Leet-Code Review

junyan.xu*

April 4, 2019

Contents

1 Important Algorithm

Before the solution, here are some summary of common graph algorithms

1.1 A Star Algorithm

The key idea is to always get minimum

*junyanxu5513@gmail.com

$$k(n) = g(n) + f(n) \tag{1}$$

as searching start

If $f(n) \leq \tilde{f}(n)$, then the $f(n)$ is admissive, and the searching result is guaranteed to be smallest. Dijstra is a special case for A* algorithm where $f(n) = 0$

Use a tree set as a priority queue for getting next.

2 Chapter 2

3 Chapter 3

4 Chapter 4

5 Chapter 5

5.1 Delete A Node In BST

Recursive is the best, remember to devide and conquer. Keep deleteing what you copied. **time complexity O(log(n)), mem O(log(n))**

```
class Solution {
  public:
  TreeNode* deleteNode(TreeNode* root, int key) {
    if(!root) return NULL;
    if(root->val < key){
      root->right = deleteNode(root->right, key);
    }
    else if(root->val > key){
      root->left = deleteNode(root->left, key);
    }
    else{
      if(!root->left || !root->right){
        root = root->left ? root->left: root->right;
      }
      else{
        auto temp = root->right;
        while(temp->left)
        temp = temp->left;
        root->val = temp->val;
        // key of the recursion
        root->right = deleteNode(root->right, temp->val);
      }
    }
```

```
        return root;
    }
};
```

5.2 Minimum Number of Arrows to Burst Balloons

Greedy Algo, very similar to merge intervals. **time complexity** $O(Nlog(N))$, **space is** $O(1)$

```
class Solution {
public:
    int findMinArrowShots(vector<pair<int, int>>& points) {
        sort(points.begin(), points.end());
        int count = 0;
        int right = 0;
        for(int i=0; i<points.size(); i++){
            if(i==0){
                right = points[i].second;
                count++;
                continue;
            }
            if(points[i].first <= right)
                right = min(right, points[i].second);
            else{
                count++;
                right = points[i].second;
            }
        }
        return count;
    }
};
```

5.3 Minimum Moves to Equal Array Elements

Find minimum first, **time complexity** $O(N)$, **space is** $O(1)$. A set of good function in **algorithm** to be used

1. nth_element(a.begin(), a.begin()+n, a.end())
2. *min_element(a.begin(), a.end())
3. *max_element(a.begin(), a.end())

```
class Solution {
public:
    int minMoves(vector<int>& nums) {
        int m = *min_element(nums.begin(), nums.end());
        long long res = 0;
        for(auto x: nums)
            res += abs((long long)x - m);
        return res;
    }
};
```

5.4 4 Sum Two

Use unordered_map. **time complexity $O(N^2)$, space is $O(N^2)$**

```
class Solution {
public:
    int fourSumCount(
        vector<int>& A,
        vector<int>& B,
        vector<int>& C,
        vector<int>& D
      ) {
        unordered_map<int, int> a, b;
        for(auto x: A) for(auto y: B) a[x+y]++;
        for(auto x: C) for(auto y: D) b[x+y]++;
        int res = 0;
        for(auto x: a){
            if(b.count(-x.first))
                res += x.second*b[-x.first];
        }
        return res;
    }
};
```

5.5 Assign Cookie

Sort and double pointer.**time complexity O(Nlog(N)), space is O(1)**

```
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
        sort(g.begin(), g.end());
        sort(s.begin(), s.end());
        int i=0, j=0, count=0;
        while(i < g.size() && j < s.size()){
```

```
            if(g[i] <= s[j]){
                i++;
                j++;
                count++;
            }
            else{
                j++;
            }
        }
        return count;
    }
};
```

5.6 132 Pattern

Using reverse stack, keep track of the second largest. **time complexity O(N)**, **space is O(N)**

```
class Solution {
public:
    bool find132pattern(vector<int>& nums) {
        stack<int> s;
        int s2 = INT_MIN;
        for(int i=nums.size()-1; i>-1; i--){
            if(nums[i] < s2)
                return true;
            while(!s.empty() && s.top() < nums[i]){
                s2 = s.top();
                s.pop();
            }
            s.push(nums[i]);
        }
        return false;
    }
};
```

5.7