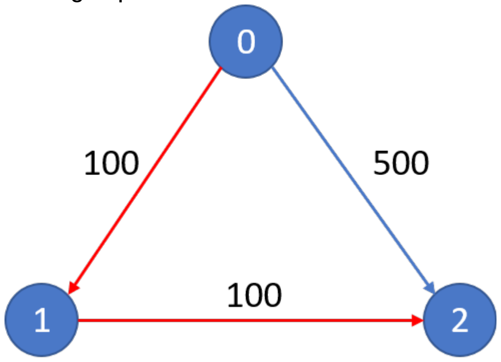
787. Cheapest Flights Within K Stops

DescriptionHintsSubmissionsDiscussSolution

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

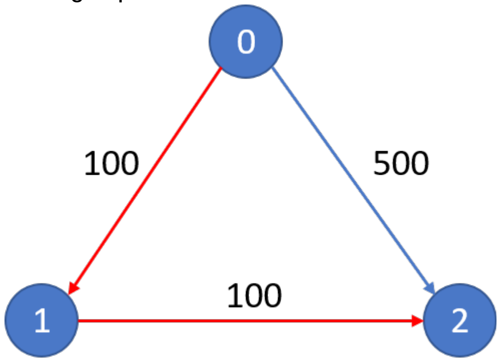
Now given all the cities and fights, together with starting city `src` and the destination `dst`, your task is to find the cheapest price from `src` to `dst` with up to `k` stops. If there is no such route, output `-1`.

```
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 0 to city 2 with at most 1 stop costs 200, as marked
red in the picture.
```

```
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 0 to city 2 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.

Seen this question in a real interview before?

- Difficulty:Medium
- Total Accepted:2.4K
- Total Submissions:7.4K
- Contributor:1337c0d3r
-
- Subscribe to see which companies asked this question.

Approach #2: Dijkstra's [Accepted]

Intuition

Instead of nodes being places, use places and number of stops. We want to find the lowest cost from source to target, which makes Dijkstra's a good candidate algorithm.

If we continually extend our potential flightpaths in order of cost, we know once we've reached the destination `dst` that it was the lowest cost way to get there. This is the idea behind Dijkstra's algorithm.

Algorithm

Dijkstra's algorithm uses a priority queue to continually search the next node with the lowest cost.

If we've come to a node and it has a lower recorded cost or we've taken too many steps, we don't need to search it. If we reach our destination, because we are searching in order of lowest cost first, it must have the lowest cost.

Otherwise, for every outbound flight from `node` that is better, we'll add it to our priority queue of things to search.

```
//C++
#include<stdio.h>
#include<iostream>
#include<vector>
#include<algorithm>
#include<string.h>
#include<unordered_map>
#include<queue>
#include<limits.h>
using namespace std;

struct comp
{
     bool operator()(vector<int> &a, vector<int> &b)
     {
          return a[0]>b[0];
     }
};

int findCheapestPrice(int n, vector<vector<int>>& flights, int src,
int dst, int K) {
     int graph[n][n];
     memset(graph,0,sizeof(graph));
     for(int i=0;i<(int)flights.size();++i)
     {
          vector<int> x = flights[i];
          graph[x[0]][x[1]] = x[2];
     }
     unordered_map<string,int> map;
     // the queue was sorted in an ascend order
     priority_queue<vector<int>,vector<vector<int>>,comp> pq;
     vector<int> start = {0,0,src}; // cost, k, node
     pq.push(start);
     while(pq.size())
     {
          vector<int> info = pq.top();
          printf("cost=%d\n",info[0]);
          pq.pop();
          int cost = info[0]; int k = info[1]; int place = info[2];
          int cost2 = INT_MAX; char tmpstr[20];
          sprintf(tmpstr,"%d+%d",k,place);
          if(map.count(string(tmpstr))) cost2 = map[string(tmpstr)];
          if(k>K+1 || cost>cost2) continue;
          // find the neighbours
          if(place==dst) return cost;
          for(int nei=0;nei<n;++nei)
          {
               if(nei==place || graph[place][nei]<=0) continue;
               int newcost = graph[place][nei] + cost;
               int cost2 = INT_MAX;char tmpstr[20];
               sprintf(tmpstr,"%d+%d",k+1,nei);
               if(map.count(string(tmpstr))) cost2 =
map[string(tmpstr)];
               if(newcost<cost2)
               {
                    pq.push({newcost,k+1,nei});
                    char tmpstr2[20];
                    sprintf(tmpstr2,"%d+%d",k+1,nei);
                    map[string(tmpstr2)] = newcost;
               }
          }
     }
     return -1;
}


int main(int argc, char *argv[])
{
     /* a simple test case
     3
     [[0,1,100],[1,2,100],[0,2,500]]
     0
     2
     1
    */
    vector<vector<int>> flights = {{0,1,100},{1,2,100},{0,2,500}};
    int ans = findCheapestPrice(3,flights,0,2,1);
    cout << ans << endl;
     return 0;
}
```

Python:

```
class Solution(object):
    def findCheapestPrice(self, n, flights, src, dst, K):
        graph = collections.defaultdict(dict)
        for u, v, w in flights:
            graph[u][v] = w

        best = {}
        pq = [(0, 0, src)]
        while pq:
            cost, k, place = heapq.heappop(pq)
            if k > K+1 or cost > best.get((k, place), float('inf')):
continue
            if place == dst: return cost

            for nei, wt in graph[place].iteritems():
```

```
            newcost = cost + wt
            if newcost < best.get((k+1, nei), float('inf')):
                heapq.heappush(pq, (newcost, k+1, nei))
                best[k+1, nei] = newcost

    return -1
```