1 - 全球最火IT公司面试揭秘

543. N数组第K大元素

http://www.lintcode.com/problem/kth-largest-in-n-arrays

在N个数组中找到第K大元素

注意事项 你可以交换数组中的元素

样例

n = 2 数组 [[9,3,2,4,7],[1,2,3,4,8]], 第三大元素是 7. n = 2 数组 [[9,3,2,4,8],[1,2,3,4,2]], 最大数是 9, 次大数是 8, 第三大数是 7

Source code:

```
struct Node {
    int value;
    int from_id;
    int index;
    Node(int _v, int _id, int _i):
        value(_v), from_id(_id), index(_i) {}
    bool operator < (const Node & obj) const {
        return value < obj.value;
    }
};
class Solution {
public:
    /**
     * @param arrays a list of array
     * @param k an integer
     * @return an integer, K-th largest element in N arrays
     */
    int KthInArrays(vector<vector<int>>& arrays, int k) {
        // Write your code here
        if (k <= 0) {
            return numeric_limits<int>::min();
        }
        priority_queue<Node> pq;
        for(int i = 0; i < arrays.size(); i++) {
            //sort from largest to smallest
            sort(arrays[i].begin(), arrays[i].end(), greater<int>());
            if(arrays[i].size() > 0) {
                pq.push(Node(arrays[i][0], i, 0));
            }
        }
                for(int i = 0; i < k; i++) {
            if(pq.empty()){
                break;
            }
            Node cur = pq.top();
            pq.pop();
            int value = cur.value;
            if (i == k-1) {
                return value;
            }
};
```

443. 两数之和 II

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

给一组整数，问能找出多少对整数，他们的和大于一个给定的目标值。

注意事项 使用 O(1) 的额外空间和 O(nlogn) 的时间。

样例

对于 numbers = [2, 7, 11, 15], target = 24 的情况，返回 1。因为只有11 + 15可以大于24。

Source code:

```
class Solution {
public:
    /**
     * @param nums: an array of integer
     * @param target: an integer
     * @return: an integer
     */
    int twoSum2(vector<int> &nums, int target) {
        // Write your code here
        sort(nums.begin(), nums.end());
        int i = 0;
        int j = nums.size() - 1;
        int count = 0;
        while(i < j)
        {
            if(nums[i]+nums[j] > target)
            {
                count += j - i;
                j--;
            }
            else
            {
                i++;
            }
        }
        return count;
    }
};
```

401. 排序矩阵中的从小到大第k个数

http://www.lintcode.com/problem/kth-smallest-number-in-sorted-matrix

在一个排序矩阵中找从小到大的第 k 个整数。

排序矩阵的定义为：每一行递增，每一列也递增。

样例

给出 k = 4 和一个排序矩阵： [[1 ,5 ,7], [3 ,7 ,8], [4 ,8 ,9],] 返回 5。

Source code:

```
class MyPoint {
public:
    int x;
    int y;
    int val;
    MyPoint(int _x, int _y, int _val){
        this->x = _x;
        this->y = _y;
        this->val = _val;
    }
    bool operator ()(const MyPoint &a, const MyPoint &b) const{
        return a.val > b.val;
    }
    bool operator <(const MyPoint &obj) const{
        return this->val > obj.val;
    }
};
class Solution {
public:
    /**
     * @param matrix: a matrix of integers
     * @param k: an integer
     * @return: the kth smallest number in the matrix
     */
    int kthSmallest(vector<vector<int> > &matrix, int k) {
        // write your code here
        int count = 0;
        priority_queue<MyPoint> pq;
        int n = matrix.size();
        if(n == 0){
            return -1;
        }
        int m = matrix[0].size();
                for(int i = 0; i < n; i++){
            pq.push(MyPoint(i, 0, matrix[i][0]));
        }
        for(int j = 0; j < k-1; j++){
            MyPoint now = pq.top();
            pq.pop();
            int x = now.x;
            if(y+1 < m){
                pq.push(MyPoint(x, y+1, matrix[x][y+1]));
            }
        }
        MyPoint ans = pq.top();
        return ans.val;
    }
};
```

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;

            }
};
```

382. 三角形计数

http://www.lintcode.com/problem/triangle-count

给定一个整数数组，在该数组中，寻找三个数，分别代表三角形三条边的长度，问，可以寻找到多少组这样的三个数来组成三角形？

样例

例如，给定数组 S = {3,4,6,7}，返回 3 其中我们可以找到的三个三角形为： {3,4,6} {3,6,7} {4,6,7} 给定数组 S = {4,4,4,4}, 返回 4 其中我们可以找到的三个三角形为： {4(1),4(2),4(3)} {4(1),4(2),4(4)} {4(1),4(3),4(4)} {4(2),4(3),4(4)}

Source code:

```
class Solution {
public:
    /**
     * @param S: A list of integers
     * @return: An integer
     */
    int triangleCount(vector<int> &S) {
        // write your code here
        int ans = 0;
        sort(S.begin(), S.end());
        for(int i = 2; i < S.size(); i++)
        {
                int j = 0;
                int k = i-1;
                while(j < k)
                {
                        if(S[j] + S[k] > S[i])
                        {
                                ans += k - j;
                                k--;
                        }
                        else
                        {
                                j++;
                        }
                }
        }
        return ans;
    }
};
```

2 - 高级数据结构（上）

433. 岛屿的个数

http://www.lintcode.com/problem/number-of-islands

给一个01矩阵，求不同的岛屿的个数。

0代表海，1代表岛，如果两个1相邻，那么这两个1属于同一个岛。我们只考虑上下左右为相邻。

样例

在矩阵： [[1, 1, 0, 0, 0], [0, 1, 0, 0, 1], [0, 0, 0, 1, 1], [0, 0, 0, 0, 0], [0, 0, 0, 0, 1]] 中有 3 个岛.

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;
             }
         }
     }
     // find its super parent
     int find(int id){
         int parent = id;
         while(father[parent] != parent){
             parent = father[parent];
         }
         return parent;
     }
     int compress_find(int id){
        int parent = id;
         while(father[parent] != parent){
             parent = father[parent];
         }
                  int fat = id;
         while(father[fat] != fat){
             int tmp = father[fat];
             father[fat] = parent;
             fat = tmp;
         }
                  return parent;
     }
     int union_both(int x, int y){
         int father_x = find(x);
         int father_y = find(y);
         if(father_x != father_y){
             father[father_x] = father_y;
         }
     }
class Solution {
public:
    /**
     * @param grid a boolean 2D matrix
     * @return an integer
     */
    // V1 : ć¸·é·śćł·
    int numIslands1(vector<vector<bool>>& grid) {
        // Write your code here
        int n = grid.size();
        if(n == 0){
            return 0;
        }
        int m = grid[0].size();
        if(m == 0){
            return 0;
        }
        int count = 0;
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(grid[i][j] != 0){
                    zeroNear(grid, i, j);
                    count++;
                }
            }
        }
        return count;
    }
    void zeroNear(vector<vector<bool>>& grid, int i, int j){
        if(grid[i][j] == 0){
            return;
        }
        grid[i][j] = 0;
        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 < grid.size() && ny >= 0 && ny < grid[0].size()){
                zeroNear(grid, nx, ny);
            }
        }
    }
    ////////////////////////////////
    // V2 : UnionFind
    int numIslands(vector<vector<bool>>& grid) {
        int n = grid.size();
        if(n == 0){
            return 0;
        }
        int m = grid[0].size();
        if(m == 0){
            return 0;
        }
        UnionFind uf(n, m);
        int islandCount = 0;
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(grid[i][j] != 0){
                    islandCount++;
                    int id = i * 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];
                        int nid = nx * m + ny;
                        if(nx >= 0 && nx < grid.size() && ny >= 0 && ny < grid[0].size()
&&
                            grid[nx][ny] != 0){
                            int fa = uf.find(id);
                            int nfa = uf.find(nid);
                            if(fa != nfa){
                                uf.union_both(id, nid);
                                islandCount--;
                            }
                        }
                    }
                }
            }
        }
        return islandCount;
    }
};
```

391. 数飞机

http://www.lintcode.com/problem/number-of-airplanes-in-the-sky

给出飞机的起飞和降落时间的列表，用 interval 序列表示. 请计算出天上同时最多有多少架飞机？

注意事项 如果多架飞机降落和起飞在同一时刻，我们认为降落有优先权。

样例

对于每架飞机的起降时间列表：[[1,10],[2,3],[5,8],[4,7]], 返回3。

Source code:

```
/**
 * Definition of Interval:
 * classs Interval {
 *     int start, end;
 *     Interval(int start, int end) {
 *         this->start = start;
 *         this->end = end;
 *     }
 */
class myPoint {
public:
    int startTime;
    int isStart;
    myPoint(int _startTime, bool _isStart){
        this->startTime = _startTime;
        this->isStart = _isStart;
    }
        // writing sort compare funtion is a pain!!!!
    // const ... const is must! or else compile error
    #if 0
    bool operator ()(const myPoint &a, const myPoint &b) const{
        if(a.startTime == b.startTime){
            return a.isStart < b.isStart;
        }
        else{
            return a.startTime < b.startTime;
        }
    }
    #endif
        // const ... const is must! or else compile error
    bool operator <(const myPoint &obj) const{
        if(startTime == obj.startTime){
            return isStart < obj.isStart;
        }
        else{
            return startTime < obj.startTime; // 从小到大排序
        }
    }
    bool operator >(const myPoint &obj) const{
        if(startTime == obj.startTime){
        }
        else{
            return startTime > obj.startTime;
        }
    }
};
// 注意此处定义myPoint 与 sort对象vector<myPoint>参数类型要一致
bool mycompare(const myPoint &a, const myPoint &b){
    if(a.startTime == b.startTime){
        return a.isStart < b.isStart;
    }
    else{
        return a.startTime < b.startTime;
    }
}
class Solution {
public:
    /**
     * @param intervals: An interval array
     * @return: Count of airplanes are in the sky.
     */
    int countOfAirplanes(vector<Interval> &airplanes) {
        // write your code here
        // 注意此处定义myPoint 与 mycompare 函数参数类型要一致
        vector<myPoint> points;
        for(Interval itv : airplanes){
            points.push_back(myPoint(itv.start, 1));
            points.push_back(myPoint(itv.end, 0));
        }

                sort(points.begin(), points.end());
        //sort(points.begin(), points.end(), mycompare);

                int count = 0;
        int maxCount = 0;
        for(int i = 0; i < points.size(); i++){
            if(points[i].isStart){
                count++;
            }
            else{
                count--;
            }
            maxCount = max(maxCount, count);
        }
        return maxCount;
    }
};
```

178. 图是否是树

http://www.lintcode.com/problem/graph-valid-tree

给出 n 个节点，标号分别从 0 到 n - 1 并且给出一个 无向 边的列表 (给出每条边的两个顶点), 写一个函数去判断这张 ` 无向 ` 图是否是一棵树

注意事项 你可以假设我们不会给出重复的边在边的列表当中. 无向边　[0, 1] 和 [1, 0]　是同一条边，　因此他们不会同时出现在我们给你的边的列表当中。

样例

给出n = 5 并且 edges = [[0, 1], [0, 2], [0, 3], [1, 4]], 返回 true. 给出n = 5 并且 edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]], 返回 false.

Source code:

```
class Solution {
private:
    unordered_map<int, int > father;
public:
    void build(int n){
        for(int i = 0; i < n; i++){
            father[i] = i;
        }
    }
    int find(int x){
        int parent = x;
        while(parent != father[parent]){
            parent = father[parent];
        }
        return parent;
    }
    int compress_find(int x){
        int parent = x;
        while(parent != father[parent]){
            parent = father[parent];
        }
                int fa = x;
        // compressed find
        while(fa != father[fa]){
            int tmp = father[fa];
            father[fa] = parent;
            fa = tmp;
        }
                return parent;
    }
    void union_both(int x, int y){
        int fa_x = compress_find(x);
        int fa_y = compress_find(y);
        if(fa_x != fa_y){
            father[fa_y] = fa_x;
        }
    }
    /**
     * @param n an integer
     */
    bool validTree(int n, vector<vector<int>>& edges) {
        // Write your code here
        // n nodes tree must have n-1 edges
        if(edges.size() != n-1){
            return false;
        }
        // union find building
        build(n);
        for(auto edge : edges){
            if(compress_find(edge[0]) == compress_find(edge[1])){
                return false;
            }
            union_both(edge[0], edge[1]);
        }
        return true;
    }
};
```

473. 单词的添加与查找

http://www.lintcode.com/problem/add-and-search-word

设计一个包含下面两个操作的数据结构：addWord(word), search(word)

addWord(word)会在数据结构中添加一个单词。而search(word)则支持普通的单词查询或是只包含.和a-z的简易正则表达式的查询。 一个 . 可以代表一个任何的字母。 注意事项 你可以假设所有的单词都只包含小写字母 a-z。

样例

addWord("bad") addWord("dad") addWord("mad") search("pad") // return false search("bad") // return true search(".ad") // return true search("b..") // return true

Source code:

```
class TrieNode {
public:
    unordered_map<char, TrieNode *> children;
    bool isWord;
    TrieNode (){
        isWord = false;
    }
};
class WordDictionary {
public:
    TrieNode * root;
    WordDictionary(){
        root = new TrieNode;
    }
    // Adds a word into the data structure.
    void addWord(string word) {
        // Write your code here
        if(word.length() == 0){
            return;
        }
        TrieNode * now = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.at(i);
            if(now->children.find(c) == now->children.end()){
                now->children[c] = new TrieNode();
            }
            now = now->children[c];
        }
        now->isWord = true;
    }
    // Returns if the word is in the data structure. A word could
    // contain the dot character '.' to represent any one letter.
    bool search(string word) {
        // Write your code here
        return helper(root, word, 0);
    }
    bool helper(TrieNode * root, string word, int start){
        if(start >= word.length()){
            if(root->isWord){
            else{
                return false;
            }
        }
        char c = word.at(start);
        if(c == '.'){
            for(auto it : root->children){
                bool isMatch = helper(it.second, word, start+1);
                if(isMatch){
                    return true;
                }
            }
            return false;
        }
        else {
            if(root->children.find(c) == root->children.end()){
                return false;
            }
            return helper(root->children[c], word, start+1);
        }
    }
};
// Your WordDictionary object will be instantiated and called as such:
// WordDictionary wordDictionary;
// wordDictionary.addWord("word");
// wordDictionary.search("pattern");
```

432. 找出有向图中的弱联通分量

http://www.lintcode.com/problem/find-the-weak-connected-component-in-the-directed-graph

请找出有向图中弱联通分量的数目。图中的每个节点包含其邻居的 1 个标签和 1 个列表。（一个有向图中的相连节点指的是一个包含 2 个通过直接边沿路径相连的顶点的子图。）

样例

给定图: A----->B C \ | | \ | | \ | | \ v v ->D E <- F 返回 {A,B,D}, {C,E,F}. 图中有 2 个相连要素，即{A,B,D} 和 {C,E,F} 。

Source code:

```
/**
 * Definition for Directed graph.
 * struct DirectedGraphNode {
 *     int label;
 *     vector<DirectedGraphNode *> neighbors;
 *     DirectedGraphNode(int x) : label(x) {};
 * };
 */
class UnionFind {
private:
    unordered_map<DirectedGraphNode*, DirectedGraphNode*> father;
public:
    UnionFind(vector<DirectedGraphNode*>& nodes){
        for(int i = 0; i < nodes.size() ; i++){
            father[nodes[i]] = nodes[i];
        }
    }
    DirectedGraphNode* find(DirectedGraphNode* x) {
        DirectedGraphNode* fa = father[x];
        while(fa != father[fa]){
            fa = father[fa];
        }
        return fa;
    }
    void union_both(DirectedGraphNode* x, DirectedGraphNode* y) {
        DirectedGraphNode* fa_x = find(x);
        DirectedGraphNode* fa_y = find(y);
        if(fa_x != fa_y){
            father[fa_x] = fa_y;
        }
    }
};
class Solution {
public:
    /**
     * @param nodes a array of directed graph node
     * @return a connected set of a directed graph
     */
     // BFS 这里行不通
    vector<vector<int>> connectedSet2(vector<DirectedGraphNode*>& nodes) {
        // Write your code here
            return result;
        }
```

```
            // hashSet 去重
        unordered_set<DirectedGraphNode *> hashSet;
        for(int i = 0; i < nodes.size(); i++){
            DirectedGraphNode * node = nodes[i];
            hashSet.insert(node);
            for(int j = 0; j < node->neighbors.size(); j++){
                hashSet.insert(node->neighbors[j]);
            }
        }
        // union find && 合并相邻节点
        UnionFind uf(nodes);
        for(unordered_set<DirectedGraphNode *>::iterator it = hashSet.begin();
            it != hashSet.end(); it++){
            DirectedGraphNode * node = *it;
            for(int j = 0; j < node->neighbors.size(); j++){
                DirectedGraphNode * neighbor = node->neighbors[j];
                uf.union_both(node, neighbor);// 合并
            }
        }
        // 用hashmap通过father作为key 来分组，相同father的节点为一组
        unordered_map<DirectedGraphNode*, vector<int> > map; // father -> nodes
            that belong to the same father
        for(unordered_set<DirectedGraphNode *>::iterator it = hashSet.begin();
                                                        it != hashSet.end();
                                                        it++){
            DirectedGraphNode * node = *it;
            DirectedGraphNode * fa = uf.find(node);
            map[fa].push_back(node->label);
        }
        //输出各组的结果
        for(unordered_map<DirectedGraphNode*, vector<int> >::iterator it = map
            .begin();
                                                                      it != map
                                    .end();
                                                                      it++){
            vector<int> oneGroup = it->second;
            sort(oneGroup.begin(), oneGroup.end());
            result.push_back(oneGroup);

                                          }
        return result;
    }
};
```

431. 找无向图的连通块

http://www.lintcode.com/problem/find-the-connected-component-in-the-undirected-graph

找出无向图中所有的连通块。

图中的每个节点包含一个label属性和一个邻接点的列表。（一个无向图的连通块是一个子图，其中任意两个顶点通过路径相连，且不与整个图中的其它顶点相连。） 注意事项 每个连通块内部应该按照label属性排序

样例

给定图: A------B C \ | | \ | | \ | | \ | | D E 返回 {A,B,D}, {C,E}。其中有 2 个连通块，即{A,B,D}, {C,E}

Source code:

```
/**
 * Definition for Undirected graph.
 * struct UndirectedGraphNode {
 *     int label;
 *     vector<UndirectedGraphNode *> neighbors;
 *     UndirectedGraphNode(int x) : label(x) {};
 * };
 */
class Solution {
public:
    /**
     * @param nodes a array of Undirected graph node
     * @return a connected set of a Undirected graph
     */
    // BFS
    vector<vector<int>> connectedSet1(vector<UndirectedGraphNode*>& nodes) {
        // Write your code here
        vector<vector<int>> result;
        unordered_map<UndirectedGraphNode *, bool> visited;// if visited or not
                for(UndirectedGraphNode * node : nodes){
            if(visited.find(node) != visited.end()){
                continue;
            }
                        queue<UndirectedGraphNode*> q;
            q.push(node);
            visited[node] = true;
                        vector<int> component;// 一个子图 或者说 一个group节点
            while(!q.empty()){
                UndirectedGraphNode * now = q.front();
                q.pop();
                component.push_back(now->label);
                                for(int j = 0; j < now->neighbors.size(); j++){
                    UndirectedGraphNode * neighbor = now->neighbors[j];
                    if(visited.find(neighbor) == visited.end()){
                        q.push(neighbor);
                        visited[neighbor] = true;
                    }
            sort(component.begin(), component.end());// this is optional to make
                it nice looking
            result.push_back(component);
        }
        return result;
    }

        // DFS
    vector<vector<int>> connectedSet(vector<UndirectedGraphNode*>& nodes) {
        vector<vector<int>> result;
        unordered_map<UndirectedGraphNode*, bool> visited;
        for(auto node : nodes){
            vector<int> component;
            dfs(component, node, visited);
            if(component.size() > 0){
                sort(component.begin(), component.end());// this is optional to
                    make it nice looking
                result.push_back(component);
            }
        }
        return result;
    }
    void dfs(vector<int> &component,
        UndirectedGraphNode* now,
        unordered_map<UndirectedGraphNode*, bool> &visited){

            if(visited.find(now) != visited.end() && visited[now]){
            return;
        }
        visited[now] = true;
        component.push_back(now->label);

                for(auto nb : now->neighbors){
            dfs(component, nb, visited);
        }
    }
};
```

123. 单词搜索

http://www.lintcode.com/problem/word-search

给出一个二维的字母板和一个单词，寻找字母板网格中是否存在这个单词。

单词可以由按顺序的相邻单元的字母组成，其中相邻单元指的是水平或者垂直方向相邻。每个单元中的字母最多只能使用一次。

样例

给出board = ["ABCE", "SFCS", "ADEE"] word = "ABCCED"， ->返回 true, word = "SEE"， -> 返回 true, word = "ABCB"， -> 返回 false.

Source code:

```
class Solution {
public:
    bool isMatch(vector<vector<char>>& board, int x, int y, string word, int
        start) {
        if(start >= word.length()){
            return true;
        }
        int n = board.size();
        int m = board[0].size();
        if(x < 0 || x >= n || y < 0 || y >= m){
            return false;
        }
        if(board[x][y] != word.at(start)){
            return false;
        }
        char backup = board[x][y];
        board[x][y] = '#';
        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(isMatch(board, nx, ny, word, start+1)){
                board[x][y]= backup;
                return true;
            }
        }
        board[x][y]= backup;
        return false;
    }
    bool exist(vector<vector<char>>& board, string word) {
        int n = board.size();
        if(n == 0){
            return false;
        }
        int m = board[0].size();
        if(m == 0){
            return false;
        }

            return false;
        }

                for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(isMatch(board, i, j, word, 0)){
                    return true;
                }
            }
        }
        return false;
    }
};
```

434. 岛屿的个数II

http://www.lintcode.com/problem/number-of-islands-ii

给定 n，m，分别代表一个2D矩阵的行数和列数，同时，给定一个大小为 k 的二元数组A。起初，2D矩阵的行数和列数均为 0，即该矩阵中只有海洋。二元数组有 k 个运算符，每个运算符有 2 个整数 A[i].x, A[i].y，你可通过改变矩阵网格中的A[i].x]，[A[i].y] 来将其由海洋改为岛屿。请在每次运算后，返回矩阵中岛屿的数量。

注意事项 0 代表海，1 代表岛。如果两个1相邻，那么这两个1属于同一个岛。我们只考虑上下左右为相邻。

样例

给定 n = 3, m = 3，　二元数组 A = [(0,0),(0,1),(2,2),(2,1)]. 返回 [1,1,2,2].

Source code:

```
/**
 * Definition for a point.
 * struct Point {
 *     int x;
 *     int y;
 *     Point() : x(0), y(0) {}
 *     Point(int a, int b) : x(a), y(b) {}
 * };
 */
class UnionFind {
public:
    unordered_map<int, int> father;
    UnionFind(int n, int m){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                int id = i * m + j;// 2D index -> 1D index
                father[id] = id;
            }
        }
    }
    // find the super father of x
    int find(int x) {
        int parent = father.at(x);
        while(parent != father.at(parent)){
            parent = father.at(parent);
        }
        return parent;
    }
    int compressed_find(int x){
        int parent = father.at(x);
        while(parent != father.at(parent)){
            parent = father.at(parent);
        }
                //带路径压缩的find
        int tmp = -1;
        int fa = father.at(x);
        while(fa != father.at(fa)){
            tmp = father.at(fa);
            father.at(fa) = parent;
            fa = tmp;
        }
                return parent;
    }
    int union_both(int x, int y) {
        int fa_x = compressed_find(x);
        int fa_y = compressed_find(y);
        if(fa_x != fa_y) {
            father.at(fa_x) = fa_y;
        }
    }
};
class Solution {
public:
    /**
     * @param n an integer
     * @param m an integer
     * @param operators an array of point
     * @return an integer array
     */
    vector<int> numIslands2(int n, int m, vector<Point>& operators) {
        // Write your code here
        vector<int> result;
        vector<vector<int> > islands(n, vector<int>(m, 0));
        UnionFind uf(n, m);
        int count = 0;
        for(int k = 0; k < operators.size(); k++){
            int x = operators[k].x;
            int y = operators[k].y;
            if(islands[x][y] != 1) {
                islands[x][y] = 1;
                count++;
                int dx[4] = {-1, 1, 0, 0};
                int dy[4] = {0, 0, -1, 1};
                for(int p = 0; p < 4; p++){
                    int nx = x + dx[p];
                    int ny = y + dy[p];
                    if(nx >= 0 && nx < n && ny >= 0 && ny < m && islands[nx][ny] == 1){
                        int id = x * m + y;
                        int nid = nx * m + ny;
                        int fa = uf.compressed_find(id);
                        int nfa = uf.compressed_find(nid);
                        if(fa != nfa){
                            uf.union_both(id, nid);
                            count--;
                        }
                    }
                }

                        }
            result.push_back(count);
        }
        return result;
    }
};
```

132. 单词搜索 II

http://www.lintcode.com/problem/word-search-ii

给出一个由小写字母组成的矩阵和一个字典。找出所有同时在字典和矩阵中出现的单词。一个单词可以从矩阵中的任意位置开始，可以向左/右/上/下四个相邻方向移动。

样例

给出矩阵： doaf agai dcan 和字典： {"dog", "dad", "dgdg", "can", "again"}

返回 {"dog", "dad", "can", "again"}

dog: doaf agai dcan dad: doaf agai dcan can: doaf agai dcan again: doaf agai dcan

Source code:

```
class TrieNode {
public:
    string s;
    unordered_map<char, TrieNode *> subtree;
    bool isWord;
    TrieNode(){
        isWord = false;
        s = "";
    }
};
class TrieTree{
public:
    TrieNode * root;
    TrieTree(){
        root = new TrieNode();
    }
    void insert(string word){
        TrieNode * node = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.at(i);
            if(node->subtree.find(c) == node->subtree.end()){
                node->subtree[c] = new TrieNode();
            }
            node = node->subtree[c];
        }
        node->s = word;
        node->isWord = true;
    }
    bool find(string word){
        TrieNode * node = root;
        for(int i = 0; i < word.length(); i++){
            char c = word.at(i);
            if(node->subtree.find(c) == node->subtree.end()){
                return false;
            }
            node = node->subtree[c];
        }
        return node->isWord;
    }
};
class Solution {
    /**
     * @param board: A list of lists of character
     * @param words: A list of string
     * @return: A list of string
     */
    vector<string> wordSearchII(vector<vector<char> > &board, vector<string> &words) {
        // write your code here
        vector<string> result;
        int n = board.size();
        if(n == 0){
            return result;
        }
        int m = board[0].size();
        if(m == 0){
            return result;
        }
        //build trie tree
        TrieTree tree;
        for(string word : words){
            tree.insert(word);
        }
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                search(result, tree.root, board, i, j);
            }
        }
        return result;
    }
    // helper funtion
    void search(vector<string> &result, TrieNode* root, vector<vector<char> > &board,
                int i, int j){
        if(root->isWord){
            // 答案去重
            if(find(result.begin(), result.end(), root->s) == result.end()){
                result.push_back(root->s);
            }
            // 不能return，比如 case ('se', 'see')
        }
        if(i < 0 || i >= board.size() || j < 0 || j >= board[0].size()){
            return;
        }
        char c = board[i][j];
        if(root->subtree.find(c) == root->subtree.end()){
            return;
        }

                board[i][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];
            // 此处不要判断nx  ny的有效范围，由下层递归去判断。否则单字符单词('a')不work
            //if(nx >= 0 && nx < board.size() && ny >= 0 && ny < board[0].size()){
                search(result, root->subtree[c], board, nx, ny);
            //}
        }
        board[i][j] = c;
    }
};
```

3 - 高级数据结构（下）

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;
    }
};
```

130. 堆化

http://www.lintcode.com/problem/heapify

给出一个整数数组，堆化操作就是把它变成一个最小堆数组。

对于堆数组A，A[0]是堆的根，并对于每个A[i]，A [i * 2 + 1]是A[i]的左儿子并且A[i * 2 + 2]是A[i]的右儿子。

说明

什么是堆？ 堆是一种数据结构，它通常有三种方法：push， pop 和 top。其中，“push”添加新的元素进入堆，“pop”删除堆中最小/最大元素，“top”返回堆中最小/最大元素。 什么是堆化？ 把一个无序整数数组变成一个堆数组。如果是最小堆，每个元素A[i]，我们将得到A[i * 2 + 1] >= A[i]和A[i * 2 + 2] >= A[i] 如果有很多种堆化的结果？ 返回其中任何一个。

Source code:

```
class Solution {
private:
    void siftdown(vector<int> &A, int i){
        while(i < A.size()){
            int left = i * 2 + 1;
            int right = i * 2 + 2;
            int small = i;
            if(left < A.size() && A[left] < A[small]){
                small = left;
            }
            if(right < A.size() && A[right] < A[small]){
                small = right;
            }
                    if(small == i) {
                break;
            }
                    int tmp = A[i];
            A[i] = A[small];
            A[small] = tmp;
                    i = small;
        }
    }
public:
    /**
     * @param A: Given an integer array
     * @return: void
     */
    void heapify(vector<int> &A) {
        // write your code here
        for(int i = A.size() / 2; i >= 0; i--){
            siftdown(A, i);
        }
    }
};
```

364. 接雨水 II

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

给出 n * m 个非负整数，代表一张X轴上每个区域为 1 * 1 的 2d 海拔图, 计算这个海拔图最多能接住多少（面积）雨水。

样例

例如，给定一个 5*4 的矩阵： [[12,13,0,12], [13,4,13,12], [13,8,10,12], [12,13,12,12], [13,13,13,13]] 返回 14.

Source code:

```
class Element {
public:
    int x;
    int y;
    int h;
    Element(int _x, int _y, int _h){
        this->x = _x;
        this->y = _y;
        this->h = _h;
    }
    bool operator()(const Element &a, const Element &b) const {
        return a.h > b.h;// min heap
    }
    bool operator <(const Element &obj) const {
        return this->h > obj.h;
    }
};
class Solution {
public:
    /**
     * @param heights: a matrix of integers
     * @return: an integer
     */
    int trapRainWater(vector<vector<int> > &heights) {
        // write your code here
        int ans = 0;
        int n = heights.size();
        if(n == 0){
            return 0;
        }
        int m = heights[0].size();
        if(m == 0){
            return 0;
        }
        vector<vector<bool> > visited(n, vector<bool>(m, false));
        priority_queue<Element> pq;
        for(int i = 0; i < n; i++){
            pq.push(Element(i, 0, heights[i][0]));
            pq.push(Element(i, m-1, heights[i][m-1]));
            visited[i][0] = true;
            visited[i][m-1] = true;
        }
            pq.push(Element(0, j, heights[0][j]));
            pq.push(Element(n-1, j, heights[n-1][j]));
            visited[0][j] = true;
            visited[n-1][j] = true;
        }

                int dx[4] = {-1, 1, 0, 0};
        int dy[4] = {0, 0, -1, 1};
        while(!pq.empty()){
            Element now = pq.top();
            pq.pop();
            for(int k = 0; k < 4; k++){
                int nx = now.x + dx[k];
                int ny = now.y + dy[k];
                if(nx < 0 || nx >= n || ny < 0 || ny >= m){
                    continue;
                }
                if(visited[nx][ny]){
                    continue;
                }
                visited[nx][ny] = true;
```

```
                    if(heights[nx][ny] < now.h){
                        ans += now.h - heights[nx][ny];
                        pq.push(Element(nx, ny, now.h));
                    }
                    else {
                        pq.push(Element(nx, ny, heights[nx][ny]));
                    }
                }

                        }
            return ans;
        }
};
```

360. 滑动窗口的中位数

http://www.lintcode.com/problem/sliding-window-median

给定一个包含 n 个整数的数组，和一个大小为 k 的滑动窗口,从左到右在数组中滑动这个窗口，找到数组中每个窗口内的最大值。(如果数组中有偶数，则在该窗口存储该数字后，返回第 N/2-th 个数字。)

样例

对于数组 [1,2,7,8,5], 滑动大小 k = 3 的窗口时，返回 [2,7,7] 最初，窗口的数组是这样的： [| 1,2,7 | ,8,5] , 返回中位数 2; 接着，窗口继续向前滑动一次。 [1, | 2,7,8 | ,5], 返回中位数 7; 接着，窗口继续向前滑动一次。 [1,2, | 7,8,5 |], 返回中位数 7;

Source code:

```
// class node
class node {
public:
    int id;
    int num; // in case duplicate numbers.
    node(node &now)
    {
        this->id = now.id;
        this->num = now.num;
    }
    node(int first, int second)
    {
        this->id = first;
        this->num = second;
    }
};
// class HashHeap
class HashHeap {
private:
    vector<int> heap;
    string mode;
    int size_t;
    unordered_map<int, node*> hash;
```

```
public:
    HashHeap(string mod)
    {
        mode = mod;
        size_t = 0;
    }
    HashHeap()
    {
        mode = "max";
        size_t = 0;
    }
        int peek()
    {
        return heap.at(0);
    }

        return size_t;
    }

        bool empty()
    {
        return (size_t == 0);
    }

        int parent(int id)
    {
        if(id == 0)
        {
            return -1;
        }
        return (id - 1)/2;
    }
    int lson(int id)
    {
        return (2 * id + 1);
    }
    int rson(int id)
    {
        return (2 * id + 2);
    }
    bool comparesmall(int a, int b)
    {
        if(a <= b)
        {
            if(mode.compare("min") == 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            if(mode.compare("min") == 0)
            {
                return false;
            }
            else
            {
                return true;
            }
        }
    }

        void swap(int idA, int idB)
    {
        int valA = heap[idA];
        int valB = heap[idB];

                hash[valB]->id = idA;
        hash[valA]->id = idB;

                heap[idA] = valB;
        heap[idB] = valA;
    }

        int poll()
    {
        size_t--;
        int now = heap[0];
        node* hashNow = hash[now];
        if(hashNow->num == 1)
        {
            swap(0, heap.size()-1);
            hash.erase(now);
            heap.pop_back();
            if(heap.size() > 0)
            {
                siftdown(0);
            }
        }
        else
        {
        hashNow->id = 0;
            hashNow->num--;
        }
        return now;
    }

        void add(int now)
    {
        size_t++;
        if(hash.find(now) != hash.end())
        {
            node* hashNow = hash[now];
            hashNow->num++;
        }
        else
        {
            heap.push_back(now);
            hash[now] = new node(heap.size()-1, 1);
        }

                siftup(heap.size()-1);
```

```
}

    void remove(int now)
{
    size_t --;
    node * hashNow = hash[now];
    int id = hashNow->id;
    int num = hashNow->num;
    if(num == 1)
    {
        swap(id, heap.size()-1);
        hash.erase(now);
        heap.pop_back();
        if(heap.size() > id)
        {
            siftup(id);
            siftdown(id);
        }
    }
    else
    {
        hashNow->num--;
    }
}

    void siftup(int id)
{
    while(parent(id) > -1)
    {
        int parentId = parent(id);
        if(comparesmall(heap[parentId], heap[id]) == true)
        {
            break;
        }
        else
        {
            swap(id, parentId);
        }
        id = parentId;
    }
}

    void siftdown(int id)
{
    while(lson(id) < heap.size())
    {
        int leftId = lson(id);
        int rightId = rson(id);
        int son;
        if (rightId >= heap.size() ||
            (comparesmall(heap[leftId], heap[rightId]) == true)) {
            son = leftId;
        } else {
            son = rightId;
        }

                    if (comparesmall(heap[id], heap[son]) == true) {
            break;
            } else {
                swap(id, son);
            }
            id = son;
        }
    }
};
class Solution {
public:
    /**
     * @param nums: A list of integers.
     * @return: The median of the element inside the window at each moving
     */
    vector<int> medianSlidingWindow(vector<int> &nums, int k)  {
        // write your code here
        vector<int> result;
        if(nums.size() < k || k < 1)
        {
            return result;
        }

                int median = nums[0];
        HashHeap minheap("min");
        HashHeap maxheap("max");

                for(int i = 0; i < nums.size(); i++)
        {
            if(i != 0)
            {
                if(nums[i] > median)
                {
                    minheap.add(nums[i]);
                }
                else
                {
                    maxheap.add(nums[i]);
                }
            }

                        if(i >= k)
            {
                // since num[i-k] is out of sliding window, we need to delete it
                if(median == nums[i-k])
                {
                    //it's median
                    if(maxheap.size() > 0)
                    {
                        median = maxheap.poll();
                    }
                    else if(minheap.size() > 0)
                    {
                        median = minheap.poll();
                    }
                }
                else if(median < nums[i-k])
                {
                    // must be in minheap
                    minheap.remove(nums[i-k]);
                }
                else
                {
                    // must be in maxheap
                    maxheap.remove(nums[i-k]);
                }
            }

                        // adjust heap size, ensure A = [B-1, B]
            // A: maxheap.size()
            // B: minheap.size()
            //
            while (maxheap.size() > minheap.size()) {
                minheap.add(median);
                median = maxheap.poll();
            }
            while (minheap.size() > (maxheap.size() + 1)) {
                maxheap.add(median);
                median = minheap.poll();
            }
            if ((i + 1) >= k) {
            result.push_back(median);
            }
        }

                return result;

            }
};
```

81. 数据流中位数

http://www.lintcode.com/problem/data-stream-median

数字是不断进入数组的，在每次添加一个新的数进入数组的同时返回当前新数组的中位数。

说明

中位数的定义： 中位数是排序后数组的中间值，如果有数组中有n个数，则中位数为A[(n-1)/2]。 比如：数组A=[1,2,3]的中位数是2，数组A=[1,19]的中位数是1。

Source code:

```
class Solution {
public:
    struct minHeapCompare {
        bool operator()(const int a, const int b)
        {
            return a > b;
        }
    };
    struct maxHeapCompare {
        bool operator()(const int a, const int b)
        {
            return a < b;
        }
    };
    /**
     * @param nums: A list of integers.
     * @return: The median of numbers
     */
    vector<int> medianII(vector<int> &nums) {
        // write your code here
        vector<int> result;
        if(nums.size() == 0)
        {
            return result;
        }
        priority_queue<int, vector<int>, maxHeapCompare> pqmax;//(int, vector
            <int>, maxHeapCompare); // left
        priority_queue<int, vector<int>, minHeapCompare> pqmin;//(int, vector
            <int>, minHeapCompare); // right
        int m = nums[0];
        result.push_back(m);
        for(int i = 1; i < nums.size(); i++)
        {
            int x = nums[i];
            if(pqmax.size() == pqmin.size())
            {
                if(x >= m)
                {
                    pqmin.push(x);
                }
                else
                {
                    pqmin.push(m);
                    m = x;
                    if(!pqmax.empty() && x < pqmax.top())
                    {
                        m = pqmax.top();
                        pqmax.pop();
                        pqmax.push(x);
                    }
                }
            }
            else if(pqmax.size() == (pqmin.size()-1))
            {
                if(x <= m)
                {
                    pqmax.push(x);
                }
                else
                {
                    pqmax.push(m);
                    m = x;
                    if(!pqmin.empty() && x > pqmin.top())
                    {
                        m = pqmin.top();
                        pqmin.pop();
                        pqmin.push(x);
                    }
                }
            }
            result.push_back(m);
        }

                return result;
    }
};
```

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;
            }
        }
    }
};
```

362. 滑动窗口的最大值

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

给出一个可能包含重复的整数数组，和一个大小为 k 的滑动窗口, 从左到右在数组中滑动这个窗口，找到数组中每个窗口内的最大值。

样例

给出数组 [1,2,7,7,8], 滑动窗口大小为 k = 3. 返回 [7,7,8]. 解释： 最开始，窗口的状态如下： [|1, 2 ,7| ,7 , 8], 最大值为 7; 然后窗口向右移动一位： [1, |2, 7, 7|, 8], 最大值为 7; 最后窗口再向右移动一位： [1, 2, |7, 7, 8|], 最大值为 8.

Source code:

```
class Solution {
public:
    /**
     * @param nums: A list of integers.
     * @return: The maximum number inside the window at each moving.
     */
    // V1 : Deque, O(N) Best
    vector<int> maxSlidingWindow(vector<int> &nums, int k) {
        // write your code here
        vector<int> result;
        deque<int> q;
        if(nums.size() < k || k < 1)
        {
            return result;
        }
        for(int i = 0; i < k -1; i++)
        {
            while(!q.empty() && nums[i] >= nums[q.back()])
            {
                q.pop_back();
            }
            q.push_back(i);
        }
        for(int left = 0, right = k - 1;
            left <= (nums.size() - k) && right < nums.size();
            left++, right++)
        {
            while(!q.empty() && nums[right] >= nums[q.back()])
            {
                q.pop_back();
            }
            q.push_back(right);
            while(!q.empty() && q.front() < left)
            {
                q.pop_front();
            }
                        result.push_back(nums[q.front()]);
        }
        return result;
};
```

4 - 两个指针

399. Nuts 和 Bolts 的问题

http://www.lintcode.com/problem/nuts-bolts-problem

给定一组 n 个不同大小的 nuts 和 n 个不同大小的 bolts。nuts 和 bolts 一一匹配。 不允许将 nut 之间互相比较，也不允许将 bolt 之间互相比较。也就是说，只许将 nut 与 bolt 进行比较， 或将 bolt 与 nut 进行比较。请比较 nut 与 bolt 的大小。

样例

给出 nuts = ['ab','bc','dd','gg'], bolts = ['AB','GG', 'DD', 'BC'] 你的程序应该找出bolts和nuts的匹配。 一组可能的返回结果是： nuts = ['ab','bc','dd','gg'], bolts = ['AB','BC','DD','GG'] 我们将给你一个匹配的比较函数，如果我们给你另外的比较函数，可能返回的结果是： nuts = ['ab','bc','dd','gg'], bolts = ['BC','AB','DD','GG'] 因此的结果完全取决于比较函数，而不是字符串本身。 因为你必须使用比较函数来进行排序。 各自的排序当中nuts和bolts的顺序是无关紧要的，只要他们一一匹配就可以。

Source code:

```
/**
 * class Comparator {
 *     public:
 *      int cmp(string a, string b);
 * };
 * You can use compare.cmp(a, b) to compare nuts "a" and bolts "b",
 * if "a" is bigger than "b", it will return 1, else if they are equal,
 * it will return 0, else if "a" is smaller than "b", it will return -1.
 * When "a" is not a nut or "b" is not a bolt, it will return 2, which is not valid.
*/
class Solution {
private:
    void qsort(vector<string> &nuts, vector<string> &bolts, Comparator compare, int
start,
        int end){
        if(start >= end){
            return;
        }
        int part_inx = partition(nuts, bolts[start], compare, start, end);
        partition(bolts, nuts[part_inx], compare, start, end);
        qsort(nuts, bolts, compare, start, part_inx-1);
        qsort(nuts, bolts, compare, part_inx+1, end);
    }
    int partition(vector<string> &str, string pivot, Comparator compare, int l, int u){
        // l 放pivot
        // mid 左边小（含自己）， 右边大
        int mid = l; // middle
        for (int i = l + 1; i <= u; i++) {
            if (compare.cmp(str[i], pivot) == -1 ||
                compare.cmp(pivot, str[i]) == 1) {
```

```
                // str[i] smaller than pivot
                mid++;
                swap(str, i, mid);
            } else if (compare.cmp(str[i], pivot) == 0 ||
                       compare.cmp(pivot, str[i]) == 0) {
                // swap nuts[l]/bolts[l] with pivot
                // l位置放pivot
                swap(str, i, l);
                i--;
            }
        }
        // move pivot to proper index
        swap(str, mid, l);
        return mid;

     }
    void swap(vector<string> &str, int p, int q){
        string tmp = str[p];
        str[p] = str[q];
        str[q] = tmp;
    }
public:
    /**
     * @param nuts: a vector of integers
     * @param bolts: a vector of integers
     * @param compare: a instance of Comparator
     * @return: nothing
     */
    void sortNutsAndBolts(vector<string> &nuts, vector<string> &bolts, Comparator
compare) {
        // write your code here
        if(nuts.size() == 0 || nuts.size() != bolts.size()){
            return;
        }
        qsort(nuts, bolts, compare, 0, nuts.size()-1);
    }
};
```

406. 和大于S的最小子数组

http://www.lintcode.com/problem/minimum-size-subarray-sum

给定一个由 n 个整数组成的数组和一个正整数 s ， 请找出该数组中满足其和 ≥ s 的最小长度子数组。如果无解，则返回 -1。

样例

给定数组 [2,3,1,2,4,3] 和 s = 7, 子数组 [4,3] 是该条件下的最小长度子数组。

Source code:

```
class Solution {
public:
    /**
     * @param nums: a vector of integers
     * @param s: an integer
     * @return: an integer representing the minimum size of subarray
     */
    //O(N) 两个指针
    int minimumSize(vector<int> &nums, int s) {
        // write your code here
        int n = nums.size();
        if(n == 0)
        {
            return -1;
        }
        int minlen = numeric_limits<int>::max();
        int i = 0, j = 0;
        int sum = 0;
        for(i = 0; i < n; i++)
        {
            while(j < n && sum < s)
            {
                sum += nums[j];
                j++;
            }
            if(sum >= s)
            {
                minlen = min(minlen, j- i);
            }
            sum -= nums[i];
        }
        return (minlen == numeric_limits<int>::max()) ? -1:minlen;
    }
        // prefix sum
    int minimumSize2(vector<int> &nums, int s) {
        int n = nums.size();
        if(n == 0)
        {
            return -1;
        }
        for(int i = 1; i <= n; i++){
            presum[i] += presum[i-1];
        }

                int minLen = numeric_limits<int>::max();
        for(int i = 0; i < n; i++){
            for(int j = i+1; j <= n; j++){
                int diff = presum[j] - presum[i];
                if(diff >= s){
                    minLen = min(minLen, j - i);
                    break;
                }
            }
        }
        return (minLen == numeric_limits<int>::max()) ? -1:minLen;

            }
};
```

383. 装最多水的容器

http://www.lintcode.com/problem/container-with-most-water

给定 n 个非负整数 a1, a2, ..., an, 每个数代表了坐标中的一个点 (i, ai)。画 n 条垂直线，使得 i 垂直线的两个端点分别为(i, ai)和(i, 0)。找到两条线，使得其与 x 轴共同构成一个容器，以容纳最多水。

注意事项 容器不可倾斜。

样例

给出[1,3,2], 最大的储水面积是2.

Source code:

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

384. 最长无重复字符的子串

http://www.lintcode.com/problem/longest-substring-without-repeating-characters

给定一个字符串，请找出其中无重复字符的最长子字符串。

样例

例如，在"abcabcbb"中，其无重复字符的最长子字符串是"abc"，其长度为 3。 对于，"bbbbb"，其无重复字符的最长子字符串为"b"，长度为1。

Source code:

```
class Solution {
public:
    /**
     * @param s: a string
     * @return: an integer
     */
    // v1
    int lengthOfLongestSubstring(string s) {
        // write your code here
        if(s.length() == 0){
            return 0;
        }
        unordered_map<char, int> map;
        int maxLen = 1;
        int left = 0;
        for(int right = 0; right < s.length(); right++){
            char c = s.at(right);
            if(map.find(c) != map.end()){
                do{
                    map.erase(s.at(left));
                }while(s.at(left++) != c);
             }
            map[c] = right;
            maxLen = max(maxLen, right - left + 1);
        }
        return maxLen;
    }
    // V2:
    int lengthOfLongestSubstring2(string s) {
        // write your code here
        if(s.length() == 0){
            return 0;
        }
                int maxLen = 0;
        vector<int> map(256, 0);
        int left = 0;
        for(int right = 0; right < s.length(); right++){
            map[s.at(right)]++;
            while(map[s.at(right)] > 1){
                map[s.at(left)]--;
                left++;
            }
            maxLen = max(maxLen, right - left + 1);
        }
        return maxLen;
    }
};
```

386. 最多有k个不同字符的最长子字符串

http://www.lintcode.com/problem/longest-substring-with-at-most-k-distinct-characters

给定一个字符串，找到最多有k个不同字符的最长子字符串。

样例

例如，给定 s = "eceba" , k = 3, T 是 "eceb"， 长度为 4.

Source code:

```
class Solution {
public:
    /**
     * @param s : A string
     * @return : The length of the longest substring
     *           that contains at most k distinct characters.
     */
    // V1
    int lengthOfLongestSubstringKDistinct(string s, int k) {
        // write your code here
        if(k <= 0 || s.length() == 0 || k > s.length()){
            return 0;
        }
        int slow = 0;
        int maxLen = 0;
        // Key: letter; value: the number of occurrences.
        unordered_map<char, int> map;
        for (int fast = 0; fast < s.length(); ++fast) {
            char c = s.at(fast);
            if (map.find(c) != map.end()) {
                map[c] += 1;
            } else {
                map[c] = 1;
                while (map.size() > k) {
                    char slowChar = s.at(slow++);
                    map[slowChar]--;
                    if (map[slowChar] == 0) {
                        map.erase(slowChar);
                    }
                }
            }
            maxLen = max(maxLen, fast - slow + 1);
        }
        return maxLen;
    }
};
```

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;
    }
};
```

56. 两数之和

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

给一个整数数组，找到两个数使得他们的和等于一个给定的数 target。

你需要实现的函数twoSum需要返回这两个数的下标, 并且第一个下标小于第二个下标。注意这里下标的范围是 1 到 n，不是以 0 开头。 注意事项 你可以假设只有一组答案。

样例

给出 numbers = [2, 7, 11, 15], target = 9, 返回 [1, 2].

Source code:

```
class Solution {
public:
    /*
     * @param numbers : An array of Integer
     * @param target : target = numbers[index1] + numbers[index2]
     * @return : [index1+1, index2+1] (index1 < index2)
     */
    vector<int> twoSum(vector<int> &nums, int target) {
        // write your code here
        vector<int> result;
        unordered_map<int, int> hashmap;
        for(int i = 0; i < nums.size(); i++)
        {
            if(hashmap.find(target - nums[i]) != hashmap.end())
            {
                result.push_back(hashmap[target - nums[i]] + 1);
                result.push_back(i + 1);
                return result;
            }
                        hashmap[nums[i]] = i;
        }
                result.push_back(-1);
        result.push_back(-1);
        return result;
    }
};
```

32. 最小子串覆盖

http://www.lintcode.com/problem/minimum-window-substring

给定一个字符串source和一个目标字符串target，在字符串source中找到包括所有目标字符串字母的子串。

注意事项 如果在source中没有这样的子串，返回""，如果有多个这样的子串，返回起始位置最小的子串。

说明

在答案的子串中的字母在目标字符串中是否需要具有相同的顺序？ ——不需要。

Source code:

```
class Solution {
public:
    bool isValid(vector<int> &srcHash, vector<int> &tgtHash){
        for(int i = 0; i < 256; i++){
            if(srcHash[i] < tgtHash[i]){
                return false;
            }
        }
        return true;
    }
    /**
     * @param source: A string
     * @param target: A string
     * @return: A string denote the minimum window
     *          Return "" if there is no such a string
     */
    string minWindow(string &source, string &target) {
        // write your code here
        vector<int> sourceHash(256, 0);
        vector<int> targetHash(256, 0);
        for(int i = 0; i < target.length(); i++){
            targetHash[target.at(i)]++;
        }
                string ans="";
        int minLen = numeric_limits<int>::max();
        int left = 0, right = 0;
        for(left = 0; left < source.length(); left++){
            while(!isValid(sourceHash, targetHash) && right < source.length()){
                sourceHash[source.at(right)]++;
                right++;
            }
            if(isValid(sourceHash, targetHash)){
                if(minLen > (right - left)){
                    minLen = right - left;
                    ans = source.substr(left, minLen);
                }
            }
            sourceHash[source[left]]--;
                    }
    }
};
```

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);
    }
};
```

5 - 动态规划（上）

397. 最长上升连续子序列

http://www.lintcode.com/problem/longest-increasing-continuous-subsequence

给定一个整数数组（下标从 0 到 n-1， n 表示整个数组的规模），请找出该数组中的最长上升连续子序列。（最长上升连续子序列可以定义为从右到左或从左到右的序列。）

注意事项 time

样例

给定 [5, 4, 2, 1, 3], 其最长上升连续子序列（LICS）为 [5, 4, 2, 1], 返回 4. 给定 [5, 1, 2, 3, 4], 其最长上升连续子序列（LICS）为 [1, 2, 3, 4], 返回 4.

Source code:

```
class Solution {
public:
    /**
     * @param A an array of Integer
     * @return  an integer
     */
    int longestIncreasingContinuousSubsequence(vector<int>& A) {
        if(A.size() <= 1)
        {
            return A.size();
        }
        int maxCount = 1;
        int count = 1;
        for(int i = 1; i < A.size(); i++)
        {
            if(A[i] > (A[i-1]))
            {
                count++;
                if(count > maxCount)
                {
                    maxCount = count;
                }
            }
            else
            {
                count = 1;
            }
        }
                count = 1;
        for(int i = 1; i < A.size(); i++)
        {
            if(A[i] < (A[i-1]))
            {
                count++;
                if(count > maxCount)
                {
                    maxCount = count;
                }
            }
            else
            {
            }
        }
        return maxCount;
    }
};
```

41. 最大子数组

http://www.lintcode.com/problem/maximum-subarray

给定一个整数数组，找到一个具有最大和的子数组，返回其最大和。

注意事项 子数组最少包含一个数

样例

给出数组[−2,2,−3,4,−1,2,1,−5,3]，符合要求的子数组为[4,−1,2,1]，其最大和为6

Source code:

```
class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A integer indicate the sum of max subarray
     */
    int maxSubArray1(vector<int> nums) {
        // write your code here
        int maxSum = numeric_limits<int>::min();
        int minSum = 0;
        int currSum = 0;
        for(int i = 0; i < nums.size(); i++) {
            currSum += nums[i];
            maxSum = max(maxSum, currSum - minSum);
            minSum = min(minSum, currSum);
        }
                return maxSum;
    }
        int maxSubArray(vector<int> nums) {
        // write your code here
        int maxSum = numeric_limits<int>::min();
        int currSum = 0;
        for(int i = 0; i < nums.size(); i++) {
            if(currSum < 0) {
                currSum = nums[i];
            }
            else {
                currSum += nums[i];
            }
            maxSum = max(maxSum, currSum);
        }
                return maxSum;
    }
};
```

436. 最大正方形

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

在一个二维01矩阵中找到全为1的最大正方形

样例

1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 返回 4

Source code:

```
class Solution {
public:
```

```
/**
 * @param matrix: a matrix of 0 and 1
 * @return: an integer
 */
 // V1: DP
int maxSquare1(vector<vector<int> > &matrix) {
    // write your code here
    int n = matrix.size();
    if(n == 0){
        return 0;
    }
    int m = matrix[0].size();
    vector<vector<int> > f(n, vector<int>(m, 0));
            int ans = 0;
    // init
    for(int i = 0; i < n; i++){
        f[i][0] = matrix[i][0];
        ans = max(ans, f[i][0]);
    }
    for(int j = 1; j < m; j++){
        f[0][j] = matrix[0][j];
        ans = max(ans, f[0][j]);
    }
    //function
    for(int i = 1; i < n; i++){
        for(int j = 1; j < m; j++){
            if(matrix[i][j] != 0){
                f[i][j] = min(min(f[i-1][j], f[i][j-1]), f[i-1][j-1])
                            + 1;
            }
            else{
                f[i][j] = 0;
            }
                            ans = max(ans, f[i][j]);
        }
    }
    return ans * ans;
int maxSquare2(vector<vector<int> > &matrix) {
    // write your code here
    int n = matrix.size();
    if(n == 0){
        return 0;
    }
    int m = matrix[0].size();
    vector<vector<int> > f(2, vector<int>(m, 0));

            int ans = 0;
    // init
    for(int i = 0; i < n; i++){
        f[i%2][0] = matrix[i][0];
        ans = max(ans, f[i%2][0]);
    }
    for(int j = 1; j < m; j++){
        f[0][j] = matrix[0][j];
        ans = max(ans, f[0][j]);
    }
    //function
    for(int i = 1; i < n; i++){
            for(int j = 1; j < m; j++){
                if(matrix[i][j] != 0){
                    f[i%2][j] = min(min(f[(i-1)%2][j], f[i%2][j-1]), f[(i-1
                        )%2][j-1])
                                + 1;
                }
                else{
                    f[i%2][j] = 0;
                }

                                ans = max(ans, f[i%2][j]);
            }
        }
        return ans * ans;
    }

        // V3: 通过滚动数组进行空间优化
    int maxSquare(vector<vector<int> > &matrix) {
        // write your code here
        int n = matrix.size();
        if(n == 0){
            return 0;
        }
        int m = matrix[0].size();
        vector<vector<int> > f(n, vector<int>(2, 0));

                int ans = 0;
        // init
        for(int i = 0; i < n; i++){
            f[i][0] = matrix[i][0];
            ans = max(ans, f[i][0]);
        }
        for(int j = 1; j < m; j++){
            f[0][j%2] = matrix[0][j%2];
            ans = max(ans, f[0][j%2]);
        }
        //function
        for(int j = 1; j < m; j++){
        for(int i = 1; i < n; i++){

                        if(matrix[i][j] != 0){
                    f[i][j%2] = min(min(f[i-1][j%2], f[i][(j-1)%2]), f[i-1][(j-1
                        )%2])
                                + 1;
                }
                else{
                    f[i][j%2] = 0;
                }

                                ans = max(ans, f[i][j%2]);
            }
        }
        return ans * ans;
    }
};
```

200. 最长回文子串

http://www.lintcode.com/problem/longest-palindromic-substring

给出一个字符串（假设长度最长为1000），求出它的最长回文子串，你可以假定只有一个满足条件的最长回文串。

样例

给出字符串 "abcdzdcab"，它的最长回文子串为 "cdzdc"。

Source code:

```
class Solution {
public:
    /**
     * @param s input string
     * @return the longest palindromic substring
     */
     // v1: 中心扩展法
    string longestPalindrome(string& s) {
        // Write your code here
        string result("");
        int maxLen = 0;
        for(int i = 0; i < s.length(); i++){
            // odd length
            for(int len = 1; len <= (s.length() + 1)/2; len++){
                int left = i - len + 1;
                int right = i + len - 1;
                if(left < 0 ||
                   right >= s.length() ||
                   s.at(left) != s.at(right)){
                       break;
                }
               if((len * 2 - 1) > maxLen){
                   maxLen = len * 2 - 1;
                   result = s.substr(left, maxLen);
               }
            }
                        // even length
            for(int offset = 0; offset <= s.length()/2; offset++){
                int left = i - offset;
                int right = i + offset + 1;
                if(left < 0 ||
                   right >= s.length() ||
                   s.at(left) != s.at(right)){
                       break;
                }
                               if((offset * 2 + 2) > maxLen){
                   maxLen = offset * 2 + 2;
                   result = s.substr(left, maxLen);
        }
        return result;
    }

        // V2 : 借鉴马拉车算法 Manache
    // 'abc'插入#变成: “#a#b#c#”
    string longestPalindrome2(string& s) {
        int n = s.length();
        if(n <= 1){
            return s;
        }
        string result = "";
        int maxCount = 0;
        for(int i = 1; i <= 2 * n - 1; i++){
            int count = 1;
            while((i - count) >= 0 &&
                  (i + count) <= (2 * n) &&
                  (getChar(s, i - count) == getChar(s, i + count))){
                count ++;
            }
            count--;
            if(count > maxCount){
                maxCount = count;
                result = s.substr((i - count)/2, count);;
            }
        }
        return s.at(index / 2);
    }
};
```

395. 硬币排成线 II

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

有 n 个不同价值的硬币排成一条线。两个参赛者轮流从左边依次拿走 1 或 2 个硬币，直到没有硬币为止。计算两个人分别拿到的硬币总价值，价值高的人获胜。

请判定 第一个玩家 是输还是赢?

样例

给定数组 A = [1,2,2], 返回 true. 给定数组 A = [1,2,4], 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
     */
    // V1: DP
    bool firstWillWin1(vector<int> &values) {
        // write your code here
        int n = values.size();
        if(n == 0){
            return false;
        }
        else if(n == 1 || n == 2){
            return true;
        }
        // f[i]表示先手在剩余i个硬币的情况所能获得的最大值，从左到右
        vector<int> f(n+1, 0);
        f[0] = 0;
        f[1] = values[0];
        f[2] = values[0] + values[1];
        f[3] = values[0] + values[1];
        for(int i = 4; i <= n; i++){
            f[i] = max(values[n-i] + min(f[i-2], f[i-3]),
                       values[n-i] + values[n-i+1] + min(f[i-3], f[i-4]));
        }
                int sum = 0;
        for(int i = 0; i < n; i++){
            sum += values[i];
        }
        return f[n] > sum/2;
    }
        //V2: Memory Search
    bool firstWillWin(vector<int> &values) {
        int n = values.size();
        if(n == 0){
            return false;
        }
        vector<int> dp(n+1, -1);
        int ans = memorySearch(n, dp, values);
```

394. 硬币排成线

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

有 n 个硬币排成一条线。两个参赛者轮流从右边依次拿走 1 或 2 个硬币，直到没有硬币为止。拿到最后一枚硬币的人获胜。

请判定 第一个玩家 是输还是赢？

样例

n = 1, 返回 true. n = 2, 返回 true. n = 3, 返回 false. n = 4, 返回 true. n = 5, 返回 true.

Source code:

```
        }
        return f[n];
    }
        ///////////////////////////////////////////
    // V2 记忆化搜索
    bool firstWillWin(int n) {
        // write your code here
        vector<int> f(n+1, -1);
        int ans = search(f, n);
        return (ans == 1);
    }
    int search(vector<int> &f, int n){
        if(f[n] != -1){
            return f[n];
        }
                if(n == 0){
            f[0] = 0;
            return 0;
        }
        else if(n == 1){
            f[1] = 1;
            return 1;
        }
        else if(n == 2){
            f[2] = 1;
            return 1;
        }
        else if(n == 3){
            f[3] = 0;
            return 0;
        }
                f[n] = (search(f, n-2) && search(f, n-3)) ||
               (search(f, n-3) && search(f, n-4));
        return f[n];
    }
    };
```

392. 打劫房屋

http://www.lintcode.com/problem/house-robber

假设你是一个专业的窃贼，准备沿着一条街打劫房屋。每个房子都存放着特定金额的钱。你面临的唯一约束条件是：相邻的房子装着相互联系的防盗系统，且 当相邻的两个房子同一天被打劫时，该系统会自动报警。

给定一个非负整数列表，表示每个房子中存放的钱， 算一算，如果今晚去打劫，你最多可以得到多少钱 在不触动报警装置的情况下。

样例

给定 [3, 8, 4], 返回 8.

Source code:

```
class Solution {
public:
    /**
     * @param A: An array of non-negative integers.
     * return: The maximum amount of money you can rob tonight
     */
     //V1 DP
    long long houseRobber1(vector<int> A) {
        // write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        vector<long long> f(n, 0);
        f[0] = A[0];
        f[1] = max(A[0], A[1]);
                for(int i = 2; i < n; i++){
            f[i] = max(f[i-1], f[i-2]+A[i]);
        }
                return f[n-1];
    }
        // V2 用滚动数组进行空间优化
    long long houseRobber(vector<int> A) {
        // write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        long long f[2];
        f[0] = A[0];
        f[1] = max(A[0], A[1]);
                for(int i = 2; i < n; i++){
            f[i%2] = max(f[(i-1)%2], f[(i-2)%2]+A[i]);
        }

};
```

191. 乘积最大子序列

http://www.lintcode.com/problem/maximum-product-subarray

找出一个序列中乘积最大的连续子序列（至少包含一个数）。

样例

比如, 序列 [2,3,-2,4] 中乘积最大的子序列为 [2,3]， 其乘积为6。

Source code:

```
class Solution {
public:
    /**
     * @param nums: a vector of integers
     * @return: an integer
     */
    // V1 : simple one
    int maxProduct1(vector<int>& nums) {
        // write your code here
        int n = nums.size();
        if(n == 0){
            return numeric_limits<int>::min();
        }
        int g_max = numeric_limits<int>::min();
        int suffix_max = 1, suffix_min = 1;
        for(int i = 0; i < n; i++){
            int local_max = max(nums[i] * suffix_min, nums[i] * suffix_max);
            int local_min = min(nums[i] * suffix_min, nums[i] * suffix_max);
            suffix_max = max(local_max, nums[i]);
            suffix_min = min(local_min, nums[i]);
            g_max = max(g_max, suffix_max);
        }
        return g_max;
    }
        // V2
    int maxProduct(vector<int>& nums) {
        // write your code here
        int n = nums.size();
        if(n == 0){
            return numeric_limits<int>::min();
        }
        int g_max = numeric_limits<int>::min();
        int suffix_max = 1, suffix_min = 1;
        for(int i = 0; i < n; i++){
            if(nums[i] > 0){
                g_max = max(g_max, suffix_max * nums[i]);
                suffix_max = max(1, suffix_max * nums[i]);
                suffix_min = min(1, suffix_min * nums[i]);
            }
            else {
                suffix_max = max(1, suffix_min * nums[i]);
                suffix_min = min(1, tmp * nums[i]);
            }
        }
        return g_max;
    }
};
```

76. 最长上升子序列

http://www.lintcode.com/problem/longest-increasing-subsequence

给定一个整数序列，找到最长上升子序列（LIS），返回LIS的长度。

说明

最长上升子序列的定义： 最长上升子序列问题是在一个无序的给定序列中找到一个尽可能长的由低到高排列的子序列，这种子序列不一定是连续的或者唯一的。
https://en.wikipedia.org/wiki/Longest_increasing_subsequence

Source code:

```
class Solution {
public:
    /**
     * @param nums: The integer array
     * @return: The length of LIS (longest increasing subsequence)
     */
         // http://www.cnblogs.com/liyukuneed/archive/2013/05/26/3090402.html
      /* 解法一：最长公共子序列法：
仔细思考上面的问题，其实可以把上面的问题转化为求最长公共子序列的问题。原数组为A{5,  6,  7,
    1,  2, 8 }，下一步，我们对这个数组进行排序，排序后的数组为A‘{1,  2,  5,  6,
    7, 8}。我们有了这样的两个数组后，
如果想求数组A的最长递增子序列，其实就是求数组A与它的排序数组A‘的最长公共子序列。我来思考下
    原问题的几个要素：最长、递增、子序列（即顺序不变）。
*/
// 解法二：动态规划法（O(N^2)）
    int longestIncreasingSubsequence(vector<int> nums) {
        // write your code here
        // f[i] 定义为以nums[i]结尾的最长上升子序列
        int n = nums.size();
        if(n <= 1){
            return n;
        }
        vector<int> f(n, 0);
        f[0] = 1;
                for(int i = 1; i < n; i++){
            int maxCount = 1;
            for(int j = 0; j < i; j++){
                if(nums[i] >= nums[j]){
                    maxCount = max(maxCount, f[j] + 1);
                }
            }
            f[i] = maxCount;
        }
                int ans = 0;
        for(int i = 0; i < n; i++){
            ans = max(ans, f[i]);
        }
        return ans;
    }
};
```

398. 最长上升连续子序列 II

http://www.lintcode.com/problem/longest-increasing-continuous-subsequence-ii

给定一个整数矩阵（其中，有 n 行， m 列），请找出矩阵中的最长上升连续子序列。（最长上升连续子序列可从任意行或任意列开始，向上/下/左/右任意

方向移动）。

样例

给定一个矩阵 [[1 ,2 ,3 ,4 ,5], [16,17,24,23,6], [15,18,25,22,7], [14,19,20,21,8], [13,12,11,10,9]] 返回 25

Source code:

```
class Solution {
private:
    int search(vector<vector<int>>& A, int i, int j,
               vector<vector<int> > &flag,
               vector<vector<int> > &dp) {
                if(flag[i][j] != 0)  {
            return dp[i][j];
        }
        int n = A.size();
        int m = A[0].size();
                int ans = 1;
        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 && A[i][j] > A[nx][ny])
            {
                ans = max(ans, search(A, nx, ny, flag, dp) + 1);
            }
        }
        dp[i][j] = ans;
        flag[i][j] = 1;
        return ans;
    }
public:
    /**
     * @param A an integer matrix
     * @return  an integer
     */
    int longestIncreasingContinuousSubsequenceII(vector<vector<int>>& A) {
        // Write your code here
        int n = A.size();
        if(n == 0) {
            return 0;
        }
        int m = A[0].size();
        if(m == 0) {
        int ans = 0;
        vector<vector<int> > dp(n, vector<int>(m, 0));
        vector<vector<int> > flag(n, vector<int>(m, 0)); // indicate A[i][j] is
            searched or not
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                dp[i][j] = search(A, i, j, flag, dp);
                ans = max(ans, dp[i][j]);
            }
        }
        return ans;
    }
};
```

6 - 动态规划（下）

476. 石子归并

http://www.lintcode.com/problem/stone-game

有一个石子归并的游戏。最开始的时候，有n堆石子排成一列，目标是要将所有的石子合并成一堆。合并规则如下：

每一次可以合并相邻位置的两堆石子 每次合并的代价为所合并的两堆石子的重量之和 求出最小的合并代价。

样例

对于石子序列：[4, 1, 1, 4]（每个数代表这堆石子的重量），最优合并方案下，合并代价为 18： 1. 合并第2堆和第3堆 => [4, 2, 4], 代价 +2 2. 合并前两堆 => [6, 4]， 代价 +6 3. 合并剩下的两堆 => [10]， 代价 +10 其他例子： [1, 1, 1, 1] 代价为 8 [4, 4, 5, 9] 代价为 43

Source code:

```
class Solution {
public:
    /**
     * @param A an integer array
     * @return an integer
     */
    int stoneGame(vector<int>& A) {
        // Write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        vector<vector<int> > f(n, vector<int>(n, 0));
        vector<vector<int> > sum(n, vector<int>(n, 0));
        vector<int> presum(n , 0);
        presum[0] = A[0];
        for(int i = 1; i < n; i++){
            presum[i] = A[i] + presum[i-1];
            sum[0][i] = presum[i];
        }
        for(int i = 1; i < n; i++){
            for(int j = i; j < n; j++){
                sum[i][j] = presum[j] - presum[i-1];
            }
        }
                for(int x = 0; x < n-1; x++){
            f[x][x+1] = A[x] + A[x+1];
        }
        for(int x = n-2; x >=0; x--){
            for(int y = x+1; y < n; y++){
                int localMin = numeric_limits<int>::max();
                for(int k = x; k < y; k++){
                    localMin = min(localMin, f[x][k] + f[k+1][y] + sum[x][y]);
                }
                f[x][y] = localMin;
            }
        }
        return f[0][n-1];
    }
};
```

395. 硬币排成线 II

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

有 n 个不同价值的硬币排成一条线。两个参赛者轮流从左边依次拿走 1 或 2 个硬币，直到没有硬币为止。计算两个人分别拿到的硬币总价值，价值高的人获胜。

请判定 第一个玩家 是输还是赢？

样例

给定数组 A = [1,2,2], 返回 true. 给定数组 A = [1,2,4], 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
     */
    // V1: DP
    bool firstWillWin1(vector<int> &values) {
        // write your code here
        int n = values.size();
        if(n == 0){
            return false;
        }
        else if(n == 1 || n == 2){
            return true;
        }
        // f[i]表示先手在剩余i个硬币的情况所能获得的最大值，从左到右
        vector<int> f(n+1, 0);
        f[0] = 0;
        f[1] = values[0];
        f[2] = values[0] + values[1];
        f[3] = values[0] + values[1];
        for(int i = 4; i <= n; i++){
            f[i] = max(values[n-i] + min(f[i-2], f[i-3]),
                       values[n-i] + values[n-i+1] + min(f[i-3], f[i-4]));
        }
                int sum = 0;
        for(int i = 0; i < n; i++){
            sum += values[i];
        }
        return f[n] > sum/2;
    }
        //V2: Memory Search
    bool firstWillWin(vector<int> &values) {
        int n = values.size();
        if(n == 0){
            return false;
        }
        vector<int> dp(n+1, -1);
        for(int x : values){
            sum += x;
        }

};
```

394. 硬币排成线

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

有 n 个硬币排成一条线。两个参赛者轮流从右边依次拿走 1 或 2 个硬币，直到没有硬币为止。拿到最后一枚硬币的人获胜。

请判定 第一个玩家 是输还是赢？

样例

n = 1, 返回 true. n = 2, 返回 true. n = 3, 返回 false. n = 4, 返回 true. n = 5, 返回 true.

Source code:

```
class Solution {
public:
    /**
     * @param n: an integer
     * @return: a boolean which equals to true if the first player will win
     */
     bool firstWillWin1(int n) {
        // write your code here
        if(n == 0){
            return false;
        }
        else if(n == 1){
            return true;
        }
        else if(n == 2){
            return true;
        }
        else if(n == 3){
            return false;
        }
        vector<bool> f(n+1, false);
        f[0] = false;
        f[1] = true;
        f[2] = true;
        f[3] = false;
        for(int i = 4; i <= n; i++){
            f[i] = (f[i-2] && f[i-3]) ||
                   (f[i-3] && f[i-4]);
        }
        return f[n];
    }
        ///////////////////////////////////////////
    // V2 记忆化搜索
    bool firstWillWin(int n) {
        // write your code here
        vector<int> f(n+1, -1);
        int ans = search(f, n);
        return (ans == 1);
    }
            return f[n];
        }

                if(n == 0){
            f[0] = 0;
            return 0;
        }
        else if(n == 1){
            f[1] = 1;
            return 1;
        }
        else if(n == 2){
            f[2] = 1;
            return 1;
        }
        else if(n == 3){
            f[3] = 0;
            return 0;
        }

                f[n] = (search(f, n-2) && search(f, n-3)) ||
               (search(f, n-3) && search(f, n-4));
        return f[n];
    }

    };
```

92. 背包问题

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

在n个物品中挑选若干物品装入背包，最多能装多满？假设背包的大小为m，每个物品的大小为A[i]

注意事项 你不可以将物品进行切割。

样例

如果有4个物品[2, 3, 5, 7] 如果背包的大小为11， 可以选择[2, 3, 5]装入背包， 最多可以装满10的空间。 如果背包的大小为12， 可以选择[2, 3, 7]装入背包， 最多可以装满12的空间。 函数需要返回最多能装满的空间大小。

Source code:

```
class Solution {
public:
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A: Given n items with size A[i]
     * @return: The maximum size
     */
    int backPack(int m, vector<int> A) {
        // write your code here
        int n = A.size();
        if(n == 0){
            return 0;
        }
        vector<vector<bool> > f(n+1, vector<bool>(m+1, false));
        f[0][0] = true;
        for(int i = 1; i <= n; i++){
            for(int j = 0; j <= m; j++){
                if((j >= A[i-1]) && f[i-1][j-A[i-1]]){
                    f[i][j] = true;
                }
                else {
                    f[i][j] = f[i-1][j];
                }
            }
        }
                int res = 0;
        for(int j = m; j >= 0; j--){
            if(f[n][j]){
                res = j;
                break;
            }
        }
        return res;
    }
};
```

430. 攀爬字符串

http://www.lintcode.com/problem/scramble-string

给定一个字符串 S1，将其递归地分割成两个非空子字符串,从而将其表示为二叉树。

下面是s1 = "great"的一个可能表达： great / \ gr eat / \ / \ g r e at / \ a t 在攀爬字符串的过程中，我们可以选择其中任意一个非叶节点，然后交换该节点的两个儿子。 例如，我们选择了 "gr" 节点，并将该节点的两个儿子进行交换，从而产生了攀爬字符串 "rgeat"。 rgeat / \ rg eat / \ / \ r g e at / \ a t 我们认为， "rgeat" 是 "great" 的一个攀爬字符串. 类似地，如果我们继续将其节点 "eat" 和 "at" 进行交换，就会产生新的攀爬字符串 "rgtae"。 rgtae / \ rg tae / \ / \ r g ta e / \ t a 同样地，"rgtae" 也是 "great"的一个攀爬字符串。 给定两个相同长度的字符串s1 和 s2，判定 s2 是否为 s1 的攀爬字符串。

样例

Source code:

```
class Solution {
public:
    /**
     * @param s1 A string
     * @param s2 Another string
     * @return whether s2 is a scrambled string of s1
     */
    bool isValid(string s1, string s2){
        if(s1.length() != s2.length()){
            return false;
        }
        sort(s1.begin(), s1.end());
        sort(s2.begin(), s2.end());
        if(s1.compare(s2) != 0){
            return false;
        }
        return true;
    }
        bool isScramble(string& s1, string& s2){
        int len1 = s1.length();
        int len2 = s2.length();
        if(len1 != len2){
            return false;
        }
        if(len1 == 0 || s1.compare(s2) == 0){
            return true;
        }
        if(!isValid(s1, s2)){
            return false;
        }
        for(int k = 0; k < len1-1; k++){
            //s1 的前k个字符，和(n - k)个字符
            string s11 = s1.substr(0, k+1);
            string s12 = s1.substr(k+1, len1 - k - 1);
            //s2 的前k个字符，和(n - k)个字符
            string s21 = s2.substr(0, k+1);
            string s22 = s2.substr(k+1, len2 - k - 1);
            //s2 的前n-k个字符，和后k个字符
            string s23 = s2.substr(0, len2 - k - 1);
            string s24 = s2.substr(len2 - k -1, k+1);
            }
            if(isScramble(s11, s24) && isScramble(s12, s23)){
                   return true;
            }
        }
        return false;
    }
    ////////////////////////////////////////////////////
    // V2: Why below also pass without DP??
    bool isScramble2(string& s1, string& s2) {
        // Write your code here

                // 1. length check
        int len1 = s1.length();
        int len2 = s2.length();
        if(len1 == 0 && len2 == 0){
            return true;
        }
        if(len1 != len2){
            return false;
        }
        // 2. check if both strings have the same word set.
        sort(s1.begin(), s1.end());
        sort(s2.begin(), s2.end());
};
```

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);
    }
};
```

7 - 面试当中的常见算法拓展

138. 子数组之和

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

给定一个整数数组，找到和为零的子数组。你的代码应该返回满足要求的子数组的起始位置和结束位置

样例

给出 [-3, 1, 2, -3, 4]，返回[0, 2] 或者 [1, 3].

Source code:

```
class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
     // hash map
    vector<int> subarraySum1(vector<int> nums){
        // write your code here
        vector<int> result;
        int n = nums.size();
        if(n == 0){
            return result;
        }
        unordered_map<int, int> map;
        map[0] = -1;
        int sum = 0;
        for(int i = 0; i < nums.size(); i++){
            sum += nums[i];
            if(map.find(sum) != map.end()){
                result.push_back(map[sum] + 1);
                result.push_back(i);
                return result;
            }
            map[sum] = i;
        }
        return result;
    }
};
```

405. 和为零的子矩阵

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

给定一个整数矩阵，请找出一个子矩阵，使得其数字之和等于0.输出答案时，请返回左上数字和右下数字的坐标。

样例

给定矩阵 [[1 ,5 ,7], [3 ,7 ,-8], [4 ,-8 ,9],] 返回 [(1,1), (2,2)]

Source code:

```
class Solution {
public:
    /**
     * @param matrix an integer matrix
     * @return the coordinate of the left-up and right-down number
     */
    vector<vector<int>> submatrixSum(vector<vector<int>>& matrix) {
        // Write your code here
        vector<vector<int>> result;
        int n = matrix.size();
        if(n == 0){
            return result;
        }
        int m = matrix[0].size();
        if(m == 0){
            return result;
        }
        vector<vector<int>> sum(n+1, vector<int>(m+1, 0));
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                sum[i][j] = matrix[i-1][j-1] + sum[i-1][j] + sum[i][j-1] - sum[i
                    -1][j-1];
            }
        }
                for(int i = 0; i < n; i++){ // number of rows
            for(int j = i+1; j <= n; j++){ // number of rows
                unordered_map<int, int> map;
                for(int k = 0; k <= m; k++){
                    int diff = sum[j][k] - sum[i][k];
                    if(map.find(diff) == map.end()){
                        map[diff] = k; // col count
                    }
                    else {
                        int l = map[diff];
                        vector<int> topLeft = {i, l};
                        vector<int> bottomRight = {j-1, k-1};
                        result.push_back(topLeft);
                        result.push_back(bottomRight);
                        return result;
                    }
};
```

402. 连续子数组求和

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

给定一个整数数组，请找出一个连续子数组，使得该子数组的和最大。输出答案时，请分别返回第一个数字和最后一个数字的下标。（如果两个相同的答案，请返回其中任意一个）

样例

给定 [-3, 1, 3, -3, 4], 返回[1,4].

Source code:

```
class Solution {
public:
    /**
     * @param A an integer array
     * @return  A list of integers includes the index of
     *          the first number and the index of the last number
     */
    vector<int> continuousSubarraySum(vector<int>& A) {
        // Write your code here
        vector<int> ans;
        if(A.size() == 0)
        {
            return ans;
        }
        int n = A.size();
        int currSum = 0;
        int maxSum = numeric_limits<int>::min();
        int first = 0, last = n-1; // index
        int minFirst = 0, minLast = n - 1;
        for(int i = 0; i < A.size(); i++)
        {
            if(currSum >= 0)
            {
                currSum += A[i];
            }
            else
            {
                currSum = A[i];
                first = i;
            }
                        if(currSum > maxSum)
            {
                maxSum = currSum;
                minFirst = first;
                minLast = i;
            }
        }
        ans.push_back(minFirst);
        ans.push_back(minLast);
        return ans;
    }
};
```

403. 连续子数组求和 II

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

给定一个整数循环数组（头尾相接），请找出一个连续的子数组，使得该子数组的和最大。输出答案时，请分别返回第一个数字和最后一个数字的值。

如果多个答案，请返回其中任意一个。

样例

给定 [3, 1, -100, -3, 4], 返回 [4,0].

Source code:

```
class Solution {
public:
    /**
     * @param A an integer array
     * @return  A list of integers includes the index of
     *          the first number and the index of the last number
     */
     // 基本思想：两种可能 1：连续数组的最大和 ； 2 : total sum -
        连续数组的最小和
    vector<int> continuousSubarraySumII(vector<int>& A) {
        // Write your code here
        vector<int> result(2, 0);
        int len = A.size();
        if(len == 0){
            return result;
        }
        int totalSum = 0;
                int globalMax = numeric_limits<int>::min();
        int curr = 0;
        int first = 0, second = -1;
        for(int i = 0; i < A.size(); i++){
            totalSum += A[i];
            if(curr >= 0){
                curr += A[i];
                second = i;
            }
            else {
                curr = A[i];
                first = second = i;
            }
            if(curr >= globalMax){
                globalMax = curr;
                result[0] = first;
                result[1] = second;
            }
        }
        int globalMin = numeric_limits<int>::max();
        curr = 0;
        first = 0;
        for(int i = 0; i < A.size(); i++){
            if(curr <= 0){
                curr += A[i];
                second = i;
            }
            else {
                curr = A[i];
                first = second = i;
            }
            if (first == 0 && second == len-1) continue;
            if (totalSum - curr >= globalMax) {
                globalMax = totalSum - curr;
                result[0] = (second + 1) % len;
                result[1] = (first - 1 + len) % len;
            }
        }

         return result;
    }
};
```

401. 排序矩阵中的从小到大第k个数

http://www.lintcode.com/problem/kth-smallest-number-in-sorted-matrix

在一个排序矩阵中找从小到大的第 k 个整数。

排序矩阵的定义为：每一行递增，每一列也递增。

样例

给出 k = 4 和一个排序矩阵： [[1 ,5 ,7], [3 ,7 ,8], [4 ,8 ,9],] 返回 5。

Source code:

```
class MyPoint {
public:
    int x;
    int y;
    int val;
    MyPoint(int _x, int _y, int _val){
        this->x = _x;
        this->y = _y;
        this->val = _val;
    }
    bool operator ()(const MyPoint &a, const MyPoint &b) const{
        return a.val > b.val;
    }
    bool operator <(const MyPoint &obj) const{
        return this->val > obj.val;
    }
};
class Solution {
public:
    /**
     * @param matrix: a matrix of integers
     * @param k: an integer
     * @return: the kth smallest number in the matrix
     */
    int kthSmallest(vector<vector<int> > &matrix, int k) {
        // write your code here
        int count = 0;
        priority_queue<MyPoint> pq;
        int n = matrix.size();
        if(n == 0){
            return -1;
        }
        int m = matrix[0].size();
                for(int i = 0; i < n; i++){
            pq.push(MyPoint(i, 0, matrix[i][0]));
        }
        for(int j = 0; j < k-1; j++){
            MyPoint now = pq.top();
            pq.pop();
            int x = now.x;
            if(y+1 < m){
                pq.push(MyPoint(x, y+1, matrix[x][y+1]));
            }
        }
        MyPoint ans = pq.top();
        return ans.val;
    }
};
```

406. 和大于S的最小子数组

http://www.lintcode.com/problem/minimum-size-subarray-sum

给定一个由 n 个整数组成的数组和一个正整数 s ， 请找出该数组中满足其和 ≥

s 的最小长度子数组。如果无解，则返回 -1。

样例

给定数组 [2,3,1,2,4,3] 和 s = 7, 子数组 [4,3] 是该条件下的最小长度子数组。

Source code:

```
class Solution {
public:
    /**
     * @param nums: a vector of integers
     * @param s: an integer
     * @return: an integer representing the minimum size of subarray
     */
    //O(N) 两个指针
    int minimumSize(vector<int> &nums, int s) {
        // write your code here
        int n = nums.size();
        if(n == 0)
        {
            return -1;
        }
        int minlen = numeric_limits<int>::max();
        int i = 0, j = 0;
        int sum = 0;
        for(i = 0; i < n; i++)
        {
            while(j < n && sum < s)
            {
                sum += nums[j];
                j++;
            }
            if(sum >= s)
            {
                minlen = min(minlen, j- i);
            }
            sum -= nums[i];
        }
        return (minlen == numeric_limits<int>::max()) ? -1:minlen;
    }
        // prefix sum
    int minimumSize2(vector<int> &nums, int s) {
        int n = nums.size();
        if(n == 0)
        {
            return -1;
        }
        for(int i = 1; i <= n; i++){
            presum[i] += presum[i-1];
        }

                int minLen = numeric_limits<int>::max();
        for(int i = 0; i < n; i++){
            for(int j = i+1; j <= n; j++){
                int diff = presum[j] - presum[i];
                if(diff >= s){
                    minLen = min(minLen, j - i);
                    break;
                }
};
```

139. 最接近零的子数组和

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

给定一个整数数组，找到一个和最接近于零的子数组。返回第一个和最有一个指数。你的代码应该返回满足要求的子数组的起始位置和结束位置

样例

给出[-3, 1, 1, -3, 5]，返回[0, 2]，[1, 3]， [1, 1]， [2, 2] 或者 [0, 4]。

Source code:

```
bool mycompare(pair<int,int> a, pair<int, int> b){
    return a.second < b.second;
}
class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySumClosest(vector<int> nums){
        // write your code here
        vector<int> result;
        int n = nums.size();
        if(n == 0){
            return result;
        }
        vector<pair<int, int> > presum(n+1, pair<int, int>(0,0));
        for(int i = 1; i <=n; i++){
            presum[i].first  = i;
            presum[i].second = presum[i-1].second + nums[i-1];
        }
        sort(presum.begin(), presum.end(), mycompare);
                int start = 0, end = 1;
        int minDiff = numeric_limits<int>::max();
        for(int i = 1; i <= n; i++){
            if(presum[i].second - presum[i-1].second < minDiff){
                minDiff = presum[i].second - presum[i-1].second;
                start = presum[i-1].first;
                end = presum[i].first;
            }
        }
        if(start > end){
            //swap
            int tmp = start;
            start = end;
            end = tmp;
        }
        result = {start, end-1};


    };
```

75. 寻找峰值

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

你给出一个整数数组(size为n)，其具有以下特点：

相邻位置的数字是不同的 A[0] < A[1] 并且 A[n - 2] > A[n - 1] 假定P是峰值的位置则满足A[P] > A[P-1]且A[P] > A[P+1]，返回数组中任意一个峰值的位置。 注意事项 数组可能包含多个峰值，只需找到其中的任何一个即可

样例

给出数组[1, 2, 1, 3, 4, 5, 7, 6]返回1, 即数值 2 所在位置, 或者6, 即数值 7 所在位置.

Source code:

```
class Solution {
public:
    /**
     * @param A: An integers array.
     * @return: return any of peek positions.
     */
    // 1. 第一类面试者 只会 for 循环
    int findPeak1(vector<int> A) {
        // write your code here
        for(int i = 1; i < A.size()-1; i++){
            if(A[i] > A[i-1] && A[i] > A[i+1]){
                return i;
            }
        }
        return -1;
    }
    // 2. 第二类面试者 知道二分法
    int findPeak2(vector<int> A) {
        if(A.size() < 3){
            return -1;
        }
        int start = 1;
        int end = A.size()-1;
        while(start <= end){
            int middle = start + (end - start) / 2;
            if(A[middle] < A[middle-1]){
                end = middle-1;
            }
            else if(A[middle] < A[middle+1]){
                start = middle+1;
            }
            else {
                return middle;
            }
        }
        return -1;//should never happen
    }
    // 3. 第二类面试者 知道二分法(模板解法)
    int findPeak(vector<int> A) {
        if(A.size() < 3){
            return -1;
        int end = A.size()-2;
        while(start+1 < end){
            int middle = start + (end - start) / 2;
            if(A[middle] < A[middle-1]){
                end = middle;
            }
            else if(A[middle] < A[middle+1]){
                start = middle;
            }
            else {
                start = middle;
            }
        }
        if(A[start] > A[end]){
            return start;
        }
        return end;
    }
};
```

400. 最大间距

http://www.lintcode.com/problem/maximum-gap

给定一个未经排序的数组，请找出其排序表中连续两个要素的最大间距。

如果数组中的要素少于 2 个，请返回 0. 注意事项 可以假定数组中的所有要素都是非负整数，且最大不超过 32 位整数。

样例

给定数组 [1, 9, 2, 5]，其排序表为 [1, 2, 5, 9]，其最大的间距是在 5 和 9 之间，= 4.

Source code:

```
class Solution {
public:
    /**
     * @param nums: a vector of integers
     * @return: the maximum difference
     */
    int maximumGap(vector<int> nums) {
        // write your code here
        int n = nums.size();
        if(n < 2){
            return 0;
        }
        // 分桶法
    // 如果是平均分布，则桶的数目和元素的数目相同时，其排序的时间复杂度是0(n)
    // 我们假设桶的个数和元素的数目相同，若是平均分布，则每个桶里有一个数，而若某个桶
        里有两个以上的数时，这时必
        //有至少一个是空桶，那么最大间隔可能就落在空桶的相邻两个桶存储的数之间，最大
        间隔不会落在同一个桶的数里
        //，因此我们不需要对每个桶再排一次序，只需要记录同一个桶的最大值和最小值，算
        出前一个有最大值的桶和//个有最小值的桶之差，则可能是最大间隔
    //步骤：1.算好用的桶的个数，用最大元素和最小元素算出平均间隔，记录在平均间隔上的
```

```
        最大值和最小值，
        int maxValue = -1;
        int minValue = numeric_limits<int>::max();
        // 1. 算出用的桶数：取平均间隔，再用最大值和最小值之差除以间隔，得到桶数
        // 因为假设所有值都是平均分布的时候，如此取桶数可得时间复杂度是0(n)
        for(int val: nums){
            maxValue = max(maxValue, val);
            minValue = min(minValue, val);
        }
        int bucketSize = (maxValue - minValue) / n + 1;
        int bucketNum =  (maxValue - minValue) / bucketSize + 1;
        vector<pair<int, int> > buckets(bucketNum, make_pair(-1, -1));
        // 2. 记录每个桶的最大值和最小值
        for(int val: nums){
            int bucketId = (val - minValue) / bucketSize;
            pair<int, int> bucket = buckets[bucketId];
            if(buckets[bucketId].first == -1){
                buckets[bucketId].first = buckets[bucketId].second = val;
            }
            else {
                buckets[bucketId].first  = max(buckets[bucketId].first, val); //
                    保存最大值
                buckets[bucketId].second = min(buckets[bucketId].second, val); //
                    保存最小值
            }
        }
        // 3. 算出最大间隔
        int maxGap = 0;
        int lastNum = buckets[0].first;
        for(int id = 1; id < bucketNum; id++){
            if(buckets[id].first == -1){
                // Ignore empty bucket!
                continue;
            }
            maxGap = max(maxGap, buckets[id].second - lastNum);
            lastNum = buckets[id].first;
        }
        return maxGap;
    }
};
```