July 7, 2018

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

425,

1. straightforward comparing
2. dfs

```
class Solution {
    typedef vector<string> vs;
    set<string> dics;
    void dfs(int i, int &l, vs&vec, vector<vs>&ans){
        if(i==l){
            ans.push_back(vec);
            return;
        }
        string target;
        for(int j=0; j<i; ++j){
            target += vec[j][i];
        }
        auto start = dics.lower_bound(target);
        while(start!=dics.end() && (*start).substr(0, i)==targ
et){
            vec.push_back(*start);
            dfs(i+1, l, vec, ans);
            vec.pop_back();
            ++start;
        }
    }
public:
    vector<vector<string>> wordSquares(vector<string>& words)
{
        int n = words.size(), l = words[0].size();
        for(string w: words) dics.insert(w);
        vector<vs> ans;
        for(string w: dics){
            vs tmp = {w};
            dfs(1, l, tmp, ans);
        }
        return ans;
    }
};
```

- [ ] See LeetCode discussion @Zebo L

510,

2. Not found

244,

1. Watch out for duplicates. Use two pointer method to find out the closest numbers as they are sorted. Trade off between query time and constructor time. The following method gives a TLE.

```
class WordDistance {

    Map<String, Integer> res;

    public WordDistance(String[] words) {
        if (words == null || words.length == 0) {
            return;
        }
        Map<String, List<Integer>> map = new HashMap<>();

        for (int i = 0; i < words.length; i++) {
            if (!map.containsKey(words[i])) {
                map.put(words[i], new ArrayList<Integer>());
            }
            map.get(words[i]).add(i);
        }

        res = new HashMap<>();

        for (int i = 0; i < words.length; i++) {
            for (int j = i + 1; j < words.length; j++) {
                if (res.containsKey(words[i] + ":" + words
[j])) {
                    continue;
                }
                List<Integer> i_list = map.get(words[i]);
                List<Integer> j_list = map.get(words[j]);
                int i_idx = 0;
                int j_idx = 0;
                int dis = Integer.MAX_VALUE;
                while (i_idx < i_list.size() && j_idx < j_lis
t.size()) {
                    if (i_list.get(i_idx) < j_list.get(j_idx))
{
                        dis = Math.min(dis, j_list.get(j_idx)
- i_list.get(i_idx));
                        i_idx++;
                    } else {
                        dis = Math.min(dis, i_list.get(i_idx)
- j_list.get(j_idx));
                        j_idx++;
                    }
                }
                if (words[i].compareTo(words[j]) < 0) {
                    res.put(words[i] + ":" + words[j], dis);
                } else {
                    res.put(words[j] + ":" + words[i], dis);
                }

            }
        }
    }

    public int shortest(String word1, String word2) {
        if (word1.compareTo(word2) < 0) {
            return res.get(word1 + ":" + word2);
        }
        return res.get(word2 + ":" + word1);

    }
}
```

2. 思路同上, c++ version:

```
class WordDistance {
    typedef vector<int> vi;
    unordered_map<string, vi> pos;
public:
    WordDistance(vector<string> words) {
        for(int i=0;i<words.size();++i) pos[words[i]].push_bac
k(i);
    }

    int shortest(string word1, string word2) {
        int i = 0, j = 0, ans = 1E9;
        auto vec1 = pos[word1], vec2 = pos[word2];
        while(i<vec1.size() && j < vec2.size()){
            ans = min(ans, abs(vec1[i] - vec2[j]));
            if(vec1[i] > vec2[j]) ++j;
            else ++i;
        }
        return ans;
    }
};
```

773,

1. BFS. Represent the board with a string, and do searching by moving the 0 to all possible directions while labelling the visited status.
2. 同上

```
#include<cassert>
class Solution {
    typedef pair<int, int> ii;
    int dp[6];
    int getEntry(int &state, int i){
        return (state/dp[i])%6;
    }
    int getZeroEntry(int &state){
        for(int i=0;i<6;++i) if(getEntry(state, i) == 0){
            return i;
        }
        return -1;
    }
    int swapEntry(int state, int j){
        int i = getZeroEntry(state);
        int vj = getEntry(state, j);
        state += vj * dp[i] - vj * dp[j];
        return state;
    }
public:
    int slidingPuzzle(vector<vector<int>>& board) {
        dp[0] = 1;
        for(int i=1;i<6;++i) dp[i] = dp[i-1]*6;
        int fin = dp[0]*1 + dp[1]*2 + dp[2]*3 + dp[3]*4 + dp
[4]*5;
        int ini = dp[0]*board[0][0] + dp[1]*board[0][1] + dp
[2] * board[0][2] +
            dp[3] * board[1][0] + dp[4] * board[1][1] + dp[5]
* board[1][2];
        set<int> S;
        queue<ii> Q;
        Q.push(ii(ini, 0));
        S.insert(ini);
        while(!Q.empty()){
            int state = Q.front().first, step = Q.front().seco
nd;
            if(state == fin) return step;
            Q.pop();
            int i = getZeroEntry(state);
            if(i==0 || i==1 || i==2){
                int tmp_state = swapEntry(state, i+3);
                if(!S.count(tmp_state)){
                    S.insert(tmp_state);
                    Q.push(ii(tmp_state, step+1));
                }
            }
            else{
                int tmp_state = swapEntry(state, i-3);
                if(!S.count(tmp_state)){
                    S.insert(tmp_state);
                    Q.push(ii(tmp_state, step+1));
                }
            }
            if(i!=5 && i!=2){
                int tmp_state = swapEntry(state, i+1);
                if(!S.count(tmp_state)){
                    S.insert(tmp_state);
                    Q.push(ii(tmp_state, step+1));
                }
            }
            if(i!=0 && i!=3){
                int tmp_state = swapEntry(state, i-1);
                if(!S.count(tmp_state)){
                    S.insert(tmp_state);
                    Q.push(ii(tmp_state, step+1));
                }
            }
        }
        return -1;
    }
};
```

563,

1. dfs recursion

```
class Solution {
    int diff=0;
    int getSum(TreeNode *root){
        if(!root) return 0;
        int l = getSum(root->left), r = getSum(root->right);
        diff += abs(l-r);
        return root->val + l + r;
    }

public:
    int findTilt(TreeNode* root) {
        getSum(root);
        return diff;
    }
};
```

2. Straightforward DFS

```
class Solution {
    int res = 0;
    public int findTilt(TreeNode root) {
        dfs(root);
        return res;
    }
    int dfs(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = dfs(root.left);
        int right = dfs(root.right);

        res += Math.abs(left - right);
        return left + right + root.val;
    }
}
```

625,

1. 分解因数

```
class Solution {
public:
    int smallestFactorization(int a) {
        if(a<=9) return a;
        vector<int> factors;
        for(int j=9;j>1;--j){
            while(a%j == 0){
                factors.push_back(j);
                a/=j;
            }
        }
        if(a>1 || factors.size()>9) return 0;
        reverse(factors.begin(), factors.end());
        long ans = 0;
        for(int v: factors) ans = ans*10 + long(v);
        if(ans > long(INT_MAX)) return 0;
        return (int)ans;
    }
};
```

2. Unique way of solving this problem.
   from 9 to 0, divide the number and add them from the smallest on the left to the largest on the right

```
class Solution {
    public int smallestFactorization(int a) {
        if (a < 2) {
            return a;
        }
        long res = 0, mul = 1;
        for (int i = 9; i >= 2; i--) {
            while (a % i == 0) {
                a = a / i;
                res = mul * i + res;
                mul *= 10;
            }
        }
        return a < 2 && res <= Integer.MAX_VALUE ? (int)res : 0;
    }
}
```

336,

1. GoodSpeed:
    a. lang: c++
    b. 很弱的solution：

```
class Solution {
    bool isPal(string s){
        int n = s.size();
        for(int i=0;i<n/2;++i) if(s[i] != s[n-i-1]) return false;
        return true;
    }
public:
    vector<vector<int>> palindromePairs(vector<string>& words) {
        map<string, int> inv;
        vector<vector<int>> ans;
        for(int i=0;i<words.size();++i){
            string s = words[i];
            if(s==""){
                for(int j=0;j<words.size();++j) if(i!=j && isPal(words[j])){
                    ans.push_back(vector<int>{i, j});
                    ans.push_back(vector<int>{j, i});
                }
                continue;
            }
            reverse(s.begin(), s.end());
            inv[s] = i;
        }
        for(auto p: inv){
            cout<<p.first<<' '<<p.second<<endl;
        }
        for(int i=0;i<words.size();++i) if(words[i] != ""){
            char c = words[i][0];
            int l1 = words[i].size();
            auto it = inv.lower_bound(string(1, c));
            cout<<it->first<<endl;
            while(it!=inv.end() && it->first[0]==c){
                string s = it->first;
                int l2 = s.size();
                if(i==it->second || s.substr(0, min(l1, l2)) != words[i].substr(0, min(l1, l2))) {
                    it++;
                    continue;
                }
                if(isPal(s.substr(min(l1, l2))) && isPal(words[i].substr(min(l1, l2)))) ans.push_back(vector<int>{i, it->second});
                it++;
            }
            cout<<i<<endl;
        }
        return ans;
    }
};
```

- [ ] See LeetCode discussion @Zebo L

4. recognize all 4 cases of palindrome and just write it.

785,

1. GoodSpeed's solution:
   a. lang: c++
   b. idea: DFS, 注意多个连通区域的情况

```
class Solution {
    unordered_map<int, bool> dp;
    unordered_set<int> rest;
    bool dfs(int i, bool grp, vector<vector<int>>& graph){
        if(dp.count(i)) return (grp==dp[i]);
        dp[i] = grp;
        rest.erase(i);
        for(int j: graph[i]) {
            if(!dfs(j, !grp, graph)) {
                return false;
            }
        }
        return true;
    }
public:
    bool isBipartite(vector<vector<int>>& graph) {
        for(int i=0;i<graph.size();++i) rest.insert(i);
        while(!rest.empty()){
            int j = *rest.begin();
            if(!dfs(j, true, graph)) return false;
        }
        return true;
    }
};
```

2 BFS to simulate the bipartite process, use a visited array to label status of each node, could be 0 unvisited, 1 blue, 2 red. actually 0,1,-1 is better. Can also use DFS, which is only replacing the queue with a stack.

```
class Solution {
    public boolean isBipartite(int[][] graph) {
        if (graph == null || graph.length == 0) {
            return true;
        }

        int[] visited = new int[graph.length];

        for (int i = 0; i < graph.length; i++) {
            if (visited[i] != 0) {
                continue;
            }
            Queue<Integer> q = new LinkedList<>();
            q.offer(i);
            visited[i] = 1;
            while(!q.isEmpty()){
                int size = q.size();
                int cur = q.poll();

                for (int g : graph[cur]) {
                    if (visited[cur] == visited[g]) {
                        return false;
                    } else if (visited[g] == 0) {
                        if (visited[cur] == 1) {
                            visited[g] = 2;
                        } else {
                            visited[g] = 1;
                        }
                        q.offer(g);
                    }

                }
            }
        }
        return true;
    }
}
```

56,

1. Sort + merge one by one:

```
class Solution {
    static bool cmp(const Interval&i1, const Interval&i2){
        if(i1.start == i2.start) return i1.end<i2.end;
        return i1.start<i2.start;
    }
public:
    vector<Interval> merge(vector<Interval>& intervals) {
        sort(intervals.begin(), intervals.end(), cmp);
        vector<Interval> ans;
        int i = 0;
        while(i<intervals.size()){
            int start=intervals[i].start, end = intervals[i].e
nd;
            while(i<intervals.size() && intervals[i].start<=en
d){
```

```
                end = max(end, intervals[i].end);
                ++i;
            }
            ans.push_back(Interval(start, end));
        }
        return ans;
    }
};
```

2. Sort the array first then scan from left to right, and merge. same as above.

108

1. Easy recusion:

```
class Solution {
    TreeNode* rangeBST(int l, int r, vector<int> nums){
        if(l>r) return NULL;
        TreeNode *root = new TreeNode(nums[(l+r)/2]);
        root->left = rangeBST(l, (l+r)/2-1, nums);
        root->right = rangeBST((l+r)/2+1, r, nums);
        return root;
    }
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return rangeBST(0, (int)nums.size()-1, nums);
    }
};
```

- [ ] See LeetCode discussion @Zebo L

```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        if (nums == null || nums.length == 0) {
            return null;
        }
        int mid = (nums.length) / 2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = sortedArrayToBST(Arrays.copyOfRange(nums,
0 , mid));
        root.right = sortedArrayToBST(Arrays.copyOfRange(nums,
mid + 1, nums.length));
        return root;
    }
}
```