1 - 全球最火IT公司面试揭秘

439. 线段树的构造 II

http://www.lintcode.com/problem/segment-tree-build-ii

线段树是一棵二叉树，他的每个节点包含了两个额外的属性start和end用于表示该节点所代表的区间。start和end都是整数，并按照如下的方式赋值:

根节点的 start 和 end 由 build 方法所给出。 对于节点 A 的左儿子，有 start=A.left, end=(A.left + A.right) / 2。 对于节点 A 的右儿子，有 start=(A.left + A.right) / 2 + 1, end=A.right。 如果 start 等于 end, 那么该节点是叶子节点，不再有左右儿子。 对于给定数组设计一个build 方法，构造出线段树

说明

wiki: Segment Tree Interval Tree

Source code:

```
/**
 * Definition of SegmentTreeNode:
 * class SegmentTreeNode {
 * public:
 *     int start, end, max;
 *     SegmentTreeNode *left, *right;
 *     SegmentTreeNode(int start, int end, int max) {
 *         this->start = start;
 *         this->end = end;
 *         this->max = max;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
private:
    // V1 我的传统做法 ,不是最优
    SegmentTreeNode * buildHelper1(vector<int>& A, int start, int end){
        if(start > end) {
            return NULL;
        }
        int maxVal = A[start];
        for(int i = start; i <= end; i++){
            if(maxVal < A[i]){
```

```
                maxVal = A[i]; //找最大值
            }
        }
        SegmentTreeNode * root = new SegmentTreeNode(start, end, maxVal);
        if(start != end){
            root->left = buildHelper(A, start, (start+end) / 2);
            root->right = buildHelper(A, (start+end) / 2 + 1, end);
        }
                return root;
    }
    // V2 分治法
    SegmentTreeNode * buildHelper(vector<int>& A, int start, int end){
        if(start > end) {
            return NULL;
        }
        SegmentTreeNode * root = new SegmentTreeNode(start, end, A[start]);
            root->left = buildHelper(A, start, (start+end) / 2);
            root->right = buildHelper(A, (start+end) / 2 + 1, end);
            root->max = max(root->left->max, root->right->max);
        }

                return root;
    }
public:
    /**
     *@param A: a list of integer
     *@return: The root of Segment Tree
     */
    SegmentTreeNode * build(vector<int>& A) {
        // write your code here
        if(A.size() == 0){
            return NULL;
        }
        return buildHelper(A, 0, A.size()-1);
    }
};
```

205. 区间最小数

http://www.lintcode.com/problem/interval-minimum-number

给定一个整数数组（下标由 0 到 n-1，其中 n 表示数组的规模），以及一个查询列表。每一个查询列表有两个整数 [start, end]。 对于每个查询，计算出数组中从下标 start 到 end 之间的数的最小值，并返回在结果列表中。

注意事项 在做此题前，建议先完成以下三道题 线段树的构造， 线段树的查询 及 线段树的修改。

样例

对于数组 [1,2,7,8,5]， 查询 [(1,2),(0,4),(2,4)]，返回 [2,1,5]

Source code:

```
/**
 * Definition of Interval:
 * classs Interval {
 *     int start, end;
 *     Interval(int start, int end) {
 *         this->start = start;
 *         this->end = end;
 *     }
 */
class SegmentTreeNodeX {
public:
      int start, end, min;
      SegmentTreeNodeX *left, *right;
      SegmentTreeNodeX(int start, int end, int min) {
          this->start = start;
          this->end = end;
          this->min = min;
          this->left = this->right = NULL;
      }
};
class Solution {
public:
    SegmentTreeNodeX * build(vector<int> &A, int start, int end) {
        if(start > end) {
            return NULL;
        }
        SegmentTreeNodeX * root = new SegmentTreeNodeX(start, end,
            numeric_limits<int>::max());
        if(start == end) {
            root->min = A[start];
        }
        else {
            int mid = (start + end) / 2;
            root->left  = build(A, start, mid);
            root->right = build(A, mid+1, end);
                        root->min = min(root->left->min, root->right->min);
        }

    }
    int query(SegmentTreeNodeX * root,  int start, int end){
        if(root->start == start && root->end == end){
```

```
            return root->min;
        }

                int leftMin = numeric_limits<int>::max(), rightMin =
numeric_limits<int
            >::max();
        int mid = (root->start + root->end) / 2;
        if(start <= mid) {
            if(mid >= end) {
                leftMin = query(root->left, start, end);
            }
            else {
                leftMin = query(root->left, start, mid);
            }
        }
        if(mid < end){
            if(start <= mid){
                rightMin = query(root->right, mid+1, end);
            }
            else {
                rightMin = query(root->right, start, end);
            }
        }

                return min(leftMin, rightMin);
    }
    /**
     *@param A, queries: Given an integer array and an query list
     *@return: The result list
     */
    vector<int> intervalMinNumber(vector<int> &A, vector<Interval> &queries)
{
        // write your code here
        vector<int> ans;
        if(A.size() == 0) {
            return ans;
        }
        SegmentTreeNodeX * root = build(A, 0, A.size()-1);
        for(int i = 0; i < queries.size(); i++){
            Interval q = queries[i];
            ans.push_back(query(root, q.start, q.end));
        }
        return ans;

            }
};
```

203. 线段树的修改

http://www.lintcode.com/problem/segment-tree-modify

对于一棵 最大线段树, 每个节点包含一个额外的 max 属性，用于存储该节点所代表区间的最大值。

设计一个 modify 的方法，接受三个参数 root、 index 和 value。该方法将 root 为跟的线段树中 [start, end] = [index, index] 的节点修改为了新的 value ，并确保在修改后，线段树的每个节点的 max 属性仍然具有正确的值。 注意事项 在做此题前，最好先完成线段树的构造和 线段树查询这两道题目。

样例

对于线段树: [1, 4, max=3] / \ [1, 2, max=2] [3, 4, max=3] / \ / \ [1, 1, max=2], [2, 2, max=1], [3, 3, max=0], [4, 4, max=3] 如果调用 modify(root, 2, 4), 返回: [1, 4, max=4] / \ [1, 2, max=4] [3, 4, max=3] / \ / \ [1, 1, max=2], [2, 2, max=4], [3, 3, max=0], [4, 4, max=3] 或 调用 modify(root, 4, 0), 返回: [1, 4, max=2] / \ [1, 2, max=2] [3, 4, max=0] / \ / \ [1, 1, max=2], [2, 2, max=1], [3, 3, max=0], [4, 4, max=0]

Source code:

```
/**
 * Definition of SegmentTreeNode:
 * class SegmentTreeNode {
 * public:
 *     int start, end, max;
 *     SegmentTreeNode *left, *right;
 *     SegmentTreeNode(int start, int end, int max) {
 *         this->start = start;
 *         this->end = end;
 *         this->max = max;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     *@param root, index, value: The root of segment tree and
     *@ change the node's value with [index, index] to the new given value
     *@return: void
     */
    void modify(SegmentTreeNode *root, int index, int value) {
        // write your code here
        if(root == NULL)
        {
            return;
        }
        if(root->start == index && root->end == index)
        {
            root->max = value;
            return;
        }
        int mid = (root->start + root->end ) / 2;
        if(root->start <= index && index <= mid)
        {
            modify(root->left, index, value);
        }
        if(mid < index && index <= root->end)
        {
            modify(root->right, index, value);
        }
        return;
    }
};
```

202. 线段树的查询

http://www.lintcode.com/problem/segment-tree-query

对于一个有n个数的整数数组，在对应的线段树中, 根节点所代表的区间为0-n-1, 每个节点有一个额外的属性max，值为该节点所代表的数组区间start到end内的最大值。

为SegmentTree设计一个 query 的方法，接受3个参数root, start和end，线段树root所代表的数组中子区间[start, end]内的最大值。 注意事项 在做此题之前，请先完成 线段树构造 这道题目。

样例

对于数组 [1, 4, 2, 3], 对应的线段树为： [0, 3, max=4] / \ [0,1,max=4] [2,3,max=3] / \ / \ [0,0,max=1] [1,1,max=4] [2,2,max=2], [3,3,max=3] query(root, 1, 1), return 4 query(root, 1, 2), return 4 query(root, 2, 3), return 3 query(root, 0, 2), return 4

Source code:

```
/**
 * Definition of SegmentTreeNode:
 * class SegmentTreeNode {
 * public:
 *     int start, end, max;
 *     SegmentTreeNode *left, *right;
 *     SegmentTreeNode(int start, int end, int max) {
 *         this->start = start;
 *         this->end = end;
 *         this->max = max;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     *@param root, start, end: The root of segment tree and
     *                         an segment / interval
     *@return: The maximum number in the interval [start, end]
     */
    // version : DFS <Recommended>
    int query(SegmentTreeNode *root, int start, int end) {
        // write your code here
        if(root == NULL)  // should never happen
        {
            return numeric_limits<int>::min();
        }
                if(start <= root->start && end >= root->end)
        {
            return root->max;
```

```
        }
        int mid = (root->start + root->end) / 2;
        int leftMax = numeric_limits<int>::min(), rightMax =
numeric_limits<int
            >::min();
        if(start <= mid) // 左子区
        {
            if(mid < end)
            {
                leftMax = query(root->left, start, mid);
            else
            {
                leftMax = query(root->left, start, end);
            }
        }

                if(mid < end) // 右子区
        {
            if(start <= mid)
            {
                rightMax = query(root->right, mid+1, end);
            }
            else
            {
                rightMax = query(root->right, start, end);
            }
        }

                return max(leftMax, rightMax);
    }
    // version 2: use BFS
    int query2(SegmentTreeNode *root, int start, int end) {
        if(root == NULL) {
            return numeric_limits<int>::min();
        }
        int ans = numeric_limits<int>::min();
        queue<SegmentTreeNode*> q;
        q.push(root);
        SegmentTreeNode * node = NULL;
        while(!q.empty()) {
            node = q.front();
            q.pop();
            if(start == node->start && node->end == end) {
                return node->max;
            }
            if(start <= node->start && node->end <= end) {
                ans = max(ans, node->max);
            }
            int middle = node->start + (node->end - node->start) / 2;
            if(start <= middle && node->left != NULL) {
                q.push(node->left);
```

```
            }
            if(end > middle && node->right != NULL) {
                q.push(node->right);
            }

                    }
        return ans;
    }
};
```

201. 线段树的构造

http://www.lintcode.com/problem/segment-tree-build

线段树是一棵二叉树，他的每个节点包含了两个额外的属性start和end用于表示该节点所代表的区间。start和end都是整数，并按照如下的方式赋值:

根节点的 start 和 end 由 build 方法所给出。 对于节点 A 的左儿子，有 start=A.left, end=(A.left + A.right) / 2。 对于节点 A 的右儿子，有 start=(A.left + A.right) / 2 + 1, end=A.right。 如果 start 等于 end, 那么该节点是叶子节点，不再有左右儿子。 实现一个 build 方法，接受 start 和 end 作为参数, 然后构造一个代表区间 [start, end] 的线段树，返回这棵线段树的根。

说明

线段树(又称区间树), 是一种高级数据结构，他可以支持这样的一些操作: 查找给定的点包含在了哪些区间内 查找给定的区间包含了哪些点 见百科： 线段树 区间树

Source code:

```
/**
 * Definition of SegmentTreeNode:
 * class SegmentTreeNode {
 * public:
 *     int start, end;
 *     SegmentTreeNode *left, *right;
 *     SegmentTreeNode(int start, int end) {
 *         this->start = start, this->end = end;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     *@param start, end: Denote an segment / interval
     *@return: The root of Segment Tree
     */
    SegmentTreeNode * build(int start, int end) {
                if(start > end) {
            return NULL;
        }
        // write your code here
        SegmentTreeNode * root = new SegmentTreeNode(start, end);
        if(start != end)
        {
            int middle = start + (end - start) / 2;
            root->left = build(start, middle);
            root->right = build(middle+1, end);
        }
        return root;
    }
};
```

5. 第k大元素

http://www.lintcode.com/problem/kth-largest-element

在数组中找到第k大的元素

注意事项 你可以交换数组中的元素的位置

样例

给出数组 [9,3,2,4,8]，第三大的元素是 4 给出数组 [1,2,3,4,5]，第一大的元素是 5，第二大的元素是 4，第三大的元素是 3，以此类推

Source code:

```
class Solution {
private:
    void swap(vector<int> &nums, int a, int b){
        int tmp = nums[a];
        nums[a] = nums[b];
        nums[b] = tmp;
    }
    int partition(vector<int> &nums, int start, int end){
        if(start >= end){
            return 0;
        }
        int left = start;
        int right = end - 1;
        int pivot = nums[end];
        while(left <= right){
            while(left <= right && nums[left] >= pivot){
                left++;
            }
            while(left <= right && pivot >= nums[right]){
                right--;
            }
            if(left < right){
                swap(nums, left, right);
            }
        }
        swap(nums, left, end);
        return left-start;
    }
    int findKth(int k, vector<int> &nums, int start, int end){
        if(start == end){
            return nums[start];
        }
        int j = partition(nums, start, end);
        if(j == (k -1)){
            return nums[j];
        }
        else if (j < (k -1)){
            return findKth(k-j-1, nums, start+j+1, end);
        }
        else {
            return findKth(k, nums, start, start+j-1);
public:
    /*
     * param k : description of k
     * param nums : description of array and index 0 ~ n-1
     * return: description of return
     */
    int kthLargestElement(int k, vector<int> nums) {
        // write your code here
        if(nums.size() == 0){
            return -1;
```

```
        }
        return findKth(k, nums, 0, nums.size() - 1);
    }
};
```

2 - 高级数据结构（上）

477. 被围绕的区域

http://www.lintcode.com/problem/surrounded-regions

给一个二维的矩阵，包含 'X' 和 'O', 找到所有被 'X' 围绕的区域，并用 'X' 填充满。

样例

给出二维矩阵： X X X X X O O X X X O X X O X X 把被 'X' 围绕的区域填充之后变为： X X X X X X X X X X X X O X X

Source code:

```
class UnionFind{
private:
    unordered_map<int, int> father;
public:
    UnionFind(int n, int m){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                int id = i * m + j;
                father[id] = id;
            }
        }
    }
    int find(int id){
        int parent = id;
        while(parent != father[parent]){
            parent = father[parent];
        }
        return parent;
    }
    int compress_find(int id){
        int parent = id;
        while(parent != father[parent]){
            parent = father[parent];
        }
                int fa = id;
```

```
        while(fa != father[fa]){
            int tmp = father[fa];
            father[fa] = parent;
            fa = tmp;
        }
                return parent;
    }
    void union_both(int a, int b){
        int fa_a = compress_find(a);
        int fa_b = compress_find(b);
        if(fa_a != fa_b){
            father[fa_a] = father[fa_b];
        }
    }
public:
    /**
     * @param board a 2D board containing 'X' and 'O'
     * @return void
     */
    // V1
    //思路：目标是要找到由X包围起来的O的区域。
    // 首先，外围一圈上的O肯定会保留下来。然后，从外围的O能达到的O也要保留。剩下
        其他的O就是内部的O。
    // 所以方法就是从外围的一圈进行DFS算法：依次对外圈的“O”做DFS，将其可以到达O
        临时设置为#。
    void surroundedRegions1(vector<vector<char>>& board) {
        // Write your code here
        if(board.size() == 0 || board[0].size() == 0)
            return;

                fillBoarder(board, 'O', '#');
        replace(board, 'O', 'X');
        fillBoarder(board, '#', 'O');
    }

        // BFS
    void fillBoarder(vector<vector<char>>& board, char patten, char c){
        int n = board.size();
        int m = board[0].size();
        queue<pair<int,int> > q;//save thoese with patten

                // set those boarder pattern with c
        for(int i = 0; i < board.size(); i++){
            if(board[i][0] == patten){
                board[i][0] = c;
                q.push(make_pair(i, 0));
            }
            if(board[i][m-1] == patten){
                board[i][m-1] = c;
                q.push(make_pair(i, m-1));
            }
```

```
        }
        for(int j = 1; j < m-1; j++){
            if(board[0][j] == patten){
                board[0][j] = c;
                q.push(make_pair(0, j));
            }
            if(board[n-1][j] == patten){
                board[n-1][j] = c;
                q.push(make_pair(n-1, j));
            }
        }

                int dx[4] = {-1, 1, 0, 0};
        int dy[4] = {0, 0, -1, 1};
        while(!q.empty()){
            int x = q.front().first;
            int y = q.front().second;
            q.pop();
            for(int k = 0; k < 4; k++){
                int nx = x + dx[k];
                int ny = y + dy[k];
                if(nx >= 0 && nx < n && ny >= 0 && ny < m && board[nx][ny]
==
                    patten){
                    board[nx][ny] = c;
                    q.push(make_pair(nx, ny));
                }
            }
        }
    }

        void replace(vector<vector<char>>& board, char patten, char c){
        for(int i = 0; i < board.size(); i++){
            for(int j = 0; j < board[0].size(); j++){
                if(board[i][j] == patten){
                    board[i][j] = c;
                }
            }
        }
    }

        // V2 类似于V1, but use DFS
    void surroundedRegions(vector<vector<char>>& board) {
        // Write your code here
        if(board.size() == 0 || board[0].size() == 0)
            return;

                fillBoarderDFS(board, 'O', '#');
        replace(board, 'O', 'X');
        fillBoarderDFS(board, '#', 'O');
    }
```

```
    // DFS
    void fillBoarderDFS(vector<vector<char>>& board, char patten, char c){
        int n = board.size();
        int m = board[0].size();
        queue<pair<int,int> > q;//save thoese with patten

                // set those boarder pattern with c
        for(int i = 0; i < board.size(); i++){
            fillDFS(board, i, 0, patten, c);
            fillDFS(board, i, m-1, patten, c);
        }
        for(int j = 1; j < m-1; j++){
            fillDFS(board, 0, j, patten, c);
            fillDFS(board, n-1, j, patten, c);
        }
    }
    void fillDFS(vector<vector<char>>& board, int x, int y, char patten,
char c
        ){
        if(board[x][y] != patten){
            return;
        }
        board[x][y] = c;
        int n = board.size();
        int m = board[0].size();
        // set those boarder pattern with c
        int dx[4] = {-1, 1, 0, 0};
        int dy[4] = {0, 0, -1, 1};
        for(int k = 0; k < 4; k++){
            int nx = x + dx[k];
            int ny = y + dy[k];
            if(nx >= 0 && nx < n && ny >= 0 && ny < m){
                fillDFS(board, nx, ny, patten, c);
            }

                    }
    }

////////////////////////////////////////////////////////////////////////////
        ///
    // V3 : Uinon Find
    void surroundedRegions3(vector<vector<char>>& board) {
        if(board.size() == 0 || board[0].size() == 0)
            return;
        int n = board.size();
        int m = board[0].size();
        UnionFind uf(n, m);
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                int dx[4] = {-1, 1, 0, 0};
                int dy[4] = {0, 0, -1, 1};
```

```
                for(int k = 0; k < 4; k++){
                    int nx = i + dx[k];
                    int ny = j + dy[k];
                    if(nx >= 0 && nx < n && ny >= 0 && ny < m && board[nx]
[ny]
                        == board[i][j]){
                        int id = i * m + j;
                        int nid = nx * m + ny;
                        uf.union_both(id, nid);
                    }
                }
            }
        }
        // 将边角以及与其相邻的‘O’ 的father 加入set
        unordered_set<int> set;
        for(int i = 0; i < n; i++){
            if(board[i][0] == 'O'){
                int fa = uf.compress_find(i*m);
                set.insert(fa);
            }
            if(board[i][m-1] == 'O'){
                int fa = uf.compress_find(i * m + m - 1);
                set.insert(fa);
            }
        }
        for(int j = 1; j < m-1; j++){
            if(board[0][j] == 'O'){
                int fa = uf.compress_find(j);
                set.insert(fa);
            }
            if(board[n-1][j] == 'O'){
                int fa = uf.compress_find((n-1)*m + j);
                set.insert(fa);
            }
        }
        // 判断是否和边角的‘O’同一个group
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                int fa = uf.find(i * m + j);
                if(board[i][j] == 'O' && set.find(fa) == set.end()){
                    board[i][j] = 'X';
                }
            }
        }
    }
};
```

247. 线段树查询 II

http://www.lintcode.com/problem/segment-tree-query-ii

对于一个数组，我们可以对其建立一棵 线段树, 每个结点存储一个额外的值 count 来代表这个结点所指代的数组区间内的元素个数. (数组中并不一定每个位置上都有元素)

实现一个 query 的方法，该方法接受三个参数 root, start 和 end, 分别代表线段树的根节点和需要查询的区间，找到数组中在区间[start, end]内的元素个数。

样例

对于数组 [0, 空，2, 3], 对应的线段树为： [0, 3, count=3] / \ [0,1,count=1] [2,3,count=2] / \ / \ [0,0,count=1] [1,1,count=0] [2,2,count=1], [3,3,count=1] query(1, 1), return 0 query(1, 2), return 1 query(2, 3), return 2 query(0, 2), return 2

Source code:

```
/**
 * Definition of SegmentTreeNode:
 * class SegmentTreeNode {
 * public:
 *     int start, end, count;
 *     SegmentTreeNode *left, *right;
 *     SegmentTreeNode(int start, int end, int count) {
 *         this->start = start;
 *         this->end = end;
 *         this->count = count;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     *@param root, start, end: The root of segment tree and
     *                         an segment / interval
     *@return: The count number in the interval [start, end]
     */
    int query(SegmentTreeNode *root, int start, int end) {
        // write your code here
        if(root == NULL) {
            return 0;
        }
                if(start <= root->start && root->end <= end) {
            return root->count;
        }
                int leftCount = 0, rightCount = 0;
        int middle = (root->start + root->end) / 2;
        if(start <= middle) {
```

```
            if(middle < end) {
                leftCount = query(root->left, start, middle);
            }
            else {
                leftCount = query(root->left, start, end);
            }
        }
                rightCount = query(root->right, middle+1, end);
            }
            else {
                rightCount = query(root->right, start, end);
            }

                    }

                return (leftCount + rightCount);

            }
};
```

465. 两个排序数组和的第K小

http://www.lintcode.com/problem/kth-smallest-sum-in-two-sorted-arrays

给定两个排好序的数组 A, B，定义集合 sum = a + b，求 sum 中第k小的元素

样例

给出 A = [1,7,11] B = [2,4,6] 当 k = 3, 返回 7. 当 k = 4, 返回 9. 当 k = 8, 返回 15.

Source code:

```
class Element {
public:
    int row;
    int col;
    int val;
    Element(int _row, int _col, int _val){
        this->row = _row;
        this->col = _col;
        this->val = _val;
    }
    bool operator() (const Element &a, const Element &b) const{
        return a.val > b.val; // min heap
    }
    bool operator< (const Element &obj) const{
        return this->val > obj.val;
```

```
    }
};
class Solution {
public:
    /**
     * @param A an integer arrays sorted in ascending order
     * @param B an integer arrays sorted in ascending order
     * @param k an integer
     * @return an integer
     */
    int kthSmallestSum(vector<int>& A, vector<int>& B, int k) {
        // Write your code here
        int n = A.size(), m = B.size();
        if(n == 0 && m == 0){
            return -1;
        }
        priority_queue<Element> pq;
        for(int i = 0; i < n; i++){
            pq.push(Element(i, 0, A[i] + B[0]));
        }
        if(k < 0 || k > m * n){
            return -1;
        }
                Element now(0,0,-1);
        int count = 0;
            pq.pop();
            int nrow = now.row;
            int ncol = now.col + 1;
            if(ncol < m){
                pq.push(Element(nrow, ncol, A[nrow] + B[ncol]));
            }
        }
        return now.val;

            }
};
```

249. 统计前面比自己小的数的个数

http://www.lintcode.com/problem/count-of-smaller-number-before-itself

给定一个整数数组（下标由 0 到 n-1， n 表示数组的规模，取值范围由 0 到10000）。对于数组中的每个 ai 元素，请计算 ai 前的数中比它小的元素的数量。

说明

在做此题前，最好先完成以下三道题： 线段树的构造， 线段树的查询 II，和 比给定数小的项目数 I 。

Source code:

```
class STNode{
public:
    int start, end;
    int count;
    STNode * left, * right;
    STNode(int start, int end){
        this->start = start;
        this->end = end;
        this->count = 0;
        this->left = NULL;
        this->right = NULL;
    }
};
class SegmentTree{
public:
    STNode * _root;
    SegmentTree(int start, int end){
        _root = build(start, end);
    }
    STNode * build(int start, int end){
        if(start > end){
            return NULL;
        }
        STNode * root = new STNode(start, end);
        if(start < end){
            int mid = start + (end - start) / 2;
            root->left = build(start, mid);
            root->right = build(mid+1, end);
        }
        return root;
    }
    int query(STNode* root, int start, int end){
        if(root == NULL || start > end){
            return 0;
        }
        if(start == root->start && end == root->end){
            return root->count;
        }
        int mid = root->start + (root->end - root->start) / 2;
                int leftCount = 0;
                leftCount = query(root->left, start, mid);
            }
            else{
                leftCount = query(root->left, start, end);
            }
```

```
        }
        int rightCount = 0;
        if(mid < end){
            if(start <= mid){
                rightCount = query(root->right, mid+1, end);
            }
            else{
                rightCount = query(root->right, start, end);
            }
        }
        return leftCount + rightCount;
    }
    void update(STNode* root, int index, int value){
        if(root == NULL){
            return;
        }
        if(root->start == index && index == root->end){
            root->count += value;
            return;
        }
        int mid = root->start + (root->end - root->start) / 2;
        if(index <= mid){
            update(root->left, index, value);
        }
        else{
            update(root->right, index, value);
        }
        root->count = root->left->count + root->right->count;
    }
};
class Solution {
public:
   /**
     * @param A: An integer array
     * @return: Count the number of element before this element 'ai' is
     *          smaller than it and return count number array
     */
    vector<int> countOfSmallerNumberII(vector<int> &A) {
        // write your code here
        SegmentTree *stree = new SegmentTree(0, 10001);
        vector<int> result;
        for(int i = 0; i < A.size(); i++){
            int count = stree->query(stree->_root, 0, A[i]-1);
            result.push_back(count);
            stree->update(stree->_root, A[i], 1);
        }
        delete stree;
        return result;
    }
};
```

207. 区间求和 II

http://www.lintcode.com/problem/interval-sum-ii

在类的构造函数中给一个整数数组, 实现两个方法 query(start, end)

和 modify(index, value):

对于 query(start, end), 返回数组中下标 start 到 end 的 和。 对于 modify(index, value), 修改数组中下标为 index 上的数为 value. 注意事项 在做此题前，建议先完成以下三题： 线段树的构造 线段树的查询 线段树的修改

样例

给定数组 A = [1,2,7,8,5]. query(0, 2), 返回 10. modify(0, 4), 将 A[0] 修改为 4. query(0, 1), 返回 6. modify(2, 1), 将 A[2] 修改为 1. query(2, 4), 返回 14.

Source code:

```
class Solution {
public:
    /* you may need to use some attributes here */
        /**
     * @param A: An integer vector
     */
    Solution(vector<int> A) {
        // write your code here
    }
        /**
     * @param start, end: Indices
     * @return: The sum from start to end
     */
    long long query(int start, int end) {
        // write your code here
    }
        /**
     * @param index, value: modify A[index] to value.
     */
    void modify(int index, int value) {
        // write your code here
    }
};
```

131. 大楼轮廓

http://www.lintcode.com/problem/building-outline

水平面上有 N 座大楼，每座大楼都是矩阵的形状，可以用三个数字表示 (start, end, height)，分别代表其在x轴上的起点，终点和高度。大楼之间从远处看可能会重叠，求出 N 座大楼的外轮廓线。

外轮廓线的表示方法为若干三元组，每个三元组包含三个数字 (start, end, height)，代表这段轮廓的起始位置，终止位置和高度。 注意事项 请注意合并同样高度的相邻轮廓，不同的轮廓线在x轴上不能有重叠。

样例

给出三座大楼： [[1, 3, 3], [2, 4, 4], [5, 6, 1]] 外轮廓线为： [[1, 2, 3], [2, 4, 4], [5, 6, 1]]

Source code:

```
// Time:  O(nlogn)
// Space: O(n)
// BST solution.
class Solution {
public:
    enum {start, end, height};
    struct Endpoint {
        int height;
        bool isStart;
    };
    /**
     * @param buildings: A list of lists of integers
     * @return: Find the outline of those buildings
     */
    vector<vector<int>> buildingOutline(vector<vector<int>> &buildings) {
        map<int, vector<Endpoint>> point_to_height;  // Ordered, no
duplicates.
        for (const auto& building : buildings) {
            point_to_height[building[start]].emplace_back
                (Endpoint{building[height], true});
            point_to_height[building[end]].emplace_back
                (Endpoint{building[height], false});
        }
        vector<vector<int>> res;
        map<int, int> height_to_count;  // BST.
        int curr_start = -1;
        int curr_max = 0;
        // Enumerate each point in increasing order.
        for (const auto& kvp : point_to_height) {
            const auto& point = kvp.first;
            const auto& heights = kvp.second;
            for (const auto& h : heights) {
                if (h.isStart) {
```

```
                ++height_to_count[h.height];
            } else {
                --height_to_count[h.height];
                if (height_to_count[h.height] == 0) {
                    height_to_count.erase(h.height);
                }
            }
        }
        if (height_to_count.empty() ||
            curr_max != height_to_count.crbegin()->first) {
            if (curr_max > 0) {
                res.emplace_back(move(vector<int>{curr_start, point,
                    curr_max}));
            }
            curr_start = point;
            curr_max = height_to_count.empty() ?
                       0 : height_to_count.crbegin()->first;
        }
    }
    return res;
  }
};
// Time:  O(nlogn)
// Space: O(n)
// Divide and conquer solution.
class Solution2 {
public:
    enum {start, end, height};
    /**
     * @param buildings: A list of lists of integers
     * @return: Find the outline of those buildings
     */
    vector<vector<int>> buildingOutline(vector<vector<int>> &buildings) {
        return ComputeSkylineInInterval(buildings, 0, buildings.size());
    }
    // Divide and Conquer.
    vector<vector<int>> ComputeSkylineInInterval(const vector<vector<int>>&
        buildings,
                                                 int left_endpoint, int
                                            right_endpoint) {
        if (right_endpoint - left_endpoint <= 1) {  // 0 or 1 skyline, just
copy
            it.
            return {buildings.cbegin() + left_endpoint,
                    buildings.cbegin() + right_endpoint};
        }
        int mid = left_endpoint + ((right_endpoint - left_endpoint) / 2);
        auto left_skyline = ComputeSkylineInInterval(buildings,
left_endpoint,
            mid);
        auto right_skyline = ComputeSkylineInInterval(buildings, mid,
```

```
            right_endpoint);
        return MergeSkylines(left_skyline, right_skyline);
    }
    // Merge Sort
    vector<vector<int>> MergeSkylines(vector<vector<int>>& left_skyline,
vector
        <vector<int>>& right_skyline) {
        int i = 0, j = 0;
        vector<vector<int>> merged;
        while (i < left_skyline.size() && j < right_skyline.size()) {
            if (left_skyline[i][end] < right_skyline[j][start]) {
                merged.emplace_back(move(left_skyline[i++]));
            } else if (right_skyline[j][end] < left_skyline[i][start]) {
                merged.emplace_back(move(right_skyline[j++]));
            } else if (left_skyline[i][start] <= right_skyline[j][start]) {
                MergeIntersectSkylines(merged, left_skyline[i], i,
                                       right_skyline[j], j);
            } else {  // left_skyline[i][start] > right_skyline[j][start].
                MergeIntersectSkylines(merged, right_skyline[j], j,
                                       left_skyline[i], i);
            }
        }
        // Insert the remaining skylines.
        merged.insert(merged.end(), left_skyline.begin() + i,
left_skyline.end
            ());
        merged.insert(merged.end(), right_skyline.begin() + j,
right_skyline.end
            ());
        return merged;
    }
    // a[start] <= b[start]
    void MergeIntersectSkylines(vector<vector<int>>& merged, vector<int>& a,
        int& a_idx,
                                vector<int>& b, int& b_idx) {
        if (a[end] <= b[end]) {
            if (a[height] > b[height]) {  // |aaa|
                if (b[end] != a[end]) {   // |abb|b
                    b[start] = a[end];
                    merged.emplace_back(move(a)), ++a_idx;
                } else {        // aaa
                    ++b_idx;    // abb
                }
            } else if (a[height] == b[height]) {  // abb
                b[start] = a[start], ++a_idx;    // abb
            } else {  // a[height] < b[height].
                if (a[start] != b[start]) {
                                //    bb
                    merged.emplace_back(move(vector<int>{a[start], b[start],
                        a[height]}));  // |a|bb
                }
```

```
                ++a_idx;
            }
        } else {  // a[end] > b[end].
            if (a[height] >= b[height]) {  // aaaa
                ++b_idx;                   // abba
            } else {
                //    |bb|
                // |a||bb|a
                if (a[start] != b[start]) {
                    merged.emplace_back(move(vector<int>{a[start], b[start],
                        a[height]}));
                }
                a[start] = b[end];
                merged.emplace_back(move(b)), ++b_idx;
            }
        }
    }
};
```

3 - 高级数据结构（下）

475. 二叉树的最大路径和 II

http://www.lintcode.com/problem/binary-tree-maximum-path-sum-ii

给一棵二叉树，找出从根节点出发的路径中，和最大的一条。

这条路径可以在任何二叉树中的节点结束，但是必须包含至少一个点（也就是根了）。

样例

给出如下的二叉树： 1 /\ 2 3 返回4。(最大的路径为1→3)

Source code:

```
/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root the root of binary tree.
     * @return an integer
     */
    int maxPathSum2(TreeNode *root) {
        // Write your code here
        if(root == NULL){
            return 0;
        }
        int leftSP = maxPathSum2(root->left);
        int rightSP = maxPathSum2(root->right);
        return max(max(leftSP, rightSP), 0) + root->val;
    }
};
```

40. 用栈实现队列

http://www.lintcode.com/problem/implement-queue-by-two-stacks

正如标题所述，你需要使用两个栈来实现队列的一些操作。

队列应支持push(element)，pop() 和 top()，其中pop是弹出队列中的第一个(最前面的)元素。pop和top方法都应该返回第一个元素的值。

样例

比如push(1), pop(), push(2), push(3), top(), pop()，你应该返回1，2和2

Source code:

```
class Queue {
public:
    stack<int> stack1;
    stack<int> stack2;
    Queue() {
        // do intialization if necessary
    }
    void push(int element) {
        // write your code here
        stack1.push(element);
    }
        int pop() {
        // write your code here
        if(stack2.empty())
        {
            while(!stack1.empty())
            {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        int elem = stack2.top();
        stack2.pop();
        return elem;
    }
    int top() {
        // write your code here
        if(stack2.empty())
        {
            while(!stack1.empty())
            {
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        return stack2.top();
};
```

12. 带最小值操作的栈

http://www.lintcode.com/problem/min-stack

实现一个带有取最小值min方法的栈，min方法将返回当前栈中的最小值。

你实现的栈将支持push，pop 和 min 操作，所有操作要求都在O(1)时间内完成。 注意事项 如果堆栈中没有数字则不能进行min方法的调用

样例

如下操作：push(1)，pop()，push(2)，push(3)，min()， push(1)，min() 返回 1，2，1

Source code:

```
class MinStack {
private:
        stack<int> normalStack;
        stack<int> minStack;
public:
    MinStack() {
        // do initialization if necessary
    }
    void push(int number) {
        // write your code here
        normalStack.push(number);
                int currentMin;
        if(minStack.empty())
        {
            currentMin = number;
        }
        else
        {
            currentMin = minStack.top();
            if(number < currentMin)
            {
                currentMin = number;
            }
        }
        minStack.push(currentMin);
    }
    int pop() {
        // write your code here
        int x = normalStack.top();
        normalStack.pop();
        minStack.pop();
        return x;
    }
    int min() {
        // write your code here
        return minStack.top();
    }
};
```

510. 最大矩形

http://www.lintcode.com/problem/maximal-rectangle

给你一个二维矩阵，权值为False和True，找到一个最大的矩形，使得里面的值全部为True，输出它的面积

样例

给你一个矩阵如下 [[1, 1, 0, 0, 1], [0, 1, 0, 0, 1], [0, 0, 1, 1, 1], [0, 0, 1, 1, 1], [0, 0, 0, 0, 1]] 输出6

Source code:

```
class Solution {
public:
    /**
     * @param matrix a boolean 2D matrix
     * @return an integer
     */
    int maximalRectangle(vector<vector<bool> > &matrix) {
        // Write your code here
        int n = matrix.size();
        if(n == 0){
            return 0;
        }
        int m = matrix[0].size();
                // 预处理为直方图和
        vector<vector<int>> sum(n, vector<int>(m,0));
        for(int j = 0; j < m; j++){
            sum[0][j] = matrix[0][j] ? 1 : 0;
        }
        for(int i = 1; i < n; i++){
            for(int j = 0; j < m; j++){
                if(matrix[i][j]){
                    sum[i][j] = 1 + sum[i-1][j];
                }
            }
        }
                // use stack to get max area
        int maxArea = -1;
        for(int i = 0; i < n; i++){
            stack<int> s;
            for(int j = 0; j <= m; j++){
                int now = (j == m) ? -1 : sum[i][j];
                while(!s.empty() && now < sum[i][s.top()]){
                    int h = sum[i][s.top()];
                    s.pop();
                    int w = s.empty() ? j : (j - s.top() - 1);
                    maxArea = max(maxArea, h * w);
                }
                s.push(j);
            }
    }
};
```

367. 表达树构造

http://www.lintcode.com/problem/expression-tree-build

表达树是一个二叉树的结构，用于衡量特定的表达。所有表达树的

叶子都有一个数字字符串值。而所有表达树的非叶子都有另一个操作字符串值。

给定一个表达数组，请构造该表达的表达树，并返回该表达树的根。

说明

什么是表达树？详见wiki百科： 表达树

Source code:

```
/**
 * Definition of ExpressionTreeNode:
 * class ExpressionTreeNode {
 * public:
 *     string symbol;
 *     ExpressionTreeNode *left, *right;
 *     ExpressionTreeNode(string symbol) {
 *         this->symbol = symbol;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
//////////////////////////////////////////
// V1 : Build expression tree from RPN
class Solution {
private:
    bool isOp(string str)
    {
        if((str.compare("+") == 0) || (str.compare("-") == 0) ||
           (str.compare("*") == 0) || (str.compare("/") == 0))
        {
            return true;
        }
        return false;
    }
        int compareOrder(string str1, string str2)
    {
        int order1 =  ((str1.compare("*") == 0) || (str1.compare("/") == 0))
? 1
            : 0;
        int order2 =  ((str2.compare("*") == 0) || (str2.compare("/") == 0))
? 1
            : 0;
                return order1 - order2;
    }
    // 将中缀表达式转换为后缀表达式（逆波兰式）
    vector<string> convertToRPN(vector<string> &expression){
        vector<string> rpnList;
```

```
    stack<string> opStack;
    for(int i = 0; i < expression.size(); i++){
        string now = expression[i];
        if(now.compare("(") == 0){
            opStack.push(now);
        }
        else if(now.compare(")") == 0){
            while(!opStack.empty() &&
                    opStack.top().compare("(") != 0){
                rpnList.push_back(opStack.top());
                opStack.pop();
            }
            if(!opStack.empty()) {
                opStack.pop();
            }
        }
        else if(isOp(now)){
            while(!opStack.empty() &&
                    isOp(opStack.top()) &&
                    compareOrder(now, opStack.top()) <= 0){
                rpnList.push_back(opStack.top());
                opStack.pop();
            }
            opStack.push(now);
        }
        else {
            rpnList.push_back(now);
        }
    }
    while(!opStack.empty()){
        rpnList.push_back(opStack.top());
        opStack.pop();
    }
    return rpnList;
}


        ExpressionTreeNode* buildFromRPN(vector<string> &expression){
    if(expression.size() == 0){
        return NULL;
    }
    stack<ExpressionTreeNode*> s;
    for(int i = 0; i < expression.size(); i++){
        string now = expression[i];
        if(isOp(now)){
            ExpressionTreeNode * right = s.top();
            s.pop();
            ExpressionTreeNode * left = s.top();
            s.pop();
            ExpressionTreeNode * node = new ExpressionTreeNode(now);
            node->left = left;
```

```
                node->right = right;
                s.push(node);
            }
            else{
                s.push(new ExpressionTreeNode(now));
            }
        }
        return s.top();
    }
public:
    /**
     * @param expression: A string array
     * @return: The root of expression tree
     */
    ExpressionTreeNode* build(vector<string> &expression) {
        // write your code here
        if(expression.size() == 0) {
            return NULL;
        }
        vector<string> rpe = convertToRPN(expression);
        return buildFromRPN(rpe);
    }
};
////////////////////////////////////////////////////////////////////////////
////
    //
/* V2 : Build directly expression tree from expression
 *    1). Leaf node is number
 *    2). Non-leaf is operator
 *    3). top level node has lower priority
 *    4). The subtree of each node is a complete expression
 */
class TNode{
public:
    int priority;
    ExpressionTreeNode* root;
    TNode(int p, string ss){
        this->priority = p;
        this->root = new ExpressionTreeNode(ss);

            }
};
class Solution2 {
private:
    int getPriority(string a, int base){
        if(a.length() == 0){
            // empty string, terminater
            return numeric_limits<int>::min();
        }
        else if((a.compare("+") == 0) || (a.compare("-") == 0)){
            return base + 1;
```

```
        }
        else if((a.compare("*") == 0) || (a.compare("/") == 0)){
            return base + 2;
        }
        return numeric_limits<int>::max();
    }
public:
    /**
     * @param expression: A string array
     * @return: The root of expression tree
     */
    ExpressionTreeNode* build(vector<string> &expression) {
        if(expression.size() == 0){
            return NULL;
        }

                // stack keeps tree nodes with increasing priority sequence.
        stack<TNode*> s;
        int priority = 0;
        int base = 0;
        for(int i = 0; i <= expression.size(); i++){
            if(i != expression.size()){
                if(expression[i].compare("(") == 0){
                    base += 10;
                    continue;
                }
                else if(expression[i].compare(")") == 0){
                    base -= 10;
                    continue;
                }
                priority = getPriority(expression[i], base);
            }
            else{
                priority = numeric_limits<int>::min();
            }
            TNode * rightNode = NULL;
            if(i == expression.size()) {
                rightNode = new TNode(numeric_limits<int>::min(), "");
            }
            else{
                rightNode = new TNode(priority, expression[i]);
            }

                        //top level node has low priority
            while(!s.empty() && rightNode->priority <= s.top()->priority){
                TNode * nowNode = s.top();
                s.pop();
                if(s.empty()){
                    rightNode->root->left = nowNode->root;
                }
                else{
```

```
                    TNode * leftNode = s.top();
                    if(leftNode->priority < rightNode->priority){
                        rightNode->root->left = nowNode->root;
                    }
                    else{
                        leftNode->root->right = nowNode->root;
                    }
                }
            }

                    s.push(rightNode);
        }
        return (s.top())->root->left;
    }
};
```

370. 将表达式转换为逆波兰表达式

http://www.lintcode.com/problem/convert-expression-to-reverse-polish-notation

给定一个表达式字符串数组，返回该表达式的逆波兰表达式（即去掉括号）。

样例

对于 [3 - 4 + 5]的表达式（该表达式可表示为["3", "-", "4", "+", "5"]），返回 [3 4 - 5 +]（该表达式可表示为 ["3", "4", "-", "5", "+"]）。

Source code:

```
class Solution {
private:
    bool isOp(string str)
    {
        if((str.compare("+") == 0) || (str.compare("-") == 0) ||
           (str.compare("*") == 0) || (str.compare("/") == 0))
        {
            return true;
        }
        return false;
    }
        int compareOrder(string str1, string str2)
    {
        int order1 =  ((str1.compare("*") == 0) || (str1.compare("/") == 0))
? 1
            : 0;
        int order2 =  ((str2.compare("*") == 0) || (str2.compare("/") == 0))
```

```
? 1
            : 0;
                return order1 - order2;
    }
public:
    /**
     * @param expression: A string array
     * @return: The Reverse Polish notation of this expression
     */
    vector<string> convertToRPN(vector<string> &expression) {
        // write your code here
        vector<string> rpnList;
        stack<string> opStack;
        for(int i = 0; i < expression.size(); i++)
        {
            string str = expression[i];
                        if(str.compare("(") == 0)
            {
                opStack.push(str);
            }
            else if(str.compare(")") == 0)
            {
                while(!opStack.empty() && opStack.top().compare("(") != 0)
                {
                    rpnList.push_back(opStack.top());
                    opStack.pop();
                }
                if(!opStack.empty())
                {
                    opStack.pop();
                }
            }
            else if(isOp(str))
            {
                while(!opStack.empty() && isOp(opStack.top()) &&
compareOrder
                    (str, opStack.top()) <= 0)
                {
                    rpnList.push_back(opStack.top());
                    opStack.pop();
                }
                opStack.push(str);
            }
            else
            {
                rpnList.push_back(str);
            }

                    }
        while(!opStack.empty())
        {
```

```
            rpnList.push_back(opStack.top());
            opStack.pop();
        }

                return rpnList;
    }
};
```

369. 将表达式转换为波兰表达式

http://www.lintcode.com/problem/convert-expression-to-polish-notation

给定一个表达式字符串数组，返回该表达式的波兰表达式。（即去掉括号）

说明

波兰表达 的定义: http://en.wikipedia.org/wiki/Polish_notation
http://baike.baidu.com/view/7857952.htm

Source code:

```
class Solution {
public:
    /**
     * @param expression: A string array
     * @return: The Polish notation of this expression
     */
    vector<string> convertToPN(vector<string> &expression) {
        // write your code here
    }
};
```

368. 表达式求值

http://www.lintcode.com/problem/expression-evaluation

给一个用字符串表示的表达式数组，求出这个表达式的值。

注意事项 表达式只包含整数, +, -, *, /, (,).

样例

对于表达式 *(26-(23+7)/(1+2)), 对应的数组为：* ["2", "", "6", "-", "(", "23", "+", "7", ")", "/", (", "1", "+", "2", ")"], 其值为 2

Source code:

```
class Solution {
public:
    bool isOp(string op){
        if(op.compare("+") == 0 ||
           op.compare("-") == 0 ||
           op.compare("*") == 0 ||
           op.compare("/") == 0 ){
            return true;
        }
                return false;
    }
        // greater: 1
    // equal: 0
    // less: -1
    int compareOrder(string op1, string op2) {
        int prio1 = 0, prio2 = 0;
        if(op1.compare("*") == 0 ||
           op1.compare("/") == 0){
            prio1 = 1;
        }
        if(op2.compare("*") == 0 ||
           op2.compare("/") == 0){
            prio2 = 1;
        }
        return prio1 - prio2;
    }
    vector<string> convertToRPN(vector<string> &expression) {
        vector<string> rpnList;
        stack<string> opStack;
        for(int i = 0; i < expression.size(); i++){
            string now = expression[i];
            if(now.compare("(") == 0){
                opStack.push(now);
            }
            else if(now.compare(")") == 0){
                while(opStack.top().compare("(") != 0){
                    rpnList.push_back(opStack.top());
                    opStack.pop();
                }
            else if(isOp(now)){
                if(!opStack.empty() &&
                    isOp(opStack.top()) &&
                    compareOrder(opStack.top(), now) >= 0){
                    // pop out those ops with higher or equal order
                    rpnList.push_back(opStack.top());
                    opStack.pop();
                }
                opStack.push(now);
```

```
            }
            else {
                rpnList.push_back(now);
            }
        }
        while(!opStack.empty()) {
            rpnList.push_back(opStack.top());
            opStack.pop();
        }
        return rpnList;
    }
    int calculate(int a, int b, string op){
        if(op.compare("+") == 0){
            return a+b;
        }
        else if(op.compare("-") == 0){
            return a-b;
        }
        else if(op.compare("*") == 0){
            return a*b;
        }
        else if(op.compare("/") == 0){
            return a/b;
        }
        return -1; // should never happen
    }
    int calculateRPN(vector<string> &expression) {
        stack<int> s;
        int number = 0;
        for(int i = 0; i < expression.size(); i++){
            string now = expression[i];
            if(isOp(now)){
                int b = s.top();
                s.pop();
                int a = s.top();
                s.pop();
                number = calculate(a, b, now);
                s.push(number );
            }
            else {
                number = stoi(now);
                s.push(number);
            }
        }
        return s.top();
    }
    /**
     * @param expression: a vector of strings;
     * @return: an integer
     */
    int evaluateExpression(vector<string> &expression) {
```

```
        // write your code here
        if(expression.size() == 0){
            return 0;
        }
        // 1. converted to reversed polish expression
        vector<string> rpe = convertToRPN(expression);

                // 2. caculate the reversed polish expression
        return calculateRPN(rpe);
    }
};
```

126. 最大树

http://www.lintcode.com/problem/max-tree

给出一个没有重复的整数数组，在此数组上建立最大树的定义如

下：

根是数组中最大的数 左子树和右子树元素分别是被父节点元素切分开的子数组中的最大值 利用给定的数组构造最大树。

样例

给出数组 [2, 5, 6, 0, 3, 1]，构造的最大树如下： 6 / \ 5 3 / / \ 2 0 1

Source code:

```
/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param A: Given an integer array with no duplicates.
     * @return: The root of max tree.
     */
    TreeNode* maxTree1(vector<int> A) {
        // write your code here
```

```
        if(A.size() == 0){
            return NULL;
        }
        // 维持递减序列的栈
        // 对于每一个元素 v
        // 1. 父亲节点是谁?
        // 2. 找左边第一个比它大的元素x, 和右边第一个比它小的元素y
        // if(x < y), v 是x的右节点
        // else if , v 是 y 的左节点
        stack<TreeNode*> s;
        for(int i = 0; i <= A.size(); i++){
            int value = (i == A.size()) ? numeric_limits<int>::max() : A[i];
            TreeNode * rightNode = new TreeNode(value);
            while(!s.empty() && value > s.top()->val){
                TreeNode * nowNode = s.top();
                s.pop();
                if(s.empty()){
                    rightNode->left = nowNode;
                }
                else {
                    TreeNode *leftNode = s.top();
                    }
                    else {
                        leftNode->right = nowNode;
                    }
                }
            }
            s.push(rightNode);
        }
        return (s.top())->left;
    }

        // V2 same way, just rewrite
    TreeNode* maxTree(vector<int> A) {
        stack<TreeNode*> s;
        for(int i = 0; i <= A.size(); i++){
            int rightValue = (i == A.size()) ? numeric_limits<int>::max() :
A[i]
                ;
            TreeNode * rightNode = new TreeNode(rightValue);
            while(!s.empty() && rightValue > s.top()->val){
                TreeNode* nowNode = s.top();
                s.pop();
                if(s.empty() || rightValue < s.top()->val){
                    rightNode->left = nowNode;
                }
                else {
                    TreeNode* leftNode = s.top();
                    leftNode->right = nowNode;
                }
            }
```

```
            s.push(rightNode);
        }
    return s.top()->left;
}
};
```

122. 直方图最大矩形覆盖

http://www.lintcode.com/problem/largest-rectangle-in-histogram

给出的n个非负整数表示每个直方图的高度，每个直方图的宽均为

1，在直方图中找到最大的矩形面积。

以上直方图宽为1，高度为[2,1,5,6,2,3]。

最大矩形面积如图阴影部分所示，含有10单位

样例

给出 height = [2,1,5,6,2,3]，返回 10

Source code:

```
class Solution {
public:
    /**
     * @param height: A list of integer
     * @return: The area of largest rectangle in the histogram
     */
    int largestRectangleArea(vector<int> &height) {
        // write your code here
        int maxArea = 0;
        stack<int> s;
        for(int i = 0; i <= height.size(); i++){
            int cur = (i == height.size()) ? -1 : height[i];
            while(!s.empty() && cur < height[s.top()]){
                int h = height[s.top()];
                s.pop();
                int w = (s.empty())? i : i - s.top() -1;
                                maxArea = max(maxArea, w * h);
            }
                        s.push(i);
        }
        return maxArea;
    }
};
```

4 - 两个指针

415. 有效回文串

http://www.lintcode.com/problem/valid-palindrome

给定一个字符串，判断其是否为一个回文串。只包含字母和数字，忽略大小写。

注意事项 你是否考虑过，字符串有可能是空字符串？这是面试过程中，面试官常常会问的问题。 在这个题目中，我们将空字符串判定为有效回文。

样例

"A man, a plan, a canal: Panama" 是一个回文。 "race a car" 不是一个回文。

Source code:

```
class Solution {
private:
    bool isValid(char c){
        if((c >= 'a' && c <= 'z') ||
           (c >= 'A' && c <= 'Z') ||
           (c >= '0' && c <= '9')){
            return true;
        }
        return false;
    }
        bool equal(char p, char q){
        if(p >= 'A' && p <= 'Z'){
            p = 'a' + p - 'A';
        }
        if(q >= 'A' && q <= 'Z'){
            q = 'a' + q - 'A';
        }
        return p == q;
    }
   public:
    /**
     * @param s A string
     * @return Whether the string is a valid palindrome
     */
    bool isPalindrome(string& s) {
        // Write your code here
        if(s.length() == 0){
            return true;
        }
        int left = 0, right = s.length()-1;
        while(left < right){
            while(left < right && !isValid(s.at(left))){
                left++;
            }
            while(left < right && !isValid(s.at(right))){
                right--;
            }
            if(!equal(s.at(left), s.at(right))){
                return false;
            right--;
        }
        return true;
    }
};
```

373. 奇偶分割数组

http://www.lintcode.com/problem/partition-array-by-odd-and-even

分割一个整数数组，使得奇数在前偶数在后。

样例

给定 [1, 2, 3, 4]，返回 [1, 3, 2, 4]。

Source code:

```
class Solution {
public:
    /**
     * @param nums: a vector of integers
     * @return: nothing
     */
    void partitionArray(vector<int> &nums) {
        // write your code here
        int left = 0;
        int right = nums.size()-1;
        while(left < right)
        {
            while(left < right && (nums[left] % 2) == 1)
            {
                left++;
            }
            while(left < right && (nums[right] % 2) == 0)
            {
                right--;
            }
                        //swap
            int tmp = nums[left];
            nums[left] = nums[right];
            nums[right] = tmp;
        }
    }
};
```

174. 删除链表中倒数第n个节点

http://www.lintcode.com/problem/remove-nth-node-from-end-of-list

给定一个链表，删除链表中倒数第n个节点，返回链表的头节点。

注意事项 链表中的节点个数大于等于n

样例

给出链表1->2->3->4->5->null和 n = 2. 删除倒数第二个节点之后，这个链表将变成1->2->3->5->null.

Source code:

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @param n: An integer.
     * @return: The head of linked list.
     */
    ListNode *removeNthFromEnd(ListNode *head, int n) {
        // write your code here
        ListNode dummy;
        dummy.next = head;
        ListNode * node = head;
        ListNode * prev = &dummy;
                for(int i = 1; i < n; i++)
        {
            if(head == NULL)
            {
                return NULL;
            }
            head = head->next;
        }
                while(head->next != NULL)
        {
            head = head->next;
            prev = node;
            node = node->next;
        }

                return dummy.next;
    }
};
```

165. 合并两个排序链表

http://www.lintcode.com/problem/merge-two-sorted-lists

将两个排序链表合并为一个新的排序链表

样例

给出 1->3->8->11->15->null，2->null， 返回 1->2->3->8->11->15->null。

Source code:

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param ListNode l1 is the head of the linked list
     * @param ListNode l2 is the head of the linked list
     * @return: ListNode head of linked list
     */
    ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
        // write your code here
    }
};
```

387. 最小差

http://www.lintcode.com/problem/the-smallest-difference

给定两个整数数组（第一个是数组 A，第二个是数组 B），在数组 A 中取 A[i]，数组 B 中取 B[j]，A[i] 和 B[j]两者的差越小越好(|A[i] - B[j]|)。返回最小差。

样例

给定数组 A = [3,4,6,7]， B = [2,3,8,9]，返回 0。

Source code:

```
class Solution {
public:
    /**
     * @param A, B: Two integer arrays.
     * @return: Their smallest difference.
     */
    int smallestDifference(vector<int> &A, vector<int> &B) {
        // write your code here
        sort(A.begin(), A.end());
        sort(B.begin(), B.end());
                int minDiff = numeric_limits<int>::max();
        int i = 0, j = 0;
        while(i < A.size() && j < B.size()){
            minDiff = min(minDiff, abs(A[i]-B[j]));
            if(A[i] > B[j]){
                j++;
            }
            else{
                i++;
            }
        }
        return minDiff;
    }
};
```

363. 接雨水

http://www.lintcode.com/problem/trapping-rain-water

给出 n 个非负整数，代表一张X轴上每个区域宽度为 1 的海拔图, 计算这个海拔图最多能接住多少（面积）雨水。

样例

如上图所示，海拔分别为 [0,1,0,2,1,0,1,3,2,1,2,1], 返回 6.

Source code:

```
class Solution {
public:
    /**
     * @param heights: a vector of integers
     * @return: a integer
     */
    int trapRainWater(vector<int> &heights) {
        // write your code here
        if(heights.size() == 0){
            return 0;
        }
        int ans = 0;
        int left = 0, right = heights.size()-1;
        int leftHeight = heights[left];
        int rightHeight = heights[right];
        while(left < right){
            if(heights[left] < heights[right]){
                left++;
                if(heights[left] < leftHeight){
                    ans += leftHeight - heights[left];
                }
                else{
                    leftHeight = heights[left];
                }
            }
            else{
                right--;
                if(heights[right] < rightHeight){
                    ans += rightHeight - heights[right];
                }
                else{
                    rightHeight = heights[right];
                }
            }
        }
        return ans;
    }
};
```

148. 颜色分类

http://www.lintcode.com/problem/sort-colors

给定一个包含红，白，蓝且长度为 n 的数组，将数组元素进行分类使相同颜色的元素相邻，并按照红、白、蓝的顺序进行排序。

我们可以使用整数 0，1 和 2 分别代表红，白，蓝。 注意事项 不能使用代码库中的排序函数

来解决这个问题。 排序需要在原数组中进行。

样例

给你数组 [1, 0, 1, 2], 需要将该数组原地排序为 [0, 1, 1, 2]。

Source code:

```
class Solution{
private:
    inline void swap(vector<int> &A, int x, int y)
    {
        int tmp = A[x];
        A[x] = A[y];
        A[y] = tmp;
    }
public:
    /**
     * @param nums: A list of integer which is 0, 1 or 2
     * @return: nothing
     */
    void sortColors(vector<int> &nums) {
        // write your code here
        int left = -1, index = 0, right = nums.size();
        while(index < right){
            if(nums[index] == 0){
                left++;
                swap(nums, left, index);
                index++;
            }
            else if(nums[index] == 2){
                swap(nums, index, --right);
            }
            else {
                index++;
            }
        }
    }
};
```

144. 交错正负数

http://www.lintcode.com/problem/interleaving-positive-and-negative-numbers

给出一个含有正整数和负整数的数组，重新排列成一个正负数交错的数组。

注意事项 不需要保持正整数或者负整数原来的顺序。

样例

给出数组[-1, -2, -3, 4, 5, 6]，重新排序之后，变成[-1, 5, -2, 4, -3, 6]或者其他任何满足要求的答案

Source code:

```
class Solution {
private:
     void swap(vector<int> &A, int x, int y)
     {
         int tmp = A[x];
         A[x] = A[y];
         A[y] = tmp;
     }
public:
    /**
     * @param A: An integer array.
     * @return: void
     */
    void rerange(vector<int> &A) {
        // write your code here
        int n = A.size();
        if(n < 2)
        {
            return;
        }
                int posIndex, negIndex;
        int posCount = 0, negCount = 0;
        for(int i = 0; i < n; i++)
        {
            if(A[i] > 0)
            {
                posCount++;
            }
            else
            {
                negCount++;
            }
        }
                if(posCount > negCount)
        {
            posIndex = 0;
            negIndex = 1;
        }
        else
            negIndex = 0;
        }

                while(posIndex < n && negIndex < n)
```

```
        {
            if(A[n-1] > 0)
            {
                swap(A, posIndex, n-1);
                posIndex += 2;
            }
            else
            {
                swap(A, negIndex, n-1);
                negIndex += 2;
            }
        }
        return;

            }
};
```

102. 带环链表

http://www.lintcode.com/problem/linked-list-cycle

给定一个链表，判断它是否有环。

样例

给出 -21->10->4->5, tail connects to node index 1，返回 true

Source code:

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: True if it has a cycle, or false
     */
    bool hasCycle(ListNode *head) {
        // write your code here
    }
};
```

59. 最接近的三数之和

http://www.lintcode.com/problem/3sum-closest

给一个包含 n 个整数的数组 S, 找到和与给定整数 target 最接近的三元组，返回这三个数的和。

注意事项 只需要返回三元组之和，无需返回三元组本身

样例

例如 S = [-1, 2, 1, -4] and target = 1. 和最接近 1 的三元组是 -1 + 2 + 1 = 2.

Source code:

```
class Solution {
public:
    /**
     * @param numbers: Give an array numbers of n integer
     * @param target: An integer
     * @return: return the sum of the three integers, the sum closest
target.
     */
    int threeSumClosest(vector<int> nums, int target) {
        // write your code here
        int n = nums.size();
        if(n < 3){
            return 0;
        }
                int closest = numeric_limits<int>::max() / 2;
        sort(nums.begin(), nums.end());
        for(int i = 0; i < n-2; i++){
                        int left = i+1;
            int right = n-1;
            while(left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if(sum == target) {
                    return sum;
                }
                else if(sum < target) {
                    left++;
                                        while(left < right && nums[left] ==
nums[left-1]) {
                        left++;
                    }
                }
                else {
                    right--;
                    while(left < right && nums[right] == nums[right+1]) {
                        right--;
                    }
                }
                                closest = abs(sum - target) < abs(closest -
target) ?
                            sum : closest;

                return closest;
    }
};
```

58. 四数之和

http://www.lintcode.com/problem/4sum

给一个包含n个数的整数数组S，在S中找到所有使得和为给定整数target的四元组(a, b, c, d)。

注意事项 四元组(a, b, c, d)中，需要满足a <= b <= c <= d 答案中不可以包含重复的四元组。

样例

例如，对于给定的整数数组S=[1, 0, -1, 0, -2, 2] 和 target=0. 满足要求的四元组集合为： (-1, 0, 0, 1) (-2, -1, 1, 2) (-2, 0, 0, 2)

Source code:

```
class Solution {
public:
    /**
     * @param numbers: Give an array numbersbers of n integer
     * @param target: you need to find four elements that's sum of target
     * @return: Find all unique quadruplets in the array which gives the sum
of
     *          zero.
     */
    vector<vector<int> > fourSum(vector<int> nums, int target) {
        // write your code here
        vector<vector<int> > result;
        int n = nums.size();
        if(n < 4)
        {
            return result;
        }
                sort(nums.begin(), nums.end());
        int sum = 0;
        for(int i = 0; i < (n - 3); i++)
        {
            if(i != 0 && nums[i] == nums[i-1])
            {
                continue;
            }
                        for(int j = (i+1); j < (n-2); j++)
            {
                if(j != (i+1) && nums[j] == nums[j-1])
                {
                    continue;
                }
                int left = j + 1;
                int right = n - 1;
                while(left < right)
                {
                    sum = nums[i] + nums[j] + nums[left] + nums[right];
                    if(sum == target)
```

```
                    {
                        vector<int> v = {nums[i], nums[j], nums[left],
                            nums[right]};
                        left++;
                        right--;
                        while(nums[left] == nums[left-1])
                        {
                            left++;
                        }
                        while(nums[right] == nums[right+1])
                        {
                            right--;
                        }
                    }
                    else if(sum < target)
                    {
                        left++;
                    }
                    else
                    {
                        right--;
                    }
                }
            }
        }

                return result;
    }
};
```

57. 三数之和

http://www.lintcode.com/problem/3sum

给出一个有n个整数的数组S，在S中找到三个整数a, b, c，找到所有使得a + b + c = 0的三元组。

注意事项 在三元组(a, b, c)，要求a <= b <= c。 结果不能包含重复的三元组。

样例

如S = {-1 0 1 2 -1 -4}, 你需要返回的三元组集合的是： (-1, 0, 1) (-1, -1, 2)

Source code:

```
class Solution {
public:
    /**
     * @param numbers : Give an array numbers of n integer
     * @return : Find all unique triplets in the array which gives the sum
of
         zero.
     */
    vector<vector<int> > threeSum(vector<int> &nums) {
        // write your code here
        vector<vector<int> > ans;
        if(nums.size() < 3){
            return ans;
        }
        int n = nums.size();
        sort(nums.begin(), nums.end());
        for(int i = 0; i < n-2; i++){
            if(i > 0 && nums[i] == nums[i-1]){
                continue;
            }
            int j = i+1;
            int k = n-1;
            while(j < k){
                int sum = nums[i] + nums[j] + nums[k];
                if(sum == 0){
                    ans.push_back({nums[i], nums[j], nums[k]});
                    j++;
                    k--;
                    while((j < k) && nums[j] == nums[j-1]) {
                        j++;
                    }
                    while((j < k) && nums[k] == nums[k+1]) {
                        k--;
                    }
                }
                else if (sum < 0){
                    j++;
                }
                else{
                    k--;
                }
            }
        return ans;
    }
};
```

49. 字符大小写排序

http://www.lintcode.com/problem/sort-letters-by-case

给定一个只包含字母的字符串，按照先小写字母后大写字母的顺序进行排序。

注意事项 小写字母或者大写字母他们之间不一定要保持在原始字符串中的相对位置。

样例

给出"abAcD"，一个可能的答案为"acbAD"

Source code:

```
class Solution {
private:
    inline void swap(string &A, int pos1, int pos2)
    {
        char temp = A.at(pos1);
        A.at(pos1) = A.at(pos2);
        A.at(pos2) = temp;
    }
public:
    /**
     * @param chars: The letters array you should sort.
     */
    void sortLetters(string &letters) {
        // write your code here
        int n = letters.length();
        int left = 0, right = n-1;
        while(left < right)
        {
            while(left <= right && letters.at(left) <= 'z' &&
letters.at(left) >=
                'a')
            {
                left++;
            }
            while(left <= right && letters.at(right) <= 'Z' &&
letters.at(right)
                >= 'A')
            {
                right--;
            }
            if(left < right)
            {
                swap(letters, left, right);
            }
            left++;
            right--;
        }
            }
};
```

31. 数组划分

http://www.lintcode.com/problem/partition-array

给出一个整数数组 nums 和一个整数 k。划分数组（即移动数组 nums 中的元素），使得：

所有小于k的元素移到左边 所有大于等于k的元素移到右边 返回数组划分的位置，即数组中第

一个位置 i，满足 nums[i] 大于等于 k。 注意事项 你应该真正的划分数组 nums，而不仅仅只是计算比 k 小的整数数，如果数组 nums 中的所有元素都比 k 小，则返回 nums.length。

样例

给出数组 nums = [3,2,2,1] 和 k = 2，返回 1.

Source code:

```
class Solution {
public:
    int partitionArray(vector<int> &nums, int k) {
        int i = 0, j = nums.size() - 1;
        while (i <= j) {
            while (i <= j && nums[i] < k) i++;
            while (i <= j && nums[j] >= k) j--;
            if (i < j) {
                int temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
                i++;
                j--;
            }
        }
        return i;
    }
};
```

558. Sliding Window Matrix Maximum

http://www.lintcode.com/problem/sliding-window-matrix-maximum

Given an array of n * m matrix, and a moving matrix window (size k * k), move the window from top left to botton right at each iteration, find the maximum number inside the window at each moving.

Return 0 if the answer does not exist.

样例

For matrix [[1, 5, 3], [3, 2, 1], [4, 1, 9],] The moving window size k = 2. return 13. At first the window is at the start of the array like this [[|1, 5|, 3], [|3, 2|, 1], [4, 1, 9],] ,get the sum 11; then the window move one step forward. [[1, |5, 3|], [3, |2, 1|], [4, 1, 9],] ,get the sum 11; then the window move one step forward again. [[1, 5, 3], [|3, 2|, 1], [|4, 1|, 9],] ,get the sum

10; then the window move one step forward again. [[1, 5, 3], [3, |2, 1|], [4, |1, 9|],] ,get the sum 13; SO finally, get the maximum from all the sum which is 13.

Source code:

```
class Solution {
public:
    /**
     * @param matrix an integer array of n * m matrix
     * @param k an integer
     * @return the maximum number
     */
    int maxSlidingWindow2(vector<vector<int>>& matrix, int k) {
        // Write your code here
    }
};
```

103. 带环链表 II

http://www.lintcode.com/problem/linked-list-cycle-ii

给定一个链表，如果链表中存在环，则返回到链表中环的起始节点的值，如果没有环，返回null。

样例

给出 -21->10->4->5, tail connects to node index 1，返回10

Source code:

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: The node where the cycle begins.
     *           if there is no cycle, return null
     */
    ListNode *detectCycle(ListNode *head) {
        // write your code here
    }
};
```

5 - 动态规划（上）

438. 书籍复印 II

http://www.lintcode.com/problem/copy-books-ii

你有 n 本书，每本书页数一样现在有 k 个人，每个人复印一本书需要花费时间 times[i] ，现在一个人只能复印连续一段编号的书，比如第一个人可以复印第一本第二本，但是不能复印第一本第三本。

求最少花费时间

样例

n = 4 times = [3,2,4] 返回4 第一个人复印第一本书花费3，第二个人复印二三两本书花费4，第三个人复印第四本书花费4

Source code:

```
class Solution {
```

```
public:
    /**
     * @param n: an integer
     * @param times: a vector of integers
     * @return: an integer
     */
    // V1 二分法 （最优解）
    int copyBooksII(int n, vector<int> &times) {
        if(n == 0){
            return 0;
        }
        int k = times.size();
        if(k == 0){
            return numeric_limits<int>::max();
        }
        else if (k == 1){
            //只有一个抄书员
            return times[0] * n;
        }
        //找出抄书时间的上下限
        // 所有人以最快速度抄书，得到下限
        // 所有人以最慢速度抄书，得到上限
        int lowBound = numeric_limits<int>::max();
        int highBound = numeric_limits<int>::min();
        for(int i = 0; i < k; i++){
            lowBound = min(lowBound, times[i]);
            highBound = max(highBound, times[i]);
        }
        lowBound =  lowBound * (( n + k - 1) / k);
        highBound = highBound * (( n + k - 1) / k);
                // 这里需要排序，将速度最快的排在前面，这样便于canCopy判断
        sort(times.begin(), times.end());
                // 二分法找出答案，每次二分时判断间值是否能在target时间内抄完
        while(lowBound < highBound){
            int mid = lowBound + (highBound - lowBound)/2;
            if(canCopy(mid, n, times)){
                highBound = mid;
            }
        }
        return lowBound;
    }
    // 判断是否能在target时间内抄写完成
    // 优先选择最快的人抄写，因此事前需要对times 排序。
    // 抄写速度最快的人尽量多抄
    bool canCopy(int target, int n, vector<int> &times){
        int totalBookCount = n;
        int sum = 0;
        int i = 0;
        while(i < times.size()){
            if(sum + times[i] <= target){
                sum += times[i];
```

```
            }
            else if (i != (times.size() - 1) &&
                     times[++i] <= target){
                sum = times[i];
            }
            else {
                return false;
            }

                        totalBookCount--;

                        if(totalBookCount <= 0){
                return true;
            }
        }
        return false;
    }

////////////////////////////////////////////////////////////////////////////////
        //////

////////////////////////////////////////////////////////////////////////////////
        /////
    // V2 : DP, time limit exceeded O(N2 * K)
    int copyBooksII1(int n, vector<int> &times) {
        // write your code here
        if(n == 0){
            return 0;
        }
        int k = times.size();
        if(k == 0){
            return numeric_limits<int>::max();
        }
        else if (k == 1){
            //只有一个抄书员
            return times[0] * n;
        }
        //f[i][j] 表示i个人抄j本书的最小耗时
        vector<vector<int> > f(k+1, vector<int>(n+1, 0));
        for(int j = 1; j <= n; j++){
            f[0][j] = numeric_limits<int>::max();
            f[1][j] = times[0] * j;
        }
        for(int i = 1; i <= k; i++){
            for(int j = 1; j <= n; j++){
                int minTime = numeric_limits<int>::max();
                for(int x = 0; x <= j; x++){
                    minTime = min(minTime,
                                  max(f[i-1][x], (j - x) * times[i-1]));
                }
                f[i][j] = minTime;
```

```
            }
        }
        return f[k][n];
    }
    //V3 : DP 优化，滚动数组空间优化，但是时间复杂度仍然比较高
    int copyBooksII2(int n, vector<int> &times) {
        // write your code here
        if(n == 0){
            return 0;
        }
        int k = times.size();
        if(k == 0){
            return numeric_limits<int>::max();
        }
        else if (k == 1){
            //只有一个抄书员
            return times[0] * n;
        }
        //f[i][j] 表示i个人抄j本书的最小耗时
        vector<vector<int> > f(2, vector<int>(n+1, 0));
        for(int j = 1; j <= n; j++){
            f[0][j] = numeric_limits<int>::max();
        }
        for(int i = 1; i <= k; i++){
            for(int j = 1; j <= n; j++){
                int minTime = numeric_limits<int>::max();
                for(int x = j; x >= 0; x--){
                    minTime = min(minTime,
                                  max(f[(i-1)%2][x], (j - x) * times[i-1]));

                                       if(f[(i-1)%2][x] < (j - x) *
times[i-1]){
                        break;
                    }

                                    }
                f[i%2][j] = minTime;
            }
        }
        return f[k%2][n];
    }

    };
```

437. 书籍复印

http://www.lintcode.com/problem/copy-books

给出一个数组A包含n个元素，表示n本书以及各自的页数。现在有

个k个人复印书籍，每个人只能复印连续一段编号的书，比如A[1],A[2]由第一个人复印，但是不能A[1],A[3]由第一个人复印，求最少需要的时间复印所有书。

样例

A = [3,2,4],k = 2 返回5，第一个人复印前两本书

Source code:

```
// http://www.jiuzhang.com/problem/2/
class Solution {
public:
    /**
     * @param pages: a vector of integers
     * @param k: an integer
     * @return: an integer
     */
    // 解法1：  动态规划
    // 设f[i][j]代表前i本书分给j个抄写员抄完的最少耗时。答案就是f[n][k]。
    // 思考最后一个人需要抄几本书
    // 状态转移方程f[i][j] = min{max(f[x][j-1], sum(x+1, i)), j<x
        <i}。其中x是在枚举第j个抄写员是从哪本书开始抄写。
    // 时间复杂度O(n^2*k)
    int copyBooks1(vector<int> &pages, int k) {
        // write your code here
        int n = pages.size(); // book number
        if(n == 0){
            return 0;
        }
        int ans = 0;
        //预处理边界条件
        if(k > n){
            for(int i = 0; i < n; i++){
                ans = max(ans, pages[i]);
            }
            return ans;
        }
                //f[i][j] 表示前i本书分给j给人抄的最少花费时间
        vector<vector<int> > f(n+1, vector<int>(k+1, 0));
        int maxPage = 0;
        for(int i = 0; i < n; i++){
            maxPage = max(maxPage, pages[i]);
        }
                for(int i = 1; i <= n; i++){
            f[i][0] = numeric_limits<int>::max();
        }
```

```
            // prepare sum start
        // sum[i] 表示从pages[0]到pages[i]的前缀和
        sum[0] = pages[0];
        for(int i = 1; i < n; i++){
            sum[i] = pages[i] + sum[i-1];
        }
        // prepare sum end
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= k; j++){
                int minTime = numeric_limits<int>::max();
                for(int x = j-1; x < i; x++) { // 枚举最后一个人从哪本书开始抄
                    // x表示前面j-1 个人抄x本书(至少j-1本，否则不够抄),
                    // 最后一个人抄第x+1本(下标x)到最后第i本(下标i-1)
                    minTime = min(minTime,
                                  max(f[x][j-1], sum[i-1] - sum[x-1]));
                }
                f[i][j] = minTime;
            }
        }

                return f[n][k];
    }
    // 解法2:  二分法
    // 二分答案，然后尝试一本本的加进来，加满了就给一个抄写员。
    // 看最后需要的抄写员数目是多余k个还是少于k个，然后来决定是将答案往上调整还
        是往下调整。
    // 时间复杂度O( n log Sum(pi) )
    int copyBooks(vector<int> &pages, int k) {
        int n = pages.size(); // book number
        if(n == 0){
            return 0;
        }
        int ans = 0;
        //预处理边界条件
        if(k > n){
            for(int i = 0; i < n; i++){
                ans = max(ans, pages[i]);
            }
            return ans;
        }

                int minTime = numeric_limits<int>::min();
        int maxTime = 0;
        for(int i = 0; i < n; i++){
                minTime = max(minTime, pages[i]); // min of books
                maxTime += pages[i];// sum of all
        }
        //可以这样想：如果一个人抄书那么耗时 maxTime , which is the sum of all
            pages （上限）.
        // 如果有足够多人，则最小耗时为所有书中最大值（下限）
        //答案必然在minTime and maxTime 之间
```

```
        // 二分法找出满足条件的答案
        int start = minTime, end = maxTime;
        while(start < end){
            int mid = start + (end - start) / 2;
            if(search(mid, pages, k)){
                // 此时已经满足条件n本书由k个人抄完，但是我们要找最小费时，所以
                    继续往左边区间找
                // 由于mid 是可能的答案之一，所以不能mid-1.
                end = mid;
            }
            else {
                // 在mid时间内无法抄完
                start = mid + 1;
            }
        }
        return start;
    }

        // search 函数返回值表示k个人在target 时间能否抄完所有书
    bool search(int target, vector<int> &pages, int k){
        int count = 1; // how many people needed
        int sum = 0;
        int i = 0;
        while(i < pages.size()){
            if(sum + pages[i] <= target){ // 每个人在target时间内尽量多抄
                sum += pages[i++];
            }
            else if(pages[i] <= target){
                count++;//上一个人抄不完，由另外一个人抄
                sum = pages[i++];
            }
            else {
                // 单本书就已经超时了，直接return
                return false;
            }
        }

                return count <= k;
    }
};
```

435. 邮局问题

http://www.lintcode.com/problem/post-office-problem

在一条线上有 n 个房子，现在给出它们的位置，选择 k 个位置建立邮局使得每个房子到最近邮局距离的总和最小

样例

[1,2,3,4,5] k = 2 答案为 3

Source code:

```
class Solution {
public:
    /**
     * @param A an integer array
     * @param k an integer
     * @return an integer
     */
    int postOffice(vector<int>& A, int k) {
        // Write your code here
        int n = A.size();
        if(n == 0 || k >= n){
            return 0;
        }
        // you must sort it first.
        sort(A.begin(), A.end());
        //dp[i][j] ďź· the minimal cost of having i post offices in j houses
        vector<vector<int> > dp(k+1, vector<int>(n+1, 0));
        //dis[x][y] : the minimal cost of have ONE post office between house
[x
            to y]. Starting from 1.
        vector<vector<int> > dis(n+1, vector<int>(n+1, 0));
        init(dis, A);
        for(int i = 0;i <= n;++i)  {
            dp[1][i] = dis[1][i];
        }
                for(int i = 2; i <= k; i++){
            for(int j = i; j <= n; j++){
                dp[i][j] = numeric_limits<int>::max();
                // the heading x houses have the (i-1) post offices
                for(int x = 0; x < j; x++){
                    if(dp[i][j] > dp[i-1][x] + dis[x+1][j]){
                        dp[i][j] = dp[i-1][x] + dis[x+1][j];
                    }
                }
            }
        }
        return dp[k][n];
    }
    // initialize the minimal cost of having single ONE post office between
        houses [xth to yth] inclusive
    void init(vector<vector<int> > &dis, vector<int>& A){
        int n = A.size();
        for(int i = 1; i <= n; i++){
            for(int j = i+1; j <= n; ++j){
```

```
                // Obviously having the post office in the midway is the
BEST!
                int middle = (i + j) / 2;
                for(int k = i; k <= j; ++k){
                    dis[i][j] += abs(A[k-1] - A[middle-1]);
                }
            }
        }
    }
};
```

396. 硬币排成线 III

http://www.lintcode.com/problem/coins-in-a-line-iii

有 n 个硬币排成一条线，每一枚硬币有不同的价值。两个参赛者轮流从任意一边取一枚硬币，知道没有硬币为止。计算拿到的硬币总价值，价值最高的获胜。

请判定 第一个玩家 是输还是赢?

样例

给定数组 A = [3,2,2], 返回 true. 给定数组 A = [1,2,4], 返回 true. 给定数组 A = [1,20,4], 返回 false.

Source code:

```
class Solution {
public:
    /**
     * @param values: a vector of integers
     * @return: a boolean which equals to true if the first player will win
     */
    // DP
    bool firstWillWin1(vector<int> &values) {
        // write your code here
        int n = values.size();
        if(n == 0){
            return false;
        }
        //f[x][y] 表示先手在剩余硬币区间[x,y]情况下能获得的最大价值
        vector<vector<int> > f(n, vector<int>(n, 0));
        for(int x = 0; x < n; x++){
            f[x][x] = values[x];
        }
```

```
        for(int x = 0; x < n-1; x++){
            f[x][x+1] = max(values[x], values[x+1]);
        }
                for(int x = n-3; x >= 0; x--){
            for(int y = x+2; y < n; y++){
                f[x][y] = max(values[x] + min(f[x+2][y], f[x+1][y-1]),
                              values[y] + min(f[x][y-2], f[x+1][y-1]));
            }
        }
                int sum = 0;
        for(int x : values){
            sum += x;
        }
        return f[0][n-1] > (sum / 2);
    }
        bool firstWillWin(vector<int> &values) {
        // write your code here
        int n = values.size();
        if(n == 0){
            return false;
            return true;
        }
        //f[x][y] 表示先手在剩余硬币区间[x,y]情况下能获得的最大价值
        vector<vector<int> > f(3, vector<int>(n, 0));
        for(int x = n-3; x >= 0; x--){
            for(int y = x; y < n; y++){
                if(y == x){
                    f[x%3][x] = values[x];
                }
                else if(y == x+1){
                    f[x%3][x+1] = max(values[x], values[x+1]);
                }
                else {
                    f[x%3][y] = max(values[x] + min(f[(x+2)%3][y],
f[(x+1)%3][y
                        -1]),
                                    values[y] + min(f[x%3][y-2], f[(x+1)%3][y-
1]
                                        ));
                }
            }
        }

                int sum = 0;
        for(int x : values){
            sum += x;
        }
        return f[0][n-1] > (sum / 2);
    }
};
```

393. 买卖股票的最佳时机 IV

http://www.lintcode.com/problem/best-time-to-buy-and-sell-stock-iv

假设你有一个数组，它的第i个元素是一支给定的股票在第i天的价格。

设计一个算法来找到最大的利润。你最多可以完成 k 笔交易。 注意事项 你不可以同时参与多笔交易(你必须在再次购买前出售掉之前的股票)

样例

给定价格 = [4,4,6,1,1,4,2,5], 且 k = 2, 返回 6.

Source code:

```
class Solution {
private:
    // 单次交易的maxprofit
    int profit(vector<int> &prices, int start, int end){
        int ans = 0;
        int minPrice = prices[start];
        for(int i = start+1; i <= end; i++){
            ans = max(ans, prices[i] - minPrice);
            minPrice = min(minPrice, prices[i]);
        }
        return ans;
    }
public:
    /**
     * @param k: An integer
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    // V1: Time Limit Exceeded
    int maxProfit1(int k, vector<int> &prices) {
        // write your code here
        if(prices.size() == 0 || k > prices.size()){
            return 0;
        }
        int n = prices.size();
        vector<vector<int> > f(n, vector<int>(k+1, 0));
        for(int i = 1; i < n; i++){
            for(int j = 1; j <= k; j++){
                int theProfit = 0;
                for(int m = 0; m < i; m++){
                    theProfit = max(theProfit, f[m][j-1] + profit(prices, m,
i
                        ));
```

```
                }
                f[i][j] = theProfit;
            }
        }
        return f[n-1][k];
    }
    // V2 : time Optimized (Prefered)
    int maxProfit(int k, vector<int> &prices) {
        // write your code here
            return 0;
        }
        int n = prices.size();
        #if 0
        if (k >= n / 2) {
            int profit = 0;
            for (int i = 1; i < n; i++) {
                if (prices[i] > prices[i - 1]) {
                    profit += prices[i] - prices[i - 1];
                }
            }
            return profit;
        }
        #endif

                //mustSell[i][j] 表示前i天，至多进行j次交易，第i天必须sell的最大获益
        vector<vector<int> > mustsell(n, vector<int>(k+1, 0));
        //globalbest[i][j]表示前i天，至多进行j次交易，第i天可以不sell的最大获益
        vector<vector<int> > globalbest(n, vector<int>(k+1, 0));
        for(int i = 1; i < n; i++){
            int gain = prices[i] - prices[i-1];
            for(int j = 1; j <= k; j++){
                mustsell[i][j] = max(globalbest[i-1][j-1] + gain,
                                     mustsell[i-1][j] + gain);

                                                        globalbest[i][j] =
max(globalbest[i-1][j],
                                    mustsell[i][j]);
            }
        }
        return globalbest[n-1][k];

            }

    };
```

6 - 动态规划（下）

564. Backpack VI

http://www.lintcode.com/problem/backpack-vi

Given an integer array nums with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

注意事项 The different sequences are counted as different combinations.

样例

Given nums = [1, 2, 4], target = 4 The possible combination ways are: [1, 1, 1, 1] [1, 1, 2] [1, 2, 1] [2, 1, 1] [2, 2] [4] return 6

Source code:

```
class Solution {
public:
    /**
     * @param nums an integer array and all positive numbers, no duplicates
     * @param target an integer
     * @return an integer
     */
    int backPackVI(vector<int>& nums, int target) {
        // Write your code here
    }
};
```

563. Backpack V

http://www.lintcode.com/problem/backpack-v

Given n items with size nums[i] which an integer array and all positive numbers. An integer target denotes the size of a backpack. Find the number of possible fill the backpack.

Each item may only be used once

样例

Given candidate items [1,2,3,3,7] and target 7, A solution set is: [7] [1, 3, 3] return 2

Source code:

```
class Solution {
public:
    /**
     * @param nums an integer array and all positive numbers
     * @param target an integer
     * @return an integer
     */
    int backPackV(vector<int>& nums, int target) {
        // Write your code here
    }
};
```

562. Backpack IV

http://www.lintcode.com/problem/backpack-iv

Given n items with size nums[i] which an integer array and all positive numbers, no duplicates. An integer target denotes the size of a backpack. Find the number of possible fill the backpack.

Each item may be chosen unlimited number of times

样例

Given candidate items [2,3,6,7] and target 7, A solution set is: [7] [2, 2, 3] return 2

Source code:

```
class Solution {
public:
    /**
     * @param nums an integer array and all positive numbers, no duplicates
     * @param target an integer
     * @return an integer
     */
    int backPackIV(vector<int>& nums, int target) {
        // Write your code here
    }
};
```

125. 背包问题 II

http://www.lintcode.com/problem/backpack-ii

给出n个物品的体积A[i]和其价值V[i]，将他们装入一个大小为m的背包，最多能装入的总价值有多大?

注意事项 A[i], V[i], n, m均为整数。你不能将物品进行切分。你所挑选的物品总体积需要小于等于给定的m。

样例

对于物品体积[2, 3, 5, 7]和对应的价值[1, 5, 2, 4], 假设背包大小为10的话，最大能够装入的价值为9。

Source code:

```
class Solution {
public:
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A & V: Given n items with size A[i] and value V[i]
     * @return: The maximum value
     */
    // V1, DP: O(n x m) memory
    int backPackII1(int m, vector<int> A, vector<int> V) {
        // write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        vector<vector<int> > f(n+1, vector<int>(m+1, 0));
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                f[i][j] = f[i-1][j];
                if(j >= A[i-1] &&
                   (f[i][j] < f[i-1][j-A[i-1]] + V[i-1])){
                    f[i][j] = f[i-1][j-A[i-1]] + V[i-1];
                }
            }
        }
        return f[n][m];
    }
    // V2 : DP O(M) memory by rolling array
    int backPackII(int m, vector<int> A, vector<int> V) {
        // write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        vector<vector<int> > f(2, vector<int>(m+1, 0));
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                f[i%2][j] = f[(i-1)%2][j];
                if(j >= A[i-1] &&
                   (f[i%2][j] < f[(i-1)%2][j-A[i-1]] + V[i-1])){
                    f[i%2][j] = f[(i-1)%2][j-A[i-1]] + V[i-1];
                }
        return f[n%2][m];
    }
};
```

91. 最小调整代价

http://www.lintcode.com/problem/minimum-adjustment-cost

给一个整数数组，调整每个数的大小，使得相邻的两个数的差小于一个给定的整数target，调整每个数的代价为调整前后的差的绝对值，求调整代价之和最小是多少。

注意事项 你可以假设数组中每个整数都是正整数，且小于等于100。

样例

对于数组[1, 4, 2, 3]和target=1，最小的调整方案是调整为[2, 3, 2, 3]，调整代价之和是2。返回2。

Source code:

```
class Solution {
public:
    /**
     * @param A: An integer array.
     * @param target: An integer.
     */
    int MinAdjustmentCost(vector<int> A, int target) {
        // write your code here
        int n = A.size();
        if(n < 2)
        {
            return 0;
        }
                vector<vector<int> > f(A.size(), vector<int>(101, 0));
        for(int j = 0; j < 101; j++)
        {
            f[0][j] = abs(j - A[0]);
        }
                for(int i = 1; i < n; i++)
        {
            for(int j = 0; j <= 100; j++)
            {
                f[i][j] = numeric_limits<int>::max();
                int diff = abs(j - A[i]);
                int upper = min(j+target, 100);
                int lower = max(j-target, 0);
                for(int k = lower; k <= upper; k++)
                {
                    f[i][j] = min(f[i][j], f[i-1][k] + diff);
                }
                            }
        }
        int ret = numeric_limits<int>::max();
        for(int j = 0; j <= 100; j++)
        {
            ret = min(ret, f[n-1][j]);
        }
};
```

168. 吹气球

http://www.lintcode.com/problem/burst-balloons

有n个气球，编号为0到n-1，每个气球都有一个分数，存在nums数组中。每次吹气球i可以得到的分数为 nums[left] * nums[i] *

nums[right]，left和right分别表示i气球相邻的两个气球。当i气球被吹爆后，其左右两气球即为相邻。要求吹爆所有气球，得到最多的分数。

注意事项 你可以假设nums[-1] = nums[n] = 1 0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100

样例

给出 [4, 1, 5, 10] 返回 270 nums = [4, 1, 5, 10] burst 1, 得分 4 * 1 * 5 = 20 nums = [4, 5, 10] burst 5, 得分 4 * 5 * 10 = 200 nums = [4, 10] burst 4, 得分 1 * 4 * 10 = 40 nums = [10] burst 10, 得分 1 * 10 * 1 = 10 总共的分数为 20 + 200 + 40 + 10 = 270

Source code:

```
class Solution {
public:
    /**
     * @param nums a list of integer
     * @return an integer, maximum coins
     */
    int maxCoins(vector<int>& nums) {
        // Write your code here
    }
};
```

440. Backpack III

http://www.lintcode.com/problem/backpack-iii

Given n kind of items with size Ai and value Vi(each item has an infinite number available) and a backpack with size m. What's the maximum value can you put into the backpack?

注意事项 You cannot divide item into small pieces and the total size of items you choose should smaller or equal to m.

样例

Given 4 items with size [2, 3, 5, 7] and value [1, 5, 2, 4], and a backpack with size 10. The maximum value is 15.

Source code:

```
class Solution {
public:
    /**
     * @param A an integer array
     * @param V an integer array
     * @param m an integer
     * @return an array
     */
    int backPackIII(vector<int>& A, vector<int>& V, int m) {
        // Write your code here
    }
};
```

393. 买卖股票的最佳时机 IV

http://www.lintcode.com/problem/best-time-to-buy-and-sell-stock-iv

假设你有一个数组，它的第i个元素是一支给定的股票在第i天的价格。

设计一个算法来找到最大的利润。你最多可以完成 k 笔交易。 注意事项 你不可以同时参与多笔交易(你必须在再次购买前出售掉之前的股票)

样例

给定价格 = [4,4,6,1,1,4,2,5], 且 k = 2, 返回 6.

Source code:

```
class Solution {
private:
    // 单次交易的maxprofit
    int profit(vector<int> &prices, int start, int end){
        int ans = 0;
        int minPrice = prices[start];
        for(int i = start+1; i <= end; i++){
            ans = max(ans, prices[i] - minPrice);
            minPrice = min(minPrice, prices[i]);
        }
        return ans;
    }
public:
    /**
     * @param k: An integer
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
```

```
    // V1: Time Limit Exceeded
    int maxProfit1(int k, vector<int> &prices) {
        // write your code here
        if(prices.size() == 0 || k > prices.size()){
            return 0;
        }
        int n = prices.size();
        vector<vector<int> > f(n, vector<int>(k+1, 0));
        for(int i = 1; i < n; i++){
            for(int j = 1; j <= k; j++){
                int theProfit = 0;
                for(int m = 0; m < i; m++){
                    theProfit = max(theProfit, f[m][j-1] + profit(prices, m,
i
                        ));
                }
                f[i][j] = theProfit;
            }
        }
        return f[n-1][k];
    }
    // V2 : time Optimized (Prefered)
    int maxProfit(int k, vector<int> &prices) {
        // write your code here
            return 0;
        }
        int n = prices.size();
        #if 0
        if (k >= n / 2) {
            int profit = 0;
            for (int i = 1; i < n; i++) {
                if (prices[i] > prices[i - 1]) {
                    profit += prices[i] - prices[i - 1];
                }
            }
            return profit;
        }
        #endif

                //mustSell[i][j] 表示前i天，至多进行j次交易，第i天必须sell的最大获益
        vector<vector<int> > mustsell(n, vector<int>(k+1, 0));
        //globalbest[i][j]表示前i天，至多进行j次交易，第i天可以不sell的最大获益
        vector<vector<int> > globalbest(n, vector<int>(k+1, 0));
        for(int i = 1; i < n; i++){
            int gain = prices[i] - prices[i-1];
            for(int j = 1; j <= k; j++){
                mustsell[i][j] = max(globalbest[i-1][j-1] + gain,
                                     mustsell[i-1][j] + gain);

                                                        globalbest[i][j] =
max(globalbest[i-1][j],
```

```
                                    mustsell[i][j]);
            }
        }
        return globalbest[n-1][k];

            }

    };
```

89. k数和

http://www.lintcode.com/problem/k-sum

给定n个不同的正整数，整数k（k < = n）以及一个目标数字。

在这n个数里面找出K个数，使得这K个数的和等于目标数字，求问有多少种方案?

样例

给出[1,2,3,4]，k=2， target=5，[1,4] and [2,3]是2个符合要求的方案

Source code:

```
class Solution {
public:
    /**
     * @param A: an integer array.
     * @param k: a positive integer (k <= length(A))
     * @param target: a integer
     * @return an integer
     */
    // V1: é··Í″·
    void helper(vector<int> A, int start, int k, int target, int &paths){
        if(target == 0 && k == 0){
            paths++;
            return;
        }
        if(target < 0 || k < 0 || start >= A.size()){
            return;
        }
                helper(A, start+1, k-1, target - A[start], paths);
        helper(A, start+1, k, target, paths);
    }
    int kSum1(vector<int> A, int k, int target) {
        // wirte your code here
        int paths = 0;
        helper(A, 0, k, target, paths);
        return paths;
    }
    // V2: DP
    int kSum(vector<int> A, int k, int target) {
        int n = A.size();
        // f[n][k][target]
        vector<vector<vector<int> > > f(n+1, vector<vector<int>>(k+1,
vector<int
            >(target+1, 0)));
        for(int i = 0; i <= n; i++){
            f[i][0][0] = 1;
        }
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= k && j <= i; j++){
                for(int t = 0; t <= target; t++){
                    if(t >= A[i-1]){
                        f[i][j][t] += f[i-1][j-1][t-A[i-1]];
                    }
                }
            }
        }
        return f[n][k][target];
    }
};
```

7 - 面试当中的常见算法拓展

551. Nested List Weight Sum

http://www.lintcode.com/problem/nested-list-weight-sum

Given a nested list of integers, return the sum of all integers in the list weighted by their depth. Each element is either an integer, or a list -- whose elements may also be integers or other lists.

样例

Given the list [[1,1],2,[1,1]], return 10. (four 1's at depth 2, one 2 at depth 1, 4 * 1 * 2 + 1 * 2 * 1 = 10) Given the list [1,[4,[6]]], return 27. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 42 + 63 = 27)

Source code:

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * class NestedInteger {
 *   public:
 *     // Return true if this NestedInteger holds a single integer,
 *     // rather than a nested list.
 *     bool isInteger() const;
 *
 *     // Return the single integer that this NestedInteger holds,
 *     // if it holds a single integer
 *     // The result is undefined if this NestedInteger holds a nested list
 *     int getInteger() const;
 *
 *     // Return the nested list that this NestedInteger holds,
 *     // if it holds a nested list
 *     // The result is undefined if this NestedInteger holds a single
integer
 *     const vector<NestedInteger> &getList() const;
 * };
 */
class Solution {
public:
    int depthSum(const vector<NestedInteger>& nestedList) {
        // Write your code here
    }
};
```

553. Bomb Enemy

http://www.lintcode.com/problem/bomb-enemy

Given a 2D grid, each cell is either a wall 'W', an enemy 'E' or empty '0' (the number zero), return the maximum enemies you can kill using one bomb.

The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed. 注意事项 You can only put the bomb at an empty cell.

样例

Given a grid: 0 E 0 0 E 0 W E 0 E 0 0 return 3. (Placing a bomb at (1,1) kills 3 enemies)

Source code:

```
class Solution {
public:
    /**
     * @param grid Given a 2D grid, each cell is either 'W', 'E' or '0'
     * @return an integer, the maximum enemies you can kill using one bomb
     */
    int maxKilledEnemies(vector<vector<char>>& grid) {
        // Write your code here
    }
};
```

541. 左旋右旋迭代器 II

http://www.lintcode.com/problem/zigzag-iterator-ii

k 个一维向量，循环地返回向量中的元素

样例

k = 3 [1,2,3] [4,5,6,7] [8,9] 返回 [1,4,8,2,5,9,3,6,7].

Source code:

```
class ZigzagIterator2 {
public:
    /**
     * @param vecs a list of 1d vectors
     */
    ZigzagIterator2(vector<vector<int>>& vecs) {
        // initialize your data structure here.
    }
    int next() {
        // Write your code here
    }
    bool hasNext() {
        // Write your code here
    }
};
/**
 * Your ZigzagIterator2 object will be instantiated and called as such:
 * ZigzagIterator2 solution(vecs);
 * while (solution.hasNext()) result.push_back(solution.next());
 * Ouptut result
 */
```

540. 左旋右旋迭代器

http://www.lintcode.com/problem/zigzag-iterator

给你两个一维向量，实现一个迭代器，交替返回两个向量的元素

样例

v1 = [1, 2] v2 = [3, 4, 5, 6] [1, 3, 2, 4, 5, 6]

Source code:

```
class ZigzagIterator {
public:
    /**
     * @param v1 v2 two 1d vectors
     */
    ZigzagIterator(vector<int>& v1, vector<int>& v2) {
        // initialize your data structure here.
    }
    int next() {
        // Write your code here
    }
    bool hasNext() {
        // Write your code here
    }
};
/**
 * Your ZigzagIterator object will be instantiated and called as such:
 * ZigzagIterator solution(v1, v2);
 * while (solution.hasNext()) result.push_back(solution.next());
 * Ouptut result
 */
```

404. 子数组求和 II

http://www.lintcode.com/problem/subarray-sum-ii

给定一个整数数组及一个区间，找到一个子数组，使得其数字的总和在给定区间范围内。请返回所有可能答案的数量。

样例

给定数组[1,2,3,4] 及 区间 = [1,3]， 返回 4。 所有可能的答案如下： [0, 0] [0, 1] [1, 1] [3, 3]

Source code:

```
class Solution {
```

```
public:
    /**
     * @param A an integer array
     * @param start an integer
     * @param end an integer
     * @return the number of possible answer
     */
    // prefix sum
    int subarraySumII1(vector<int>& A, int start, int end) {
        // Write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        // 先求出前缀和，然后枚举求两个前缀和的差。
        // 如果差在start与end之间，就给res+1。注意前缀和数组前要插入一个0。
        vector<int> presum(A);
        presum.insert(presum.begin(), 0); // size = n + 1
        for(int i = 1; i <= n; i++){
            presum[i] += presum[i-1];
        }
                // O(N2)
        int cnt = 0;
        for(int i = 0; i < n; i++){
            for(int j = i+1; j <= n; j++){
                int diff = presum[j] - presum[i];
                if(diff >= start && diff <= end){
                    cnt++;
                }
            }
        }
        return cnt;
    }
        // V2, 二分法 optimized to O(N lgN)
    int find(vector<int>& A, int len, int value) {
        if (A[len-1] < value )
            return len;

            int mid = (l + r) / 2;
            if (value <= A[mid]) {
                ans = mid;
                r = mid - 1;
            }  else
                l = mid + 1;
        }
        return ans;
    }
    int subarraySumII(vector<int>& A, int start, int end) {
        int len = A.size();
        for (int i = 1; i <len; ++i)
            A[i] += A[i-1];
```

```
        sort(A.begin(), A.end());
        int cnt = 0;
        for (int i = 0; i <len; ++i) {
            if (A[i] >= start && A[i] <= end)
                cnt ++;
            int l = A[i] - end;
            int r = A[i] - start;
            cnt += find(A, len, r+1) - find(A, len, l);
        }
        return cnt;
    }
};
```

390. 找峰值 II

http://www.lintcode.com/problem/find-peak-element-ii

一个整数矩阵有如下一些特性：

相邻的整数都是不同的 矩阵有 n 行 m 列。 对于所有的 i < m, 都有 A[0][i] < A[1][i] && A[n - 2][i] > A[n - 1][i]. 对于所有的 j < n, 都有 A[j][0] < A[j][1] && A[j][m - 2] > A[j][m - 1]. 我们定义一个位置 P 是一个峰，如果有 A[j][i] > A[j+1][i] && A[j][i] > A[j-1][i] && A[j][i] > A[j][i+1] && A[j][i] > A[j][i-1]。 找出该矩阵的一个峰值元素，返回他的坐标。 注意事项 可能会存在多个峰值，返回任意一个即可。

样例

给一个矩阵： [[1 ,2 ,3 ,4 ,5], [16,41,23,22,6], [15,17,24,21,7], [14,18,19,20,8], [13,12,11,10,9]] 返回 41 的坐标[1,1], 或者 24 的坐标[2,2]。

Source code:

```
class Solution {
private:
    int find(int row, vector<vector<int> > &A)
    {
        int col = 0;
        for(int j = 0; j < A[0].size(); j++)
        {
            if(A[row][j] > A[row][col])
            {
                col = j;
            }
        }
                return col;
    }
public:
    /**
```

```
     * @param A: An integer matrix
     * @return: The index of the peak
     */
    vector<int> findPeakII(vector<vector<int> > A) {
        // write your code here
        vector<int>  res;
        int n = A.size();
        if(n == 0)
        {
            return res;
        }
        int m = A[0].size();
                int start = 1;
        int end = n - 2;
        while(start <= end)
        {
            int mid = start + (end - start) / 2;
            int col = find(mid, A);
            if(A[mid][col] < A[mid-1][col])
            {
                end = mid - 1;
            }
            else if(A[mid][col] < A[mid+1][col])
            }
            else
            {
                res.push_back(mid);
                res.push_back(col);
                return res;
            }
        }

                return res;
    }
};
```