July 11, 2018

844

1. 从后往前扫，注意结束条件。如果一个已经扫完，另一个没有，不一定两者不同，有可能另一个剩下的都被删了。

```
class Solution {
public:
    bool backspaceCompare(string S, string T) {
        int i, j, ni, nj;
        for(i=S.size()-1, j=T.size()-1, ni=0, nj=0;true;){
            while(i>=0 && (ni || S[i]=='#')){
                if(S[i]=='#') ++ni;
                else --ni;
                --i;
            }
            while(j>=0 && (nj || T[j]=='#')){
                if(T[j]=='#') ++nj;
                else --nj;
                --j;
            }
            if(i<0) return j<0;
            if(j<0) return false;
            if(S[i] != T[j]) return false;
            --i;
            --j;
        }
    }
};
```

2. Or use stack, pop when seeing A #, otherwise just keep pushing.

821

1. 先记录所有`C`的位置，然后on pass 即可

```
class Solution {
public:
    vector<int> shortestToChar(string S, char C) {
        vector<int> pos{-20000}, ans;
        for(int i=0;i<S.size();++i) if(S[i]==C) pos.push_back
(i);
        pos.push_back(20000);
        for(int i=0,j=0;i<S.size();++i){
            if(i==pos[j+1]){
                ++j;
                ans.push_back(0);
            }
            else{
                ans.push_back(min(i-pos[j], pos[j+1]-i));
            }
        }
        return ans;
    }
};
```

2. 左到右，右到左scan

```
class Solution {
    public int[] shortestToChar(String S, char C) {
        char[] sc = S.toCharArray();

        int idx = -1;

        int[] res = new int[sc.length];
        Arrays.fill(res, Integer.MAX_VALUE);
        for (int i = 0; i < sc.length; i++) {
            if (idx != -1 && sc[i] != C) {
                res[i] = i - idx;
            } else {
                if (sc[i] == C) {
                    res[i] = 0;
                    idx = i;
                    continue;
                }
            }
        }
        idx = -1;
        for (int i = sc.length - 1; i >= 0; i--) {
            if (idx != -1 && sc[i] != C) {
                res[i] = Math.min(res[i], idx - i);
            } else {
                if (sc[i] == C) {
                    idx = i;
                    continue;
                }
            }
        }
        return res;
    }
}
```

334

1. 用stack，然后在maintain 第三个数的下限

```
class Solution {
public:
    bool increasingTriplet(vector<int>& nums) {
        int lower_bound = INT_MAX;
        stack<int> S;
        for(int n: nums){
            if(n > lower_bound) return true;
            while(!S.empty() && S.top()>=n){
                S.pop();
            }
            if(!S.empty()) lower_bound = min(lower_bound, n);
            S.push(n);
        }
        return false;
    }
};
```

2. 奇技淫巧

```
class Solution {
    public boolean increasingTriplet(int[] nums) {
        int s = Integer.MAX_VALUE;
        int b = Integer.MAX_VALUE;
        for (int n : nums) {
            if (n <= s) {
                s = n;
            } else if (n <= b) {
                b = n;
            } else {
                return true;
            }
        }
        return false;
    }
}
```

17

1. 组合数法，注意一下空`string`即可

```
class Solution {
    const vector<string> ref = {
        "", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"
    };
public:
    vector<string> letterCombinations(string digits) {
        vector<string> ans;
        if(digits.empty()) return ans;
        int prod = 1;
        for(char c: digits) prod *= (int)ref[int(c-'0')].size();
        for(int k=0;k<prod;++k){
            int m = k;
            string tmp;
            for(char c: digits){
                int i = int(c-'0');
                tmp += ref[i][m%(int)ref[i].size()];
                m /= (int)ref[i].size();
            }
            ans.push_back(tmp);
        }
        return ans;
    }
};
```

2. typical dfs

```
class Solution {
    String[] str = {"", "", "abc", "def", "ghi", "jkl", "mno",
"pqrs", "tuv", "wxyz"};

    public List<String> letterCombinations(String digits) {
        List<String> res = new ArrayList<>();
        if (digits == null || digits.length() == 0) {
            return res;
        }
        char[] dc = digits.toCharArray();
        dfs(res, dc, 0, new StringBuilder());
        return res;
    }
    void dfs(List<String> res, char[] dc, int idx, StringBuild
er sb) {
        if (idx == dc.length) {
            StringBuilder temp = new StringBuilder(sb);
            res.add(temp.toString());
            return;
        }
        char[] cur = str[dc[idx] - '0'].toCharArray();
        for (char c : cur) {
            sb.append(c);
            dfs(res, dc, idx + 1, sb);
            sb.deleteCharAt(sb.length() - 1);
        }
    }
}
```

268

1. one 年级题

```
class Solution {
```

```
public:
    int missingNumber(vector<int>& nums) {
        long sum=0, n=nums.size();
        for(int k: nums) sum += (long)k;
        return (int)(n*(n+1)/2 - sum);
    }
};
```

142

1. 三步：
   a. 通过一个`fast pointer`和`slow pointer`来判定是否有`loop` （非常经典的算法）
   b. 然后`slow pointer`不动，`fast pointer`以每步走一格的速度绕场一周来判定`loop`的周长。
   c. 用相隔为这个周长的两个指针从头出发，步长为一，相遇点即为`loop`开始点。

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        if(!head || !head->next) return NULL;
        ListNode *fast = head->next, *slow = head;
        while(fast && fast->next && fast!=slow){
            fast = fast->next->next;
            slow = slow->next;
        }
        if(!fast || !fast->next) return NULL;
        ListNode *lead = head->next;
        while(fast->next != slow){
            lead = lead->next;
            fast = fast->next;
        }
        slow = head;
        while(slow != lead){
            slow = slow->next;
            lead = lead->next;
        }
        return lead;
    }
};
```

2. fast slow pointer

```
public class Solution {
    public ListNode detectCycle(ListNode head) {
        if (head == null || head.next == null) {
            return null;
        }

        ListNode fast = head;
        ListNode slow = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            if(fast == slow) {
                ListNode slow2 = head;
                while (slow2 != slow) {
                    slow = slow.next;
                    slow2 = slow2.next;
                }
                return slow;
            }
        }
        return null;
    }
}
```

55

1. 每次更行跳跃上限。

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        if(nums.empty()) return false;
        for(int i=0, j=nums[0]; i<nums.size() && i<=j; ++i){
            j = max(j, i+nums[i]);
            if(j>=nums.size()-1) return true;
        }
        return false;
    }
};
```

2. 1d dp but the upper one is better

109

1. 经典 `fast` `slow` pointers + recursion:

```
class Solution {
public:
    TreeNode* sortedListToBST(ListNode* head) {
        if(!head) return NULL;
        if(!head->next) return new TreeNode(head->val);
        ListNode *fast=head->next->next, *slow=head;
        while(fast && fast->next){
            fast=fast->next->next;
            slow=slow->next;
        }
        TreeNode *root = new TreeNode(slow->next->val);
        root->right = sortedListToBST(slow->next->next);
        slow->next = NULL;
        root->left = sortedListToBST(head);
        return root;
    }
};
```

29

1. 很烦，注意上下界即可

```
class Solution {
public:
    int divide(int dividend, int divisor) {
        if(!dividend) return 0;
        if(!divisor) return (dividend>0?INT_MAX:INT_MIN);
        long sign = ((dividend>0 && divisor>0)||(dividend<0 &&
divisor<0)? 1:-1);
        long x = abs(long(dividend));
        long y = abs(long(divisor));
        vector<long> pwr(1, y), cnt(1, 1L);
        for(int i=1; pwr[i-1]<=x-pwr[i-1]; ++i) {
            pwr.push_back(pwr[i-1] + pwr[i-1]);
            cnt.push_back(cnt[i-1] + cnt[i-1]);
        }
        long j = (long)cnt.size()-1, ans = 0;
        for(; j>=0; --j) if(x>=pwr[j]){
            x -= pwr[j];
            ans += cnt[j];
        }
        ans *= (long)sign;
        if(ans > long(INT_MAX)) return INT_MAX;
        if(ans < long(INT_MIN)) return INT_MIN;
        return (int)ans;
    }
};
```

690

1. Easy recursion:

```
class Solution {
    unordered_map<int, int> id2idx;
    int getI(vector<Employee*> &employees, int id){
        int ans = employees[id2idx[id]]->importance;
        for(int j: employees[id2idx[id]]->subordinates) ans += 
getI(employees, j);
        return ans;
    }
public:
    int getImportance(vector<Employee*> employees, int id) {
        for(int i=0; i<employees.size(); ++i) id2idx[employees
[i]->id] = i;
        return getI(employees, id);
    }
};
```