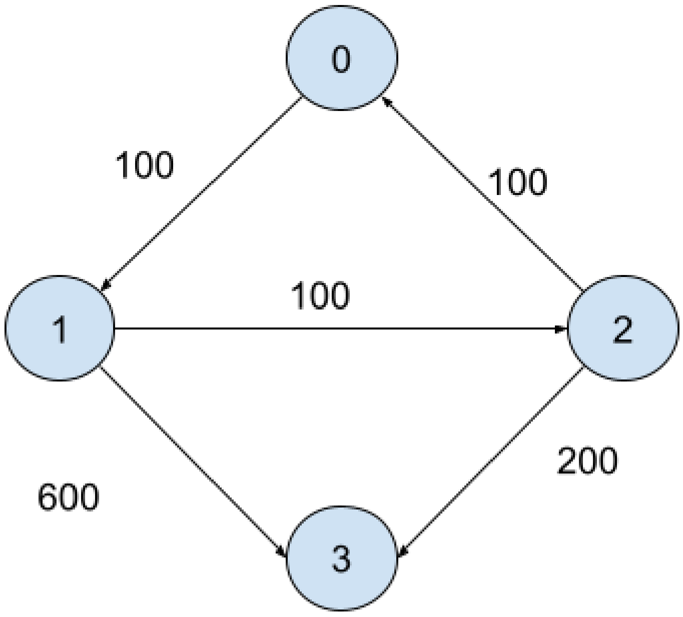
MEDIUM

Cheapest Flights within K stops

Intuition



Source = 0

Destination = 3

K = 1

With 1 stop we have 2 options :
0 -> 1 -> 3 : 700
0 -> 2 -> 3 : 300

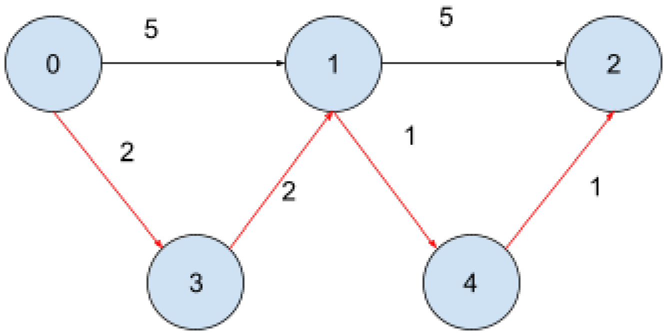
Dijkstra can be used to solve it but using a slight modification.

We cannot solely use the distance as a perimeter; we have to use stops used to count the answer.

We will keep emphasis more on the stops used as we need to check for [stops, [node, distance]] instead of [distance, [node, stops]] to change the emphasis

Priority Queue :

We don’t actually need a priority queue to solve it we can solve it using queue



Distance : // initial configuration

0	inf	inf	inf	inf

Queue :

[0, [0,0]]

Distance :

0	5	7	2	6

Queue :

[0, [0,0]]

[1, [3,2]]
[1, [1,5]]

[2,[2,10]]
[2,[4,6]]
[1,[3,2]]

[3,[2,7]]
[2,[2,10]]

[3,[2,7]]
[2,[2,10]] // its the destination we can stop here and return the distance value

Approach

- Creating an adjacency list
- Creating an empty queue containing pair<int, pair<int, int>>
- Inserting the source with 0 stops and 0 distance into the queue

```
-   Create a distance vector and initialize all elements with inf
-   In distance vector mark the source to have a distance 0
-   Traverse until the queue becomes empty :
        -   Extract the first element of the queue
        -   Pop the first element of queue
        -   Check if the stops are greater than k :
                -   Skip the element
        -   Traverse for the adjacent elements :
                -   Check if the distance + distance to reach adjacent node are smaller than
                    the distance of adjacent node :
                        -   Update the distance of adjacent node with the smaller distance
                        -   Push the adjacent node with smaller distance into the queue
-   Check if the distance to reach destination is infinity :
        -   Return -1
-   Return distance to reach the destination
```

Function Code

```
int CheapestFLight(int n, vector<vector<int>>& flights, int src, int dst,
int K)  {
        // Create an adjacency list
        vector<vector<pair<int,int>>> adj(n);
        for(auto it:flights)
        {
            int from = it[0];
            int to = it[1];
            int price = it[2];
            adj[from].push_back({to,price});
        }
        // creating a queue to store [ stops, [ node, distance]]
        queue<pair<int,pair<int, int>>> q;
        // inserting the source element into the queue
        q.push({0,{src,0}});
        // creating a distance vector
        vector<int> distance(n,1e9);
        // source distance marking 0
        distance[src] = 0;
        // traversing until the queue becomes empty
        while(!q.empty())
        {
            // extracting the first element from the queue
            auto it = q.front();
            // popping the first element from the queue
            q.pop();
            // extracting perimeters
            int stops = it.first;
            int node = it.second.first;
            int dist = it.second.second;
            // checking if stops more than required
            if(stops>K)
            {
                // move on from the node
                continue;
            }
            // traversing for the adjacent elements of the node
            for(auto it:adj[node])
            {
                int adjnode = it.first;
                int adjdist = it.second;
                // checking if distance to reach smaller than current
distance
                if(dist+adjdist<distance[adjnode])
                {
                    // updating distance
                    distance[adjnode] = dist+adjdist;
                    // pushing to the queue
                    q.push({stops+1,{adjnode,distance[adjnode]}});
                }
            }
        }
        // checking if the distance to reach is infinity
        if(distance[dst]==1e9)
        {
            return -1;
        }
        // returning distance to destination
        return distance[dst];

    }
```

Time Complexity

O(n)