DJ Krisa T

Ели е момиче, и като такова винаги закъснява. В момента тя закъснява за концерта на небезизвестната ди-джей-ка DJ Krisa T., също позната като нейната приятелка Крис. До началото на концерта остават броени минути (всъщност точно K минути), а Ели тъкмо излиза от тях. Разбира се, тя може да си вземе такси и да стигне на време, но кой дава пари за глупости. Вместо това тя може да си вземе карта за градския транспорт и да спести някой лев (който да бъде инвестиран в нещо по- пенливо). Градският транспорт в града, където живеят, е малко по-странно устроен. В него различните линии могат да имат различна цена – ако са по-бързи или по-нови те могат да са по-скъпи от други. Ако Ели си купи карта за градската мрежа на стойност X, тя може да пътува с всички линии, чиято цена е по-малка или равна на X колкото си пъти иска, но не може да пътува с по-скъпи. По дадена информация къде се намира тя, къде трябва да стигне, както и всички линии на градския транспорт (от къде до къде водят, колко струват, и колко минути отнема за да се стигне от едната точка до другата) намерете цената на най-евтината карта, с която Ели може да стигне на време (тоест за не повече от K минути).

Жокер: Класическа задача за прилагане на двоично търсене с предикатна функция. Ще направим двоично търсене по цената на картата за градски транспорт на Криста (left = 1; right=MAX_COST=100000). При всяка фиксирана цена ще викаме алгоритъма на Дийкстра с един добавен от нас параметър - фиксираната от двоичното търсене цена на картата. Към всяко от ребрата на графа ще добавим още един параметър (освен времето за изминаване на реброто). Той ще е разбира се минималната цена на картата при която можем да преминем по него. Ако цената на дадено ребро е по-малка от фиксираната цена от двоичното търсене ще смятаме че това ребро е проходимо (използваемо) при текущото извикване на нашата модифицирана Дийкстра, в противен случай реброто е неизползуваемо и просто го прескачаме (все едно че не съществува), когато го итерираме.

Графа ще си дефинираме така:

vector<vector<int, pair<int, int>>> graph;
i - номер на възел в графа;
j - номера (използува се само за итерация) на поредното ребро започващо от възел i

graph[i][j].first - номера на възела в другия край на поредното ребро (i, graph[i][j].first), започващо от възел i
graph[i][j].second.first - времето, за което се преминава по реброто (i, graph[i][j].first)
graph[i][j].second.second - минималната цена на картата, с която може да се премине по текущото ребро (i, graph[i][j].first). Действа, като маска за текущото ребро: ако graph[i][j].second.second <= fixed_from_binary_search_card_price, то реброто е валидно, иначе все едно то не съществува за текущото извикване на нашата модифицирана Дийкстра.

Извадка от нашата модифицирана Dijkstra:

```
void ModifiedDijkstra(vector<vector<int, pair<int, int>>> & graph,     int start_node, int found_distances[], int fixed_from_binary_search_card_price)
{
...
   for (auto edge: graph[u])
   {
      if (edge.second.second > fixed_from_binary_search_card_price)
         continue;
   ...
   }
...
}
```

Условието за избор на ляв или десен интервал в двоичното търсене е: ...
middle = (left + right + 1) / 2;
ModifiedDijkstra(..., found_distances, middle);
if (found_distances[N] <= K)
...

Input Format

На първия ред на стандартния вход ще бъде зададен броят тестове T. Всеки от тестовете ще започва с ред, който съдържа целите числа N, M и K – съответно броят спирки, броят маршрути и оставащото време до концерта. На всеки от следващите M реда ще бъде зададен един маршрут чрез четворката цели числа From To Cost Time, указваща, че има линия (насочено ребро) от спирка From до спирка To с цена Cost и продължителност Time. Считаме, че Ели живее до спирка 1, а концертът се намира до спирка N.

Constraints

$1 \le T \le 10$
$1 \le N \le 10,000$
$1 \le M \le 100,000$
$1 \le From, To \le N$
$1 \le Cost, Time, K \le 100,000$

Output Format

За всеки тест изведете по един ред с едно единствено число – минималната цена, с която Ели може да стигне навреме. Ако тя не може да стигне, дори ако вземе карта, с която да ползва всички линии, вместо това изведете -1 за съответния тест.

Пояснение към примерните тестовите данни: Забележете, че има пътища с цена 5, но тяхното време за преминаване е съответно 43 и 45, при ограничение 42. Също така има път с време 28, но пък неговата цена е 13. Оптималният път е 1-3-5-6-7, чиято цена е 7 и време за преминаване 40.

Sample Input 0

```
2
7 11 42
1 3 7 11
3 1 7 13
1 2 3 3
1 4 13 1
6 1 14 8
4 6 1 7
2 4 1 13
2 6 4 20
3 5 2 5
5 6 6 4
6 7 5 20
2 2 3
1 2 3 5
1 2 1 9
```

Sample Output 0

```
7
-1
```