July 3, 2018

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

344

straightforward

```
class Solution {
public:
    string reverseString(string s) {
        reverse(s.begin(), s.end());
        return s;
    }
};
```

213

do 2 house robber 1, which is 1d dp.

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

        if (nums.length == 1) {
            return nums[0];
        }

        return Math.max(helper(nums, 0, nums.length - 2), help
er(nums, 1, nums.length - 1));
    }
    int helper(int[] nums, int start, int end) {
        int rob = 0;
        int not_rob = 0;
```

```
        for (int i = start; i <= end; i++) {
            int temp = rob;
            rob = not_rob + nums[i];
            not_rob = Math.max(temp, not_rob);
        }
        return Math.max(rob, not_rob);
    }
```

2. Similar as above but `O(1)` space:

```
class Solution {
    typedef vector<int> vi;
    int getAns(int start, int end, vi& nums){
        int u = 0, v = nums[start];
        for(int i=start+1; i<end; ++i){
            int tmp = max(u + nums[i], v);
            u = max(u, v);
            v = tmp;
        }
        return max(u, v);
    }
public:
    int rob(vector<int>& nums) {
        if(nums.empty()) return 0;
        if(nums.size() == 1) return nums[0];
        return max(getAns(0, nums.size()-1, nums), getAns(1, n
ums.size(), nums));
    }
};
```

456

1. find the following pattern 1 < 3 > 2, kind of greedy, from right to left, use a stack to store the 2, and search for 3, the 1 are the minimum from left to right. VERY

EASY TO MAKE MISTAKES

```
class Solution {
    public boolean find132pattern(int[] nums) {
        if (nums == null || nums.length <= 2) {
            return false;
        }
        int third = Integer.MIN_VALUE;
        Stack<Integer> st = new Stack<>();

        for (int i = nums.length - 1; i >= 0; i--) {
            if (nums[i] < third) {
                return true;
            } else {
                while (!st.empty() && nums[i] > st.peek()) {
                    third = st.peek();
                    st.pop();
                }
            }
            st.push(nums[i]);
        }
        return false;
        }
    }
```

2. Using `map`: Complexity `O(n*log(n))`

```
#include<cassert>
class Solution {
public:
    bool find132pattern(vector<int>& nums) {
        if(nums.size()<3) return false;
```

```
        map<int, int> cnt;
        for(int n: nums) cnt[n] += 1;
        int m = nums[0];
        for(int n: nums){
            assert(cnt.count(n));
            cnt[n] -= 1;
            if(!cnt[n]) cnt.erase(n);
            m = min(m, n);
            if(n > m && cnt.upper_bound(m) != cnt.lower_bound
(n)) return true;
        }
        return false;
    }
};
```

3. 借用楼上的Idea，`O(n)` 解法，c++版本：

```
class Solution {
public:
    bool find132pattern(vector<int>& nums) {
        if(nums.size()<3) return false;
        stack<int> S;
        for(int i=nums.size()-1, m=INT_MIN; i>=0; --i){
            if(nums[i] < m) return true;
            while(!S.empty() && nums[i] > S.top()){
                m = S.top();
                S.pop();
            }
            S.push(nums[i]);
        }
        return false;
    }
```

```
};
```

765

1. Easy to write, very hard to think *******
2. 找循环节：

```
#include<cassert>
class Solution {
    inline int getNeighbor(int n){return (n%2)? (n/2)*2: (n/2)
*2+1;}
public:
    int minSwapsCouples(vector<int>& row) {
        int n = row.size(), ans = 0;
        assert(n%2==0);
        vector<int> pos(n);
        for(int i=0;i<n;++i) pos[row[i]] = i;
        unordered_set<int> res;
        for(int i=0;i<n/2;++i) res.insert(i);
        while(!res.empty()){
            int i = *res.begin();
            res.erase(i);
            int start = i*2, tmp = i*2;
            while((tmp = getNeighbor(row[getNeighbor(pos[tm
p])])) != start){
                res.erase(tmp/2);
                ++ans;
            }
        }
        return ans;
    }
};
```

- [ ] Look at solutions in discussion. @Zebo L

549

1. DFS notice what to return on each level, my method is to return the left, right, and the up down direction, many if statements
2. DFS:

```
class Solution {
    int res = 0;
    void dfs(TreeNode*root, int &inc, int &dec){
        if(!root) {
            inc = dec = 0;
            return;
        }
        inc = dec = 0;
        int linc, ldec, rinc, rdec;
        dfs(root->left, linc, ldec);
        dfs(root->right, rinc, rdec);
        if(root->left && abs(root->val - root->left->val)==1){
            if(root->left->val == root->val + 1) inc = max(in
c, linc + 1);
            else dec = max(dec, ldec + 1);
        }
        if(root->right && abs(root->val - root->right->val)==
1){
            if(root->right->val == root->val + 1) inc = max(in
c, rinc + 1);
            else dec = max(dec, rdec + 1);
        }
        res = max(res, dec + inc + 1);
    }
public:
    int longestConsecutive(TreeNode* root) {
        int inc, dec;
```

```
        dfs(root, inc, dec);
        return res;
    }
};
```

501

1. I would use a hashmap
2. Nothing to say:

```
class Solution {
    unordered_map<int, int> cnt;
    int ans = 0;
    void dfs(TreeNode *root){
        if(!root) return;
        ++cnt[root->val];
        ans = max(ans, cnt[root->val]);
        dfs(root->left);
        dfs(root->right);
    }
public:
    vector<int> findMode(TreeNode* root) {
        dfs(root);
        vector<int> res;
        for(auto p: cnt) if(p.second == ans) res.push_back(p.f
irst);
        return res;
    }
};
```

100

1. straightforward dfs, compare using left and right
2. Easy recursion:

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(!p) return q==NULL;
        if(!q) return p==NULL;
        return p->val == q->val && isSameTree(p->left, q->lef
t) && isSameTree(p->right, q->right);
    }
};
```

329

1. DFS + memo
2. 同上，c++ version：

```
class Solution {
    typedef pair<int, int> ii;
    int dx[4] = {1, -1, 0, 0}, dy[4] = {0, 0, 1, -1}, n, m;
    vector<vector<int>> dp;
    int dfs(int i, int j, vector<vector<int>>& matrix){
        if(dp[i][j]) return dp[i][j];
        dp[i][j] = 1;
        for(int k=0; k<4; ++k){
            int i1 = i+dx[k], j1 = j + dy[k];
            if(i1>=0 && i1<n && j1>=0 && j1<m && matrix[i1][j
1]>matrix[i][j]){
                dp[i][j] = max(dp[i][j], 1 + dfs(i1, j1, matri
x));
            }
        }
        return dp[i][j];
    }
public:
```

```
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        if(matrix.empty() || matrix[0].empty()) return 0;
        n = matrix.size();
        m = matrix[0].size();
        int ans = 1;
        dp = vector<vector<int>>(n, vector<int>(m, 0));
        for(int i=0;i<n;++i) for(int j=0;j<m;++j) ans = max(an
s, dfs(i, j, matrix));
        return ans;
    }
};
```

285

1. iterative inorder traversal
2. 在BST中查找, complexity: `O(log n)`

```
class Solution {
public:
    TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
        if(!p) return NULL;
        TreeNode *q = root, *ans = p->right;
        if(ans){
            while(ans->left) ans = ans->left;
            return ans;
        }
        int tar = p->val;
        while(q!=p){
            if(q->val < tar) q = q->right;
            else{
                ans = q;
                q = q->left;
            }
```

```
        }
        return ans;
    }
};
```

612

SQL