



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment1.3

Student Name: Himanshu

Branch: BE-CSE

Semester: 6th

Subject Name: Competitive Coding-II

UID: 20BCS7944

Section/Group: 905/A

Date of Performance: 22/02/2023

Subject Code: 20CSP-351

1. Aim:

To implement the concept of string heap model.

2. Objective:

- The objective is to build problem solving capability and to learn the basic concepts of data structures.
- The implementation of last weight stone which shows and brushes up the concept of heap data structures.
- The implementation of cheapest flight within k-stops + in which the concept of kmp was introduced.

3. LeetCode code and output:

- **LAST STONE WEIGHT-**

```
class Solution {
public:
    int lastStoneWeight(vector<int>& stones) {
        priority_queue<int> pq(stones.begin(),stones.end());
        while(pq.size()!=1){
            int a = pq.top();
            pq.pop();
            int b = pq.top();
            pq.pop();
            pq.push(a-b);
        }
        return pq.top();
    }
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

OUTPUT:

LeetCode Problem 1046: Last Stone Weight

Easy 4.2K 83

Companies

You are given an array of integers `stones` where `stones[i]` is the weight of the i^{th} stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with $x \leq y$. The result of this smash is:

- If $x == y$, both stones are destroyed, and
- If $x \neq y$, the stone of weight x is destroyed, and the stone of weight y has new weight $y - x$.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return `0`.

Example 1:

Input: `stones = [2,7,4,1,8,1]`
Output: `1`
Explanation:

```
1 class Solution {
2 public:
3     int lastStoneWeight(vector<int>& stones) {
4         priority_queue<int> pq(stones.begin(), stones.end());
5         while(pq.size() > 1)
6         {
7             int a = pq.top();
8             pq.pop();
9             int b = pq.top();
10            pq.pop();
11            pq.push(a-b);
12        }
13        return pq.top();
14    }
15 }
```

Testcase Result Accepted Runtime: 2 ms

Case 1 Case 2

Input

stones =
[2, 7, 4, 1, 8, 1]

Console Run Submit

• CHEAPEST FLIGHT WITHIN K-STOP

```
class Solution {
const int inf = 1e8;
int dp[101][101];

public:
int dfs(int node, int k, int dest, vector<vector<int>> &cost, vector<int> adj[])
{
    if (k < 0)
        return inf;

    if (node == dest)
        return 0;

    if (dp[node][k] != -1)
        return dp[node][k];

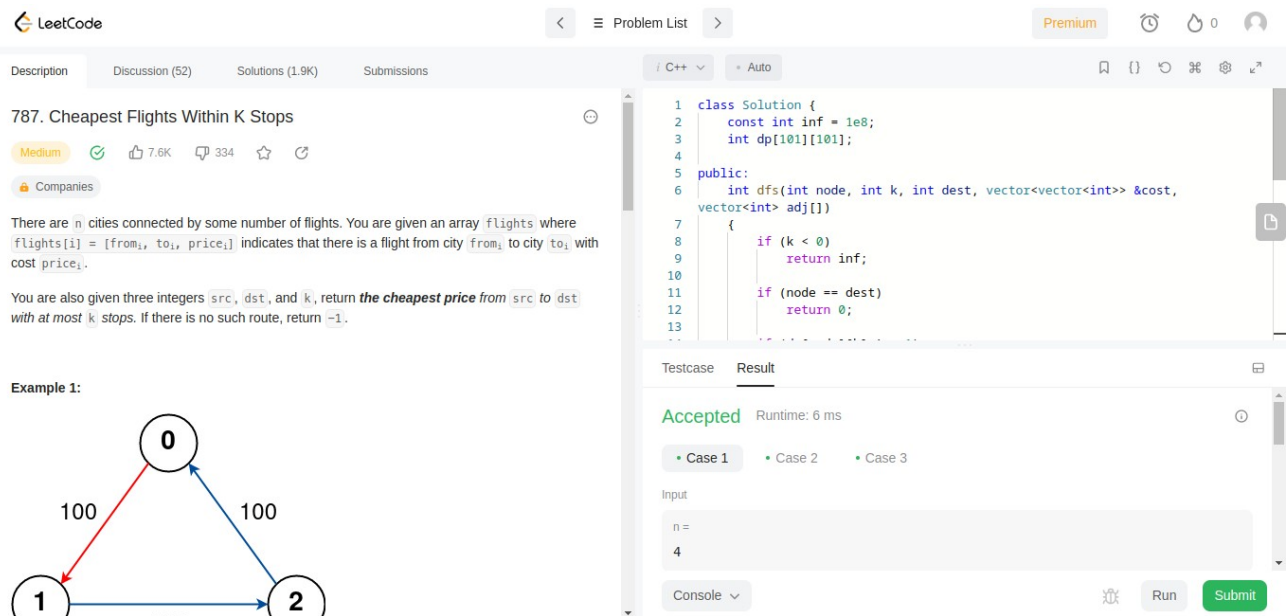
    int ans = inf;
    for (auto i : adj[node])
        ans = min(ans, cost[node][i] + dfs(i, k - 1, dest, cost, adj));
}
```

```
return dp[node][k] = ans;
}
int findCheapestPrice(int n, vector<vector<int>> &flights, int src, int dst, int k){
memset(dp, -1, sizeof dp);

vector<vector<int>> cost(n, vector<int>(n));
vector<int> adj[n];
for (auto e : flights){
adj[e[0]].push_back(e[1]);
cost[e[0]][e[1]] = e[2];
}

int ans = dfs(src, k + 1, dst, cost, adj);
return ans == inf ? -1 : ans; }
};
```

OUTPUT:



787. Cheapest Flights Within K Stops

Medium 7.6K 334

Companies

There are n cities connected by some number of flights. You are given an array `flights` where `flights[i] = [fromi, toi, pricei]` indicates that there is a flight from city `fromi` to city `toi` with cost `pricei`.

You are also given three integers `src`, `dst`, and `k`, return **the cheapest price from `src` to `dst` with at most `k` stops**. If there is no such route, return `-1`.

Example 1:

```

graph TD
    0((0)) -- 100 --> 1((1))
    0((0)) -- 100 --> 2((2))
    1((1)) -- 100 --> 2((2))
  
```

Testcase Result: Accepted Runtime: 6 ms

Case 1 Case 2 Case 3

Input: `n = 4`

Console Run Submit