总结

1. 580，和上次相比又进步了，开心
2. 第三题的大数据集没做出来，不知道最多5个用例的测试集特别大是啥意思？不过看了大佬的解答后，觉得自己确实不应该想不到做法
3. 最后一题做出来了，可喜可贺，但是其实就是使用树状数组，一开始没想到正确的奇偶划分，后来想到了之后把树状数组给忘了，浪费了好多时间啊哭哭哭，不然多出来的时间没准能做出来第三题呢哭哭哭~

第二题

1. 题目描述

Problem

Apollo is playing a game involving polyominos. A polyomino is a shape made by joining together one or more squares edge to edge to form a single connected shape. The game involves combining **N** polyominos into a single rectangular shape without any holes. Each polyomino is labeled with a unique character from `A` to `Z`.

Apollo has finished the game and created a rectangular wall containing **R** rows and **C** columns. He took a picture and sent it to his friend Selene. Selene likes pictures of walls, but she likes them even more if they are *stable* walls. A wall is stable if it can be created by adding polyominos one at a time to the wall so that each polyomino is always *supported*. A polyomino is supported if each of its squares is either on the ground, or has another square below it.

Apollo would like to check if his wall is stable and if it is, prove that fact to Selene by telling her the order in which he added the polyominos.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing the two integers **R** and **C**. Then, **R** lines follow, describing the wall from top to bottom. Each line contains a string of **C** uppercase characters from `A` to `Z`, describing that row of the wall.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is a string of **N** uppercase characters, describing the order in which he built them. If there is more than one such order, output any of them. If the wall is not stable, output `-1` instead.

Limits

Time limit: 20 seconds per test set. Memory limit: 1GB. 1 ≤ **T** ≤ 100. 1 ≤ **R** ≤ 30. 1 ≤ **C** ≤ 30. No two polyominos will be labeled with the same letter. The input is guaranteed to be valid according to the rules described in the statement.

Test set 1

1 ≤ **N** ≤ 5.

Test set 2

1 ≤ **N** ≤ 26.

Sample

```
Input
4
4 6
ZOAAMM
ZOAOMM
ZOOOOM
ZZZZOM
4 4
XXOO
XFFO
XFXO
XXXO
5 3
XXX
XPX
XXX
XJX
XXX
3 10
AAABBCCDDE
AABBCCDDEE
AABBCCDDEE

Output
Case #1: ZOAM
Case #2: -1
Case #3: -1
Case #4: EDCBA
```

In sample case #1, note that `ZOMA` is another possible answer.

In sample case #2 and sample case #3, the wall is not stable, so the answer is `-1`.

In sample case #4, the only possible answer is `EDCBA`.

2. 比赛实现

拓扑排序题，对于相邻两行的元素，同一列上下两个元素不一样，则上边的元素依赖于下边的元素，因此可以建立拓扑图，若图可以拓扑排序，则是稳定的

```
#include <bits/stdc++.h>
using namespace std;

void solve() {
```

```
    int r,c;
    cin >> r >> c;
    unordered_map<char, unordered_set<char>> m;//拓扑图
    unordered_map<char, int> ins;//入度
    string pre;//上一行
    cin >> pre;
    for(int i = 0; i < c; i++){//第一行
        if(ins.find(pre[i]) == ins.end())
            ins[pre[i]] = 0;
    }
    for(int i = 1; i < r; i++){
        string cur;//当前行
        cin >> cur;
        for(int j = 0; j < c; j++){//各列上下比较
            if(ins.find(cur[j]) == ins.end())
                ins[cur[j]] = 0;
            if(cur[j] != pre[j] && m[cur[j]].find(pre[j]) == m[cur[j]].end()){//产
生拓扑依赖
                m[cur[j]].insert(pre[j]);
                ins[pre[j]]++;
            }
        }
        pre = cur;
    }
    //BFS拓扑排序
    string ans = "";
    queue<char> q;
    for(auto it = ins.begin(); it != ins.end(); it++){
        if(it->second == 0){
            q.push(it->first);
            it->second = -1;
        }
    }
    while(!q.empty()){
        char cur = q.front();
        q.pop();
        ans += cur;
        if(m.find(cur) == m.end())
            continue;
        for(auto it2 = m[cur].begin(); it2 != m[cur].end(); it2++){
            if(--ins[*it2] == 0){
                q.push(*it2);
                ins[*it2] = -1;
            }
        }
    }
    if(ans.size() == ins.size())
        cout << ans << "\n";
    else
        cout << "-1\n";
}

int main() {
```

```
    ios::sync_with_stdio(0);
    cin.tie(0);

    int t, i=1;
    cin >> t;
    while(t--) {
        cout << "Case #" << i << ": ";
        solve();
        ++i;
    }
}
```

第三题

1. 题目描述

Problem

Cristobal has an array of **N** (possibly negative) integers. The i-th integer in his array is **Ai**. A contiguous non-empty subarray of Cristobal's array is *perfect* if its total sum is a perfect square. A perfect square is a number that is the product of a non-negative integer with itself. For example, the first five perfect squares are 0, 1, 4, 9 and 16.

How many subarrays are *perfect*? Two subarrays are different if they start or end at different indices in the array, even if the subarrays contain the same values in the same order.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains the integer **N**. The second line contains **N** integers describing Cristobal's array. The i-th integer is **Ai**.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the number of perfect subarrays.

Limits

Memory limit: 1GB. 1 ≤ **T** ≤ 100. -100 ≤ **Ai** ≤ 100, for all i.

Test set 1

Time limit: 20 seconds. 1 ≤ **N** ≤ 1000.

Test set 2

Time limit: 30 seconds. For up to 5 cases, 1 ≤ **N** ≤ 105. For the remaining cases, 1 ≤ **N** ≤ 1000.

Sample

```
Input
3
3
2 2 6
5
30 30 9 1 30
4
4 0 0 16

Output
Case #1: 1
Case #2: 3
Case #3: 9
```

In sample case #1, there is one perfect subarray: `[2 2]` whose sum is 22.

In sample case #2, there are three perfect subarrays:

- `[9]`, whose total sum is 32.
- `[1]`, whose total sum is 12.
- `[30 30 9 1 30]`, whose total sum is 102.

In sample case #3, there are nine perfect subarrays:

- `[4]`, whose total sum is 22.
- `[4 0]`, whose total sum is 22.
- `[4 0 0]`, whose total sum is 22.
- `[0]`, whose total sum is 02.
- `[0 0]`, whose total sum is 02.
- `[0 0 16]`, whose total sum is 42.
- `[0]`, whose total sum is 02.
- `[0 16]`, whose total sum is 42.
- `[16]`, whose total sum is 42.

Note: We do not recommend using interpreted/slower languages for the test set 2 of this problem.

2. 比赛实现

比赛时只过了小数据集，实在不懂大数据集是啥意思，方法就是简单的前缀和+两层遍历判断，又加了各种加速的trick，也还是过不了大数据集，下面代码里没带trick

```cpp
#include <bits/stdc++.h>
using namespace std;

unordered_map<long, bool> dic;
bool judge(long n){//判断是否为平方数
    if(dic.find(n) != dic.end())
        return dic[n];
    long cur = sqrt(n);
    long num = cur * cur;
    dic[n] = (num == n);
    return num == n;
}
void solve() {
```

```
    int n;
    cin >> n;
    vector<int> dp(n+1, 0);//前缀和
    for(int i = 1; i <= n; i++){
        int cur;
        cin >> cur;
        dp[i] = dp[i-1] + cur;
    }
    int ans = 0;
    for(int l = 0; l < n; l++){
        for(int r = l+1; r <= n; r++){
            if(judge(dp[r]-dp[l]))
                ans++;
        }
    }
    cout << ans << "\n";
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int t, i=1;
    cin >> t;
    while(t--) {
        cout << "Case #" << i << ": ";
        solve();
        ++i;
    }
}
```

3. 正确做法

逆向思维，看注释

```
#include<bits/stdc++.h>
using namespace std;
using LL = long long;
constexpr int maxn = -1;
void work(){
    int N;
    cin >> N;
    int S = 0;//目前前缀和
    int MS = 0;//前缀和最小值
    unordered_map<int, int> mp;//统计目前各种前缀和的出现次数
    mp[S] += 1;//前缀和为0的记一次，因为要记录当sum[0...i]为平方数的情况
    LL ans = 0;
    for(int i = 1, A; i <= N; i += 1){
        cin >> A;
        S += A;
        for(int j = 0;; j += 1){
            int T = S - j * j;//若存在0<=i<j,sum[i...j]为完全平方数，则T=sum[0...i]应该
在mp中有计数
```

```
            if(T < MS) break;//之后的不用遍历了
            if(mp.count(T)) ans += mp[T];//加上计次
        }
        MS = min(MS, S);
        mp[S] += 1;//当前前缀和计次
    }
    cout << ans << "\n";
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int T;
    cin >> T;
    for(int t = 1; t <= T; t += 1){
        cout << "Case #" << t << ": ";
        work();
    }
    return 0;
}
```

第四题

1. 题目描述

Problem

Carl has an array of **N** candies. The i-th element of the array (indexed starting from 1) is **Ai** representing *sweetness value* of the i-th candy. He would like to perform a series of **Q** operations. There are two types of operation:

- Update the sweetness value of a candy in the array.
- Query the *sweetness score* of a subarray.

The sweetness score of a subarray from index l to r is: **Al** × 1 - **Al+1** × 2 + **Al+2** × 3 - **Al+3** × 4 + **Al+4** × 5 ...

More formally, the sweetness score is the sum of (-1)i-l**Ai** × (i - l + 1), for all i from l to r inclusive.

For example, the sweetness score of:

- [3, 1, 6] is 3 × 1 - 1 × 2 + 6 × 3 = 19
- [40, 30, 20, 10] is 40 × 1 - 30 × 2 + 20 × 3 - 10 × 4 = 0
- [2, 100] is 2 × 1 - 100 × 2 = -198

Carl is interested in finding out the total sum of sweetness scores of all queries. If there is no query operation, the sum is considered to be 0. Can you help Carl find the sum?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing **N** and **Q**. The second line contains **N** integers describing the array. The i-th integer is **Ai**. The j-th of the following **Q** lines describe the j-th operation. Each line begins with a single character describing the type of operation (`U` for update, `Q` for query).

- For an update operation, two integers **Xj** and **Vj** follow, indicating that the **Xj**-th element of the array is changed to **Vj**.
- For a query operation, two integers **Lj** and **Rj** follow, querying the sweetness score of the subarray from the **Lj**-th element to the **Rj**-th element (inclusive).

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the total sum of sweetness scores of all the queries.

Limits

Time limit: 20 seconds per test set. Memory limit: 1GB. 1 ≤ **T** ≤ 100. 1 ≤ **Ai** ≤ 100, for all i. 1 ≤ **N** ≤ 2 × 105 and 1 ≤ **Q** ≤ 105 for at most 6 test cases. For the remaining cases, 1 ≤ **N** ≤ 300 and 1 ≤ **Q** ≤ 300. If the j-th operation is an update operation, 1 ≤ **Xj** ≤ **N** and 1 ≤ **Vj** ≤ 100. If the j-th operation is a query operation, 1 ≤ **Lj** ≤ **Rj** ≤ **N**.

Test set 1

There will be at most 5 update operations.

Test set 2

There are no special constraints.

Sample

```
Input
2
5 4
1 3 9 8 2
Q 2 4
Q 5 5
U 2 10
Q 1 2
3 3
4 5 5
U 1 2
U 1 7
Q 1 2

Output
Case #1: -8
Case #2: -3
```

In sample case #1:

- The first query asks for the sweetness score of [3, 9, 8] which is 3 × 1 - 9 × 2 + 8 × 3 = 9.
- The second query asks for the sweetness score of [2] which is 2 × 1 = 2.
- The third query asks for the sweetness score of [1, 10] which is 1 × 1 - 10 × 2 = -19.

Thus, the final output should be 9 + 2 - 19 = -8.

In sample case #2:

- The first and only query asks for the sweetness score of [7, 5] which is 7 × 1 - 5 × 2 = -3.

Thus, the final output should be -3.

2. 比赛实现

因为奇偶位数的正负值不一样，所以分奇偶分别记录，可以使用树状数组记录

```
#include <bits/stdc++.h>
using namespace std;

class NumArray {//树状数组，从leetcode题里拿来的
public:
    vector<long long> nums, tree;
    int size;
    long long lowbit(long long x){
        return x & (-x);
    }
    NumArray(long long size) {
        this->size = size+1;//树状数组下标以1开始
        //初始化nums和trees
        this->nums = vector<long long>(size+1, 0);
        tree = vector<long long>(size+1, 0);
    }
    void update(int i, long long val) {
        i++;//下标对应
        val -= nums[i];//变化值
        nums[i] += val;//更新值
        while(i < size){
            tree[i] += val;
            i += lowbit(i);
        }
    }
    long long getsum(int pos){
        long long sum=0;
        pos++;//下标对应
        while(pos > 0){
            sum += tree[pos];
            pos -= lowbit(pos);
        }
        return sum;
    }
    long long sumRange(int i, int j) {
        return getsum(j) - getsum(i-1);
    }

};
void solve() {
```

```
    int n,q;
    cin >> n >> q;
    NumArray odd(n+1);//下标从1开始存储，奇数位的加权和，A[i]权值为i
    NumArray even(n+1);//下标从1开始存储，偶数位的加权和，A[i]权值为i
    NumArray sumodd(n+1);//下标从1开始存储，奇数位的和
    NumArray sumeven(n+1);//下标从1开始存储，偶数位的和
    for(int i = 1; i <= n; i++){//初始化，读入数据
        int cur;
        cin >> cur;
        if(i % 2 == 0){
            sumeven.update(i, cur);
            even.update(i, i*cur);//加权存储
        }
        else{
            sumodd.update(i, cur);
            odd.update(i, i*cur);//加权存储
        }
    }
    long long ans = 0;
    while(q--){
        char c;
        cin >> c;
        if(c == 'Q'){
            int l,r;
            cin >> l >> r;
            long cur = 0;
            if(l % 2 == 0){//以偶数位为起始
                cur -= odd.sumRange(l, r)-(l-1)*sumodd.sumRange(l, r);//奇数位在Q中为偶数位，-
                cur += even.sumRange(l, r)-(l-1)*sumeven.sumRange(l, r);//偶数位在Q中为奇数位，+
            }
            else{
                cur += odd.sumRange(l, r)-(l-1)*sumodd.sumRange(l, r);//奇数位在Q中为奇数位，+
                cur -= even.sumRange(l, r)-(l-1)*sumeven.sumRange(l, r);//偶数位在Q中为偶数位，+
            }
            /*减去的l-1倍的和值是整体移位，如整个数组的权重是1，-2，3，-4，5，-6，取Q[3,5]对应的是3，-4，5，需要调整奇数位整体权重减2，偶数位加2；取[2，5]对应-2，3，-4，5，需要调整奇数位整体取负加1，偶数位整体取负减1*/
            ans += cur;
        }
        else{//更新
            int idx, v;
            cin >> idx >> v;
            if(idx % 2 == 0){
                even.update(idx, idx*v);
                sumeven.update(idx, v);
            }
            else{
                odd.update(idx, idx*v);
                sumodd.update(idx, v);
```

```
            }
        }
    }
    cout << ans << "\n";
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int t, i=1;
    cin >> t;
    while(t--) {
        cout << "Case #" << i << ": ";
        solve();
        ++i;
    }
}
```