# July 2, 2018

@Mingze X @Chong T @Zhaorui D @Yu S

## 261

1. DFS if loop or separated node = not a vaid tree.

```
class Solution {
    public boolean validTree(int n, int[][] edges) {
        if (edges == null || edges.length == 0 || edges[0] ==
null || edges[0].length == 0) {
            if (n > 1) {
                return false;
            }
            return true;
        }
        boolean[] visited = new boolean[n];

        Map<Integer, Set<Integer>> map = new HashMap<>();

        for (int[] e : edges) {
            if (map.get(e[0]) == null) {
                map.put(e[0], new HashSet<Integer>());
            }
            if (map.get(e[1]) == null) {
                map.put(e[1], new HashSet<Integer>());
            }
            map.get(e[0]).add(e[1]);
            map.get(e[1]).add(e[0]);
        }
        visited[edges[0][0]] = true;
```

```
        if (loop(visited, edges, map, edges[0][0], -1)) {
            return false;
        }
        for (boolean v : visited) {
            if (!v) {
                return false;
            }
        }
        return true;
    }

    boolean loop(boolean[] visited, int[][] edges, Map<Intege
r, Set<Integer>> map, int start, int parent){
        if (map.get(start) == null) {
            return false;
        }

        for (int child: map.get(start)) {
            if (child == parent) {
                continue;
            }
            if (visited[child]) {
                return true;
            }
            visited[child] = true;
            if (loop(visited, edges, map, child, start)) {
                return true;
            }
        }
        return false;
```

```
    }
}
```

2. Typical Union Find problem (have not written the code yet)
3. 楼上应该指并查集：(c++)

```
class Solution {
    vector<int> P;
    void init(int n){
        P.resize(n);
        for(int i=0; i<n; ++i) P[i] = i;
    }
    int findRoot(int i){
        if(P[i] == i) return i;
        return P[i] = findRoot(P[i]);
    }
    void addEdge(int i, int j){
        P[findRoot(j)] = findRoot(i);
    }
public:
    bool validTree(int n, vector<pair<int, int>>& edges) {
        if((int)edges.size() != n-1) return false;
        init(n);
        for(auto p: edges){
            if(findRoot(p.first) == findRoot(p.second)) return
false;
            addEdge(p.first, p.second);
        }
        return true;
    }
};
```

My simple DFS solution:

```
   //This question basically is to 1. verify it is fully conne
cted 2. detect if there is any loop(backedge)


  //Time: O(m+n) Space: O(m+n)
 class Solution {
  public:
    vector<bool> visited;
    map<int, vector<int>> neighbors;
    bool hasLoop = false;
    bool isConnected = true;
    bool validTree(int n, vector<pair<int, int>>& edges) {
        if (edges.size() == 0){
            if (n == 1 || n == 0)
                return true;
            else
                return false;
        }
        visited = vector<bool>(n, false);
        for (auto i : edges){
            if (!neighbors.count(i.first)){
                neighbors[i.first] = {};
            }
             else if (!neighbors.count(i.second)){
                 neighbors[i.second] = {};
            }
            neighbors[i.first].push_back(i.second);
            neighbors[i.second].push_back(i.first);
        }
```

```
        DFS(edges[0].first, -1);

        for (int i = 0; i < visited.size(); i++){
            if (!visited[i]){
                cout<<i<<endl;
                isConnected = false;
            }
        }
        return isConnected && !hasLoop;


    }

    void DFS(int n, int prev){
        if (visited[n]){
            hasLoop = true;
            return;
        }

        visited[n] = true;
        vector<int> neigh = neighbors[n];
        for (auto i : neigh){
            if (i == prev){
                continue;
            }
            DFS(i, n);
        }
    }
};
```

# 543

1. DFS

```
class Solution {
    int res = 0;
    public int diameterOfBinaryTree(TreeNode root) {
        if (root == null) {
            return res;
        }
        dfs(root);
        return res - 1;
    }
    int dfs(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = dfs(root.left);
        int right = dfs(root.right);
        res = Math.max(res, left + right + 1);
        return Math.max(left + 1, right + 1);
    }
}
```

2. 跟上面类似：diameter = 左depth + 右depth

```
class Solution {
    unordered_map<TreeNode*, int> dp;
    int depth(TreeNode* root){
        if(!root) return 0;
        if(dp.count(root)) return dp[root];
        return dp[root] = 1 + max(depth(root->left), depth(roo
t->right));
    }
```

```
public:
    int diameterOfBinaryTree(TreeNode* root) {
        if(!root) return 0;
        return max(
            depth(root->left) + depth(root->right), max(diamet
erOfBinaryTree(root->left), diameterOfBinaryTree(root->righ
t)));
    }
};
```

## 376

1. DP maintain up and down and discuss the relationship of nums[i] with nums[i - 1]

```
class Solution {
    public int wiggleMaxLength(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        int up = 1;
        int down = 1;

        for(int i = 1; i < nums.length; i++) {
            if (nums[i] > nums[i - 1]) {
                up = down + 1;
            }
            if (nums[i] < nums[i - 1]) {
                down = up + 1;
            }
        }

        return Math.max(up, down);
```

```
    }
}
```

2. 同上，wiggle 远期无效性，最长的length只取决于第一步是向上还是向下，所以各扫一遍，求最大即可：

```
class Solution {
    bool needSwap(bool &goUp, int&cur, int&k){
        return (goUp && k>cur) || (!goUp && k<cur);
    }
    int getLength(bool goUp, vector<int>&nums){
        int ans = 1;
        for(int i=1, cur=nums[0]; i<nums.size(); ++i){
            if(needSwap(goUp, cur, nums[i])){
                ++ans;
                goUp = !goUp;
            }
            cur = nums[i];
        }
        return ans;
    }
public:
    int wiggleMaxLength(vector<int>& nums) {
        if(nums.empty()) return 0;
        // go up first
        return max(getLength(true, nums), getLength(false, num
s));
    }
};
```

## 593

1. sort then calculate the distance between the sides and the diagonal side, square if they are equal.

```
    double dist(int[] a, int[] b) {
        return (a[0] - b[0]) * (a[0] - b[0]) + (a[1] - b[1]) * (a[1] - b[1]);
    }
    public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {
        int[][] p = new int[][]{p1,p2,p3,p4};
        Arrays.sort(p, (a,b) -> {
            if (a[0] == b[0]) {
                return a[1] - b[1];
            }
            return a[0] - b[0];
        });
        return dist(p[0], p[1]) != 0 && dist(p[0], p[1]) == dist(p[1], p[3]) &&  dist(p[1], p[3]) == dist(p[2], p[3]) &&  dist(p[2], p[3]) == dist(p[2], p[0]) &&  dist(p[0], p[3]) == dist(p[1], p[2]);
    }
```

2. 简单的几何学判定：

```
class Solution {
    typedef vector<int> vi;
    bool verticalEqual(vi&a, vi&c, vi&d){
        int x1 = c[0]-a[0], y1 = c[1]-a[1], x2 = d[0]-a[0], y2=d[1]-a[1];
        return (x1==y2 && x2==-y1) || (x1==-y2 && x2==y1);
    }
    bool valid(vi&a, vi&b, vi&c, vi&d){
        if(a[0]==b[0] && a[1]==b[1]) return false;
        return verticalEqual(a,c,d) && verticalEqual(b,c,d);
    }
public:
```

```
    bool validSquare(vector<int>& p1, vector<int>& p2, vector<
int>& p3, vector<int>& p4) {
        return valid(p1,p2,p3,p4) || valid(p1,p3,p2,p4) || val
id(p1,p4,p2,p3);
    }
};
```

## 191

1. straightforward counting
2. 同上:

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int cnt=0;
        while(n){
            n &= n-1;
            ++cnt;
        }
        return cnt;
    }
};
```

## 33

1. 1 round binary search

```
class Solution {
    public int search(int[] nums, int target) {
        if (nums == null || nums.length == 0) {
            return -1;
        }

        int start = 0;
```

```
        int end = nums.length - 1;
        while (start + 1 < end) {
            int mid = start + (end - start) / 2;
            if (nums[mid] == target) {
                return mid;
            }
            if (target > nums[end]) {
                if (nums[mid] < target && nums[mid] > nums[en
d]) {
                    start = mid;
                }else {
                    end = mid;
                }
            }else {
                if (nums[mid] > target && nums[mid] < nums[en
d]) {
                    end = mid;
                }else {
                    start = mid;
                }
            }
        }
            if (nums[start] == target) {
                return start;
            }
            if (nums[end] == target) {
                return end;
            }

        return -1;
```

```
    }
}
```

2. 2*bisection:

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int l=0, r=nums.size(), n=nums.size();
        if(!n) return -1;
        if(nums[0] > nums[n-1]){
            while(l < r-1){
                int c = (l+r)/2;
                if(nums[c] > nums[0]) l = c;
                else r = c;
            }
            l = r;
        }
        r = l + n;
        while(l < r-1){
            int c = (l+r)/2;
            if(nums[c%n] > target) r = c;
            else l = c;
        }
        if(nums[l%n] == target) return l%n;
        return -1;
    }
};
```

# 691

1. Bitset dp:

```
class Solution {
    int dp[1<<15 + 2], N;
```

```
    unordered_map<int, vector<int>> pos;
    vector<unordered_map<char, int>> stks;
    int getMin(int stat){
        if(stat >= N) return 0;
        if(dp[stat]) return dp[stat];
        dp[stat] = N;
        for(auto m: stks){
            int tmp = stat;
            for(auto p: m){
                char c = p.first;
                int cnt = p.second;
                for(int j: pos[c]) if(!(1&(tmp>>j))){
                    tmp |= (1<<j);
                    --cnt;
                    if(!cnt) break;
                }
            }
            if(tmp != stat){
                dp[stat] = min(dp[stat], 1 + getMin(tmp));
            }
        }
        return dp[stat];
    }
public:
    int minStickers(vector<string>& stickers, string target) {
        set<char> ref;
        for(int i=0;i<target.size();++i) pos[target[i]].push_b
ack(i);
        for(string w: stickers){
            unordered_map<char, int> tmp;
```

```
            for(char c: w) if(pos.count(c)){
                tmp[c]++;
                if(!ref.count(c)) ref.insert(c);
            }
            if(!tmp.empty()) stks.push_back(tmp);
        }
        if(ref.size() < pos.size()) return -1;
        N = (1 << (int)target.size()) - 1;
        memset(dp, 0, sizeof(dp));
        return getMin(0);
    }
};
```

- [ ] Look at other people's solution (Leetcode discussion) @Zebo L

## 52

1. dfs:

```
class Solution {
    typedef vector<int> vi;
    typedef vector<bool> vb;
    int res = 0;
    void dfs(int i, int n, vi &cur, vector<vb> &mark){
        if(i==n){
            ++res;
            return;
        }
        for(int j=0; j<n; ++j) if(!mark[0][j] && !mark[1][i+j]
&& !mark[2][i-j+n]){
            cur.push_back(j);
            mark[0][j] = mark[1][i+j] = mark[2][i-j+n] = true;
            dfs(i+1, n, cur, mark);
```

```
            mark[0][j] = mark[1][i+j] = mark[2][i-j+n] = fals
e;
        }
    }
public:
    int totalNQueens(int n) {
        vi cur;
        vector<vb> mark(3, vb(n*2, false));
        dfs(0, n, cur, mark);
        return res;
    }
};
```

## 333

1. recusion:

```
class Solution {
    int res = 0;
    void getStatus(TreeNode *root, int &m, int &M, bool & bst,
int &n){
        if(!root){
            m = INT_MAX;
            M = INT_MIN;
            bst = true;
            n = 0;
            return;
        }
        int m1,m2,M1,M2,n1,n2;
        bool bst1,bst2;
        getStatus(root->left, m1, M1, bst1, n1);
        getStatus(root->right, m2, M2, bst2, n2);
        bst = bst1&&bst2&&(M1<root->val)&&(root->val<m2);
```

```
        n = 1 + n1 + n2;
        M = max(root->val, M2);
        m = min(root->val, m1);
        if(bst) res = max(res, n);
    }
public:
    int largestBSTSubtree(TreeNode* root) {
        int m,M,n;
        bool bst;
        getStatus(root, m, M, bst, n);
        return res;
    }
};
```

## 275

1. Bisection search:

```
class Solution {
public:
    int hIndex(vector<int>& citations) {
        if(citations.empty()) return 0;
        int l = 0,r = citations.size(),n = citations.size();
        if(citations[0] >= n) return n;
        if(citations[n-1] == 0) return 0;
        while(l < r-1){
            int c = (l+r)/2;
            if(citations[n-c] >= c) l = c;
            else r = c;
        }
        return l;
    }
```

```
};
```