July 9, 2018

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

327

1. Preprocess to the sum array first, then do merge sort merging as 315.
2. Counting BST:

```
#include<cassert>

struct BST{
    long val;
    int cnt, gt, lt;
    BST *left, *right;
    BST(long v): val(v), cnt(1), gt(0), lt(0), left(NULL), right(NULL) {}
};

BST *insertNode(BST *root, long val){
    if(!root) return new BST(val);
    if(root->val < val){
        root->gt++;
        root->right = insertNode(root->right, val);
    }
    else if(root->val > val){
        root->lt++;
        root->left = insertNode(root->left, val);
    }
    else{
        root->cnt++;
    }
    return root;
```

```
}

int greaterThan(BST *root, long val){
    if(!root) return 0;
    if(root->val > val) return root->cnt + root->gt + greaterT
han(root->left, val);
    else if(root->val == val) return root->gt;
    else return greaterThan(root->right, val);
}

int lessThan(BST *root, long val){
    if(!root) return 0;
    if(root->val < val) return root->cnt + root->lt + lessThan
(root->right, val);
    else if(root->val == val) return root->lt;
    else return lessThan(root->left, val);
}

class Solution {
public:
    int countRangeSum(vector<int>& nums, int lower, int upper)
{
        if(upper < lower) return 0;
        int ans = 0, n = nums.size();
        vector<long> P(n+1, 0);
        for(int i=0;i<n;++i) P[i+1] = P[i] + (long)nums[i];
        BST *root = NULL;
        root = insertNode(root, P[n]);
        for(int i=n-1, cnt=1; i>=0; --i){
            long new_lower = (long)lower + P[i], new_upper =
(long)upper + P[i];
```

```
            ans += cnt - greaterThan(root, new_upper) - lessTh
an(root, new_lower);
            insertNode(root, P[i]);
            ++cnt;
        }
        return ans;
    }
};
```

- [ ] Check Leetcode discussion @Zebo L

431

empty

651

1. DFS very slow:

```
class Solution {
    void dfs(int rest, int cv, int cur, int& ans){
        if(rest < 3){
            cur += cv * rest;
            ans = max(ans, cur);
            return;
        }
        dfs(rest-1, cv, cur+cv, ans);
        dfs(rest-3, cur, 2*cur, ans);
        return;
    }
public:
    int maxA(int N) {
        if(N <= 6) return N;
        int ans = N;
        for(int i=3;i<=N-3;++i){
            int tmp = N;
            dfs(N-i-3, i, 2*i, tmp);
            ans = max(ans, tmp);
        }
        return ans;
    }
};
```

2. LeetCode discussion 里面的一个 `dp` 方法，非常nb：

```
class Solution {
public:
    int maxA(int N) {
        vector<int> dp(N+1, 0);
        for(int i=1;i<=N;++i) dp[i] = i;
        for(int i=1;i<=N;++i){
            for(int j=i+3, cnt=2; j<=N; ++j,++cnt) dp[j]=max(d
p[j], cnt*dp[i]);
        }
        return dp[N];
    }
};
```

3. dp is better

```
class Solution {
    public int maxA(int N) {
        int[] dp = new int[N + 1];
        for (int i = 0; i < N + 1; i++) {
            dp[i] = i;
        }

        for (int i = 0; i < N + 1; i++) {
            for (int j = 1; j <= i - 3; j++) {
                dp[i] = Math.max(dp[i], dp[j] * (i - j - 1));
            }
        }

        return dp[N];
    }
}
```

343

1. dp:

```
class Solution {
    int dp[60];
    int dfs(int n){
        if(n<4) return n-1;
        if(dp[n]) return dp[n];
        for(int i=2; i<=n/2; ++i){
            dp[n] = max(dp[n], i * (n-i));
            dp[n] = max(dp[n], i * dfs(n-i));
            dp[n] = max(dp[n], (n-i) * dfs(i));
            dp[n] = max(dp[n], dfs(i) * dfs(n-i));
        }
        return dp[n];
    }
public:
    int integerBreak(int n) {
        memset(dp, 0, sizeof(dp));
        return dfs(n);
    }
};
```

2. dp, consider the sub problems, number i may be the sum of i and i - j. but i, i - j may contain the results of dp[i] and dp[i - j], the larger one shall be used.

```
class Solution {
    public int integerBreak(int n) {
        int[] dp = new int[n + 1];
        dp[1] = 1;

        for (int i = 2; i < n + 1; i++) {
            for (int j = 1; j < i; j++) {
                dp[i] = Math.max(dp[i], Math.max(j,dp[j]) * Ma
th.max(i-j, dp[i-j]));
            }
        }
        return dp[n];
    }
}
```

28,

1. KMP:

```
class Solution {
    vector<int> kmp;
    void init(string s){
        kmp = vector<int>(s.size(), -1);
        kmp[0] = -1;
        for(int j=1, i=0; j<s.size(); ++j){
            while(i && s[j] != s[i]) i = kmp[i-1] + 1;
            if(s[i]==s[j]) kmp[j] = i++;
            else kmp[j] = -1;
        }
    }
public:
    int strStr(string haystack, string needle) {
        if(needle.empty()) return 0;
        if(needle.size() > haystack.size()) return -1;
        init(needle);
        int i=0, j=0;
        while(j<haystack.size() && i<needle.size()){
            while(i && haystack[j]!=needle[i]) i = kmp[i-1]+1;
            if(haystack[j] == needle[i]) ++i;
            ++j;
        }
        if(i==needle.size()) return j-i;
        return -1;
    }
};
```

- [ ] Remember this @Zebo L

2. I would use the regular O(n^2) instead of the KMP.

442

1. 不停swap，一开始想one pass搞定，搞了1个小时没搞定，最终还是two pass：
    a. Note: Questions need inplace swappings always requires at least two passes:
        - 1st pass: doing swap.
        - 2nd pass: gather results.

```
class Solution {
public:
    vector<int> findDuplicates(vector<int>& nums) {
        int n = nums.size(), i=0, j=0;
        while(i<n){
            while(nums[i]!=i+1 && nums[i]!=nums[nums[i]-1]){
                swap(nums[i], nums[nums[i]-1]);
            }
            ++i;
        }
        for(i=0;i<n;++i){
            if(nums[i]!=i+1) nums[j++] = nums[i];
        }
        nums.resize(j);
        return nums;
    }
};
```

2. Negating technique

```
class Solution {
    public List<Integer> findDuplicates(int[] nums) {
        List<Integer> res = new ArrayList<>();
        if (nums == null || nums.length == 0) {
            return res;
        }

        for (int i = 0; i < nums.length; i++) {
            int idx = Math.abs(nums[i]) - 1;
            if (nums[idx] < 0) {
                res.add(Math.abs(nums[i]));
            } else {
                nums[idx] = -nums[idx];
            }
        }
        return res;
    }
}
```

1. Use a stack to look back. Not difficult but need a lot of attention.
2. Recursion:
    a. Note: should use `.add()` instead of `.getList().push_back()` to add new objects
    b. Do not get into recursion if the string is empty:

```
class Solution {
    string ref;
    NestedInteger dfs(int l, int r){
        NestedInteger ni;
        if(ref[l] != '[') ni.setInteger(stoi(ref.substr(l, r-l)));
        else{
            int start = l+1;
            for(int j=start, cnt=0; j<r-1; ++j){
                if(ref[j]=='[') ++cnt;
                else if(ref[j]==']') --cnt;
                else if(ref[j]==',' && !cnt){
                    ni.add(dfs(start, j));
                    start = j+1;
                }
            }
            if(r-1 > start) ni.add(dfs(start, r-1));
        }
        return ni;
    }
public:
    NestedInteger deserialize(string s) {
        ref = s;
        return dfs(0, (int)s.size());
    }
};
```

246

1. Very straightforward
2. 正如楼上所说：

```
class Solution {
    map<char, char> ref;
public:
    bool isStrobogrammatic(string num) {
        ref['0'] = '0';
        ref['1'] = '1';
        ref['6'] = '9';
        ref['8'] = '8';
        ref['9'] = '6';
        int n = num.size();
        for(int i=0; i<(n+1)/2; ++i) if(!ref.count(num[i]) ||
ref[num[i]] != num[n-i-1]) return false;
        return true;
    }
};
```

542

1. Straightforward BFS
2. DP?!
3. BFS:

```
class Solution {
    int di[4]={1,-1,0,0}, dj[4]={0,0,1,-1};
    typedef pair<int, int> ii;
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& matr
ix) {
        int n = matrix.size(), m = matrix[0].size();
        vector<vector<int>> ans(n, vector<int>(m, -1));
        queue<ii> Q;
        for(int i=0;i<n;++i) for(int j=0;j<m;++j) if(matrix[i][j]==0){
            ans[i][j] = 0;
            Q.push(ii(i, j));
        }
        while(!Q.empty()){
            int i = Q.front().first, j = Q.front().second;
            Q.pop();
            for(int k=0;k<4;++k){
                int i1 = i+di[k], j1= j+dj[k];
                if(i1>=0 && i1<n && j1>=0 && j1<m && ans[i1][j1]<0){
                    ans[i1][j1] = ans[i][j] + 1;
                    Q.push(ii(i1, j1));
                }
            }
        }
        return ans;
    }
};
```

612

SQL