1200. 最小绝对差（简单）

1. 题目描述

   给你个整数数组 `arr`，其中每个元素都 **不相同。**

   请你找到所有具有最小绝对差的元素对，并且按升序的顺序返回。

   示例 1:

   ```
   输入: arr = [4,2,1,3]
   输出: [[1,2],[2,3],[3,4]]
   ```

   示例 2:

   ```
   输入: arr = [1,3,6,10,15]
   输出: [[1,3]]
   ```

   示例 3:

   ```
   输入: arr = [3,8,-10,23,19,-4,-14,27]
   输出: [[-14,-10],[19,23],[23,27]]
   ```

   提示:

   - `2 <= arr.length <= 10^5`
   - `-10^6 <= arr[i] <= 10^6`

2. 简单实现

```cpp
class Solution {
public:
    vector<vector<int>> minimumAbsDifference(vector<int>& arr) {
        sort(arr.begin(), arr.end());
        vector<vector<int>> ans;
        int len = arr.size();
        int min_diff = arr[len-1] - arr[0];
        for(int i = 1; i < len; i++)
            min_diff = min(min_diff, arr[i] - arr[i-1]);
        for(int i = 1; i < len; i++)
            if(arr[i] - arr[i-1] == min_diff)
                ans.push_back({arr[i-1], arr[i]});
        return ans;
    }
};
```

1201. 丑数 III（中等）

1. 题目描述

请你帮忙设计一个程序，用来找出第 `n` 个丑数。

丑数是可以被 `a` **或** `b` **或** `c` 整除的 **正整数**。

示例 1：

```
输入：n = 3, a = 2, b = 3, c = 5
输出：4
解释：丑数序列为 2, 3, 4, 5, 6, 8, 9, 10... 其中第 3 个是 4。
```

示例 2：

```
输入：n = 4, a = 2, b = 3, c = 4
输出：6
解释：丑数序列为 2, 3, 4, 6, 8, 9, 12... 其中第 4 个是 6。
```

示例 3：

```
输入：n = 5, a = 2, b = 11, c = 13
输出：10
解释：丑数序列为 2, 4, 6, 8, 10, 11, 12, 13... 其中第 5 个是 10。
```

示例 4：

```
输入：n = 1000000000, a = 2, b = 217983653, c = 336916467
输出：1999999984
```

提示：

- `1 <= n, a, b, c <= 10^9`
- `1 <= a * b * c <= 10^18`
- 本题结果在 `[1, 2 * 10^9]` 的范围内

2. 简单实现

二分法，划分条件为：求小于等于mid的整数中有几个丑数

```
class Solution {
public:
    int gcd(int a, int b){
        if(b > a)
            swap(a, b);
        return b == 0 ? a : gcd(b, a%b);
    }
    int nthUglyNumber(int n, long a, long b, long c) {
        int l = 1, r = 2e9;
        while(l < r){
            long mid = l + (r - l) / 2;
            //类似于求A，B，C的并集，注意交集部分是最小公倍数，即乘积再除以最大公约数
            long cnt = mid / a + mid / b + mid / c;
```

```
            cnt -= mid / (a*b/gcd(a,b)) + mid / (a*c/gcd(a,c)) + mid /
(b*c/gcd(b,c));
            long mult = a * b / gcd(a,b);
            mult = mult * c / gcd(mult, c);
            cnt += mid / mult;
            if(cnt >= n)
                r = mid;
            else if(cnt < n)
                l = mid + 1;
        }
        return l;
    }
};
```

1202. 交换字符串中的元素（中等）

1. 题目描述

给你一个字符串 `s`，以及该字符串中的一些「索引对」数组 `pairs`，其中 `pairs[i] = [a, b]` 表示字符串中的两个索引（编号从 0 开始）。

你可以 **任意多次交换** 在 `pairs` 中任意一对索引处的字符。

返回在经过若干次交换后，`s` 可以变成的按字典序最小的字符串。

示例 1:

```
输入: s = "dcab", pairs = [[0,3],[1,2]]
输出: "bacd"
解释:
交换 s[0] 和 s[3], s = "bcad"
交换 s[1] 和 s[2], s = "bacd"
```

示例 2:

```
输入: s = "dcab", pairs = [[0,3],[1,2],[0,2]]
输出: "abcd"
解释:
交换 s[0] 和 s[3], s = "bcad"
交换 s[0] 和 s[2], s = "acbd"
交换 s[1] 和 s[2], s = "abcd"
```

示例 3:

```
输入: s = "cba", pairs = [[0,1],[1,2]]
输出: "abc"
解释:
交换 s[0] 和 s[1], s = "bca"
交换 s[1] 和 s[2], s = "bac"
交换 s[0] 和 s[1], s = "abc"
```

提示:

- `1 <= s.length <= 10^5`
- `0 <= pairs.length <= 10^5`
- `0 <= pairs[i][0], pairs[i][1] < s.length`
- `s` 中只含有小写英文字母

2. 简单实现

并查集

```
class Solution {
public:
    vector<int> father;
    int find(int i){
        while(i != father[i]){
            father[i] = father[father[i]];
            i = father[i];
        }
        return i;
    }
    void join(int i, int j){
        i = find(i);
        j = find(j);
        if(i != j)
            father[i] = j;
    }
    string smallestStringWithSwaps(string s, vector<vector<int>>& pairs) {
        int len = s.size();
        father = vector<int>(len);
        for(int i = 0; i < len; i++)
            father[i] = i;
        for(int i = 0; i < pairs.size(); i++)
            join(pairs[i][0], pairs[i][1]);

        unordered_map<int, string> m;//记录每个单独集合下的所有字符
        for(int i = 0; i < len; i++){
            int id = find(i);
            father[i] = id;//确保树的高度为1
            if(m.count(id) == 0)
                m[id] = s[i];
            else
                m[id] += s[i];
        }

        for(auto it = m.begin(); it != m.end(); it++)
            sort((it->second).rbegin(), (it->second).rend());//从大到小排序

        for(int i = 0; i < len; i++){
            s[i] = m[father[i]].back();
            m[father[i]].pop_back();//消除最后的比消除开头的省时间
        }
        return s;
    }
};
```

1203. 项目管理（困难）

1. 题目描述

   公司共有 `n` 个项目和 `m` 个小组，每个项目要不没有归属，要不就由其中的一个小组负责。

   我们用 `group[i]` 代表第 `i` 个项目所属的小组，如果这个项目目前无人接手，那么 `group[i]` 就等于 `-1`。（项目和小组都是从零开始编号的）

   请你帮忙按要求安排这些项目的进度，并返回排序后的项目列表：

   - 同一小组的项目，排序后在列表中彼此相邻。
   - 项目之间存在一定的依赖关系，我们用一个列表 `beforeItems` 来表示，其中 `beforeItems[i]` 表示在进行第 `i` 个项目前（位于第 `i` 个项目左侧）应该完成的所有项目。

   结果要求：

   如果存在多个解决方案，只需要返回其中任意一个即可。

   如果没有合适的解决方案，就请返回一个 **空列表**。

   示例 1：

Item	Group	Before
0	-1	
1	-1	6
2	1	5
3	0	6
4	0	3, 6
5	1	
6	0	
7	-1	

```
输入：n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],
[3,6],[],[],[]]
输出：[6,3,4,1,5,2,0,7]
```

   示例 2：

```
输入：n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3],
[],[4],[]]
输出：[]
解释：与示例 1 大致相同，但是在排序后的列表中，4 必须放在 6 的前面。
```

   提示：

   - `1 <= m <= n <= 3*10^4`
   - `group.length == beforeItems.length == n`
   - `-1 <= group[i] <= m-1`
   - `0 <= beforeItems[i].length <= n-1`
   - `0 <= beforeItems[i][j] <= n-1`
   - `i != beforeItems[i][j]`

2. 正确解法——双层拓扑排序

- 预处理：若group ID为-1，则为其赋予一个新的group ID
- 基于项目的组间依赖关系，对group ID进行拓扑排序，若排序不成功，直接退出
- 基于项目的组内依赖关系，对上一步结果中的每一个group内的项目进行拓扑排序，若排序不成功，直接退出
- 由于group和每个group内的项目都是可进行排序的实体，则可以构造解

```
class Solution {
public:
    vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>&
beforeItems) {
        //预处理
        int maxGroup = m;
        for(int i = 0;i < group.size(); ++i){
            if(group[i] == -1){
                group[i] = maxGroup++;
            }
        }

        vector<set<int>> groupItem(maxGroup);//记录各组要完成的工作id
        vector<int> groupIndegree(maxGroup, 0);//各组的入度
        vector<set<int>> groupGraph(maxGroup);//各组之间拓扑图
        vector<int> itemIndegree(n, 0);//各个工作的入度
        vector<set<int>> itemGraph(n);//各工作之间拓扑图
        queue<int> qu;
        for(int i = 0;i < group.size(); ++i)
            groupItem[group[i]].insert(i);
        for(int i = 0;i < beforeItems.size(); ++i){
            for(auto it : beforeItems[i]){//i依赖it
                if(group[i] == group[it]){//组内依赖
                    itemIndegree[i]++;
                    itemGraph[it].insert(i);
                }else{//组间依赖
                    if(!groupGraph[group[it]].count(group[i])){//i不依赖于某个已经依赖
于it的作业
                        groupIndegree[group[i]]++;
                        groupGraph[group[it]].insert(group[i]);
                    }
                }
            }
        }

        //组件拓扑排序
        vector<int> ans;//排序结果
        for(int i = 0;i < maxGroup; ++i)
            if(groupIndegree[i] == 0)
                qu.push(i);

        while(!qu.empty()){
            int curr = qu.front();
            qu.pop();
            ans.push_back(curr);
            for(auto neg : groupGraph[curr]){
                groupIndegree[neg]--;
```

```
                if(groupIndegree[neg] == 0)
                    qu.push(neg);
            }
        }
        if(ans.size() != maxGroup)//组件冲突
            return vector<int>();

        //组内拓扑排序
        vector<int> res;//最终答案
        for(int i = 0;i < ans.size(); ++i){//各组内进行拓扑排序
            for(auto it : groupItem[ans[i]])
                if(itemIndegree[it] == 0)
                    qu.push(it);

            int count = 0;
            while(!qu.empty()){
                int curr = qu.front();
                count++;
                qu.pop();
                res.push_back(curr);
                for(auto neg : itemGraph[curr]){
                    itemIndegree[neg]--;
                    if(itemIndegree[neg] == 0)
                        qu.push(neg);
                }
            }

            if(count != groupItem[ans[i]].size())//组内冲突
                return vector<int>();
        }
        return res;
    }
};
```