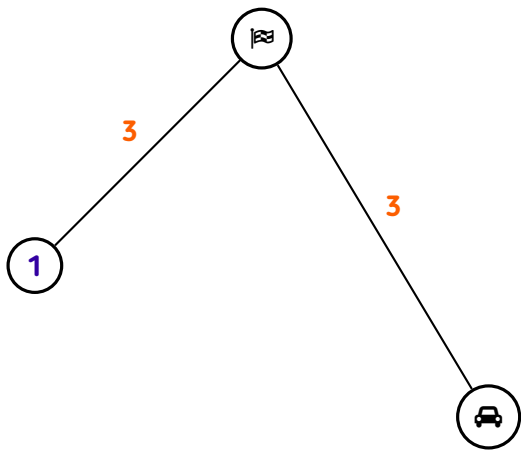
3 2 5

1 2 3

2 3 3

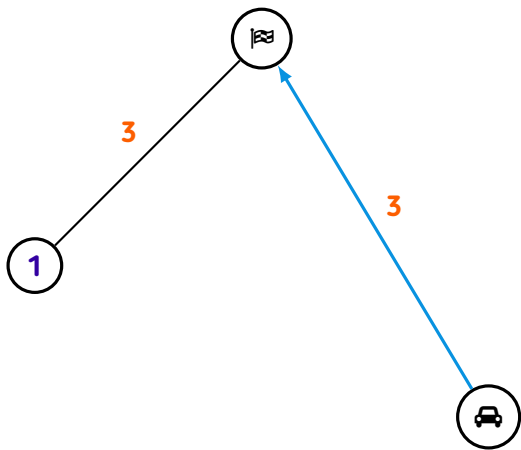
2

3 2



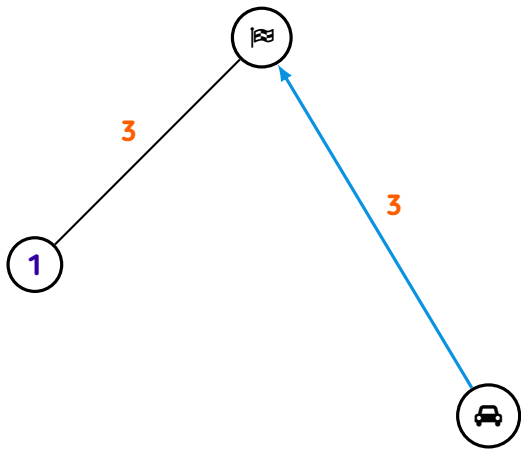
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2



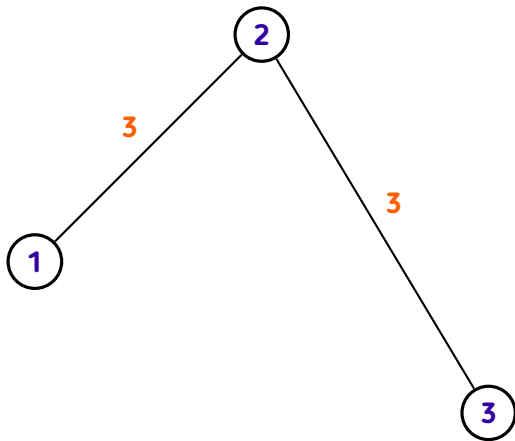
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
↓
0



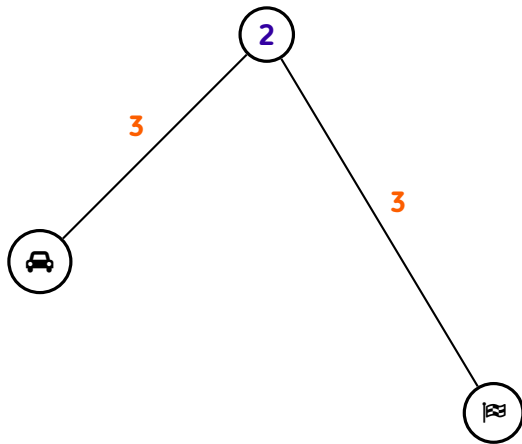
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
1 3



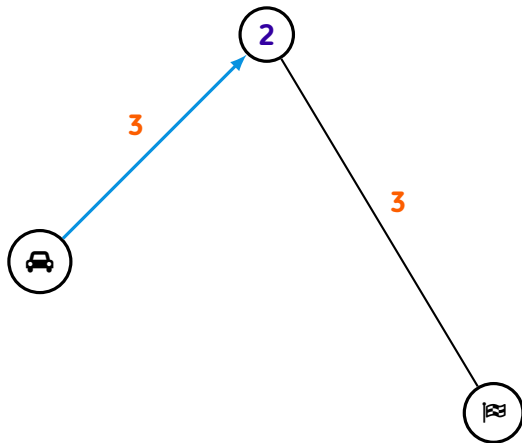
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
1 3



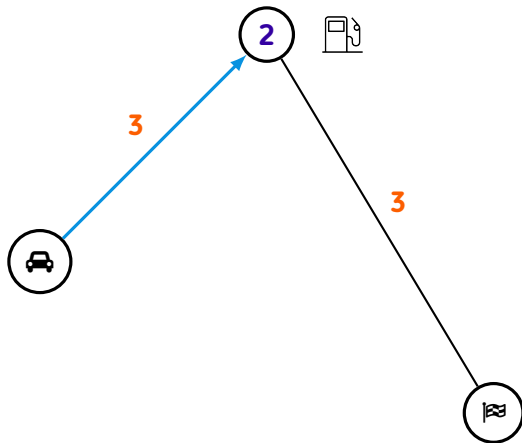
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
1 3



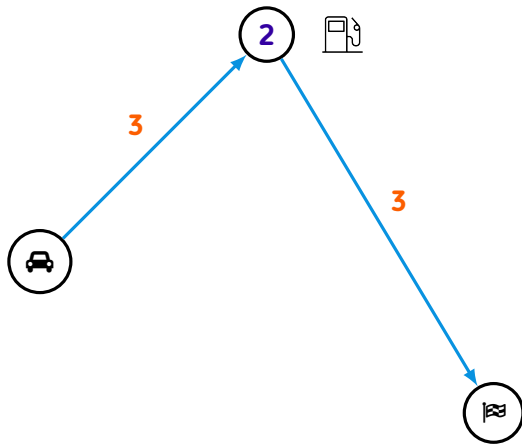
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
1 3



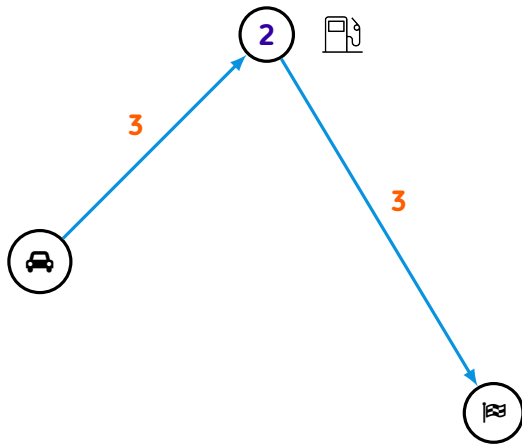
Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
1 3



Exemplo de entrada e saída

3 2 5
1 2 3
2 3 3
2
3 2
1 3
↓
1



Solução

Solução

- ★ É possível pré-computar todos os caminhos possíveis usando Dijkstra

Solução

- ★ É possível pré-computar todos os caminhos possíveis usando Dijkstra
- ★ Na fila de prioridades devem ser inseridas a distância e a gasolina restante

Solução

- ★ É possível pré-computar todos os caminhos possíveis usando Dijkstra
- ★ Na fila de prioridades devem ser inseridas a distância e a gasolina restante
- ★ A ordenação deve ser: menor distância, maior gasolina restante

Solução

- ★ É possível pré-computar todos os caminhos possíveis usando Dijkstra
- ★ Na fila de prioridades devem ser inseridas a distância e a gasolina restante
- ★ A ordenação deve ser: menor distância, maior gasolina restante
- ★ O relaxamento deve ser feito também para distâncias iguais, porém com mais gasolina restante

Solução

- ★ É possível pré-computar todos os caminhos possíveis usando Dijkstra
- ★ Na fila de prioridades devem ser inseridas a distância e a gasolina restante
- ★ A ordenação deve ser: menor distância, maior gasolina restante
- ★ O relaxamento deve ser feito também para distâncias iguais, porém com mais gasolina restante
- ★ Pré-computados os caminhos, cada consulta pode ser respondida em $O(1)$

```
void dijkstra(int s, ll N, ll L)
{
    for (ll i = 1; i <= N; ++i)
        dist[s][i] = fuel[i] = oo;

    dist[s][s] = 0;
    fuel[s] = L;
    processed.reset();

    priority_queue<iii, vector<iii>, greater<iii>> pq;
    pq.push(iii(0, -L, s));

    while (not pq.empty())
    {
        auto [d, f, u] = pq.top(); pq.pop();

        if (processed[u])
            continue;

        processed[u] = true;
```

```

for (const auto& [v, c] : adj[u]) {
    if (c <= fuel[u] and (dist[s][v] > dist[s][u] or
        (dist[s][v] == dist[s][u] and fuel[u] - c > fuel[v])))
    {
        dist[s][v] = dist[s][u];
        fuel[v] = fuel[u] - c;
        pq.push(III(dist[s][v], -fuel[v], v));
        continue;
    }

    if (c <= L and (dist[s][v] > dist[s][u] + 1 or
        (dist[s][v] == dist[s][u] + 1 and L - c > fuel[v])))
    {
        dist[s][v] = dist[s][u] + 1;
        fuel[v] = L - c;
        pq.push(III(dist[s][v], -fuel[v], v));
    }
}
}
}

```

```
vector<ll> solve(ll N, ll L, const vector<ii>& qs)
{
    vector<ll> ans(qs.size());

    for (int s = 1; s <= N; ++s)
        dijkstra(s, N, L);

    for (size_t i = 0; i < qs.size(); ++i)
    {
        auto [s, t] = qs[i];
        ans[i] = (dist[s][t] == oo ? -1 : dist[s][t]);
    }

    return ans;
}
```

Créditos

Gas station icon made by Kiranshastry from www.flaticon.com