[LeetCode](https://leetcode.com/problems/cheapest-flights-within-k-stops/description/?envType=daily-question&envId=2024-02-23)

<https://github.com/AdityaKonda6/-50DaysOfCoding>

<https://leetcode.com/problems/cheapest-flights-within-k-stops/description/?envType=daily-question&envId=2024-02-23>

<https://www.linkedin.com/in/aditya-adi-konda/>

 Day 4 of [#50dayscodingchallenge](https://www.linkedin.com/feed/hashtag/?keywords=50dayscodingchallenge&highlightedUpdateUrns=urn:li:activity:7166316239483461633):  
[#leetcode](https://www.linkedin.com/feed/hashtag/?keywords=leetcode&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#leetcodechallenge](https://www.linkedin.com/feed/hashtag/?keywords=leetcodechallenge&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#leetcodestreak](https://www.linkedin.com/feed/hashtag/?keywords=leetcodestreak&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#leetcode2024](https://www.linkedin.com/feed/hashtag/?keywords=leetcode2024&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#leetcode50day](https://www.linkedin.com/feed/hashtag/?keywords=leetcode50day&highlightedUpdateUrns=urn:li:activity:7166316239483461633)  
   
Just kicked off my coding journey with a fascinating problem - "Successfully solved LeetCode Problem “787. Cheapest Flights Within K Stops” !”  
   
✨ Task: Travel Optimization Expert

Excited to share my recent solution to a challenging problem in optimizing flight routes!

Problem: Given a set of cities and flights between them, each with associated costs, find the cheapest price to travel from a source city to a destination city with at most k stops.

Challenge: Considering the constraints of the problem (number of cities, available flights, source, destination, and maximum stops), I implemented a solution using a modified version of Dijkstra's algorithm with memoization for efficiency.

- Example 1: From city 0 to city 3 with at most 1 stop, the optimal path costs 700 (as shown in the graph).

- Example 2: Traveling from city 0 to 2 with at most 1 stop, the most cost-effective path is 200.

- Example 3: For a direct flight from city 0 to 2 (0 stops), the cost is 500.

Key Techniques:

- Utilized a priority queue for efficient exploration of paths.

- Employed memoization to store and retrieve minimum costs for each city and stop combination.

Unravel the mystery using your coding skills!  [#CodingChallenge](https://www.linkedin.com/feed/hashtag/?keywords=codingchallenge&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#TownJudge](https://www.linkedin.com/feed/hashtag/?keywords=townjudge&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#Algorithm](https://www.linkedin.com/feed/hashtag/?keywords=algorithm&highlightedUpdateUrns=urn:li:activity:7166316239483461633) [#LinkedInPost](https://www.linkedin.com/feed/hashtag/?keywords=linkedinpost&highlightedUpdateUrns=urn:li:activity:7166316239483461633) #Algorithm #Optimization #DataStructures #TravelOptimization #CodingChallenge  
  
Excited about the progress and challenges ahead!  
   
Make Sure You Follow My GitHub For Solutions: [https://lnkd.in/d7EApJ2m](https://lnkd.in/d7EApJ2m" \t "https://www.linkedin.com/feed/_self)  
  
  
Happy coding!

**Solution:-**

class Solution {

 public:

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

                        int k) {

    vector<vector<pair<int, int>>> graph(n);

    for (const vector<int>& flight : flights) {

      const int u = flight[0];

      const int v = flight[1];

      const int w = flight[2];

      graph[u].emplace\_back(v, w);

    }

    return dijkstra(graph, src, dst, k);

  }

 private:

  int dijkstra(const vector<vector<pair<int, int>>>& graph, int src, int dst,

               int k) {

    vector<vector<int>> dist(graph.size(), vector<int>(k + 2, INT\_MAX));

    using T = tuple<int, int, int>;  // (d, u, stops)

    priority\_queue<T, vector<T>, greater<>> minHeap;

    dist[src][k + 1] = 0;

    minHeap.emplace(dist[src][k + 1], src, k + 1);

    while (!minHeap.empty()) {

      const auto [d, u, stops] = minHeap.top();

      minHeap.pop();

      if (u == dst)

        return d;

      if (stops == 0)

        continue;

      for (const auto& [v, w] : graph[u])

        if (d + w < dist[v][stops - 1]) {

          dist[v][stops - 1] = d + w;

          minHeap.emplace(dist[v][stops - 1], v, stops - 1);

        }

    }

    return -1;

  }

};

