1. Daleka droga

Sigismund Dijkstra zawsze mówił, że im więcej komnat w zamku, tym lepiej - przynajmniej jest się gdzie ukryć. I choć Dijkstra rzadko zmieniał zdanie, złamana noga jest wystarczającym powodem, by znielubić dowolny budynek, po którym trzeba tak daleko łazić.
Zbliżający się obiad natchnął Dijkstrę do pewnych przemyśleń, a mianowicie - jadalnia znajdowała się niemal na drugim końcu zamku, licząc od jego biura. „Jakieś 200 metrów, może lepiej" - mruczał pod nosem. Nie byłby jednak najlepszym szpiegiem Czterech Królestw, gdyby nie wiedział tego, co inni często pomijają - tłumy czarodziejek przetaczające się tu regularnie spowodowały, że klasyczna geometria nie ma zbyt wiele wspólnego z tretogorskim zamkiem i niektóre odległości okazują się być zadziwiająco krótkie albo irracjonalnie długie. „W końcu płacą mi tu za myślenie" - mruknął i zaczął żmudne obliczenia.

1 Zadanie

Napisz program, który na podstawie odległości między poszczególnymi komnatami w tretogorskim zamku znajdzie najkrótszą trasę między biurem Dijkstry a jadalnią.

2 Wejście

Pierwsza linia zawiera dwie liczby całkowite N i K będące odpowiednio liczbą komnat i liczbą istniejących między nimi przejść.
W kolejnych K wierszach znajdują się opisy poszczególnych tras - numer komnaty, z której wychodzimy, numer komnaty, do której wchodzimy i długość przejścia (w metrach).
Dwie ostatnie liczby oznaczają numer komnaty, która jest biurem Dijkstry oraz tej, w której mieści się jadalnia.

W szablonie programu znajdziesz kilka miejsc do uzupełnienia.

3 Wyjście

Na standardowym wyjściu powinna się pojawić jedna liczba, będąca odległością w metrach, jaką Dijkstra musi przekuśtykać do jadalni.

4 Przykład

4.1 Wejście

```
5 6
0 1 1
1 2 1
2 3 1
3 4 1
0 2 2
2 4 4
0 4
```

4.2 Wyjście

```
4
```

5 Dla dociekliwych

W tym zadaniu użyłem kilku klas z biblioteki standardowej C++. Przyjrzyjmy się im nieco bliżej.

pair - trzymają dwie wartości o zdefiniowanym typie, np. $pair < int, string >$ przechowuje liczbę całkowitą ($pair.first$) i napis ($pair.second$)

vector - można go rozumieć jako tablicę z dynamicznym rozmiarem - możemy do niego dodawać dowolnie wiele elementów ($vector.push\_back(sth)$) i sprawdzić, jak dużo elementów przechowuje ($vector.size()$) oraz odwoływać się do poszczególnych elementów tak, jakby to była tablica ($vector[i]$). Tak samo, jak w przypadku pary, musimy podać, jaki typ będzie w vectorze przechowywany.

priority_queue - kolejka priorytetowa. Obsługuje podstawowe operacje *push*(), *top*() i *pop*(). Warto zwrócić uwagę na deklarację - musimy podać, jaki typ będzie rzechowywany w kolejce, możemy także podać, na bazie jakiej

struktury danych kolejka będzie oparta oraz jaką funkcję wykorzystać do porównywania elementów w kolejce.