算法设计之分治算法

(1)53. 最大子序和

Description

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

Solution

可以使用贪心算法：数字能拿就拿，拿完和若为负数，则根据贪心思想前面拿的全部舍弃。时间是O（n）

但是本题重点是分治算法的使用，虽然它所花时间是 O(NlogN)，比贪心算法所花时间更多。

> 数组分成左右两部分，考虑三种情况：
> 答案数组完全落在左边。
> ……..完全落在右边。
> ……..落在两边。
> 前两者递归解决。
> 第三种为左边的一个后缀数组 + 右边的一个前缀数组。
> 所以问题转化为求数组的最大前缀/后缀数组。

```
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>
using namespace std;
```

```
class Solution {
private:
    int maxPreArray(vector<int> nums) { //最大前缀
        int ans = -INT_MAX, tmp = 0;
        for(int i = 0; i < nums.size(); i++){
            tmp += nums[i];
            ans = max(ans, tmp);
        }
        return ans;
    }
    int maxSufArray(vector<int> nums) { //最大后缀
        reverse(nums.begin(), nums.end());
        return maxPreArray(nums);
    }
public:
    int maxSubArray(vector<int>& nums) {
        if (nums.empty()) return -INT_MAX;
        if (nums.size() == 1) return nums[0];//这两个是递归结束

        int mid = nums.size() >> 1;
        vector<int> nums_left(nums.begin(), nums.begin() + mid);//把nums对
半分成两组
        vector<int> nums_right(nums.begin() + mid, nums.end());

        //这就是那三种情况，返回最大的值
        //要比较 a,b,c 三数大小，可以用 max( max(a,b), c )
        return max( max(maxSubArray(nums_left), maxSubArray(nums_right)),
 maxSufArray(nums_left) + maxPreArray(nums_right) );
    }
};

int main(){
    Solution solu;
    vector<int> nums={-2,1,-3,4,-1,2,1,-5,4};
    cout<<solu.maxSubArray(nums);
    return 0;
}
```

---

(2)169. 求众数

Description

给定一个大小为 n 的数组，找到其中的众数。众数是指在数组中出现次数大于 ⌊ n/2 ⌋ 的元素。你可以假设数组是非空的，并且给定的数组总是存在众数。

Solution

若和擂主同派系，则擂主势力++，否则擂主势力–；台上没人，则成为新的擂主。时间复杂度： O(n)

```
#include <iostream>
#include <vector>
#include <limits>
using namespace std;

class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int ans = nums[0];//擂主
        int cnt = 0;//擂主的势力
        for(int i=0; i<nums.size(); i++){
            // 打擂
            if(nums[i] == ans) cnt++;
            else cnt--;

            // 判断擂主属于谁
            if(cnt < 0) {
                ans = nums[i];
                cnt = 1;
            }
        }
        return ans;
    }
};

int main(){
    Solution solu;
    vector<int> nums={2,2,1,1,1,2,2};
    cout<<solu.majorityElement(nums);
    return 0;
}
```

(3)215. 数组中的第K个最大元素

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
```

```
        k = nums.size() - k;
        nth_element(nums.begin(), nums.begin() + k, nums.end());
        return nums[k];
    }
};

int main(){
    Solution solu;
    vector<int> nums = {3,2,3,1,2,4,5,5,5,6};
    int k = 4;
    cout<<solu.findKthLargest(nums,k);
    return 0;
}
```

(4)74. 搜索二维矩阵

```
class Solution {
    public:
        bool searchMatrix(vector<vector<int> >& matrix, int target) {
            if (matrix.empty() || matrix.begin()->empty()) return false;

            int n = matrix.size();
            int m = matrix[0].size();
            int col = m - 1;

            for (int row = 0; row < n; row++) {
                while (col>=0 && matrix[row][col]>target) col--;
                if (col < 0) return false;
                else if (matrix[row][col] == target) return true;
            }
            return false;
        }
};
```

不要把它当成一个二维数组来看，直接把它当成一个排好序的一维数组来对待：
注意坐标转换：matrix[i][j] <=> matrix[a] 其中a=i*n+j， i=a/n, j=a%n;

```
#include <iostream>
#include <vector>
using namespace std;

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix[0].empty()) return false;

        //二分查找目标值
```

```
        int m = matrix.size(), n = matrix[0].size(), l= 0, r = m*n - 1;
        while (l <= r){
            int mid = l + (r - l) / 2;
            if(target < matrix[mid/n][mid%n])  r = mid-1;
            else if(target > matrix[mid/n][mid%n]) l = mid+1;
            else return true;
        }
        return false;
    }
};

int main() {
    Solution solu;
    vector<vector<int>> matrix = {{1, 3}};
    cout<<solu.searchMatrix(matrix, 3);
    return 0;
}
```

---

(5)23. 合并K个排序链表

Description

合并 k 个排序链表，返回合并后的排序链表。

Input & Output

> 输入:
> [
> 1->4->5,
> 1->3->4,
> 2->6
>]
> 输出: 1->1->2->3->4->4->5->6

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
```

```
};

template <class T>
struct ListNode *CreatList(T &a){ //建立动态链表
    int ArrayLength =  sizeof(a)/sizeof(a[0]);
    if(ArrayLength==0) return NULL;

    struct ListNode *head;
    struct ListNode *p;
    struct ListNode *newnode;
    for(int i=0; i<ArrayLength; i++){
        newnode = (struct ListNode *)malloc( sizeof(struct ListNode) );
//开辟空间
        newnode->val = a[i];
        if(i==0) head = p = newnode; //记下首地址
        else p->next = newnode;
        p = newnode;
    }
    p->next = NULL;
    return head;
}


class Solution {
    private:
        ListNode * merge2Lists(ListNode * list1, ListNode * list2) {
            ListNode head(0);
            ListNode * tail = &head;
            while (list1 && list2) {
                if (list1->val > list2->val) {
                    swap(list1, list2);
                }
                tail->next = list1;
                tail = list1;
                list1 = list1->next;
            }
            if (list1) tail->next = list1;
            if (list2) tail->next = list2;
            return head.next;
        }
    public:
        ListNode* mergeKLists(vector<ListNode*>& lists) {
            if (lists.empty()) return NULL;
            if (lists.size() == 1) return lists[0];
            int mid = lists.size() + 1 >> 1;
            for (int i = mid; i < lists.size(); i++) {
                lists[i - mid] = merge2Lists(lists[i - mid], lists[i]);
            }
            lists.resize(mid);
            return mergeKLists(lists);
        }
};
```

```
void output(struct ListNode *head){
    struct ListNode *p;
    p = head;
    while(p != NULL){
        cout << p->val << " ";
        p = p->next;
    }
}

int main() {
    Solution solu;
    int a1[] = {1,4,5};
    int a2[] = {1,3,4};
    int a3[] = {2,6};
    vector<ListNode*> lists;
    lists.push_back(CreatList(a1));
    lists.push_back(CreatList(a2));
    lists.push_back(CreatList(a3));
    output(solu.mergeKLists(lists));
    return 0;
}
```

---

(6)493. 翻转对

Description

给定一个数组 nums ，如果 i < j 且 nums[i] > 2*nums[j] 我们就将 (i, j) 称作一个重要翻转对。你需要返回给定数组中的重要翻转对的数量。

Solution

实质是归并排序算法过程中计算对数

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
private:
    vector<int> merge(vector<int> left, vector<int> right) { //归并排序方法

        vector<int> ans;
```

```
            int i=0;
            int j=0;
            while(i<left.size() && j<right.size()) {
                if(left[i]<right[j])  ans.push_back(left[i++]);
                else  ans.push_back(right[j++]);
            }
            while(i<left.size()) ans.push_back(left[i++]);
            while(j<right.size()) ans.push_back(right[j++]);

            return ans;
        }
    public:
        int reversePairs(vector<int>& nums) {
            if (nums.size() < 2) return 0; //递归结束

            int mid = nums.size() >> 1; //对半分成左右两个
            vector<int> left(nums.begin(), nums.begin() + mid);
            vector<int> right(nums.begin() + mid, nums.end());

            int ans = reversePairs(left) + reversePairs(right); //递归

            int j=0;
            for(int i=0; i<left.size(); i++) {
                for(; j<right.size() && left[i]/2.0-right[j]>0; j++);
                ans += j;
            }
            nums = merge(left, right);//排序和归并
            return ans;
        }
    };

    int main(){
        Solution solu;
        vector<int> nums = {1,3,2,3,1};
        cout<<solu.reversePairs(nums);
        return 0;
    }
```

(7)4. 寻找两个有序数组的中位数

Description

给定两个大小为 m 和 n 的有序数组 nums1 和 nums2。请你找出这两个有序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。

你可以假设 nums1 和 nums2 不会同时为空。

Solution

模拟合并两个数组：每次选数组中较小的数，直选到中间元素(分数组长度奇偶情况)就是要找的中位数。算法复杂度为O(n1+n2)

解法中用到一个比较巧妙的方法：

```
nums1.push_back(INT_MAX);
nums2.push_back(INT_MAX);
```

通过在尾部追加最大值，有力的防止越界，让需复杂的情况判断简化了

```
#include<iostream>
#include<vector>
#include<limits>
using namespace std;

class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2)
 {
        int n1 = nums1.size(), n2 = nums2.size();
        nums1.push_back(INT_MAX);//防止越界，简化操作
        nums2.push_back(INT_MAX);
        int mid = n1+n2+2>>1, i=0, j=0;
        vector<int> nums;
        while(mid--) { //合并操作只需进行到一半就行
            nums1[i]<nums2[j] ? nums.push_back(nums1[i++]):nums.push_back
(nums2[j++]);
        }
        return (nums[n1+n2-1>>1]+nums[n1+n2>>1])/2.0;
    }
};

int main(){
    Solution solu;
    vector<int> nums1 = {1,2};
    vector<int> nums2 = {3,4};
    cout<<solu.findMedianSortedArrays(nums1,nums2);
    return 0;
}
```

---

(8)312. 戳气球

Description

有 n 个气球，编号为0 到 n-1，每个气球上都标有一个数字，这些数字存在数组 nums 中。

现在要求你戳破所有的气球。每当你戳破一个气球 i 时，你可以获得 nums[left] * nums[i] * nums[right] 个硬币。 这里的 left 和 right 代表和 i 相邻的两个气球的序号。注意当你戳破了气球 i 后，气球 left 和气球 right 就变成了相邻的气球。

求所能获得硬币的最大数量。

Solution

分治算法

为了让子体和整体有相同性质问题，我们逆向思考

从戳爆最后一个气球往前看，这个气球的位置可以把整个气球数组分成两部分。

需要注意的是本题二分法不在合适了

```
XXXXlOOOOiOOOrXXXXXXX //假设i是子体OOOOiOOO中最后戳爆的
     OOOOiOOO
XXXXl OOOOiOOO rXXXXXXX //那么戳爆i时应该这样算：l*i*r
     OOOO OOO //那么该子体再分割子体后，就有了......
```

dp[i][j]表示把第 i 个气球和第 j 个气球之间最大能得到的分值

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
    int maxCoins(vector<int>& nums) {
        nums.insert(nums.begin(),1);
        nums.push_back(1);
        int n=nums.size();

        vector<vector<int>> dp(n,vector<int>(n,0));
        for (int length=3; length<=n; length++) { //从3个一组，到4个4个一
组......到最后n个一组，不断更新dp数组
            for (int i=0; i+length<=n; i++) {
                int j = i+length-1;
                dp[i][j] = -INT_MAX;
                for (int k=i+1; k<j; k++) {
                    dp[i][j] = max(dp[i][j], dp[i][k] + dp[k][j] + nums
[k] * nums[i] * nums[j]);
                }
            }
        }

        return  dp[0][n-1];
    }
};
```

```
int main() {
    Solution solu;
    vector<int> nums = {3,1,5,8};
    cout<<solu.maxCoins(nums);
    return 0;
}
```

---

(9)241. 为运算表达式设计优先级

Description

给定一个含有数字和运算符的字符串，为表达式添加括号，改变其运算优先级以求出不同的结果。你需要给出所有可能的组合的结果。有效的运算符号包含 +, - 以及 * 。

Solution

```
分治算法：
    把整体分为两个个体解决，个体与整体要解决的方法一样，所以依照把个体再分为次个体
    所有可能的结果：  (..)   op1  (...........)          (......) op1 (.......)          (...........) op1 (..)
                  (.)op2(.) op1 (..)op2(.......)              .                              .
                (.)op2(.) op1 (.)op3(.)op2(..)op3(...)        .                              .
                   .                                          .                              .
                   .
                   .
```

```
#include <string>
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
    vector<int> diffWaysToCompute(string input) {
        vector<int> result;
        for (int i=0; i<input.length(); i++) {
            if( isdigit(input[i]) )  continue; //不是运算符

            auto v1 = diffWaysToCompute(input.substr(0, i)); //运算符input
[i]左边，所有组合的结果
            auto v2 = diffWaysToCompute(input.substr(i + 1)); //运算符inpu
t[i]右边，所有组合的结果
            for (auto e1 : v1) for (auto e2 : v2) switch (input[i]) {
                case '+' : result.push_back(e1 + e2); break;
                case '-' : result.push_back(e1 - e2); break;
                case '*' : result.push_back(e1 * e2); break;
            }
```

```
            }

            //result为空说明递归到底了，input只有数字字符
            if (result.empty())  result.push_back(stoi(input));
            return result;
        }
    };

    void output(vector<int> nums){
        for(auto e:nums)
            cout << e << endl;
    }

    int main() {
        Solution solu;
        string input = "2*3-4*5";
        output(solu.diffWaysToCompute(input));
        return 0;
    }
```

---

(10)282. 给表达式添加运算符

Description

给定一个仅包含数字 0-9 的字符串和一个目标值，在数字之间添加二元运算符（不是一元）+、- 或 * ，返回所有能够得到目标值的表达式。

Solution

```
分治算法，把整体问题划分为多个子问题
   例如：0000      0+子        0+0+子
                               0+0-子
                               0+0*子
                   0-子        0-0+子
                               0-0-子
                               0-0*子
           I       0*子        ....
```

```
    #include <stack>
    #include <vector>
    #include <iostream>
    #include <algorithm>
    using namespace std;

    class Solution {
```

```
private:
    void dfs(string& num, int& target, int pos, string& exp, int len, lon
g prev, long curr, vector<string> &result) {
        if (pos == num.length() && curr == target) {
            result.push_back(exp.substr(0, len));
            return;
        }

        long tmp = 0;
        int s = pos;
        int l = len;
        if (s != 0) ++len;
        while (pos < num.size()) {
            tmp = tmp*10 + (num[pos]-'0');
            if (num[s]=='0' && pos!=s) break; //不能出现00*0、1*05的情况
            if (tmp>INT_MAX) break; //n值太大
            exp[len++] = num[pos++];
            if (s == 0) {
                  dfs(num, target, pos, exp, len, tmp, tmp, result);
                  continue;
            }
            exp[l] = '+';
            dfs(num, target, pos, exp, len, tmp, curr+tmp, result);
            exp[l] = '-';
            dfs(num, target, pos, exp, len, -tmp, curr-tmp, result);
            exp[l] = '*';
            dfs(num, target, pos, exp, len, prev*tmp, curr-prev+prev*tmp,
 result);
        }
    }
public:
    vector<string> addOperators(string num, int target) {
        vector<string> result;
        string exp(num.length() * 2, '\0');
        dfs(num, target, 0, exp, 0, 0, 0, result);
        return result;
    }
};

void output(vector<string> str){
    for(auto e:str)
        cout << e << endl;
}

int main() {
    Solution solu;
    string nums = "000";
    int target = 0;
    output(solu.addOperators(nums, target));
    return 0;
}
```