总结

1. 本次周赛比较简单，五十分钟完成，排了39/3079，应该是厉害的人都放假玩去了吧2333.

1436. 旅行终点站（简单）

1. 题目描述

   给你一份旅游线路图，该线路图中的旅行线路用数组 `paths` 表示，其中 `paths[i] = [cityAi, cityBi]` 表示该线路将会从 `cityAi` 直接前往 `cityBi` 。请你找出这次旅行的终点站，即没有任何可以通往其他城市的线路的城市。

   题目数据保证线路图会形成一条不存在循环的线路，因此只会有一个旅行终点站。

   示例 1：

   ```
   输入：paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]
   输出："Sao Paulo"
   解释：从 "London" 出发，最后抵达终点站 "Sao Paulo" 。本次旅行的路线是 "London" -> "New
   York" -> "Lima" -> "Sao Paulo" 。
   ```

   示例 2：

   ```
   输入：paths = [["B","C"],["D","B"],["C","A"]]
   输出："A"
   解释：所有可能的线路是：
   "D" -> "B" -> "C" -> "A".
   "B" -> "C" -> "A".
   "C" -> "A".
   "A".
   显然，旅行终点站是 "A" 。
   ```

   示例 3：

   ```
   输入：paths = [["A","Z"]]
   输出："Z"
   ```

   提示：

   - `1 <= paths.length <= 100`
   - `paths[i].length == 2`
   - `1 <= cityAi.length, cityBi.length <= 10`
   - `cityAi != cityBi`
   - 所有字符串均由大小写英文字母和空格字符组成。

2. 比赛实现

   每一对`[cityAi, cityBi]`中，cityAi是起点, cityBi是终点，则出现在终点的集合里，却不出现在起点的集合里的那个城市，就是终点站

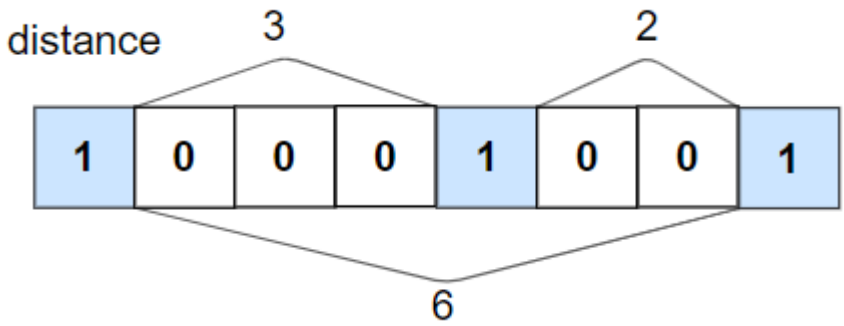
```cpp
class Solution {
public:
    string destCity(vector<vector<string>>& paths) {
        unordered_set<string> ans;
        for(int i = 0; i < paths.size(); i++)
            ans.insert(paths[i][1]);
        for(int i = 0; i < paths.size(); i++)
            if(ans.find(paths[i][0]) != ans.end())
                ans.erase(paths[i][0]);
        return *(ans.begin());
    }
};
```

1437. 是否所有1都至少相隔k个元素（中等）

1. 题目描述

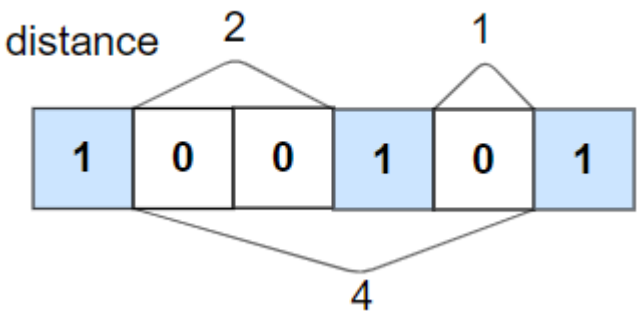
给你一个由若干 `0` 和 `1` 组成的数组 `nums` 以及整数 `k`。如果所有 `1` 都至少相隔 `k` 个元素，则返回 `True` ；否则，返回 `False` 。

示例 1：




```
输入：nums = [1,0,0,0,1,0,0,1], k = 2
输出：true
解释：每个 1 都至少相隔 2 个元素。
```

示例 2：



```
输入：nums = [1,0,0,1,0,1], k = 2
输出：false
解释：第二个 1 和第三个 1 之间只隔了 1 个元素。
```

示例 3：

```
输入：nums = [1,1,1,1,1], k = 0
输出：true
```

示例 4：

```
输入：nums = [0,1,0,1], k = 1
输出：true
```

提示：

- `1 <= nums.length <= 10^5`
- `0 <= k <= nums.length`
- `nums[i]` 的值为 `0` 或 `1`

2. 比赛实现

一次遍历滑动窗口

```cpp
class Solution {
public:
    bool kLengthApart(vector<int>& nums, int k) {
        int len = nums.size();
        int l = 0;
        while(l < len && nums[l] != 1) l++;//第一个1
        int r = l + 1;
        while(r < len){
            while(r < len && nums[r] != 1) r++;//下一个1
            if(r >= len) return true;//不存在下一个1，结束了
            if(r - l - 1 < k) return false;//间隔不足k
            l = r;
            r++;
        }
        return true;
    }
};
```

1438. 绝对差不超过限制的最长连续子数组（中等）

1. 题目描述

给你一个整数数组 `nums` ，和一个表示限制的整数 `limit`，请你返回最长连续子数组的长度，该子数组中的任意两个元素之间的绝对差必须小于或者等于 `limit` 。

如果不存在满足条件的子数组，则返回 `0` 。

示例 1：

```
输入：nums = [8,2,4,7], limit = 4
输出：2
解释：所有子数组如下：
[8] 最大绝对差 |8-8| = 0 <= 4.
[8,2] 最大绝对差 |8-2| = 6 > 4.
[8,2,4] 最大绝对差 |8-2| = 6 > 4.
[8,2,4,7] 最大绝对差 |8-2| = 6 > 4.
[2] 最大绝对差 |2-2| = 0 <= 4.
[2,4] 最大绝对差 |2-4| = 2 <= 4.
[2,4,7] 最大绝对差 |2-7| = 5 > 4.
[4] 最大绝对差 |4-4| = 0 <= 4.
[4,7] 最大绝对差 |4-7| = 3 <= 4.
[7] 最大绝对差 |7-7| = 0 <= 4.
因此，满足题意的最长子数组的长度为 2 。
```

示例 2：

```
输入：nums = [10,1,2,4,7,2], limit = 5
输出：4
解释：满足题意的最长子数组是 [2,4,7,2]，其最大绝对差 |2-7| = 5 <= 5 。
```

示例 3：

```
输入：nums = [4,2,2,2,4,4,2,2], limit = 0
输出：3
```

提示：

- `1 <= nums.length <= 10^5`
- `1 <= nums[i] <= 10^9`
- `0 <= limit <= 10^9`

2. 比赛实现

按题意和示例1，貌似不会出现不满足条件的子数组？

滑动窗口，且需要时刻维护窗口左右边界以及窗口内最小值和最大值，因此用multiset，可以按顺序存储重复元素

```cpp
class Solution {
public:
    int longestSubarray(vector<int>& nums, int limit) {
        multiset<int> s;
        int l = 0;//左边界
        s.insert(nums[0]);
        int ans = 1;
        for(int i = 1; i < nums.size(); i++){
            s.insert(nums[i]);//插入右边界
            while(1){
                auto it = s.end();
                it--;//最大值
                if(*it - *(s.begin()) > limit){//窗口内最大元素绝对差超出限制
                    //窗口左边界向右收缩
```

```
                    auto e = s.find(nums[l]);
                    s.erase(e);
                    l++;
                }
                else//满足条件
                    break;
            }
            ans = max(ans, i-l+1);
        }
        return ans;
    }
};
```

1439. 有序矩阵中的第K个最小数组和（困难）

1. 题目描述

给你一个 `m * n` 的矩阵 `mat`，以及一个整数 `k` ，矩阵中的每一行都以非递减的顺序排列。

你可以从每一行中选出 1 个元素形成一个数组。返回所有可能数组中的第 k 个 **最小** 数组和。

示例 1:

```
输入: mat = [[1,3,11],[2,4,6]], k = 5
输出: 7
解释: 从每一行中选出一个元素，前 k 个和最小的数组分别是:
[1,2], [1,4], [3,2], [3,4], [1,6]。其中第 5 个的和是 7 。
```

示例 2:

```
输入: mat = [[1,3,11],[2,4,6]], k = 9
输出: 17
```

示例 3:

```
输入: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7
输出: 9
解释: 从每一行中选出一个元素，前 k 个和最小的数组分别是:
[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]。其中第 7 个的和是 9 。
```

示例 4:

```
输入: mat = [[1,1,10],[2,2,9]], k = 7
输出: 12
```

提示:

- `m == mat.length`
- `n == mat.length[i]`
- `1 <= m, n <= 40`
- `1 <= k <= min(200, n ^ m)`

- `1 <= mat[i][j] <= 5000`
- `mat[i]` 是一个非递减数组

2. 比赛实现

首先定义主要使用的数据结构priority_queue内的状态

- 定义priority_queue内保存的数组状态为<int, string>，且为小顶堆，即优先以int升序排序，可以保证堆顶始终是当前堆终所有状态里数组和最小的状态
- 其中string的格式为"idx0 idx1 ... idxm-1 "表示当前构成的数组在矩阵内每行选取的元素的索引值，int为当前形成的数组和
- 初始状态为<每行第一个元素的和， "0 0 ... 0 ">，由于每行元素非递减，因此该数组和是最小的数组和

然后定义转换规则，有些类似于BFS的思想

- 进行k次循环，每次循环只弹出并处理当前的堆顶状态Q<sum, choice>，则第i次循环的堆顶一定是第i小的数组和，第k次循环的堆顶就是答案
- Q的所有可以到达的下一个状态集合为N{<string字段对应的数组和, choice在第j行对应的索引后移一位>, j=0...m-1}，将N内所有未访问过的状态加入priority_queue
- 由于每行元素的非递减性，可以保证，N内所有状态的最小数组和一定小于等于N内所有状态的下一状态集合内所有状态的最小数组和，即第i+1小的数组此时和一定在priority_queue内，且就是堆顶对应的状态

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& mat, int k) {
        int m = mat.size();
        if(m == 1)  return mat[0][k-1];
        int n = mat[0].size();
        priority_queue<pair<int, string>, vector<pair<int, string>>,
greater<pair<int, string>>> q;
        int sum = 0;
        string choice = "";
        for(int i = 0; i < m; i++){
            sum += mat[i][0];
            choice += "0 ";
        }
        q.push(make_pair(sum, choice));//初始状态
        unordered_set<string> visited;//记录访问过的状态
        visited.insert(choice);
        int ans;
        while(k--){//k次循环，每次找到下一个最小的数组和
            ans = q.top().first;
            if(k == 0) return ans;//找到第k小数组和
            string cur = q.top().second;//当前状态对应的选择choice
            q.pop();
            stringstream ss(cur);
            vector<int> idxs(m);
            for(int i = 0; i< m; i++)//解析cur
                ss >> idxs[i];
            //状态转移
            for(int i = 0; i< m; i++){//依次修改每一行选择的元素索引
                if(idxs[i] < n-1){//没到最后一个，可以后移
                    int sum = ans - mat[i][idxs[i]] + mat[i][idxs[i]+1];//更新数组和
                    idxs[i]++;
```

```
                    string choice = "";
                    for(int j = 0; j < m; j++)
                        choice += to_string(idxs[j]) + ' ';
                    if(visited.find(choice) == visited.end()){//状态未访问过
                        q.push(make_pair(sum, choice));
                        visited.insert(choice);
                    }
                    idxs[i]--;
                }
            }
        }
        return -1;
    }
};
```

3. 最优解法——二分法

就是先确定左右边界，即最小和与最大和，然后二分得到mid，每次判断和小于mid的数组有多少个，如果大于等于k那么更新r，否则更新了。

```
class Solution {
public:
    vector<vector<int>>temp;
    int m,n;
    int kthSmallest(vector<vector<int>>& mat, int k) {
        temp=mat;
        m=mat.size(),n=mat[0].size();
        int l=0,r=0;
        for(int i=0;i<m;i++) l+=mat[i][0];
        for(int i=0;i<m;i++) for(int j=0;j<n;j++) r+=mat[i][j];
        int init=l;
        while(l<r){
            int mid=(l+r)>>1;
            int num=1;
            dfs(mid,0,init,num,k);
            if(num>=k) r=mid;
            else l=mid+1;
        }
        return l;
    }
    void dfs(int mid,int index,int sum,int& num,int k){
        if(sum>mid||index==m||num>k) return;
        dfs(mid,index+1,sum,num,k);
        for(int i=1;i<n;i++){
            if(sum+temp[index][i]-temp[index][0]<=mid){
                num++;
                dfs(mid,index+1,sum+temp[index][i]-temp[index][0],num,k);
            }else{
                break;
            }
        }
    }
};
```