总结

1. 和上次相比进步了1500名，只有最后一题的大数据集没有通过，整体有进步
2. 然而，在做最后一题的时候，找规律明明找对了，手算验证却总是算错，浪费了很多时间，不然再多一些时间，或许可以尝试继续找大数据集的规律（毕竟在超时的时候已经有了预感）
3. 整体做题速度还需要提高
4. 只要四道都作对/飞快做完前三道半，就能前200了嘤嘤嘤！！！
5. 卧槽才发现如果通过所有测试集前，前面有通过测试样子的提交失败是有罚时的，我她妈，不要乱提交啊啊啊！！！

第一题

1. 题目描述

Problem

Li has planned a bike tour through the mountains of Switzerland. His tour consists of **N** checkpoints, numbered from 1 to **N** in the order he will visit them. The i-th checkpoint has a height of **Hi**.

A checkpoint is a *peak* if:

- It is not the 1st checkpoint or the **N**-th checkpoint, and
- The height of the checkpoint is *strictly greater than* the checkpoint immediately before it and the checkpoint immediately after it.

Please help Li find out the number of peaks.

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 integer **N**. The second line contains **N** integers. The i-th integer is **Hi**.

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 peaks in Li's bike tour.

Limits

Time limit: 10 seconds per test set. Memory limit: 1GB. 1 ≤ **T** ≤ 100. 1 ≤ **Hi** ≤ 100.

Test set 1

3 ≤ **N** ≤ 5.

Test set 2

3 ≤ **N** ≤ 100.

Sample

```
Input
4
3
10 20 14
4
7 7 7 7
5
10 90 20 90 10
3
10 3 10

Output
Case #1: 1
Case #2: 0
Case #3: 2
Case #4: 0
```

- In sample case #1, the 2nd checkpoint is a peak.
- In sample case #2, there are no peaks.
- In sample case #3, the 2nd and 4th checkpoint are peaks.
- In sample case #4, there are no peaks.

2. 比赛实现

遍历

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

void solve() {
    int n;
    vector<int> nums;
    int pre;
    cin >> pre;
    int cur;
    cin >> cur;
    int ans = 0;
    for(int i = 2; i < n-1; i++){
        int next;
        cin >> next;
        if(cur > pre && cur > next)
            ans++;
        pre = cur;
        cur = next;
    }

    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;
    }
}
```

第二题

1. 题目描述

Problem

Bucket is planning to make a very long journey across the countryside by bus. Her journey consists of **N** bus routes, numbered from 1 to **N** in the order she must take them. The buses themselves are very fast, but do not run often. The i-th bus route only runs every **Xi** days.

More specifically, she can only take the i-th bus on day **Xi**, 2**Xi**, 3**Xi** and so on. Since the buses are very fast, she can take multiple buses on the same day.

Bucket must finish her journey by day **D**, but she would like to start the journey as late as possible. What is the latest day she could take the first bus, and still finish her journey by day **D**?

It is guaranteed that it is possible for Bucket to finish her journey by day **D**.

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 **N** and **D**. Then, another line follows containing **N** integers, the i-th one is **Xi**.

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 latest day she could take the first bus, and still finish her journey by day **D**.

Limits

Time limit: 10 seconds per test set. Memory limit: 1GB. 1 ≤ **T** ≤ 100. 1 ≤ **Xi** ≤ **D**. 1 ≤ **N** ≤ 1000. It is guaranteed that it is possible for Bucket to finish her journey by day **D**.

Test set 1

1 ≤ **D** ≤ 100.

Test set 2

1 ≤ **D** ≤ 1012.

Sample

```
Input:
3
3 10
3 7 2
4 100
11 10 5 50
1 1
1

Output:
Case #1: 6
Case #2: 99
Case #3: 1
```

In Sample Case #1, there are **N** = 3 bus routes and Bucket must arrive by day **D** = 10. She could:

- Take the 1st bus on day 6 (**X1** = 3),
- Take the 2nd bus on day 7 (**X2** = 7) and
- Take the 3rd bus on day 8 (**X3** = 2).

In Sample Case #2, there are **N** = 4 bus routes and Bucket must arrive by day **D** = 100. She could:

- Take the 1st bus on day 99 (**X1** = 11),
- Take the 2nd bus on day 100 (**X2** = 10),
- Take the 3rd bus on day 100 (**X3** = 5) and
- Take the 4th bus on day 100 (**X4** = 50),

In Sample Case #3, there is **N** = 1 bus route and Bucket must arrive by day **D** = 1. She could:

- Take the 1st bus on day 1 (**X1** = 1).

2. 比赛实现

从大数据集的数据量判断可以用二分法，判断从第x天开始坐车能否满足要求，判断方法为贪心

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

int n;
//从cur天开始坐车，能否在d天前完成
bool judge(vector<unsigned long long>& x, unsigned long long cur, unsigned long
long d){
    unsigned long long now = cur;//当前日期
    for(int i = 0; i < n; i++){
        unsigned long long temp = now / x[i];
        if(x[i]*temp < now)
            temp += 1;
        now = temp*x[i];//乘坐车x[i]的最早日期
        if(now > d) return false;//超出期限
    }
    return true;
}
void solve() {
```

```
    unsigned long long d;//注意数据范围
    cin >> n>> d;
    vector<unsigned long long> x(n);
    for(int i = 0; i < n; i++)
        cin >> x[i];
    unsigned long long l = 1, r = d;
    while(l < r){
        unsigned long long mid = l + (r-l) / 2;
        if(judge(x, mid, d))
            l = mid+1;
        else
            r = mid;
    }
    //二分法没用好Hhh，虽然有bug，但是多判断一次就好了，这种trick要记得鸭
    if(judge(x,l,d))
       cout << l << "\n";
    else
       cout << l-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

Your country's space agency has just landed a rover on a new planet, which can be thought of as a grid of squares containing 109 columns (numbered starting from 1 from west to east) and 109 rows (numbered starting from 1 from north to south). Let (w, h) denote the square in the w-th column and the h-th row. The rover begins on the square (1, 1).

The rover can be maneuvered around on the surface of the planet by sending it a *program*, which is a string of characters representing movements in the four cardinal directions. The robot executes each character of the string in order:

- `N` : Move one unit north.
- `S` : Move one unit south.
- `E` : Move one unit east.
- `W` : Move one unit west.

There is also a special instruction `X(Y)`, where `X` is a number between 2 and 9 inclusive and `Y` is a non-empty subprogram. This denotes that the robot should repeat the subprogram `Y` a total of `X` times. For example:

- `2(NWE)` is equivalent to `NWENWE`.
- `3(S2(E))` is equivalent to `SEESEESEE`.
- `EEEE4(N)2(SS)` is equivalent to `EEEENNNNSSSS`.

Since the planet is a torus, the first and last columns are adjacent, so moving east from column 109 will move the rover to column 1 and moving south from row 109 will move the rover to row 1. Similarly, moving west from column 1 will move the rover to column 109 and moving north from row 1 will move the rover to row 109. Given a program that the robot will execute, determine the final position of the robot after it has finished all its movements.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains a single string: the program sent to the rover.

Output

For each test case, output one line containing `Case #x: w h`, where `x` is the test case number (starting from 1) and `w h` is the final square (w, h) the rover finishes in.

Limits

Time limit: 10 seconds per test set. Memory limit: 1GB. $1 \leq \mathbf{T} \leq 100$. The string represents a valid program. The length of each program is between 1 and 2000 characters inclusive.

Test set 1

The total number of moves the robot will make in a single test case is at most 104.

Test set 2

No additional constraints.

Sample

```
Input
4
SSSEEE
N
N3(S)N2(E)N
2(3(NW)2(W2(EE)W))

Output
Case #1: 4 4
Case #2: 1 1000000000
Case #3: 3 1
Case #4: 3 999999995
```

In Sample Case #1, the rover moves three units south, then three units east.

In Sample Case #2, the rover moves one unit north. Since the planet is a torus, this moves it into row 109.

In Sample Case #3, the program given to the rover is equivalent to `NSSSNEEN`.

In Sample Case #4, the program given to the rover is equivalent to `NWNWNWWEEEEWWEEEEWNWNWNWWEEEEWWEEEEW`.

2. 比赛实现

看到大测试集之后就知道，不能单纯地解析字符串再按顺序走，否则肯定不行，随后观察发现，字符串内NSWE满足以下规律：

- 交换律：任意替换两个字符的顺序，最终机器人位置不变
- 可消：N与S可相互抵消，W与E可相互抵消（不抵消也无所谓，反正就算呗）

因此，可以统计抵消后的各个字符数量，再直接在坐标上做加减法，统计各个字符个数的方法为：每个括号为一层，括号内每个字符增加的数量为括号外所有数字的乘积

另外需要注意本题涉及到大数问题，要及时以1E9取余（即走一圈回到起点）

```
#include <bits/stdc++.h>
using namespace std;
int MOD = 1e9;
void solve() {
    string s;
    cin >> s;
    int len = s.size();
    int r = 0, c = 0;//把坐标先按0~1e9-1记，方便处理边界
    vector<int> cnt(4, 0);//N,S,E,W
    stack<long> cache;//栈顶缓存当前括号外的所有数字乘积，初值为1
    cache.push(1);
    long num = 0;//当前待处理数字
    for(int i = 0; i < len; i++){
        if(s[i] >= '2' && s[i] <= '9'){//处理数字
            num = (num*10 + int(s[i]-'0')) % MOD;
        }
        else if(s[i] == '('){//加一层
            num = (num * cache.top()) % MOD;
            cache.push(num);
            num = 0;
        }
        else if(s[i] == ')'){//退一层
            cache.pop();
        }
        else{//字母计数
            if(s[i] == 'N')
                cnt[0] = (cnt[0] + cache.top()) % MOD;
            else if(s[i] == 'S')
                cnt[1] = (cnt[1] + cache.top()) % MOD;
            else if(s[i] == 'E')
                cnt[2] = (cnt[2] + cache.top()) % MOD;
            else if(s[i] == 'W')
                cnt[3] = (cnt[3] + cache.top()) % MOD;
        }
```

```
    }
    if(cnt[0] > 0)
        r = (r + MOD - cnt[0]) % MOD;
    if(cnt[1] > 0)
        r = (r + cnt[1]) % MOD;
    if(cnt[2] > 0)
        c = (c + cnt[2]) % MOD;
    if(cnt[3] > 0)
        c = (c + MOD - cnt[3]) % MOD;
    cout << c+1 << ' '<< r+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;
    }
}
```

第四题

1. 题目描述

Problem

Jemma is competing in a robotics competition. The challenge for today is to build a robot that can navigate around a hole in the arena.

The arena is a grid of squares containing **W** columns (numbered 1 to **W** from left to right) and **H** rows (numbered 1 to **H** from top to bottom). The square in the x-th column and y-th row is denoted (x, y). The robot begins in the top left square (1,1) and must navigate to the bottom right square (**W**, **H**).

A rectangular subgrid of squares has been cut out of the grid. More specifically, all the squares that are in the rectangle with top-left square (**L**, **U**) and bottom-right square (**R**, **D**) have been removed.

Jemma did not have much time to program her robot, so it follows a very simple algorithm:

- If the robot is in the rightmost column, it will always move to the square directly below it. Otherwise,
- If the robot is in the bottommost row, it will always move to the square directly right of it. Otherwise,
- The robot will randomly choose to either move to the square directly to the right, or to the square directly below it with equal probability.

Jemma passes the challenge if her robot avoids falling into the hole and makes it to the square (**W**, **H**). What is the probability she passes the challenge?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of a single line containing **W**, **H**, **L**, **U**, **R** and **D**.

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 real number between 0 and 1 inclusive, the probability that Jemma passes the challenge.

`y` will be considered correct if it is within an absolute or relative error of 10-5 of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

Time limit: 15 seconds per test set. Memory limit: 1GB. 1 ≤ **T** ≤ 100. 1 ≤ **U** ≤ **D** ≤ **H**. 1 ≤ **L** ≤ **R** ≤ **W**. Neither the top-left nor bottom-right squares will be missing.

Test set 1

1 ≤ **W** ≤ 300. 1 ≤ **H** ≤ 300.

Test set 2

1 ≤ **W** ≤ 105. 1 ≤ **H** ≤ 105.

Sample

```
Input
4
3 3 2 2 2 2
5 3 1 2 4 2
1 10 1 3 1 5
6 4 1 3 3 4

Output
Case #1: 0.5
Case #2: 0.0625
Case #3: 0.0
Case #4: 0.3125
```

2. 比赛实现

只找到了初步规律：`p[i][j]=(p[i+1][j] + p[i][j+1]) / 2`，`p[i][j]`为从(i,j)开始走，成功的概率，dp数组可以压缩到一维

这题一开始浪费了太多时间，还是要自信一些啊，不要觉得到最后一题了就随便了就不动脑子了、、、

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

int w,h,l,u,r,d;
```

```
bool in(int a, int b){//判断a、b是否是洞
    if(a >= l && a <= r && b >= u && b <= d)
        return true;
    else
        return false;
}
void solve() {
    cin >> w >> h >> l >> u >> r >> d;
    w = min(w,r+1);
    h = min(h,d+1);
    /*比赛时为了过大测试集加的trick，没啥用，当时时间不够了只能这样挣扎一下
    if((l == 1 && r == w) || (u == 1 && d == h))
        cout << 0<< "\n";
        return;
    }
    if(l == 1)
        h = min(h,u);
    if(r == w)
        w = min(w, l);
    if(d == h)
        h = min(h, u);
    if(u == 1)
        w = min(w, l);
    */

    vector<double> dp(w+1);//当前行里，各列的成功概率p
    //最后一行
    dp[w] = 1;
    bool flag = true;//本行存在p>0的起始位置
    for(int i = w-1; i > 0; i--){
        if(in(i, h))
            dp[i] = 0;
        else
            dp[i] = dp[i+1];
    }
    int idx = h-1;
    while(idx >= 1){
        flag = false;
        if(in(w, idx))
            dp[w] = 0;
        if(dp[w] > 0.00000001)
            flag = true;
        for(int j = w-1; j > 0; j--){
            if(in(j,idx))
                dp[j]= 0;
            else
                dp[j] = (dp[j] + dp[j+1]) / 2;
            if(dp[j] > 0.00000001)
                flag = true;
        }
        if(!flag){//本行全0，直接没有任何成功的希望了
            cout << 0<< "\n";
            return;
```

```
            }
            idx--;
        }
        cout << dp[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;
        }
    }
```

3. 正确做法

   那什么鬼规律，代码看不懂，不管了2333