June 29, 2018

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

730

`Add solution here`
goodspeed solution:
idea:
`dp[i][j] := dp[i][j-1] + dp[index of first 'a' from i + 1][j-1] - dp[index of first 'a' from i + 1][index of last 'a' from j]`, where assume `s[j] = 'a'`
code:

```
class Solution {
    int M = 1E9 + 7;
    void addM(int &x, int y){
        x += y;
        if(x >= M) x-=M;
    }
    int dp[1002][1002];
    vector<int> pre;
    vector<map<char, int>> next;
    string ref;
    int getCnt(int i, int j){
        if(i > j) return 0;
        if(i == j) return 1;
        if(dp[i][j] > 0) return dp[i][j];
        char c = ref[j];
        if(pre[j] < i){
            return dp[i][j] = getCnt(i, j-1) + 1;
        }
        dp[i][j] = getCnt(i, j-1);
```

```
        addM(dp[i][j], getCnt(next[i][c]+1, j-1));
        addM(dp[i][j], M - getCnt(next[i][c]+1, pre[j] - 1));
        if(pre[j] == next[i][c]) addM(dp[i][j], 1);
        return dp[i][j];
    }
public:
    int countPalindromicSubsequences(string S) {
        pre = vector<int>(S.size(), -1);
        next.resize(S.size());
        ref = S;
        memset(dp, -1, sizeof(dp));
        map<char, int> pos;
        for(int i=0; i<S.size(); ++i){
            if(!pos.count(S[i])) pos[S[i]] = i;
            else{
                pre[i] = pos[S[i]];
                pos[S[i]] = i;
            }
        }
        for(char c: "abcd") pos[c] = (int)S.size();
        for(int i=S.size()-1; i>=0; --i){
            pos[S[i]] = i;
            next[i] = pos;
        }
        return getCnt(0, (int)S.size()-1);
    }
};
```

693

goodspeed solution:

idea: nothing to say
code:

```
class Solution {
public:
    bool hasAlternatingBits(int n) {
        for(int j=1;n>>j;++j) if((1&n>>j) == (1&n>>(j-1))) ret
urn false;
        return true;
    }
};
```

340

goodspeed solution:
idea: sliding window
code:

```
class Solution {
public:
    int lengthOfLongestSubstringKDistinct(string s, int k) {
        int n = s.size(), i=0, j=0, ans = 0;
        map<char, int> cnt;
        while(i<n && j<n){
            while(j<n && cnt.size() <= k){
                cnt[s[j++]] += 1;
            }
            if(cnt.size() <= k) ans = max(ans, j - i);
            else ans = max(ans, j-i-1);
            while(i<n && cnt.size()>k){
                cnt[s[i]] -= 1;
                if(!cnt[s[i]]) cnt.erase(s[i]);
                ++i;
            }
```

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

32

1. BFS remove 1 in every step, then put on queue until getting the first valid one
2. DFS binary tree, remove one left, remove one right
3. find the number and kind to remove then use DFS to do it, return when doing invalid moves

goodspee solution:
idea: stack one pass
code:

```
class Solution {
    typedef pair<int, int> ii;
public:
    int longestValidParentheses(string s) {
        stack<ii> S;
        int ans = 0;
        S.push(ii(0, -1));
        for(int i=0, cur=0; i<s.size(); ++i){
            cur += (s[i]=='('? 1:-1);
            while(!S.empty() && S.top().first > cur) S.pop();
            if(!S.empty() && S.top().first==cur){
                ans = max(ans, i-S.top().second);
                continue;
            }
            else{
                S.push(ii(cur, i));
            }
```

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

190

goodspeed solution:
idea: nothing to say
code:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        int ans = 0;
        for(int i=0;i<32;++i) if(1&(n>>i)) ans |= 1<<(31-i);
        return ans;
    }
};
```

655

goodspeed solution:
idea: dfs and set the string values recursively
code:

```
class Solution {
    int depth(TreeNode *root){
        if(!root) return 0;
        return 1 + max(depth(root->left), depth(root->right));
    }
    vector<int> cnt;
    int d;
    void setTree(TreeNode *root, int dep, int pos, vector<vect
or<string>> &ans){
        if(!root) return;
```

```
        ans[dep][pos] = to_string(root->val);
        if(dep == d) return;
        setTree(root->left, dep+1, pos-1-cnt[dep+1], ans);
        setTree(root->right, dep+1, pos+1+cnt[dep+1], ans);
    }
public:
    vector<vector<string>> printTree(TreeNode* root) {
        d = depth(root);
        cnt.resize(d);
        cnt[d-1] = 0;
        for(int j=d-2;j>=0;--j){
            cnt[j] = 2 * cnt[j+1] + 1;
        }
        vector<vector<string>> ans(d, vector<string>(cnt[0]*2
+ 1, ""));
        setTree(root, 0, cnt[0], ans);
        return ans;
    }
};
```

820

- [ ] Review question @Zebo L

goodspeed solution:
idea: reverse prefix
code:

```
class Solution {
public:
    int minimumLengthEncoding(vector<string>& words) {
        set<string, greater<string>> S;
        for(string s: words) if(!s.empty()){
            reverse(s.begin(), s.end());
```

```
            S.insert(s);
        }
        int ans = 0;
        while(!S.empty()){
            string cur = *S.begin();
            ans += int(cur.size()) + 1;
            vector<string> tmp;
            for(string s: S){
                if(s[0] != cur[0]) break;
                if(cur.substr(0, s.size()) != s) continue;
                else{
                    tmp.push_back(s);
                    cur = s;
                }
            }
            for(string s: tmp) S.erase(s);
        }
        return ans;
    }
};
```

454

goodspeed solution:
idea: do pairing
code:

```
class Solution {
public:
    int fourSumCount(vector<int>& A, vector<int>& B, vector<in
t>& C, vector<int>& D) {
        unordered_map<int, int> AB, CD;
        for(int a: A) for(int b: B){
```

```
            AB[a+b] += 1;
        }
        for(int c: C) for(int d: D){
            CD[c+d] += 1;
        }
        int ans = 0;
        for(auto p: AB) if(CD.count(-p.first)){
            ans += p.second * CD[-p.first];
        }
        return ans;
    }
};
```

155

goodspeed solution:
idea: double stack
code:

```
class MinStack {
public:
    /** initialize your data structure here. */
    stack<int> S, mS;
    MinStack() {

    }

    void push(int x) {
        S.push(x);
        if(mS.empty() || x<=mS.top()) mS.push(x);
    }

    void pop() {
```

```
        if(S.empty()) return;
        if(S.top() == mS.top()) mS.pop();
        S.pop();
    }

    int top() {
        return S.top();
    }

    int getMin() {
        return mS.top();
    }
};
```

107

goodspeed solution:
idea: recursion, (using stack is the same idea)
code:

```
class Solution {
    void setTree(TreeNode *root, vector<vector<int>> &      an
s, int level){
        if(!root) return;
        if(ans.size()<=level) ans.resize(level+1);
        ans[level].push_back(root->val);
        setTree(root->left, ans, level+1);
        setTree(root->right, ans, level+1);
    }
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> ans;
        setTree(root, ans, 0);
```

```
        reverse(ans.begin(), ans.end());
        return ans;
    }
};
```