July 15, 2018

648

1. typical trie problem, be careful.
2. 可以用set，也可以用Trie （这种渣题我居然错了好几次）

```
class Solution {
    struct Tier{
        bool isRoot;
        vector<Tier*> chl;
        Tier() : isRoot(false), chl(vector<Tier*>(26, NULL))
{}
    };
    void insertRoot(Tier *root, string word){
        for(auto c: word){
            if(root->isRoot) return;
            int i = int(c - 'a');
            if(!root->chl[i]) root->chl[i] = new Tier();
            root = root->chl[i];
        }
        root->isRoot = true;
    }
public:
    string replaceWords(vector<string>& dict, string sentence)
{
        Tier lead;
        for(string s: dict) insertRoot(&lead, s);
        string ans;
        auto i = sentence.find_first_not_of(" ");
        while(i<sentence.size()){
            auto j = sentence.find_first_of(" ", i);
```

```
            if(j==string::npos) j=sentence.size();
            Tier *root=&lead;
            for(int k=i; k<j; ++k){
                if(!root) {
                    ans += sentence.substr(k, j-k);
                    break;
                }
                else if(root->isRoot) break;
                else {
                    ans += sentence[k];
                    root = root->chl[int(sentence[k]-'a')];
                }
            }
            ans += " ";
            i = sentence.find_first_not_of(" ", j);
            if(i==string::npos) i = sentence.size();
        }
        ans.pop_back();
        return ans;
    }
};
```

2

1. straightforward adding, watch out for the carry
2. 同上:

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode lead(0);
        int x = 0;
        for(auto p=&lead; l1||l2||x; p=p->next){
```

```
            x += (l1?l1->val:0) + (l2?l2->val:0);
            p->next = new ListNode(x%10);
            x /= 10;
            if(l1) l1 = l1->next;
            if(l2) l2 = l2->next;
        }
        return lead.next;
    }
};
```

575

1. Math problem. min(n/2, unique numbers)
2. 同上：

```
class Solution {
public:
    int distributeCandies(vector<int>& candies) {
        return min(candies.size()/2, set<int>(candies.begin(), candies.end()).size());
    }
};
```

652

1. 只要找一种方法 `hash tree` 的结构即可

```
class Solution {
    unordered_map<string, vector<TreeNode*>> dp;
    string dfs(TreeNode *root){
        if(!root) return "#";
        string ans = to_string(root->val) + "(" + dfs(root->left) + ")" + "(" + dfs(root->right) + ")";
        dp[ans].push_back(root);
        return ans;
```

```
    }
public:
    vector<TreeNode*> findDuplicateSubtrees(TreeNode* root) {
        dfs(root);
        vector<TreeNode *> ans;
        for(auto p: dp){
            if(p.second.size() > 1) ans.push_back(p.second
[0]);
        }
        return ans;
    }
};
```

2. 同上，hashtree with dfs, hash the tree from left to right, use a hash map to count the occurrence of the subtree

```
class Solution {
    public List<TreeNode> findDuplicateSubtrees(TreeNode root)
{
        Map<String, Integer> map = new HashMap<>();
        List<TreeNode> res = new ArrayList<>();
        dfs(root, map, res);
        return res;
    }
    String dfs(TreeNode root, Map<String, Integer> map, List<T
reeNode> res) {
        if (root == null) {
            return "#";
        }
        String cur = root.val + "," + dfs(root.left, map, res)
+ "," + dfs(root.right, map, res);
        map.put(cur, map.getOrDefault(cur, 0) + 1);
        if (map.get(cur) == 2) {
```

```
            res.add(root);
        }
        return cur;
    }
}
```

472

1. 又是Tier :

```
class Solution {
    struct Tier{
        bool isWord;
        map<char, Tier*> chl;
        Tier(): isWord(false) {}
    };
    void insertWord(Tier *root, string word){
        for(auto c: word){
            if(!root->chl.count(c)) root->chl[c] = new Tier();
            root = root->chl[c];
        }
        root->isWord = true;
    }
    bool isConc(Tier *root, string &word, int i, int &cnt){
        auto p = root;
        if(i == word.size()){
            return cnt>1;
        }
        while(p && i<word.size()){
            if(!p->chl[word[i]]) break;
            p = p->chl[word[i++]];
            int tmp = cnt+1;
```

```
            if(p->isWord && isConc(root, word, i, tmp)) return
true;
        }
        return false;
    }
public:
    vector<string> findAllConcatenatedWordsInADict(vector<stri
ng>& words) {
        vector<string> ans;
        Tier lead;
        for(auto s: words) insertWord(&lead, s);
        for(auto s: words) {
            int i = 0, cnt = 0;
            if(isConc(&lead, s, i, cnt)) ans.push_back(s);
        }
        return ans;
    }
};
```

- [ ] See LeetCode Discussion @Zebo L

2. DP, advanced version of 139 Word Break

```
class Solution {
    public List<String> findAllConcatenatedWordsInADict(String
[] words) {
        Set<String> before = new HashSet<>();
        List<String> res = new ArrayList<>();
        Arrays.sort(words, (a, b) -> {
            return a.length() - b.length();
        });
        before.add(words[0]);
        for (int i = 1; i < words.length; i++) {
            if (found(words[i], before)) {
```

```
                res.add(words[i]);
            }
            before.add(words[i]);
        }
        return res;
    }
    boolean found(String str, Set<String> before) {
        boolean[] dp = new boolean[str.length() + 1];
        dp[0] = true;
        for (int i = 1; i < str.length() + 1; i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j] && before.contains(str.substring(j,
i))) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[str.length()];
    }
}
```

680

1. One Pass Check :

```
class Solution {
    bool isPal(string &s, int i, int j){
        while(i<j){
            if(s[i++] != s[j--]) return false;
        }
        return true;
    }
```

```
public:
    bool validPalindrome(string s) {
        int i = 0, j = s.size()-1;
        while(i < j){
            if(s[i] != s[j]) return isPal(s, i+1, j) || isPal
(s, i, j-1);
            ++i;
            --j;
        }
        return true;
    }
};
```

2. straightforward

353

1. 注意一种情况：蛇追着自己跑，这样实际上没有game over

```
class SnakeGame {
    typedef pair<int, int> ii;
public:
    /** Initialize your data structure here.
        @param width - screen width
        @param height - screen height
        @param food - A list of food positions
        E.g food = [[1,1], [1,0]] means the first food is posi
tioned at [1,1], the second is at [1,0]. */
    int W, H, L, food_idx, i, j;
    vector<ii> F;
    queue<ii> Q;
    set<ii> B;
    SnakeGame(int width, int height, vector<pair<int, int>> fo
od) {
```

```
        W = width;
        H = height;
        i = j = food_idx = 0;
        F = food;
        Q.push(ii(0, 0));
        B.insert(ii(0, 0));
    }

    /** Moves the snake.
        @param direction - 'U' = Up, 'L' = Left, 'R' = Right, 'D' = Down
        @return The game's score after the move. Return -1 if game over.
        Game over when snake crosses the screen boundary or bites its body. */
    int move(string direction) {
        if(direction == "U") --i;
        else if(direction == "D") ++i;
        else if(direction == "R") ++j;
        else --j;
        if(food_idx < F.size() && ii(i, j) == F[food_idx]) ++food_idx;
        else{
            B.erase(Q.front());
            Q.pop();
            if(i<0 || i>=H || j<0 || j>=W || B.count(ii(i, j))) return -1;
        }
        Q.push(ii(i, j));
        B.insert(ii(i, j));
        return food_idx;
```

```
    }
};
```

2. Use Deque to simulate the movement of the snake

802

1. memo dp

```
class Solution {
    int dp[10005], inf = 6000000;
    int dfs(int i, vector<vector<int>>& G){
        if(dp[i] >= 0) return dp[i];
        if(G[i].empty()) return dp[i] = 0;
        dp[i] = inf;
        int ans = 0;
        for(auto j: G[i]) ans = max(ans, dfs(j, G));
        return dp[i] = ans;
    }
public:
    vector<int> eventualSafeNodes(vector<vector<int>>& graph)
{
        memset(dp, -1, sizeof(dp));
        vector<int> ans;
        for(int i=0; i<graph.size(); ++i) if(dfs(i, graph) < i
nf) ans.push_back(i);
        return ans;
    }
};
```

31

1. 老题：

```
class Solution {
public:
```

```
    void nextPermutation(vector<int>& nums) {
        int j = nums.size()-1;
        while(j && nums[j] <= nums[j-1]) --j;
        if(j){
            int k = j;
            while(k < nums.size()-1 && nums[k+1] > nums[j-1])
++k;
            swap(nums[j-1], nums[k]);
        }
        reverse(nums.begin() + j, nums.end());
    }
};
```

89

1. Easy recursion:

```
class Solution {
public:
    vector<int> grayCode(int n) {
        if(n==0) return {0};
        vector<int> ans(1<<n);
        auto pre = grayCode(n-1);
        copy(pre.begin(), pre.end(), ans.begin());
        for(auto &k: pre) k += (1<<(n-1));
        reverse(pre.begin(), pre.end());
        copy(pre.begin(), pre.end(), ans.begin() + (1<<(n-
1)));
        return ans;
    }
};
```