# 贪心

## 455 分发饼干

class Solution {

public:

    int findContentChildren(vector<int>& g, vector<int>& s) {

        sort(g.begin(), g.end());

        sort(s.begin(), s.end());

        int id\_g = 0, id\_s = 0;

        while(id\_g < g.size() && id\_s < s.size()) {

            if(s[id\_s] >= g[id\_g]) {

                ++id\_s;

                ++id\_g;

            }

            else {

                ++id\_s;

            }

        }

        return id\_g;

    }

};

## 135 分发糖果

class Solution {

public:

    int candy(vector<int>& ratings) {

        int n = ratings.size();

        vector<int> candy(n, 1);

        for(int i=1; i<n; ++i) {

            if(ratings[i] > ratings[i-1]) {

                candy[i] = candy[i-1] + 1;

            }

        }

        for(int i=n-2; i>=0; --i) {

            if(ratings[i] > ratings[i+1]) {

                candy[i] = max(candy[i], candy[i+1] + 1);

            }

        }

        return accumulate(candy.begin(), candy.end(), 0);

    }

};

## 605 种花问题

这个题虽然简单，但是如果思考放花的条件，还是比较麻烦的。这时候想起了在验证二叉搜索树里面的一个小技巧。我们可以考虑问题的反面，什么时候不能放花呢？排除掉不能放的情况，当然就能放啦。代码如下：

class Solution {

public:

    bool canPlaceFlowers(vector<int>& flowerbed, int n) {

        int num = 0;

        for(int i=0; i<flowerbed.size(); ++i) {

            if(flowerbed[i] == 0) {

                if(i-1 >=0 && flowerbed[i-1] == 1) continue;

                if(i+1 < flowerbed.size() && flowerbed[i+1] == 1) continue;

                flowerbed[i] = 1;

                ++num;

            }

        }

        return num >= n;

    }

};

## 763 划分字母区间

这道题类似跳跃游戏，也是双指针类型的贪心问题

class Solution {

public:

    vector<int> partitionLabels(string S) {

        vector<int> res;

        unordered\_map<char, int> char\_to\_last;

        int n = S.size();

        for(int i=0; i<n; ++i) {

            char\_to\_last[S[i]] = i;

        }

        int start = 0, end = 0;

        for(int i=0; i<n; ++i) {

            end = max(end, char\_to\_last[S[i]]);

            if( i == end ) {

                res.push\_back(end-start+1);

                start = end + 1;

                end = end + 1;

            }

        }

        return res;

    }

};

## 406 根据身高重建队列

class Solution {

public:

    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {

        int n = people.size();

        sort(people.begin(), people.end(), [](const vector<int>& p1, const vector<int>& p2) {

            return p1[0] > p2[0] || (p1[0] == p2[0] && p1[1] < p2[1]);

        });

        vector<vector<int>> res;

        for(int i=0; i<n; ++i) {

            res.insert(res.begin() + people[i][1], people[i]);

        }

        return res;

    }

};

这个题理解起来还是有难度的。首先要明确插入顺序为先高后矮，因为矮个子的插入，不会影响前面高个子的属性。

用deque转一下，会更好

class Solution {

public:

    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {

        int n = people.size();

        sort(people.begin(), people.end(), [](const vector<int>& p1, const vector<int>& p2) {

            return p1[0] > p2[0] || (p1[0] == p2[0] && p1[1] < p2[1]);

        });

        deque<vector<int>> dq;

        for(int i=0; i<n; ++i) {

            dq.insert(dq.begin() + people[i][1], people[i]);

        }

        return vector<vector<int>>(dq.begin(), dq.end());

    }

};

## 665 非递减数列

一定要学会计算机的思考方式：寻找在当前状态下的局部最优处理

class Solution {

public:

    bool checkPossibility(vector<int>& nums) {

        int cnt = 0;

        for(int i=1; i<nums.size() && cnt < 2; ++i) {

            if(nums[i] >= nums[i-1]) continue;

            ++cnt;

            if(i-2 >= 0 && nums[i-2] > nums[i]) {

                nums[i] = nums[i-1];

            }

            else {

                nums[i-1] = nums[i];

            }

        }

        return cnt <= 1;

    }

};

# 双指针

## 88 合并两个有序数组

class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        int i = m-1, j=n-1, k=m+n-1;

        while(i>=0 && j>=0) nums1[k--] = nums1[i] > nums2[j] ? nums1[i--] : nums2[j--];

        while(i>=0) nums1[k--] = nums1[i--];

        while(j>=0) nums1[k--] = nums2[j--];

    }

};

## 633 平方数之和

class Solution {

public:

    bool judgeSquareSum(int c) {

        long long i = 0, j = sqrt(c);

        while(i <= j) {

            long long pow\_sum = i \* i + j \* j;

            if(pow\_sum == c) return true;

            else if(pow\_sum > c) --j;

            else ++i;

        }

        return false;

    }

};

类似两数之和的解法

## 680 验证回文字符串 Ⅱ

class Solution {

public:

    bool validPalindrome(string s) {

        int i=0, j=s.size()-1;

        while(i<j) {

            if(s[i] != s[j]) {

                return isValid(s, i+1, j) || isValid(s, i, j-1);

            }

            else {

                ++i; --j;

            }

        }

        return true;

    }

    bool isValid(const string& s, int i, int j) {

        while(i<j) {

            if(s[i++] != s[j--]) return false;

        }

        return true;

    }

};

双指针验证回文的一个变种版本。

## 524通过删除字母匹配到字典里最长单词

这道题正好可以用到二分查找来高效判定子序列

<https://mp.weixin.qq.com/s?__biz=MzAxODQxMDM0Mw==&mid=2247484479&idx=1&sn=31a3fc4aebab315e01ea510e482b186a&chksm=9bd7fa37aca0732103ca82e6f2cc23f475cf771696958456fc17d7662abb6b0879e8dfbaf7a1&scene=21#wechat_redirect>

这道题其实就是子序列

class Solution {

    bool isSubsequence(string s, string target){

        int i = 0, j = 0;

        int m = s.size(), n = target.size();

        while(i<m && j<n) {

            if(s[i] == target[j]) ++j;

            ++i;

        }

        return j==n;

    }

public:

    string findLongestWord(string s, vector<string>& d) {

        string res = "";

        for(const auto& target : d) {

            if(target.length() < res.length()) continue;

            if(target.length() == res.length() && res <= target) continue;

            if(isSubsequence(s, target)) {

                res = target;

            }

        }

        return res;

    }

};

二分查找版本：

class Solution {

    int left\_bound(vector<int>& nums, int tgt) {

        int lo = 0, hi = nums.size(), mid;

        while(lo < hi) {

            mid = lo + (hi-lo) / 2;

            if(nums[mid] < tgt) lo = mid + 1;

            else hi = mid;

        }

        return lo;

    }

    bool isSubsequence(string s, string target, vector<vector<int>>& char\_to\_index){

        int j = 0;

        for(int i=0; i<target.length(); ++i) {

            char c = target[i];

            if(char\_to\_index[c-'a'].empty()) return false;

            int pos = left\_bound(char\_to\_index[c-'a'], j);

            if(pos == char\_to\_index[c-'a'].size()) return false;

            j = char\_to\_index[c-'a'][pos] + 1;

        }

        return true;

    }

public:

    string findLongestWord(string s, vector<string>& d) {

        string res = "";

        // 预处理s

        vector<vector<int>> char\_to\_index(26);

        for(int i=0; i<s.length(); ++i) {

            char c = s[i];

            char\_to\_index[c-'a'].push\_back(i);

        }

        // 循环

        for(const auto& target : d) {

            if(res.length() > target.length()) continue;

            if(res.length() == target.length() && res <= target) continue;

            // 判断每个target是不是子序列，如果是更新结果

            if(isSubsequence(s, target, char\_to\_index)) res = target;

        }

        return res;

    }

};

挺考验代码能力的，需要慢慢写

从leetcode实际来看，确实是节省了时间。

## 340 至多包含 K 个不同字符的最长子串

典型的滑动窗口问题，套用模板即可

class Solution {

public:

    int lengthOfLongestSubstringKDistinct(string s, int k) {

        int left = 0, right = 0;

        unordered\_map<char, int> window;

        int res = 0;

        while(right < s.length()) {

            char c = s[right];

            ++right;

            ++window[c];

            while(window.size() > k) {

                char d = s[left];

                ++left;

                --window[d];

                if(window[d] == 0) {

                    window.erase(d);

                }

            }

            res = max(res, right - left);

        }

        return res;

    }

};

# 排序

## 347 前k个高频元素

桶排序解法：

class Solution {

public:

    vector<int> topKFrequent(vector<int>& nums, int k) {

        unordered\_map<int, int> counter;

        int max\_count;

        for(auto num : nums) {

            ++counter[num];

            max\_count = max(max\_count, counter[num]);

        }

        vector<vector<int>> buckets(max\_count + 1);

        for(const auto& p : counter) {

            buckets[p.second].push\_back(p.first);

        }

        vector<int> ans;

        for(int i=max\_count; i>=0 && ans.size() < k; --i) {

            for(auto num : buckets[i]) {

                ans.push\_back(num);

                if(ans.size() == k) break;

            }

        }

        return ans;

    }

};

优先级队列解法：

与上面差不多，但是使用小顶堆，把太小的从堆顶扔出去，堆里面剩下的就是大的

class Solution {

public:

    static bool comp(const pair<int,int>& a, const pair<int,int>& b) {

        return a.second > b.second;

    }

    vector<int> topKFrequent(vector<int>& nums, int k) {

        unordered\_map<int, int> counter;

        for(auto num : nums) {

            ++counter[num];

        }

        priority\_queue<pair<int, int>, vector<pair<int,int>>, decltype(&comp)> pq(comp);

        for(const auto& p : counter) {

            if(pq.size() == k) {

                if(pq.top().second < p.second) {

                    pq.pop();

                    pq.push(p);

                }

            }

            else {

                pq.emplace(p.first, p.second);

            }

        }

        vector<int> ans;

        while(!pq.empty()) {

            ans.push\_back(pq.top().first);

            pq.pop();

        }

        return ans;

    }

};

注意这里优先级队列的定义方式

## 451 根据字符出现频率排序

这里也可以用桶排序和优先级队列两种解法

优先级队列：

class Solution {

    static bool comp(const pair<char,int>& a, const pair<char,int>& b) {

        return a.second < b.second;

    }

public:

    string frequencySort(string s) {

        unordered\_map<char, int> counter;

        for(auto c : s) ++counter[c];

        priority\_queue<pair<char,int>, vector<pair<char,int>>, decltype(&comp)> pq(comp);

        for(const auto& p : counter) {

            pq.push(p);

        }

        string ans;

        while(!pq.empty()) {

            auto [ch, num] = pq.top();

            pq.pop();

            for(int i=0; i<num; ++i) {

                ans.push\_back(ch);

            }

        }

        return ans;

    }

};

## 75 颜色分类（荷兰国旗问题）

<https://leetcode-cn.com/problems/sort-colors/solution/yan-se-fen-lei-by-leetcode-solution/>

// 单指针两次循环解法，第一次换0，第二次换1

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int n = nums.size();

        int ptr = 0;

        for(int i=0; i<n; ++i) {

            if(nums[i]==0) {

                swap(nums[i], nums[ptr]);

                ++ptr;

            }

        }

        for(int i=ptr; i<n; ++i) {

            if(nums[i]==1) {

                swap(nums[i], nums[ptr]);

                ++ptr;

            }

        }

    }

};

// 双指针，一个找0， 一个找1。只需要一次循环

// 重点是要理解nums[i] == 0的情形，有疑惑去找官网

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int n = nums.size();

        int p0 = 0, p1 = 0;

        for(int i=0; i<n; ++i) {

            if(nums[i] == 1) {

                swap(nums[i], nums[p1]);

                ++p1;

            }

            if(nums[i] == 0) {

                swap(nums[i], nums[p0]);

                if(p0 < p1) swap(nums[p1], nums[i]);

                ++p0; ++p1;

            }

        }

    }

};

// 双指针，一个找0， 一个找2

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int n = nums.size();

        int p0 = 0, p2 = n-1;

        for(int i=0; i<=p2; ++i) {

            while(i<=p2 && nums[i] == 2) {

                swap(nums[p2], nums[i]);

                --p2;

            }

            if(nums[i] == 0) {

                swap(nums[p0], nums[i]);

                ++p0;

            }

        }

    }

};

# 搜索

## 695 岛屿的最大面积

栈

vector<int> dir{-1, 0, 1, 0, -1};

class Solution {

public:

    int maxAreaOfIsland(vector<vector<int>>& grid) {

        int m = grid.size(), n = m ? grid[0].size() : 0;

        int x, y;

        int local\_area = 0, global\_area = 0;

        for(int i=0; i<m; ++i) {

            for(int j=0; j<n; ++j) {

                if(grid[i][j] == 1) {

                    local\_area = 1;

                    grid[i][j] = 0;

                    stack<pair<int, int>> island;

                    island.push({i, j});

                    while(!island.empty()) {

                        auto [r, c] = island.top();

                        island.pop();

                        for(int i=0; i<4; ++i) {

                            x = r + dir[i];

                            y = c + dir[i+1];

                            if(x>=0 && x<m && y>=0 && y<n && grid[x][y]==1) {

                                grid[x][y] = 0;

                                local\_area++;

                                island.push({x,y});

                            }

                        }

                    }

                    global\_area = max(global\_area, local\_area);

                }

            }

        }

        return global\_area;

    }

};

递归一：