There are n cities connected by m flights. Each fight starts from city u and arrives at v with a price w.

Now given all the cities and fights, together with starting city src and the destination dst, your task is to find the cheapest price from src to dst with up to k stops. If there is no such route, output -1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Example 1:  Input:  n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]]  src = 0, dst = 2, k = 1  Output: 200  Explanation:  The graph looks like this:    The cheapest price from city <code>0</code> to city <code>2</code> with at most 1 stop costs 200, as marked red in the picture. |



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Example 2:  Input:  n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]]  src = 0, dst = 2, k = 0  Output: 500  Explanation:  The graph looks like this:  The cheapest price from city <code>0</code> to city <code>2</code> with at most 0 stop costs 500, as marked blue in the picture. |



**Note:**

* The number of nodes n will be in range [1, 100], with nodes labeled from 0 to n - 1.
* The size of flights will be in range [0, n \* (n - 1) / 2].
* The format of each flight will be (src, dst, price).
* The price of each flight will be in the range [1, 10000].
* k is in the range of [0, n - 1].
* There will not be any duplicated flights or self cycles.

Solution 1:

int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int k) {

//1. Create an Adjacency List

vector<vector<vector<int>>> adjList(n);

//example :

//0 : {1, 100}, {2, 500}

//1 : {2, 100}

//2 :

for(auto f : flights) {

int from = f[0];

int to = f[1];

int cost = f[2];

adjList[from].push\_back({to, cost});

//from : source

//to : dest

//cost: cost from 'from' to 'to'

}

//2. Create a queue for performing BFS

queue<vector<int>> q;

//3. Push source in q as {src, money spent so far, number of stops b/w current city and source}

q.push({src, 0, -1});

//Note : if {A, B} are two directly connected cities, then number of stops b/w them is Zero,

//so for convenience i'm assuming number of stops b/w A and A as -1

int minCost = INT\_MAX; //this keeps track of minimum cost

while(!q.empty()) {

vector<int> curStation = q.front(); q.pop();

int curCity = curStation[0];

int curCost = curStation[1];

int curK = curStation[2]; //this is the number of stops seen so far from source to current city

if(curCity == dst) {

minCost = min(minCost, curCost);

continue;

}

for(auto p : adjList[curCity]) {

//we include a city in our route only if :

// > it doesn't exceed number of stops alloted

// > it keeps cost less than mincost

//\*\*\*

if(curK+1 <= k and curCost + p[1] < minCost)

q.push({p[0], p[1] + curCost, curK+1});

}

}

return minCost==INT\_MAX?-1:minCost;

}

dp[k][i]: min cost from src to i taken up to k flights (k-1 stops)

init: dp[0:k+2][src] = 0

transition: dp[k][i] = min(dp[k-1][j] + price[j][i])

ans: dp[K+1][dst]

Time complexity: O(k \* |flights|) / O(k\*n^2)

Space complexity: O(k\*n) -> O(n)

w/o space compression O(k\*n)

|  |  |
| --- | --- |
|  | // Running time: 15 ms  class Solution {  public:    int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int K) {      constexpr int kInfCost = 1e9;      vector<vector<int>> dp(K + 2, vector<int>(n, kInfCost));      dp[0][src] = 0;        for (int i = 1; i <= K + 1; ++i) {        dp[i][src] = 0;        for (const auto& p : flights)            dp[i][p[1]] = min(dp[i][p[1]], dp[i-1][p[0]] + p[2]);      }        return dp[K + 1][dst] >= kInfCost ? -1 : dp[K + 1][dst];    }  }; |