有趣的排序思想

```cpp
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<queue>
#include<stack>
#include<algorithm>
#include<string>
#include<map>
#include<set>
#include<vector>
using namespace std;

#define low16(a) ((a) & 0xffff)
#define __high16(a) ((a & 0xffff0000) >> 16)
#define high16(a) (__high16(a) > 32767 ? (__high16(a) - 32768) : (__high16(a) + 32768))

void radix_sort(vector<int> &arr, int n) {
    vector<int> cnt(65536, 0), temp(n, 0);
    // low 16bit sort
    for(int i = 0; i < n; i++) cnt[low16(arr[i])] += 1;
    for(int i = 1; i < 65536; i++) cnt[i] += cnt[i - 1];
    for(int i = n - 1; i >= 0; i--) temp[--cnt[low16(arr[i])]] = arr[i];

    for(int i = 0; i < 65536; i++) cnt[i] = 0;

    // high 16bit sort
    for(int i = 0; i < n; i++) cnt[high16(arr[i])] += 1;
    for(int i = 1; i < 65536; i++) cnt[i] += cnt[i - 1];
    for(int i = n - 1; i >= 0; i--) arr[--cnt[high16(temp[i])]] = temp[i];
    return ;
}

void getRandData(vector<int> &arr, int n) {
    for(int i = 0; i < n; i++) {
        // arr[i] = rand() % 100;
        arr[i] = (rand() % 2 ? -1 : 1) * (rand() % 100);
    }
    return ;
}

void output(vector<int> &arr, int n) {
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
```

```
    cout << endl;
    return ;
}

int main() {
    int n = 20;
    vector<int> arr(n);
    getRandData(arr, n);
    radix_sort(arr, n);
    output(arr, n);
    return 0;
}
```

164. 最大间距

给定一个无序的数组 `nums`，返回 *数组在排序之后，相邻元素之间最大的差值* 。如果数组元素个数小于 2，则返回 `0` 。

您必须编写一个在「线性时间」内运行并使用「线性额外空间」的算法。

示例：

```
输入: nums = [3,6,9,1]
输出: 3
解释: 排序后的数组是 [1,3,6,9], 其中相邻元素 (3,6) 和 (6,9) 之间都存在最大差值 3。
```

```
class Solution {
public:
    int maximumGap(vector<int>& nums) {
        int cnt[65536] = {0};
        vector<int> temp(nums.size());
        for(int i = 0; i < nums.size(); i++) {
            cnt[nums[i] % 65536]++;
        }
        for(int i = 1; i < 65536; i++) {
            cnt[i] += cnt[i - 1];
        }
        for(int i = nums.size() - 1; i >= 0; i--) {
            temp[--cnt[nums[i] % 65536]] = nums[i];
        }
        memset(cnt, 0, sizeof(cnt));
        for(int i = 0; i < temp.size(); i++) {
            cnt[temp[i] / 65536]++;
        }
        for(int i = 1; i < 65536; i++) {
            cnt[i] += cnt[i - 1];
        }
        for(int i = nums.size() - 1; i >= 0; i--) {
            nums[--cnt[temp[i] / 65536]] = temp[i];
```

```
        }

        int ans = 0;
        for(int i = 1; i < nums.size(); i++) {
            ans = max(ans, nums[i] - nums[i - 1]);
        }
        return ans;
    }
};

```

207. 课程表

你这个学期必须选修 `numCourses` 门课程，记为 `0` 到 `numCourses - 1` 。

在选修某些课程之前需要一些先修课程。 先修课程按数组 `prerequisites` 给出，其中 `prerequisites[i] = [ai, bi]` ，表示如果要学习课程 `ai` 则 **必须** 先学习课程 `bi` 。

- 例如，先修课程对 `[0, 1]` 表示：想要学习课程 `0` ，你需要先完成课程 `1` 。

请你判断是否可能完成所有课程的学习？如果可以，返回 `true` ；否则，返回 `false` 。

示例：

```
输入：numCourses = 2, prerequisites = [[1,0]]
输出：true
解释：总共有 2 门课程。学习课程 1 之前，你需要完成课程 0 。这是可能的。
```

```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        queue<int> q;
        vector<vector<int>> g(numCourses);
        vector<int> indeg(numCourses);
        for(auto x : prerequisites) {
            indeg[x[0]]++;
            g[x[1]].push_back(x[0]);
        }
        for(int i = 0; i < numCourses; i++) {
            if(indeg[i] == 0) q.push(i);
        }
        int ans = 0;
        while(!q.empty()) {
            int ind = q.front();
            q.pop();
            ans++;
            for(auto to : g[ind]) {
                indeg[to]--;
                if(indeg[to] == 0) q.push(to);
            }
```

```
        }
        return ans == numCourses;
    }
};
```

210. 课程表 II

现在你总共有 `numCourses` 门课需要选，记为 `0` 到 `numCourses - 1`。给你一个数组 `prerequisites` ，其中 `prerequisites[i] = [ai, bi]` ，表示在选修课程 `ai` 前 **必须** 先选修 `bi` 。

- 例如，想要学习课程 `0` ，你需要先完成课程 `1` ，我们用一个匹配来表示：`[0,1]` 。

返回你为了学完所有课程所安排的学习顺序。可能会有多个正确的顺序，你只要返回 **任意一种** 就可以了。如果不可能完成所有课程，返回 **一个空数组** 。

示例：

```
输入：numCourses = 2, prerequisites = [[1,0]]
输出：[0,1]
解释：总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1] 。
```

```cpp
class Solution {
public:
    vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {
        queue<int> q;
        vector<vector<int>> g(numCourses); // 邻接表存图
        vector<int> indeg(numCourses); // 存每一个点的入度
        for(auto x : prerequisites) {
            indeg[x[0]]++;
            g[x[1]].push_back(x[0]); // x1 -> x0
        }
        for(int i = 0; i < numCourses; i++) {
            if(indeg[i] == 0) q.push(i); // 入度为0的点入队列
        }

        vector<int> ans;
        while(!q.empty()) {
            int ind = q.front();
            // cout << "pop : " << ind << endl;
            ans.push_back(ind);
            q.pop();
            for(auto to : g[ind]) {
                indeg[to]--;
                if(indeg[to] == 0) q.push(to);
            }
        }
        if(ans.size() != numCourses) ans.clear();
        return ans;
    }
```

```
};
```

1122. 数组的相对排序

给你两个数组，`arr1` 和 `arr2`，`arr2` 中的元素各不相同，`arr2` 中的每个元素都出现在 `arr1` 中。

对 `arr1` 中的元素进行排序，使 `arr1` 中项的相对顺序和 `arr2` 中的相对顺序相同。未在 `arr2` 中出现过的元素需要按照升序放在 `arr1` 的末尾。

示例：

```
输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
输出：[2,2,2,1,4,3,3,9,6,7,19]
```

```
class Solution {
public:
    vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
        vector<int> ret(1010);
        for(int i = 0; i < arr1.size(); i++) {
            ret[arr1[i]]++;
        }
        int k = 0;
        for(int i = 0; i < arr2.size(); i++) {
            while(ret[arr2[i]]--) {
                arr1[k++] = arr2[i];
            }
        }
        for(int i = 0; i < 1001; i++) {
            if(ret[i] <= 0) continue;
            while(ret[i]--) {
                arr1[k++] = i;
            }
        }
        return arr1;
    }
};
```

274. H 指数

给你一个整数数组 `citations` ，其中 `citations[i]` 表示研究者的第 `i` 篇论文被引用的次数。计算并返回该研究者的 **`h` 指数**。

根据维基百科上 h 指数的定义：h 代表“高引用次数”，一名科研人员的 **`h` 指数**是指他（她）的 （`n` 篇论文中）**总共**有 `h` 篇论文分别被引用了**至少** `h` 次。且其余的 *`n` - `h`* 篇论文每篇被引用次数 **不超过** *`h`* 次。

如果 `h` 有多种可能的值，**`h` 指数** 是其中最大的那个。

示例：

```
输入：citations = [3,0,6,1,5]
输出：3
解释：给定数组表示研究者总共有 5 篇论文，每篇论文相应的被引用了 3, 0, 6, 1, 5 次。
     由于研究者有 3 篇论文每篇 至少 被引用了 3 次，其余两篇论文每篇被引用 不多于 3 次，所以她的 h 指数是 3。
```

```cpp
class Solution {
public:
    int hIndex(vector<int>& arr) {
        sort(arr.begin(), arr.end());
        for(int i = 0; i < arr.size(); i++) {
            if(arr[i] >= arr.size() - i) {
                return arr.size() - i;
            }
        }
        return 0;
    }
};
```

1288. 删除被覆盖区间

给你一个区间列表，请你删除列表中被其他区间所覆盖的区间。

只有当 `c <= a` 且 `b <= d` 时，我们才认为区间 `[a,b)` 被区间 `[c,d)` 覆盖。

在完成所有删除操作后，请你返回列表中剩余区间的数目。

示例：

```
输入：intervals = [[1,4],[3,6],[2,8]]
输出：2
解释：区间 [3,6] 被区间 [2,8] 覆盖，所以它被删除了。
```

```cpp
class Solution {
public:
    static bool cmp(const vector<int> &a, const vector<int> &b) {
        if(a[0] == b[0]) return a[1] > b[1];
        return a[0] < b[0];
    }
    int removeCoveredIntervals(vector<vector<int>>& intervals) {
        int ans = intervals.size();
        sort(intervals.begin(), intervals.end(), cmp);

        int cur_j = intervals[0][1];
        for(int i = 1; i < intervals.size(); i++) {
            if(cur_j >= intervals[i][1]) {
                ans--;
            }
```

```
            cur_j = max(cur_j, intervals[i][1]);
        }
        return ans;
    }
};
```

56. 合并区间

以数组 `intervals` 表示若干个区间的集合，其中单个区间为 `intervals[i] = [starti, endi]` 。请你合并所有重叠的区间，并返回 *一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间* 。

示例：

```
输入：intervals = [[1,3],[2,6],[8,10],[15,18]]
输出：[[1,6],[8,10],[15,18]]
解释：区间 [1,3] 和 [2,6] 重叠，将它们合并为 [1,6].
```

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        sort(intervals.begin(), intervals.end());
        vector<vector<int>> ret;
        int cur_i = intervals[0][0];
        int cur_j = intervals[0][1];
        for(int i = 0; i < intervals.size(); i++) {
            if(cur_j >= intervals[i][0]) {
                cur_j = max(cur_j, intervals[i][1]);
            } else {
                ret.push_back({cur_i, cur_j});
                cur_i = intervals[i][0];
                cur_j = intervals[i][1];
            }
        }
        ret.push_back({cur_i, cur_j});
        return ret;
    }
};
```

1094. 拼车

车上最初有 `capacity` 个空座位。车 **只能** 向一个方向行驶（也就是说，**不允许掉头或改变方向**）

给定整数 `capacity` 和一个数组 `trips` , `trip[i] = [numPassengersi, fromi, toi]` 表示第 `i` 次旅行有 `numPassengersi` 乘客，接他们和放他们的位置分别是 `fromi` 和 `toi` 。这些位置是从汽车的初始位置向东的公里数。

当且仅当你可以在所有给定的行程中接送所有乘客时，返回 `true`，否则请返回 `false`。

示例：

```
输入：trips = [[2,1,5],[3,3,7]], capacity = 4
输出：false
```

```cpp
class Solution {
public:
    bool carPooling(vector<vector<int>>& trips, int capacity) {
        map<int, int> mp;
        for(int i = 0; i < trips.size(); i++) {
            mp[trips[i][1]] += trips[i][0];
            mp[trips[i][2]] -= trips[i][0];
        }
        int ans = 0;
        for(auto it : mp) {
            ans += it.second;
            if(ans > capacity) return false;
        }
        return true;
    }
};
```

491. 递增子序列

给你一个整数数组 `nums` ，找出并返回所有该数组中不同的递增子序列，递增子序列中 **至少有两个元素** 。你可以按 **任意顺序** 返回答案。

数组中可能含有重复元素，如出现两个整数相等，也可以视作递增序列的一种特殊情况。

示例：

```
输入：nums = [4,6,7,7]
输出：[[4,6],[4,6,7],[4,6,7,7],[4,7],[4,7,7],[6,7],[6,7,7],[7,7]]
```

```cpp
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> temp;

    void dfs(int cur, vector<int> &nums) {
        if(cur == nums.size()) {
            if(temp.size() >= 2) ans.push_back(temp);
            return ;
        }

        if(temp.size() == 0 || nums[cur] >= temp.back()) {
            temp.push_back(nums[cur]);
            dfs(cur + 1, nums);
            temp.pop_back();
        }
```

```
        if(temp.size() == 0 || nums[cur] != temp.back()) {
            dfs(cur + 1, nums);
        }
        return ;
    }

    vector<vector<int>> findSubsequences(vector<int>& nums) {
        dfs(0, nums);
        return ans;
    }
};
```

面试题 04.12. 求和路径

给定一棵二叉树，其中每个节点都含有一个整数数值(该值或正或负)。设计一个算法，打印节点数值总和等于某个给定值的所有路径的数量。注意，路径不一定非得从二叉树的根节点或叶节点开始或结束，但是其方向必须向下(只能从父节点指向子节点方向)。

示例:
给定如下二叉树，以及目标和 `sum = 22`，

```
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
```

返回:

```
3
解释：和为 22 的路径有：[5,4,11,2], [5,8,4,5], [4,11,7]
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int dfs(TreeNode *root, int sum) {
        if(root == NULL) return 0;
```

```
        sum -= root->val;
        return (sum == 0) + dfs(root->left, sum) + dfs(root->right, sum);
    }
    int pathSum(TreeNode* root, int sum) {
        if(root == NULL) return 0;
        int a = pathSum(root->left, sum);
        int b = pathSum(root->right, sum);
        return a + b + dfs(root, sum);
    }
};
```