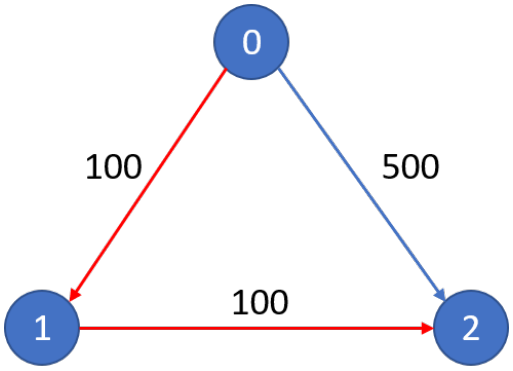
Cheapest Flights Within K Stops

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

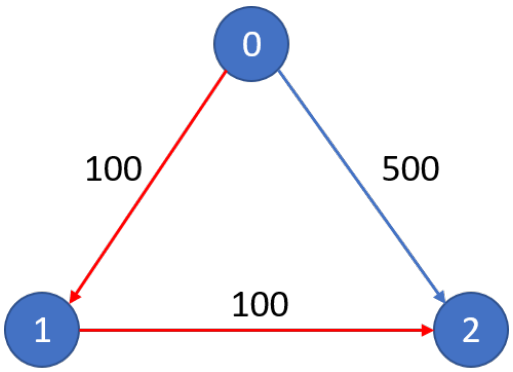
Now given all the cities and fights, together with starting city `src` and the destination `dst`, your task is to find the cheapest price from `src` to `dst` with up to `k` stops. If there is no such route, output `-1`.

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.

Solution 1

```
class Solution {
public:
    int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int
K)
    {
        const int INF = 1e9;
        K++;
        vector<vector<int>> ans(n, vector<int>(K+1));
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j <= K; j++)
            {
                ans[i][j] = INF;
            }
        }
        ans[src][0] = 0;
        for(int i = 1; i <= K; i++)
        {
            for(int j = 0; j < n; j++)
                ans[j][i] = ans[j][i-1];
            for(const vector<int>& f: flights)
            {
                ans[f[1]][i] = min(ans[f[1]][i], ans[f[0]][i-1] + f[2]);
            }
        }
        if(ans[dst][K] == INF) return -1;
        return ans[dst][K];
    }
};
```

written by chwangthu original link here

Solution 2

> **Note: The code has been updated at 2018-2-18 17:33:41 (MST) to fix a bug pointed by @chang17 and @jray319. Thanks!**
> Their test case should be added in. The testcases for the contest did not cover those.
> The fix makes the solution not really a Dijkstra that it is losing some property that Dijkstra has to use a priority queue. Instead, it falls into a BFS like solution.
> But since I effectively adapt it from Dijkstra, I decide to remain the original code in the bottom for anyone who is interested in the changes.

This is basically an implementation for the Dijkstra algorithm based on the description in the book “Cracking the coding interview”, page 634. Its description is really clear.
The only thing that is “adapted” is highlighted in the code `# this two lines are important` below.

Using vanila Dijkstra can help us figure out the shortest weighted path from the `src` to `dst`.
But then we lose the information of those paths that can reach `dst` with less stop.
So I record the information into the `results` list. Once it somehow reaches the `dst` from a path, we record it.

```python
class Solution(object):
    def findCheapestPrice(self, n, flights, src, dst, K):
        remain, results = [], []
        graph = [{} for i in range(n)]
        previous = [None for i in range(n)]
        weights = [sys.maxint for i in range(n)]
        for s,d,w in flights:
            graph[s][d]=w

        heapq.heappush(remain, (0, src))
        weights[src] = 0
        stop = 0
        while remain:
            tmp, remain = remain, []
            while tmp:
                weight, node = heapq.heappop(tmp)
                for tonode, toweight in graph[node].items():
                    if weights[tonode] > weight + toweight:
                        weights[tonode] = weight + toweight
                        heapq.heappush(remain, (weights[tonode], tonode))
                        previous[tonode] = node
                    # these two lines are important
                    if tonode == dst:
                        heapq.heappush(results, (weights[tonode], stop))
            stop+=1
        while results:
            w, count = heapq.heappop(results)
            if count<=K:
                return w
        return -1
```

==========

Below is the original code, that passed the testcases in the system but did not pass the testcase from @chang17 and @jray319
This adaptation and fix itself to me is a very interesting and great lesson to learn Dijkstra, maybe just because this is my first time implementing a Dijkstra.
hope you enjoy it too!

```
class Solution(object):
    def findCheapestPrice(self, n, flights, src, dst, K):
        def getcount():
            pre, count = dst, 0
            while previous[pre] is not None:
                count += 1
                pre = previous[pre]
            return count-1

        remain, results = [], []
        graph = [{} for i in range(n)]
        previous = [None for i in range(n)]
        weights = [sys.maxint for i in range(n)]
        for s,d,w in flights:
            graph[s][d]=w

        heapq.heappush(remain, (0, src))
        weights[src] = 0

        while remain:
            weight, node = heapq.heappop(remain)
            for tonode, toweight in graph[node].items():
                if weights[tonode] > weight + toweight:
                    weights[tonode] = weight + toweight
                    heapq.heappush(remain, (weights[tonode], tonode))
                    previous[tonode] = node
                if tonode == dst:
                    heapq.heappush(results, (weights[tonode], getcount()))

        while results:
            w, count = heapq.heappop(results)
            if count<=K:
                return w
        return -1
```

The test cases:
from @chang17,

```
5
[[0,1,5],[1,2,5],[0,3,2],[3,1,2],[1,4,1],[4,2,1]]
0
2
2
```

from @jray319,

```
4
[[0,1,1],[1,2,1],[2,3,1],[0,2,100]]
0
3
1
```

written by licaiuu original link here

Solution 3

```
class Solution {
public:
    //bellman ford.
    //just run it k+1 iterations.
    int findCheapestPrice(int n, vector<vector<int>>& a, int src, int sink, int k) {

        vector<int> c(n, 1e8);
        c[src] = 0;

        for(int z=0; z<=k; z++){
            vector<int> C(c);
            for(auto e: a)
                C[e[1]] = min(C[e[1]], c[e[0]] + e[2]);
            c = C;
        }
        return c[sink] == 1e8 ? -1 : c[sink];
    }
};
```

written by brendon4565 original link here