

2023.4第三四周

绝句 宋 志南

古木阴中系短篷，

杖藜扶我过桥东。

沾衣欲湿杏花雨，

吹面不寒杨柳风。

省赛 省赛 省赛 省赛 省赛 省赛 省赛 省赛.....

拿到名额啦！全队都很开心，下一个目标是省赛银。

算法学习：

学点博弈论，爽一下。

1. 博弈论

1. [SG函数.md](#)
2. [阶梯nim.md](#)
3. [翻硬币问题.md](#)
4. [anti-sg.md](#)
5. [d阶nim.md](#)

2. 博弈论问题：

1. [BJwc2009 取石子游戏.md](#)
2. [SDOI 2009, E&D.md](#)

dp

1. [换根dp求树直径.md](#)

图论

1. [树的直径.md](#)

codeforces 补题：

1. [D. The Butcher.md](#) 对check的理解上不够深刻：

codeforces刷题：

1. 构造

1. [B. Letter Exchange.md](#)
2. [King's Puzzle.md](#)
3. [C. Recover an RBS.md](#) (未补)
4. [D. Super-Permutation.md](#)
5. [D. Unique Palindromes.md](#)

2. 数论：

1. [C. Candy Store.md](#)

3. dp

1. 博弈， dp [D. Letter Picking.md](#)
2. [城市里的间谍.md](#)

3. [tour.md](#)
4. [单向TSP.md](#)
5. [学霸题，与原神の终极之战.md](#)
6. [学霸题，帮原神分割布丁.md](#)

4. 博弈论:

1. [E. Removing Graph.md](#)

SG函数

如果一种公平组合游戏可以转换成一个有向图。将问题转变成，在一个有向无环图中，只有一个起点，上面有一个棋子，两个玩家轮流推动棋子，不能走的玩家判负。

SG函数定义

状态 x 和它所有 k 个后继状态 y_1, y_2, \dots, y_k

$$SG_x = mex(SG_{y_1}, SG_{y_2}, SG_{y_3}, \dots, SG_{y_k})$$

游戏的和:

G_i 表示若干个游戏:

$$G = G_1 + G_2 + G_3$$

每步选择一个 G_i 移动一步。

SG定理:

对于由 n 个有向图游戏组成的组合游戏，设它们的起点分别为 $s_1 \dots s_n$ 有:

$$SG(s) = SG(s_1) \oplus SG(s_2) \oplus \dots \oplus SG(s_n)$$

先手必胜当且仅当， $SG(s) > 0$

问题:

1. SG函数的初值怎么定义?

1. 将终点状态定义为0.

2. 把握SG函数的使用前提:

1. SG函数和SG定理可以解决一大类的博弈问题,这一大类的博弈问题称为**ICG游戏**.

1. 游戏有两人参与，两人轮流做出决策，并且两人做出的决策都是对自己最优的
2. **当有一人无法决策的时候，该人失败**。无论两人如何决策，该游戏都必然会在有限时间内结束
3. 游戏中同一个状态不能达到多次，且游戏没有平局。游戏者在某个确定状态做出的决策集合只与状态有关，与游戏者无关

例题:

first

1. 设有一堆石子，两个人轮流取，每次可以取 $1 \dots k$ 个，谁不能动就算输，问谁会获胜。

$SG(i)$ 表示当前这堆石子有 i 个其SG值的大小:

$$SG(0 \dots i): 0 \ 1 \ 2 \ 3 \ \dots \ k \ 0 \ 1 \ 2 \ 3 \ \dots \ k \ \dots$$

规律就是 $\text{mod } (k + 1) = 0$ 必输。

second

2. n堆石子。两个人轮流取，每次可以在一堆石头里面选取任意多的石头，或者把一堆石头分裂成两堆。谁不能操作谁输。

1. 思路：打表找出sg大小的计算规律：

1. 关于sg函数的转移：

$$\begin{aligned} &1. \text{取石头: } sg(0 \dots (i - 1)) \\ &2. \text{分裂: } sg(j) \text{ xor } sg(j - i) \end{aligned} \quad (1)$$

2.

3. 然后找出规律后，就可以统计多个有向图组合成的组合游戏sg情况了。

生长：对于第二种分裂的统计，可以感受到多个有向图游戏之间有一定的独立性。

打表角度如下：

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6
7  int sg[N];
8  int cal(int x) {
9      if (x == 0 || (x - 1) % 4 == 0 || (x - 1) % 4 == 1) {
10         return x;
11     } else if ((x - 1) % 4 == 2)
12         return x + 1;
13     else return x - 1;
14 }
15 int main()
16 {
17     ios::sync_with_stdio(false);
18     cin.tie(0);
19     int n = 100;
20     for (int i = 1; i <= n; i++) {
21         set<int> rec;
22         for (int j = 1; j <= i; j++)
23             rec.insert(sg[i - j]);
24         for (int j = 1; j < i; j++)
25             rec.insert(sg[j] ^ sg[i - j]);
26         for (int j = 0; j <= 1000; j++)
27             if (rec.count(j) == 0) {
28                 sg[i] = j;
29                 break;
30             }
31         assert(sg[i] == cal(i));
32         cout << i << " " << sg[i] << "\n";
33     }
34 }
35 /* stuff you should look for
36 * int overflow, array bounds
```

```

37 * special cases (n=1?)
38 * do smth instead of nothing and stay organized
39 * WRITE STUFF DOWN
40 * DON'T GET STUCK ON ONE APPROACH
41 */

```

third

1. 有 $1*n$ 的东西，每个人轮流拿连续（序号必须连续）两个，谁不能拿就输。求sg的值：

对于一个初始序号连续的序列，拿到两个条之后，就可以将问题转换成两个独立的公平组合游戏：

$$sg(i) = sg(j) \text{ xor } sg(i - j - 2)$$

//这点还有待理解：

然后打表，寻找规律即可。

1	0
2	1
3	1
4	2
5	0
6	3
7	1
8	1
9	0
10	3
11	3
12	2
13	2
14	4
15	0
16	5
17	2
18	2
19	3
20	3
21	0
22	1
23	1
24	3
25	0
26	2
27	1
28	1
29	0
30	4
31	5
32	2
33	7
34	4
35	0
36	1
37	1
38	2
39	0

一眼没有看出很显然的归规律，但是可以看到一些数字出现的频率，可以猜测出现循环结。

阶梯nim

问题描述：

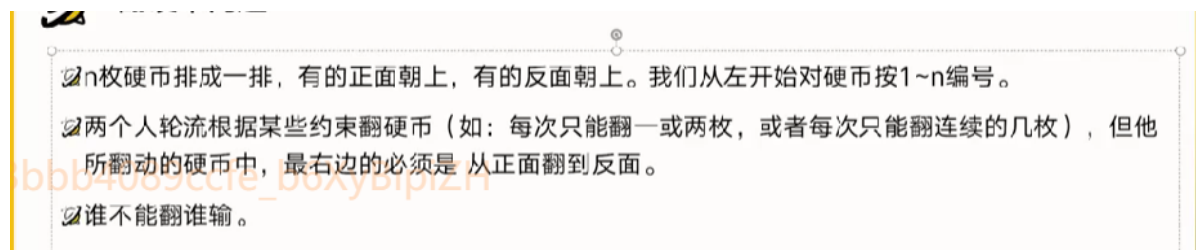
有 n 堆石头，两个人进行操作：选择第 i 堆石头，拿出一部分石头放在 $i - 1$ 堆之中，直到只剩下一堆。使得一方不能再拿。

结论：

结果与偶数堆的sg异或和相关：

1. 偶数的sg异或和为0：先手必败。

翻硬币问题



n 枚硬币排成一排，有的正面朝上，有的反面朝上。我们从左开始对硬币按 $1 \sim n$ 编号。
 两个人轮流根据某些约束翻硬币（如：每次只能翻一或两枚，或者每次只能翻连续的几枚），但他所翻动的硬币中，最右边的必须是 从正面翻到反面。
 谁不能翻谁输。

理解：

上述一个特殊的约束：

1. 最右边的硬币的变化要求
2. 这样决定了每一枚硬币只能作为一种端点：

结论：

1. 整个游戏的sg值，等于每个硬币的单独的sg的异或和。

硬币的sg值计算：

$$sg(i) = mex(0, sg(j), (1 \leq j < i))$$

原理概述：

??? 真抽象。

anti-SG



Anti-SG

对于任意一个Anti-SG游戏，如果我们规定当局面中所有的单一游戏的SG值为0时，游戏结束(操作的人失败)，则先手必胜当且仅当：

- (1) 游戏的SG函数不为0且游戏中某个单一游戏的SG函数大于1。
- (2) 游戏的SG函数为0且游戏中没有单一游戏的SG函数大于1。

小头-U

d阶nim



d阶nim

小头-U

单击此处添加文本

N 阶 Nim 游戏：有 k 堆石子，各包含 x_1, x_2, \dots, x_k 颗石子。双方玩家轮流操作，每次操作选择其中非空的若干堆，至少一堆但不超过 N 堆，在这若干堆中的每堆各取走其中的若干颗石子（1 颗，2 颗...甚至整堆），数目可以不同，取走最后一颗石子的玩家获胜。

结论：当且仅当在每一个不同的二进制位上， x_1, x_2, \dots, x_k 中在该位上 1 的个数是 $N+1$ 的倍数时，后手方有必胜策略，否则先手必胜。

如上，不是普通人neng'gou

取石子游戏

<http://oj.daimayuan.top/course/30/problem/1239>

描述

提交

自定义测试

统计 附加文件

小H和小Z正在玩一个取石子游戏。取石子游戏的规则是这样的，每个人每次可以从一堆石子中取出若干个石子，每次取石子的个数有限制，谁不能取石子时就会输掉游戏。小H先进行操作，他想问你他是否有必胜策略，如果有，第一步如何取石子。

输入格式

第一行一个整数 $n(n \leq 10)$ ，表示石子堆数。

接下来 n 行，每行一个数 $a_i(a_i \leq 1000)$ ，表示每堆石子的个数。

接下来一行，一个整数 $m(m \leq 10)$ ，表示每次取石子个数的种类数。

接下来 m 行，每行一个数 $b_i(b_i \leq 10)$ ，表示每次可以取的石子个数，保证这 m 个数按照递增顺序排列。

输出格式

第一行为yes或者no，表示小H是否有必胜策略。

若结果为yes,则第二行包含两个数，第一个数表示从哪堆石子取，第二个数表示取多少个石子。若有多种答案，取第一个数最小的答案。若仍有多种答案，取第二个数最小的答案。

样例输入：

solve

首先是个公平组合游戏：博弈的集合减少，两人操作。当集合缩减为0时结束。

1. sg函数的计算方式：

由于数据范围比较小，直接计算就行。

2. 判断先手是否必胜的方法：

sg定理。

3. 找到先手必胜的第一步：

枚举各种操作，找出使得后手必输的一个操作。（也是利用sg定理：）

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int N = 1E6 + 10;
5  int a[N] , b[N] , sg[N];
6  int main()
7  {
8      ios::sync_with_stdio(false);
9      cin.tie(0);
10     int n;
11     cin >> n;
12     for (int i = 1; i <= n; i++)
13         cin >> a[i];
14     int m;
15     cin >> m;
16     for (int i = 1; i <= m; i++)
17         cin >> b[i];
18     int ans = 0;
19     for (int i = 1; i <= 1100; i++) {
20         set<int> rec;
21         for (int j = 1; j <= m && b[j] <= i; j++) {
22             rec.insert(sg[i - b[j]]);
23         }
24         while (rec.count(sg[i])) sg[i]++;
25     }
26     for (int i = 1; i <= n; i++) {
27         ans ^= sg[a[i]];
28     }
29     if (ans == 0) cout << "NO\n";
30     else {
31         cout << "YES\n";
32         for (int i = 1; i <= n; i++) {
33             for (int j = 1; j <= m && b[j] <= a[i]; j++) {
34                 if ((ans ^ sg[a[i]] ^ sg[a[i] - b[j]]) == 0) {
35                     cout << i << ' ' << b[j] << "\n";
36                     return 0;
37                 }
38             }
39         }
40     }
41 }
```



```

42  /* stuff you should look for
43  * int overflow, array bounds
44  * special cases (n=1?)
45  * do smth instead of nothing and stay organized
46  * WRITE STUFF DOWN
47  * DON'T GET STUCK ON ONE APPROACH
48  */

```

SDOI 2009, E&D

小 E 与小 W 进行一项名为 E&D 游戏。

游戏的规则如下：桌子上有 $2n$ 堆石子，编号为 $1 \sim 2n$ 。其中，为了方便起见，我们将第 $2k-1$ 堆与第 $2k$ 堆 ($1 \leq k \leq n$) 视为同一组。第 i 堆的石子个数用一个正整数 S_i 表示。

一次分割操作指的是，从桌子上任取一堆石子，将其移走。然后分割它同一组的另一堆石子，从中取出若干个石子放在被移走的位置，组成新的一堆。操作完成后，所有堆的石子数必须保证大于 0。显然，被分割的一堆的石子数至少要为 2。两个人轮流进行分割操作。如果轮到某人进行操作时，所有堆的石子数均为 1，则此时没有石子可以操作，判此人输掉比赛。

小 E 进行第一次分割。他想知道，是否存在某种策略使得他一定能战胜小 W。因此，他求助于小 F，也就是你，请你告诉他是否存在必胜策略。例如，假设初始时桌子上有 4 堆石子，数量分别为 1, 2, 3, 1。小 E 可以选择移走第 1 堆，然后将第 2 堆分割（只能分出 1 个石子）。接下来，小 W 只能选择移走第 4 堆，然后将第 3 堆分割为 1 和 2。最后轮到小 E，他只能移走后两堆中数量为 1 的一堆，将另一堆分割为 1 和 1。这样，轮到小 W 时，所有堆的数量均为 1，则他输掉了比赛。故小 E 存在必胜策略。

solve

sg 函数转移：

$sg_{i,j} = mex(sg_{x,i-x}, sg_{x,j-x})$ 不太精确意会即可。

然后到打出了一张表：

1	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1
2	1	1	2	2	1	1	3	3	1	1	2	2	1	1	4	4	1	1	2	2	1	1	3	3	1	1	2	2	1	1
3	0	2	0	2	0	3	0	3	0	2	0	2	0	4	0	4	0	2	0	2	0	3	0	3	0	2	0	2	0	5
4	2	2	2	2	3	3	3	3	2	2	2	2	4	4	4	4	2	2	2	2	3	3	3	3	2	2	2	2	5	5
5	0	1	0	3	0	1	0	3	0	1	0	4	0	1	0	4	0	1	0	3	0	1	0	3	0	1	0	5	0	1
6	1	1	3	3	1	1	3	3	1	1	4	4	1	1	4	4	1	1	3	3	1	1	3	3	1	1	5	5	1	1
7	0	3	0	3	0	3	0	3	0	4	0	4	0	4	0	4	0	3	0	3	0	3	0	3	0	5	0	5	0	5
8	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	5	5	5	5	5	5
9	0	1	0	2	0	1	0	4	0	1	0	2	0	1	0	4	0	1	0	2	0	1	0	5	0	1	0	2	0	1
10	1	1	2	2	1	1	4	4	1	1	2	2	1	1	4	4	1	1	2	2	1	1	5	5	1	1	2	2	1	1
11	0	2	0	2	0	4	0	4	0	2	0	2	0	4	0	4	0	2	0	2	0	5	0	5	0	2	0	2	0	5
12	2	2	2	2	4	4	4	4	2	2	2	2	4	4	4	4	2	2	2	2	5	5	5	5	2	2	2	2	5	5
13	0	1	0	4	0	1	0	4	0	1	0	4	0	1	0	4	0	1	0	5	0	1	0	5	0	1	0	5	0	1
14	1	1	4	4	1	1	4	4	1	1	4	4	1	1	4	4	1	1	5	5	1	1	5	5	1	1	5	5	1	1
15	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	5	0	5	0	5	0	5	0	5	0	5	0	5
16	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5
17	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1
18	1	1	2	2	1	1	3	3	1	1	2	2	1	1	5	5	1	1	2	2	1	1	3	3	1	1	2	2	1	1
19	0	2	0	2	0	3	0	3	0	2	0	2	0	5	0	5	0	2	0	2	0	3	0	3	0	2	0	2	0	5
20	2	2	2	2	3	3	3	3	2	2	2	2	5	5	5	5	2	2	2	2	3	3	3	3	2	2	2	2	5	5
21	0	1	0	3	0	1	0	3	0	1	0	5	0	1	0	5	0	1	0	3	0	1	0	3	0	1	0	5	0	1
22	1	1	3	3	1	1	3	3	1	1	5	5	1	1	5	5	1	1	3	3	1	1	3	3	1	1	5	5	1	1
23	0	3	0	3	0	3	0	3	0	5	0	5	0	5	0	5	0	3	0	3	0	3	0	3	0	5	0	5	0	5
24	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	5	3	3	3	3	3	3	3	3	5	5	5	5	5	5
25	0	1	0	2	0	1	0	5	0	1	0	2	0	1	0	5	0	1	0	2	0	1	0	5	0	1	0	2	0	1
26	1	1	2	2	1	1	5	5	1	1	2	2	1	1	5	5	1	1	2	2	1	1	5	5	1	1	2	2	1	1
27	0	2	0	2	0	5	0	5	0	2	0	2	0	5	0	5	0	2	0	2	0	5	0	5	0	2	0	2	0	5
28	2	2	2	2	5	5	5	5	2	2	2	2	5	5	5	5	2	2	2	2	5	5	5	5	2	2	2	2	5	5
29	0	1	0	5	0	1	0	5	0	1	0	5	0	1	0	5	0	1	0	5	0	1	0	5	0	1	0	5	0	1
30	1	1	5	5	1	1	5	5	1	1	5	5	1	1	5	5	1	1	5	5	1	1	5	5	1	1	5	5	1	1

很容易发现规律：

1. 从小到大构造这张表：类似于当前表左上角的正方形向下贴三份，并且将其中的一个数字 x 改成 $x+1$;其中 x 和正方形的长有关。
2. 那么怎么利用这种现象，精确的计算 sg 的值呢？

模仿上述的递归构造方式，是可以计算出目标的 sg 函数的（时间复杂度 $O(n \log(a))$ ）

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int oo = 0xffffffff;
5  const int N = 1010;
6  int a[N] , sg[N][N];
7  int calsg(int i , int j) {
8      if (sg[i][j] != -1) return sg[i][j];
9      sg[i][j] = 0;
10     set<int> rec;
11     for (int x = 1; x < i; x++)
12         rec.insert(calsg(x , i - x));
13     for (int x = 1; x < j; x++)
14         rec.insert(calsg(x , j - x));
15     while (rec.count(sg[i][j]))sg[i][j]++;
16     return sg[i][j];
17 }
18 int calc2(int i , int j , int d) {
19     if (d == -1) return 0;
20     if ((i & (1 << d)) || (j & (1 << d))) {
21         int ans = calc2(i & ((1 << d) - 1), j & ((1 << d) - 1) , d - 1);
22         ans += ans == d;
23         return ans;
24     } else return calc2(i , j , d - 1);
25 }
26 void work(int testNo)
27 {
28     int n; cin >> n;
29     memset(sg , -1 , sizeof sg);
30     int ans = 0;
31     for (int i = 0; i < n; i += 2) {
32         int x , y;
33         cin >> x >> y;
34         ans ^= calc2(x - 1 , y - 1 , 30);
35     }
36     if (ans) cout << "YES\n";
37     else cout << "NO\n";
38 }
39 int main()
40 {
41     ios::sync_with_stdio(false);
42     cin.tie(0);
43     int t; cin >> t;
44     for (int i = 1; i <= t; i++)work(i);
45 }
46 /* stuff you should look for
47 * int overflow, array bounds

```

```

48 * special cases (n=1?)
49 * do smth instead of nothing and stay organized
50 * WRITE STUFF DOWN
51 * DON'T GET STUCK ON ONE APPROACH
52 */

```

D. A Wide, Wide Graph

<https://codeforces.com/contest/1805/problem/D>

You are given a tree (a connected graph without cycles) with n vertices.

Consider a fixed integer k . Then, the graph G_k is an undirected graph with n vertices, where an edge between vertices u and v exists if and only if the distance between vertices u and v in the given tree is **at least** k .

For each k from 1 to n , print the number of connected components in the graph G_k .

solve

最特殊的，关注树的直径。发现 k 变大的过程中，如果一个点脱离了图，就将变成孤立点。

某点是否变成孤立点的结论是：

1. 到两端点的最大距离小于 k .

于是问题转变成了：求点到其它点的最大距离，其实就是该点到直径两端点的距离的较大值：

有更简单的方式求：但用换根dp做，会使用到换根dp的经典内容，经典优化：\

指标函数定义

d_i 以1为根的树中， d 到该子树点的最大距离。

par_i 以该点父亲为一个根，除 i 子树点，所有点作为其子树中的点，该子树的最大高度（根据阶段而定。类似滚动数组，存储空间重利用。）

状态转移：

d_i dfs扫一遍即可

par_i

1. max比较对象：
 1. 父亲的父亲
 2. i 的兄弟。
2. 上述信息可以通过前后缀和维护。

初始化：

1. 根节点的没有父亲。 $par[1] = -1$
2. 前后缀初始化负数.因为可能没有。

code

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 const int N = 1E6 + 10;
5 vector<int> g[N];
6 int d[N];

```

```

7  int pa[N];
8  int sum[N];
9  //子树深度
10 void dfs(int u, int fa) {
11     d[u] = 0;
12     for (auto v : g[u]) {
13         if (fa != v) {
14             dfs(v, u);
15             d[u] = max(d[v] + 1, d[u]);
16         }
17     }
18 }
19 //换根dp;
20 void dfs2(int u, int fa) {
21
22     int sz = g[u].size();
23     vector<int>pre(sz + 5, -1), suf(sz + 5, -1);
24     d[u] = max(d[u], pa[fa] + 1);
25     for (int i = 1; i <= sz; i++) {
26         int v = g[u][i - 1];
27         if (fa != v) {
28             pre[i] = max(pre[i - 1], d[g[u][i - 1]]);
29         }
30         else pre[i] = pre[i - 1];
31     }
32     for (int i = sz; i >= 1; i--) {
33         int v = g[u][i - 1];
34         if (fa != v) {
35             suf[i] = max(suf[i + 1], d[g[u][i - 1]]);
36         }
37         else suf[i] = suf[i + 1];
38     }
39     for (int i = 1; i <= sz; i++) {
40         int v = g[u][i - 1];
41         if (fa != v) {
42             pa[u] = max({ pre[i - 1] , suf[i + 1] , pa[fa] }) + 1;
43             dfs2(g[u][i - 1], u);
44         }
45     }
46 }
47 int main()
48 {
49     ios::sync_with_stdio(false);
50     cin.tie(0);
51
52     int n; cin >> n;
53     for (int i = 1; i < n; i++) {
54         int u, v;
55         cin >> u >> v;
56         g[u].push_back(v);
57         g[v].push_back(u);
58     }
59     dfs(1, 0);
60     pa[0] = -1;
61     dfs2(1, 0);

```

```

62     for (int i = 1; i <= n; i++) {
63         sum[d[i] + 1]++;
64     }
65     for (int i = 1; i <= n; i++) {
66         sum[i] = sum[i - 1] + sum[i];
67         cout << min(1 + sum[i], n) << " \n"[i == n];
68     }
69 }

```

树的直径

概念

直径：两点之间的最大距离。

求直径的方法：

1. 树形dp求直接直径：
2. 两次bfs求直径：

拓展

1. 给定一个点，求出其与其它点之间的最大距离：
 1. 换根dp求法：比较困难。
 2. 从直径上的两个点开始进行bfs。求出最大距离。

bfs方法下求

依然是下述的例题：

jis代码

```

1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  int main() {
6      std::ios::sync_with_stdio(false);
7      std::cin.tie(nullptr);
8
9      int n;
10     std::cin >> n;
11
12     std::vector<std::vector<int>> adj(n);
13     for (int i = 1; i < n; i++) {
14         int u, v;
15         std::cin >> u >> v;
16         u--, v--;
17         adj[u].push_back(v);
18         adj[v].push_back(u);
19     }
20

```

```

21     std::vector<int> d(n, -1);
22     auto bfs = [&](int x) {
23         std::queue<int> q;
24         d.assign(n, -1);
25         q.push(x);
26         d[x] = 0;
27
28         while (!q.empty()) {
29             int x = q.front();
30             q.pop();
31
32             for (auto y : adj[x]) {
33                 if (d[y] == -1) {
34                     d[y] = d[x] + 1;
35                     q.push(y);
36                 }
37             }
38         }
39
40         return std::max_element(d.begin(), d.end()) - d.begin();
41     };
42
43     int x = bfs(0);
44     int y = bfs(x);
45     auto dx = d;
46     bfs(y);
47     auto dy = d;
48
49     std::vector<int> ans(n + 1);
50     for (int i = 0; i < n; i++) {
51         ans[std::max(dx[i], dy[i]) + 1] += 1;
52     }
53     ans[0] += 1;
54     ans[dx[y] + 1] -= 1;
55     for (int i = 1; i <= n; i++) {
56         ans[i] += ans[i - 1];
57     }
58     for (int i = 1; i <= n; i++) {
59         std::cout << ans[i] << " \n"[i == n];
60     }
61
62     return 0;
63 }

```

例题：

D. A Wide, Wide Graph

[Problem - D - Codeforces](#)

[换根dp求树直径.md](#)

换根dp处理给定一个点，到其它点的最大距离：

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int N = 1E6 + 10;
5  vector<int> g[N];
6  int d[N];
7  int pa[N];
8  int sum[N];
9  //子树深度
10 void dfs(int u, int fa) {
11     d[u] = 0;
12     for (auto v : g[u]) {
13         if (fa != v) {
14             dfs(v, u);
15             d[u] = max(d[v] + 1, d[u]);
16         }
17     }
18 }
19 //换根部dp;
20 void dfs2(int u, int fa) {
21     int sz = g[u].size();
22     vector<int>pre(sz + 5, -1), suf(sz + 5, -1);
23     d[u] = max(d[u], pa[fa] + 1);
24     for (int i = 1; i <= sz; i++) {
25         int v = g[u][i - 1];
26         if (fa != v) {
27             pre[i] = max(pre[i - 1], d[g[u][i - 1]]);
28         }
29         else pre[i] = pre[i - 1];
30     }
31     for (int i = sz; i >= 1; i--) {
32         int v = g[u][i - 1];
33         if (fa != v) {
34             suf[i] = max(suf[i + 1], d[g[u][i - 1]]);
35         }
36         else suf[i] = suf[i + 1];
37     }
38     for (int i = 1; i <= sz; i++) {
39         int v = g[u][i - 1];
40         if (fa != v) {
41             pa[u] = max({ pre[i - 1], suf[i + 1], pa[fa] }) + 1;
42             dfs2(g[u][i - 1], u);
43         }
44     }
45 }
46 }
47 int main()
48 {
49     ios::sync_with_stdio(false);
50     cin.tie(0);
51
52     int n; cin >> n;
53     for (int i = 1; i < n; i++) {

```

```

54     int u, v;
55     cin >> u >> v;
56     g[u].push_back(v);
57     g[v].push_back(u);
58 }
59 dfs(1, 0);
60 // for (int i = 1; i <= n; i++) {
61 //     cout << d[i] << " \n"[i == n];
62 // }
63 pa[0] = -1;
64 dfs2(1, 0);
65 for (int i = 1; i <= n; i++) {
66     sum[d[i] + 1]++;
67 }
68 // for (int i = 1; i <= n; i++) {
69 //     cout << d[i] << " \n"[i == n];
70 // }
71 for (int i = 1; i <= n; i++) {
72     sum[i] = sum[i - 1] + sum[i];
73     cout << min(1 + sum[i], n) << " \n"[i == n];
74 }
75 }
76 /* stuff you should look for
77 * int overflow, array bounds
78 * special cases (n=1?)
79 * do smth instead of nothing and stay organized
80 * WRITE STUFF DOWN
81 * DON'T GET STUCK ON ONE APPROACH
82 */

```

D. The Butcher

<https://codeforces.com/contest/1820/problem/D>

Anton plays his favorite game "Defense of The Ancients 2" for his favorite hero — The Butcher. Now he wants to make his own dinner. To do this he will take a rectangle of height h and width w , then make a vertical or horizontal cut so that both resulting parts have integer sides. After that, he will put one of the parts in the box and cut the other again, and so on.

More formally, a rectangle of size $h \times w$ can be cut into two parts of sizes $x \times w$ and $(h - x) \times w$, where x is an integer from 1 to $(h - 1)$, or into two parts of sizes $h \times y$ and $h \times (w - y)$, where y is an integer from 1 to $(w - 1)$.

He will repeat this operation $n - 1$ times, and then put the remaining rectangle into the box too. Thus, the box will contain n rectangles, of which $n - 1$ rectangles were put in the box as a result of the cuts, and the n -th rectangle is the one that the Butcher has left after all $n - 1$ cuts.

Unfortunately, Butcher forgot the numbers h and w , but he still has n rectangles mixed in random order. Note that Butcher **didn't rotate the rectangles**, but only shuffled them. Now he wants to know all possible pairs (h, w) from which this set of rectangles can be obtained. And you have to help him do it!

It is guaranteed that there exists at least one pair (h, w) from which this set of rectangles can be obtained.

solve

几个关键：

1. 该矩形的面积是确定的。
2. 特殊的裁剪方式，并且丢弃其中一部分。

考虑第一步：

1. 最终有两种可能，沿着高裁剪，沿着宽裁剪：
 1. 两种裁剪方式下，导致了一些结果：最大高，或者最大宽。
2. 然后根据上述的结论，可以明确：
 1. 枚举两种可能，最大高 最大宽之后，由于面积确定。宽与高就确定了。
 2. 所以一共只有两种可能结果，最后就是check这两种结果的可行性即可。
3. check设计：(从几个角度入手：)
 - 1.

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  typedef pair<ll , ll>p11;
5  #define fi first
6  #define se second
7  const int oo = 0xffffffff;
8  const int N = 1E6 + 10;
9  int n;
10
11 bool check(ll H , ll w , multiset<p11 , greater<p11> > h , multiset<p11 ,
    greater<p11> > w) {
12     //维护当前的两种情形。
13     //交替变换处理怎么实现？
14     // cout << "\n\n\n";
15     for (int i = 0 ; i < n; i++) {
16         p11 a = *h.begin();
17         p11 b = *w.begin();
18         // cout << H << " " << w << "\n";
19         if (H == a.fi) {
20             w -= a.se;
21             h.erase(h.begin());
22             w.erase(w.lower_bound({a.se , a.fi}));
23         } else if (w == b.fi) {
24             H -= b.se;
25             w.erase(w.begin());
26             h.erase(h.lower_bound({b.se , b.fi}));
27         } else return false;
28     }
29     return true;
30 }
31
32 void work(int testNo)
33 {

```

```

34     cin >> n;
35     multiset<pll , greater<pll> >h1 , w1;
36     ll s = 0;
37     for (int i = 0; i < n; i++) {
38         ll h , w;
39         cin >> h >> w;
40         s += h * w;
41         h1.insert({h , w});
42         w1.insert({w , h});
43     }
44     ll mxh = h1.begin()->fi;
45     ll mxw = w1.begin()->fi;
46     set<pll> ans;
47     if (s % mxh == 0 && check(mxh , s / mxh , h1 , w1)) {
48         ans.insert({mxh , s / mxh});
49     }
50     if (s % mxw == 0 && check(s / mxw , mxw , h1 , w1)) {
51         ans.insert({s / mxw , mxw});
52     }
53     cout << ans.size() << "\n";
54     for (auto [x , y] : ans) {
55         cout << x << " " << y << "\n";
56     }
57 }
58
59 int main()
60 {
61     ios::sync_with_stdio(false);
62     cin.tie(0);
63
64     int t; cin >> t;
65     for (int i = 1; i <= t; i++)work(i);
66 }
67
68 /* stuff you should look for
69 * int overflow, array bounds
70 * special cases (n=1?)
71 * do smth instead of nothing and stay organized
72 * WRITE STUFF DOWN
73 * DON'T GET STUCK ON ONE APPROACH
74 */

```

B. Letter Exchange

<https://codeforces.com/contest/1785/problem/B>

t宝的题!!!

A cooperative game is played by m people. In the game, there are $3m$ sheets of paper: m sheets with letter 'w', m sheets with letter 'i', and m sheets with letter 'n'.

Initially, each person is given three sheets (possibly with equal letters).

The goal of the game is to allow each of the m people to spell the word "win" using their sheets of paper. In other words, everyone should have one sheet with letter 'w', one sheet with letter 'i', and one sheet with letter 'n'.

To achieve the goal, people can make *exchanges*. Two people participate in each exchange. Both of them choose exactly one sheet of paper from the three sheets they own and exchange it with each other.

Find the shortest sequence of exchanges after which everyone has one 'w', one 'i', and one 'n'.

solve

容易将人的持牌情况分类：

策略如下： 关注一个人缺什么牌，多了什么牌。

1. 建立一个图描述其中的关系：

1. 如果i 缺y 多x 建立一条边, $x \rightarrow y$. 并且将这条边标记为i.

2. 基于上图的一个结论：

1. 如果同时存在 $x \rightarrow y, y \rightarrow x$. 交换两个人的x, y 牌。

2. 上述1不断执行完之后， 剩下了若干个三元环。通过两次交换即可。

这个结论怎么这么熟悉呀？ 得证证啦！

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int N = 1E6 + 10;
5  map<char, int> mp;
6  map<int, char> restore;
7  vector<int> g[3][3];
8  int ans[N][4], tot;
9  void add(int a, int b, int c, int d) {
10     ++tot;
11     ans[tot][0] = a;
12     ans[tot][1] = b;
13     ans[tot][2] = c;
14     ans[tot][3] = d;
15 }
16 void solve() {
17     tot = 0;
18     int n; cin >> n;
19     for (int i = 0; i < 3; i++) {
20         for (int j = 0; j < 3; j++) {
21             g[i][j].clear();
22         }

```

```

23     }
24     for (int i = 1; i <= n; i++) {
25         int c[3] {};
26         string s;
27         cin >> s;
28         for (int j = 0; j < 3; j++) {
29             c[mp[s[j]]]++;
30         }
31         //建图。
32         for (int j = 0; j < 3; j++)
33             for (int k = 0; k < 3; k++) {
34                 if (c[j] > 1 && c[k] < 1)g[j][k].push_back(i);
35             }
36     }
37     //建图完成。
38     //然后先处理
39     for (int i = 0; i < 3; i++)
40         for (int j = 0; j < 3; j++) {
41             while (g[i][j].size() && g[j][i].size()) {
42                 add(g[i][j].back(), i, g[j][i].back(), j);
43                 g[i][j].pop_back();
44                 g[j][i].pop_back();
45             }
46         }
47     //接着处理答辨
48     int u = 0, v = 1, w = 2;
49     if (g[v][u].size()) {
50         swap(u, v);
51     };
52     for (int i = 0; i < (int)g[u][v].size(); i++) {
53         int a = g[u][v][i];
54         int b = g[v][w][i];
55         int c = g[w][u][i];
56         add(a, u, b, v);
57         add(b, u, c, w);
58     }
59     cout << tot << "\n";
60     for (int i = 1; i <= tot; i++) {
61         // cout << ans[i][0] << " " << ans[i][1] << " " << ans[i][2] << " "
62         << ans[i][3] << "\n";
63         cout << ans[i][0] << " " << restore[ans[i][1]] << " " << ans[i][2]
64         << " " << restore[ans[i][3]] << "\n";
65     }
66 }
67 int main()
68 {
69     ios::sync_with_stdio(false);
70     cin.tie(0);
71     mp['w'] = 0;
72     restore[0] = 'w';
73     mp['i'] = 1;
74     restore[1] = 'i';
75     mp['n'] = 2;
76     restore[2] = 'n';
77     int t; cin >> t;

```

```

76     for (int i = 1; i <= t; i++) solve();
77 }

```

King's Puzzle

King's Puzzle

Input file: `standard input`
 Output file: `standard output`
 Time limit: 3 seconds
 Memory limit: 1024 megabytes

King Kendrick is a sovereign ruler of Kotlin Kingdom. He is getting ready for the next session of the government. Kotlin Kingdom consists of n cities. These cities need to be connected by several bidirectional roads. Since ministries are responsible for aspects of safety and comfort of the kingdom's residents, some of them have made the following requirements:

- “All the cities should be connected by new roads, i.e. there should be a path from any city to any other city via the roads” — Ministry of Transport and Digital Infrastructure.
- “There may not be a loop road — a road that connects a city with itself” — Ministry of Environment.
- “There should be at most one road between a pair of cities” — Treasury Department.
- “If a_i is the number of roads connected to i -th city, then the set $\{a_1, \dots, a_n\}$ should consist of exactly k distinct numbers” — Ministry of ICPC.

King Kendrick has issues with the requirements from the Ministry of ICPC. He asks you to help him. Find any set of roads that suits all the requirements above or say that it is impossible.

solve

体现出几点特征：

1. 特殊性：
2. 简洁：
3. 有序：

从简单的起点开始构造：从平凡图开始

有序构造：处理若干割点：

特殊性：度数的值域连续：接近 $1 \dots k$

然后方案如下：（有解前提下：）

1. 1 -> 度数为 k 。向后边连边：
2. 2 -> 度数为 $k - 1$ ，向后边连边。

然后按照这种规律，逐渐的把所有的点处理完。

最后剩下的点，并入一个第一个点中。（妙）

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  int main()

```

```

7  {
8      ios::sync_with_stdio(false);
9      cin.tie(0);
10     int n , k;
11     cin >> n >> k;
12     if (n == 1 && k == 1) {
13         cout << "YES\n";
14         cout << 0 << "\n";
15         return 0;
16     }
17     if (k >= n) {
18         cout << "NO\n";
19         return 0;
20     }
21     cout << "YES\n";
22     if (k == 1 && n == 2) {
23         cout << 1 << "\n";
24         cout << 1 << " " << 2 << "\n";
25     }
26     else if (k == 1) {
27         cout << n << "\n";
28         for (int i = 1; i <= n; i++) {
29             cout << i << " " << i % n + 1 << "\n";
30         }
31     }
32     else {
33         vector<pair<int , int>> ans;
34         int d = k;
35         for (int i = 1; i <= k ; i++ , d -= 2) {
36             for (int j = 1; j <= d; j++) {
37                 ans.emplace_back(i , i + j);
38             }
39         }
40         for (int i = k + 2; i <= n; i++) {
41             ans.emplace_back(1 , i);
42         }
43         // set<int> rec;
44         // vector<int >deg(n + 1, 0);
45         // for (auto [x , y] : ans) {
46         //     deg[x]++;
47         //     deg[y]++;
48         // }
49         // for (int i = 1; i <= n; i++) {
50         //     rec.insert(deg[i]);
51         // }
52         // cout << rec.size() << "\n";
53         cout << ans.size() << "\n";
54         for (auto [x , y] : ans) {
55             cout << x << " " << y << "\n";
56         }
57     }
58 }
59 /* stuff you should look for
60 * int overflow, array bounds
61 * special cases (n=1?)

```

```

62  * do smth instead of nothing and stay organized
63  * WRITE STUFF DOWN
64  * DON'T GET STUCK ON ONE APPROACH
65  */

```

C. Recover an RBS

Problem - 1709C - Codeforces

A bracket sequence is a string containing only characters "(" and ")". A regular bracket sequence (or, shortly, an RBS) is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters "1" and "+" between the original characters of the sequence. For example:

- bracket sequences "()" and "(())" are regular (the resulting expressions are "(1)+(1)" and "((1+1)+1)");
- bracket sequences ")", "(", "(" and ")" are not.

There was an RBS. Some brackets have been replaced with question marks. Is it true that there is a **unique** way to replace question marks with brackets, so that the resulting sequence is an RBS?

Input

The first line contains a single integer t ($1 \leq t \leq 5 \cdot 10^4$) — the number of testcases.

The only line of each testcase contains an RBS with some brackets replaced with question marks. Each character is either '(', ')' or '?'. At least one RBS can be recovered from the given sequence.

The total length of the sequences over all testcases doesn't exceed $2 \cdot 10^5$.

Output

For each testcase, print "YES" if the way to replace question marks with brackets, so that the resulting sequence is an RBS, is **unique**. If there is more than one way, then print "NO".

→ Virt

Virtual (past co-particip. ICPC m-seen th-not for archive. problem is not fo archive. read the other pr

→ Clo

You can

solve

序号模型：

1. 追求括号合法性：

1. 能放（就放（。如果必然有合法解的话，当我们无法构造合法化，是因为）不足。应该尽量将）放在后面。

2. 判断合法性的方法：

1. 将（抽象为1.
2. 将）抽象为-1.
3. 关注前缀和。合法要保证每一段前缀和都要大于等于0.

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  using ll = long long;
5  using i64 = long long;
6  using ull = unsigned long long;
7  using ld = long double;
8  using uint = unsigned int;
9  using pii = pair<int , int>;
10 using pli = pair<ll , int>;
11 using pll = pair<ll , ll>;
12
13
14 #define dbg(x) cerr << "[" << __LINE__ << "]" << ": " << x << "\n"
15
16 #define pb push_back
17 #define all(x) (x).begin(),(x).end()
18 #define fi first

```

```

19 #define se second
20 #define sz(x) (int)(x).size()
21
22 const int inf = 1 << 29;
23 const ll INF = 1LL << 60;
24 const int N = 1E6 + 10;
25
26 int a[N];
27
28 void work(int testNo)
29 {
30     string s;
31     cin >> s;
32     int n = sz(s);
33     int l = 0 , r = 0;
34     int low = -1 , high = inf;
35
36     for (auto t : s) {
37         if (t == ')') r++;
38         else if (t == '(') l++;
39     }
40     // dbg(n);
41     // dbg(l);
42     for (int i = 0; i < n; i++) {
43         if (s[i] == '(') {
44             a[i] = 1;
45         } else if (s[i] == ')') {
46             a[i] = -1;
47         } else if (s[i] == '?') {
48             if (l < n / 2) {
49                 low = max(low , i);
50                 // dbg(low);
51                 l++;
52                 a[i] = 1;
53             } else {
54                 high = min(i , high);
55                 r++;
56                 a[i] = -1;
57             }
58         }
59     }
60     // dbg(high);
61     // dbg(low);
62     // dbg(high);
63     if (low == -1 || high == inf) {
64         cout << "YES\n";
65         return;
66     }
67     swap(a[low] , a[high]);
68     int sum = 0;
69     for (int i = 0; i < n; i++) {
70         sum = sum + a[i];
71         if (sum < 0) {
72             cout << "YES\n";
73             return;

```



```

74     }
75     }
76     cout << "NO\n";
77 }
78 signed main()
79 {
80     ios::sync_with_stdio(false);
81     cin.tie(0);
82
83     int t; cin >> t;
84     for (int i = 1; i <= t; i++)work(i);
85 }
86
87 /* stuff you should look for
88 * int overflow, array bounds
89 * special cases (n=1?)
90 * do smth instead of nothing and stay organized
91 * WRITE STUFF DOWN
92 * DON'T GET STUCK ON ONE APPROACH
93 */

```

D. Super-Permutation

<https://codeforces.com/contest/1822/problem/D>

A permutation is a sequence n integers, where each integer from 1 to n appears exactly once. For example, $[1]$, $[3, 5, 2, 1, 4]$, $[1, 3, 2]$ are permutations, while $[2, 3, 2]$, $[4, 3, 1]$, $[0]$ are not.

Given a permutation a , we construct an array b , where $b_i = (a_1 + a_2 + \dots + a_i) \bmod n$.

A permutation of numbers $[a_1, a_2, \dots, a_n]$ is called a *super-permutation* if $[b_1 + 1, b_2 + 1, \dots, b_n + 1]$ is also a permutation of length n .

Grisha became interested whether a *super-permutation* of length n exists. Help him solve this non-trivial problem. Output any *super-permutation* of length n , if it exists. Otherwise, output -1 .

solve

分类讨论：

1. 奇数的解存在情况：

1. 1显然有解
2. 其余奇数无解。容易证明，至少有两个位置的前缀和是 n 的倍数。

2. 偶数的情况

1. 初始 $n - 1$. 然后奇数，偶数分类。详细处理看下代码。一种左右对称构造的想法。

code

```

1 void work(int testNo)
2 {
3     int n;
4     cin >> n;
5     if (n == 1) {
6         cout << 1 << "\n";
7         return;
8     }

```

```

9      if (n % 2) {
10         cout << -1 << "\n";
11         return;
12     }
13     ans[1] = n;
14     ans[2] = n - 1;
15     for (int i = 3; i <= n; i++) {
16         if (i % 2) ans[i] = n + 1 - ans[i - 1];
17         else ans[i] = n - 1 - ans[i - 1];
18     }
19     for (int i = 1; i <= n; i++)
20         cout << ans[i] << " \n"[i == n];
21 }

```

D. Unique Palindromes

<https://codeforces.com/contest/1823/problem/D>

A *palindrome* is a string that reads the same backwards as forwards. For example, the string `abcba` is palindrome, while the string `abca` is not.

Let $p(t)$ be the number of *unique palindromic substrings* of string t , i. e. the number of substrings $t[l \dots r]$ that are palindromes themselves. Even if some substring occurs in t several times, it's counted exactly once. (The whole string t is also counted as a substring of t).

For example, string $t = \text{abcbcbcabcb}$ has $p(t) = 6$ unique palindromic substrings: `a`, `b`, `c`, `bb`, `bcb` and `cbbc`.

Now, let's define $p(s, m) = p(t)$ where $t = s[1 \dots m]$. In other words, $p(s, m)$ is the number of palindromic substrings in the prefix of s of length m . For example, $p(\text{abcbcbcabcb}, 5) = p(\text{abcbcb}) = 5$.

You are given an integer n and k "conditions" ($k \leq 20$). Let's say that string s , consisting of n lowercase Latin letters, is **good** if all k conditions are satisfied **at the same time**. A condition is a pair (x_i, c_i) and have the following meaning:

- $p(s, x_i) = c_i$, i. e. a prefix of s of length x_i contains exactly c_i unique palindromic substrings.

Find a good string s or report that such s doesn't exist.

Look in Notes if you need further clarifications.

solve

首先注意到：

1. 长度为 n 的字符串中，本质不同的回文子串最多有 n 个。
2. 增加字符串的过程中，可以控制回文串的种数增加0或1。
3. 对于每一个段，可以分段构造贡献：

对每一段贡献为了防止前后产生干扰：

1. 使用循环串选择段落尾： `abcbabc.....`

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  using ll = long long;
5  using i64 = long long;
6  using ull = unsigned long long;
7  using ld = long double;
8  using uint = unsigned int;
9  using pii = pair<int, int>;
10 using pli = pair<ll, int>;

```

```

11 using pll = pair<ll, ll>;
12
13
14 #define dbg(x) cerr << "[" << __LINE__ << "]" << ": " << x << "\n"
15
16 #define all(x) (x).begin(),(x).end()
17 #define sz(x) (int)(x).size()
18 #define pb push_back
19 #define fi first
20 #define se second
21
22 const int inf = 1 << 29;
23 const ll INF = 1LL << 60;
24 const int N = 1E6 + 10;
25
26 int a[N], b[N];
27
28 void work(int testNo)
29 {
30     int n, k;
31     cin >> n >> k;
32     for (int i = 1; i <= k; i++) {
33         cin >> a[i];
34     }
35     for (int i = 1; i <= k; i++) {
36         cin >> b[i];
37     }
38     string res;
39     string rem;
40     while (sz(rem) <= n) rem += "abc";
41     int st = 0;
42     char cur = 'd';
43     for (int i = 1; i <= k; i++, cur++) {
44         //第一段位置， 特殊的情况怎么解决？
45         //题中的数据比较特殊。
46         // dbg(res);
47         if (a[i] - a[i - 1] < b[i] - b[i - 1]) {
48             cout << "NO\n";
49             return;
50         }
51         int t = b[i] - b[i - 1];
52         if (i == 1) {
53             res += "abc";
54             st = 3;
55             t -= 3;
56             for (int j = 1; j <= t; j++)
57                 res += cur;
58             int d = a[i] - sz(res);
59             res += rem.substr(st, d);
60             st += d;
61         } else {
62             for (int j = 1; j <= t; j++)
63                 res += cur;
64             int d = a[i] - sz(res);
65             res += rem.substr(st, d);

```

```

66         st += d;
67     }
68 }
69 cout << "YES" << "\n";
70 cout << res << "\n";
71 assert(sz(res) == n);
72 }
73 signed main()
74 {
75     ios::sync_with_stdio(false);
76     cin.tie(0);
77
78     int t; cin >> t;
79     for (int i = 1; i <= t; i++) work(i);
80 }
81 /* stuff you should look for
82 * int overflow, array bounds
83 * special cases (n=1?)
84 * do smth instead of nothing and stay organized
85 * WRITE STUFF DOWN
86 * DON'T GET STUCK ON ONE APPROACH
87 */

```

C. Candy Store

<https://codeforces.com/contest/1798/problem/C>

The store sells n types of candies with numbers from 1 to n . One candy of type i costs b_i coins. In total, there are a_i candies of type i in the store.

You need to pack all available candies in packs, each pack should contain only one type of candies. Formally, for each type of candy i you need to choose the integer d_i , denoting the number of type i candies in one pack, so that a_i is divided without remainder by d_i .

Then the cost of one pack of candies of type i will be equal to $b_i \cdot d_i$. Let's denote this cost by c_i , that is, $c_i = b_i \cdot d_i$.

After packaging, packs will be placed on the shelf. Consider the cost of the packs placed on the shelf, in order c_1, c_2, \dots, c_n . Price tags will be used to describe costs of the packs. One price tag can describe the cost of all packs from l to r inclusive if $c_l = c_{l+1} = \dots = c_r$. Each of the packs from 1 to n must be described by at least one price tag. For example, if $c_1, \dots, c_n = [4, 4, 2, 4, 4]$, to describe all the packs, a 3 price tags will be enough, the first price tag describes the packs 1, 2, the second: 3, the third: 4, 5.

You are given the integers $a_1, b_1, a_2, b_2, \dots, a_n, b_n$. Your task is to choose integers d_i so that a_i is divisible by d_i for all i , and the required number of price tags to describe the values of c_1, c_2, \dots, c_n is the minimum possible.

For a better understanding of the statement, look at the illustration of the first test case of the first test:

20mins

?? 子问题:

1. 相当于一个分组问题: 对于一个数, 怎么确定同组元素?

1. 能选就选?

1. ?? 不知道: 该结论正确情况:

1. 手推: ??

2. 纯逻辑推导: ??

2. 如果允许，尽量往小了选。（这个是显然的，因为这样遇到了更小的数字时，可以获得更多的选择。）
2. 两个数可以经过因数的选取，归于同一个标签的技巧是什么？
1. 从因数的角度上看：

$$\begin{aligned} \text{对于数组 } i, j \text{ 都有} \\ a_i \% \text{lcm}(b_i, b_j) = 0 \\ a_j \% \text{lcm}(b_i, b_j) = 0 \end{aligned} \quad (2)$$

读假题啦！！这里的区间是可以连续的。

所以只需要集中精力解决：最长连续可用作同一个区间的问题即可。

solve

角度就是从最左端开始，能选就选。

抄大哥的博客：

对我来说是个有点新的题，以前没做过这样的，vp 时考虑了很久大力 dp。

考虑一段能共用同一个价签 c 的区间 $[l, r]$ ，有 $c = b_l d_l = b_{l+1} d_{l+1} = \dots b_r d_r$ 。于是显然 $\text{lcm}_{i=l}^r b_i | c$ 。

再考虑构造出的 $d_i | a_i$ ，而 $d_i = \frac{c}{b_i}$ 。于是 $\frac{c}{b_i} | a_i$ ，即 $c | a_i b_i$ 。于是 $c | \text{gcd}_{i=l}^r a_i b_i$ 。

于是一个区间能共用一个价签的条件就是 $\text{lcm}(b_l, b_{l+1}, \dots b_r) | \text{gcd}(a_l b_l, a_{l+1} b_{l+1}, \dots a_r b_r)$ 。

下一个问题是怎么安排价签。这是一个简单的贪心：只需要从最左端开始，每次尽可能地把多的糖果共用一个价签即可。证明上可以考虑如果为了后面某段区间调整了前面的价格，会多出一个区间端点，然后减少一个区间端点，答案不会变优。

[【CF1798C】Candy Store - 一扶苏一 的博客 - 洛谷博客 \(luogu.com.cn\)](#)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int oo = 0x0fffffff;
5  const int N = 1E6 + 10;
6  ll a[N], b[N];
7  ll gcd(ll x, ll y) {
8      return y == 0 ? x : gcd(y, x % y);
9  }
10 ll lcm(ll x, ll y) {
11     return x * y / gcd(x, y);
12 }
13 void work(int testNo)
14 {
15     int n;
16     cin >> n;
17     for (int i = 1; i <= n; i++)
18         cin >> a[i] >> b[i];
19     int ans = 0;
20     for (int i = 1; i <= n; i++) {
21         ans++;
    
```

```

22         ll x = a[i] * b[i], y = b[i];
23         for (int j = i + 1; j <= n; j++) {
24             x = gcd(x, a[j] * b[j]);
25             y = lcm(y, b[j]);
26             if (x % y) {
27                 i = j - 1;
28                 break;
29             }
30             if (j == n) {
31                 i = j;
32                 break;
33             }
34         }
35     }
36     cout << ans << "\n";
37 }
38 int main()
39 {
40     ios::sync_with_stdio(false);
41     cin.tie(0);
42
43     int t; cin >> t;
44     for (int i = 1; i <= t; i++) work(i);
45 }
46 /* stuff you should look for
47 * int overflow, array bounds
48 * special cases (n=1?)
49 * do smth instead of nothing and stay organized
50 * WRITE STUFF DOWN
51 * DON'T GET STUCK ON ONE APPROACH
52 */

```

D. Letter Picking

<https://codeforces.com/contest/1728/problem/D>

Alice and Bob are playing a game. Initially, they are given a non-empty string s , consisting of lowercase Latin letters. The length of the string is even. Each player also has a string of their own, initially empty.

Alice starts, then they alternate moves. In one move, a player takes either the first or the last letter of the string s , removes it from s and **prepends** (adds to the beginning) it to their own string.

The game ends when the string s becomes empty. The winner is the player with a lexicographically smaller string. If the players' strings are equal, then it's a draw.

A string a is lexicographically smaller than a string b if there exists such position i that $a_j = b_j$ for all $j < i$ and $a_i < b_i$.

What is the result of the game if both players play optimally (e. g. both players try to win; if they can't, then try to draw)?

solve

一般而言，当博弈的资源规模较小时，可以使用dp转移记录。上述就是例子：

状态定义

$f_{i,j}$ 表示当前选到 i, j 区间时候，先手的情况。为了方便转移，保证 $j - i + 1$ 为偶数长度。从而保证了当前的先手为alice。

其中其解有三种 0, 1, 2. 分别表示 先手输, 平, 胜

转移方程

基于一个状态, 枚举可能发生的情况: 两个人分别取

1. low, low + 1
2. low, high
3. high, low
4. high, high - 1

初始化

在这个例中, 与其想办法为, 最小状态转移得到的, 不能存在的状态分配一个合理值。不如直接初始化zui'xiao

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int oo = 0xffffffff;
5  const int N = 2E3 + 10;
6  int f[N][N];
7  void work(int testNo){
8      string s;
9      cin >> s;
10     const int n = s.size();
11     //0表示更小, 1, 表示更多。
12     for (int i = 1; i < n; i++) {
13         if (s[i] == s[i - 1])f[i - 1][i] = 1;
14         //表示更小。
15         //表示更大。
16         else f[i - 1][i] = 2;
17     }
18     for (int len = 4; len <= n; len += 2)
19         for (int low = 0, high = len - 1; high < n; low++, high++) {
20             //一共有四种选择情况。
21             //low low + 1
22             int t = f[low + 2][high];
23             int a = 0;
24             if (s[low] == s[low + 1]) {
25                 a = t;
26             } else if (s[low] < s[low + 1]) {
27                 if (t == 1 || t == 2) a = 2;
28                 else a = 0;
29             } else {
30                 if (t == 2) {
31                     a = 2;
32                 } else if (t == 1) a = 1;
33                 else a = 0;
34             }
35             t = f[low + 1][high - 1];
36             int b = 0;
37             if (s[low] == s[high]) {
38                 b = t;
39             } else if (s[low] < s[high]) {

```

```

40         if (t == 2 || t == 1) b = 2;
41         else b = 0;
42     } else {
43         if (t == 2) {
44             b = 2;
45         } else b = 0;
46     }
47     //第二大类。
48     t = f[low][high - 2];
49     int c = 0;
50     if (s[high] == s[high - 1]) {
51         c = t;
52     } else if (s[high] < s[high - 1]) {
53         if (t == 2 || t == 1) c = 2;
54         else c = 0;
55     } else {
56         if (t == 2) {
57             c = 2;
58         } else c = 0;
59     }
60     t = f[low + 1][high - 1];
61     int d = 0;
62     if (s[high] == s[low]) {
63         d = t;
64     } else if (s[high] < s[low]) {
65         if (t == 2 || t == 1) d = 2;
66         else d = 0;
67     } else {
68         if (t == 2) {
69             d = 2;
70         } else d = 0;
71     }
72     f[low][high] = max(min(a , b) , min(c , d));
73 }
74 if (f[0][n - 1] == 1) {
75     cout << "Draw\n";
76 } else cout << "Alice\n";
77 }
78 int main(){
79     ios::sync_with_stdio(false);
80     cin.tie(0);
81     int t; cin >> t;
82     for (int i = 1; i <= t; i++)work(i);
83 }
84 /* stuff you should look for
85 * int overflow, array bounds
86 * special cases (n=1?)
87 * do smth instead of nothing and stay organized
88 * WRITE STUFF DOWN
89 * DON'T GET STUCK ON ONE APPROACH
90 */

```


城市里的间谍：

紫书页号： p268

20mins

解决dp之前， 是否会使用暴力？

1. 怎么写一个暴力？

关注所有方案， 阶段中的一些标志性的量：

1. 时间发生的时间。
2. 当前在什么站点。

solve

这种面向一些现实中的模型的问题中， 时间是天然有序的。感受这种天然有序的模型性质的其它问题， 例如： [免费馅饼.md](#)

分阶段的关注角度：时间。

状态定义

$dp_{i,j}$ ， 表示， 当前处于i时刻， 间谍处于j站点的最小等待时间。

转移方程：

1. 等待一分钟。
2. 搭乘正向火车。
3. 搭乘反向火车。

初始化

1. 不存在解为无穷大。
2. 最小状态： $dp_{t,n} = 0$

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int Nmax = 100;
5  const int Tmax = 1000;
6  const int inf = 1e9 + 7;
7
8  int dt[Nmax] , tx[Nmax] , ty[Nmax];
9  int n , T , m1 , m2;
10 bool has_train[Tmax][Nmax][2];
11 int dp[Tmax][Nmax];
12 int sum[Nmax];
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(0);
16     int _ = 1;
17     while (cin >> n && n) {
18         int T;
19         cin >> T;
20         for (int i = 1; i < n; i++) {
21             cin >> dt[i];

```

```

22         sum[i + 1] = sum[i] + dt[i];
23     }
24     for (int i = T; i >= 0; i--)
25         for (int j = 1; j <= n; j++) {
26             has_train[i][j][0] = has_train[i][j][1] = false;
27         }
28     cin >> m1;
29     for (int i = 1; i <= m1; i++) {
30         int t;
31
32         cin >> t;
33         for (int j = 1; j <= n; j++) {
34             // cout << sum[j] + t << " \n"[j == n];
35             if (sum[j] + t > T) continue;
36             has_train[sum[j] + t][j][0] = true;
37         }
38     }
39     cin >> m2;
40     for (int i = 1; i <= m2; i++) {
41         int t;
42         cin >> t;
43         for (int j = n; j >= 1; j--) {
44             // cout << sum[n] - sum[j] + t << " \n"[j == 1];
45             if (sum[n] - sum[j] + t > T) continue;
46             has_train[sum[n] - sum[j] + t][j][1] = true;
47         }
48     }
49     for (int i = 1; i < n; i++)
50         dp[T][i] = inf;
51     dp[T][n] = 0;
52     for (int i = T - 1; i >= 0; i--)
53         for (int j = 1; j <= n; j++) {
54             dp[i][j] = dp[i + 1][j] + 1;
55             if (j < n && has_train[i][j][0] && i + dt[j] <= T) {
56                 dp[i][j] = min(dp[i][j], dp[i + dt[j]][j + 1]); //右
57             }
58             if (j > 1 && has_train[i][j][1] && i + dt[j - 1] <= T)
59                 dp[i][j] = min(dp[i][j], dp[i + dt[j - 1]][j - 1]); //
左
60         }
61     cout << "Case Number " << _++ << ": ";
62     if (dp[0][1] >= inf) cout << "impossible\n";
63     else cout << dp[0][1] << "\n";
64 }
65 }
66 /* stuff you should look for
67 * int overflow, array bounds
68 * special cases (n=1?)
69 * do smth instead of nothing and stay organized
70 * WRITE STUFF DOWN
71 * DON'T GET STUCK ON ONE APPROACH
72 */

```

John Doe, a skilled pilot, enjoys traveling. While on vacation, he rents a small plane and starts visiting beautiful places. To save money, John must determine the shortest closed tour that connects his destinations. Each destination is represented by a point in the plane $p_i = \langle x_i, y_i \rangle$. John uses the following strategy: he starts from the leftmost point, then he goes strictly left to right to the rightmost point, and then he goes strictly right back to the starting point. It is known that the points have distinct x -coordinates.

Write a program that, given a set of n points in the plane, computes the shortest closed tour that connects the points according to John's strategy.

Input

The program input is from a text file. Each data set in the file stands for a particular set of points. For each set of points the data set contains the number of points, and the point coordinates in ascending order of the x coordinate. White spaces can occur freely in input. The input data are correct.

Output

For each set of data, your program should print the result to the standard output from the beginning of a line. The tour length, a floating-point number with two fractional digits, represents the result.

Note: An input/output sample is in the table below. Here there are two data sets. The first one contains 3 points specified by their x and y coordinates. The second point, for example, has the x coordinate 2, and the y coordinate 3. The result for each data set is the tour length, (6.47 for the first data set in the given example).

solve

1. 问题可以等闲成，两个人从同一个起点出发。达到终点。途中，两人的经过点除了起点终点外不相同。

状态设计：

$dp_{i,j}$ 表示，两个人到达分别到达 $\max(i, j)$ 个点时，其经过的最短距离。

转移方程：

1. 一种处理技巧是：规定 i 大于 j 。 $dp_{i,j}$ 和 $dp_{j,i}$ 等效性启发下的一个处理。
2. 因为途中的点都将被经历，每一步决定谁先走到 $i + 1$ 。类似于完全背包的优化思想
3. 从小贡献大状态的方向更加容易处理。

初始化：

1. $dp[1][2] = g(1, 2)$

生长思考：

1. 基于等效性的启发的体会：

1. 其作用：

1. 优化常数：/2
2. 降低编码难度。不需要分类讨论谁大谁小。

2. 原理体会：

1. 关于两个问题，它们是等效的

1. 它们的大小相同：问题的解系中存在单射，满射的关系。
2. 它们对更大规模问题的贡献相同，两者其中一个有贡献计算记录即可。

2. 基于状态设计角度的体会：

1. 类比完全背包问题：

1. 选择1，2，3步，等效于当前选择1步进入下一个状态。
2. 这种设计角度保证 $\langle \max(i, j) \rangle$ 的所有点经过。从而不需要具体研究解集点的经历情况。

3. 并不可以往回走，跳一大步之后的转移是可预测的。
4. 转换思维：将问题分为两种阶段：分化成两个人的交互移动问题。像传纸条也有类似的，双向搜索有着类似的思想精华；

code

[Source code - Virtual Judge \(csgrandeur.cn\)](https://csgrandeur.cn)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  #define fi first
6  #define se second
7
8  const int N = 1100;
9  const double inf = 1E16;
10 pair<double , double> c[N];
11 int tot;
12 double f[N][N];
13
14 double g(int i , int j) {
15     double dx = c[i].fi - c[j].fi;
16     double dy = c[i].se - c[j].se;
17     return sqrt(dx * dx + dy * dy);
18 }
19
20 int main()
21 {
22     ios::sync_with_stdio(false);
23     cin.tie(0);
24
25     int n;
26     while (cin >> n) {
27         tot = 0;
28         for (int i = 1; i <= n; i++) {
29             double x , y;
30             cin >> x >> y;
31             c[++tot] = make_pair(x, y);
32         }
33         for (int i = 1; i <= n; i++)
34             for (int j = 1; j <= n; j++) {
35                 f[i][j] = inf;
36             }
37         f[1][1] = 0;
38         f[2][1] = g(2 , 1);
39         // cout << f[2][1] << "\n";
40         for (int i = 1; i <= n; i++) {
41             for (int j = 1; j < i; j++) {
42                 f[i + 1][i] = min(f[i + 1][i] , f[i][j] + g(i + 1, j));
43                 f[i + 1][j] = min(f[i + 1][j] , f[i][j] + g(i , i + 1));
44             }
45         }
46         //那么最终的解是和哪一个指标函数有关呢？
47         for (int i = 1; n - i >= 1; i++) {

```

```

48         f[n][n] = min(f[n][n - i] + g(n, n - i) , f[n][n]);
49     }
50     cout << fixed << setprecision(2) << f[n][n] << "\n";
51 }
52 }
53 /* stuff you should look for
54  * int overflow, array bounds
55  * special cases (n=1?)
56  * do smth instead of nothing and stay organized
57  * WRITE STUFF DOWN
58  * DON'T GET STUCK ON ONE APPROACH
59  */

```

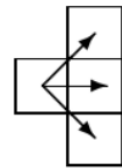
单向TSP

[Source code - Virtual Judge \(csgrandeur.cn\)](#)

Problems that require minimum paths through some domain appear in many different areas of computer science. For example, one of the constraints in VLSI routing problems is minimizing wire length. The Traveling Salesperson Problem (TSP) — finding whether all the cities in a salesperson's route can be visited exactly once with a specified limit on travel time — is one of the canonical examples of an NP-complete problem; solutions appear to require an inordinate amount of time to generate, but are simple to check.

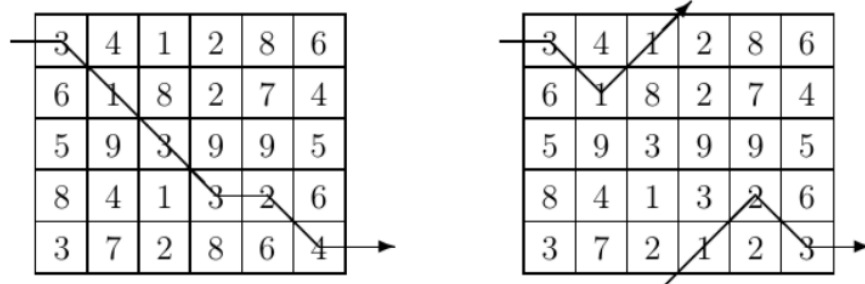
This problem deals with finding a minimal path through a grid of points while traveling only from left to right.

Given an $m \times n$ matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 1 (the first column) and consists of a sequence of steps terminating in column n (the last column). A step consists of traveling from column i to column $i + 1$ in an adjacent (horizontal or diagonal) row. The first and last rows (rows 1 and m) of a matrix are considered adjacent, i.e., the matrix “wraps” so that it represents a horizontal cylinder. Legal steps are illustrated on the right.



The *weight* of a path is the sum of the integers in each of the n cells of the matrix that are visited.

For example, two slightly different 5×6 matrices are shown below (the only difference is the numbers in the bottom row).



The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

[Unidirectional TSP - UVA 116 - Virtual Judge \(csgrandeur.cn\)](#)

solve

自己的:

正推转移:

状态定义:

$f_{i,j}$: 从第一列格子开始 到达 (i, j) 中最大经历权重和。

$pre_{i,j}, f_{i,j}$ 表示的最优方案中, i, j 点的前驱。

状态转移:

按照题意, 三个方向:

1. 就是三个状态转移的同时记录初始化点即可
2. 一个技巧: 循环可以通过mod运算天然地实现。

初始化 + 递归起点

将第一列的初始化即可。

书上, 逆推角度

状态定义:

$f_{i,j}$, 从格子 (i, j) 出发到最后一列的最小开销。

$nxt_{i,j}$, 记录最优解, 且最小的转移行号。

生长:

遇到非常多的问题:

1. 第一构造过程中, 保证每一个方案都是最优。除此之外, 比较还有一个维度, 题目中的问题是针对列的而不是针对某一个点, 在列集合中再额外完成一个还原比较字典序的工作。
2. 这个角度上, 逆推似乎恰到好处。

书上思路: code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E3 + 10;
6  const int inf = 1 << 29;
7
8  int a[N][N], f[N][N];
9  int nxt[N][N];
10
11
12 int main()
13 {
14     ios::sync_with_stdio(false);
15     cin.tie(0);
16     int n, m;
17     while (cin >> n >> m) {
18         for (int i = 1; i <= n; i++)
19             for (int j = 1; j <= m; j++) {
20                 cin >> a[i][j];
21             }
22         //尝试逆推实现
23         int ans = inf, first = 0;
24         for (int j = m; j > 0; j--) {
25             for (int i = 1; i <= n; i++) {
26                 if (j == m) f[i][j] = a[i][j];

```

```

27         else {
28             int rows[3] = {i , i - 1 , i + 1};
29             if (i == 1) rows[1] = n;
30             if (i == n) rows[2] = 1;
31             sort(rows , rows + 3);
32             f[i][j] = inf;
33             for (int k = 0; k < 3; k++) {
34                 int t = f[rows[k]][j + 1] + a[i][j];
35                 if (t < f[i][j]) f[i][j] = t , nxt[i][j] = rows[k];
36             }
37         }
38         if (j == 1 && f[i][j] < ans) {
39             ans = f[i][j];
40             first = i;
41         }
42     }
43 }
44 cout << first;
45 for (int i = nxt[first][1] , j = 2; j <= m; i = nxt[i][j] , j++) {
46     cout << " " << i;
47 }
48 cout << "\n" << ans << "\n";
49 }
50 }
51
52 /* stuff you should look for
53 * int overflow, array bounds
54 * special cases (n=1?)
55 * do smth instead of nothing and stay organized
56 * WRITE STUFF DOWN
57 * DON'T GET STUCK ON ONE APPROACH
58 */

```

code 自己的思路 wa了 (未解之谜)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  using pii = pair<int , int>;
5  const int N = 1E2 + 10;
6  const ll inf = 1LL << 40;
7
8  ll a[N][N] , f[N][N] , pre[N][N];
9  int n , m;
10 vector<int> g(int t) {
11     vector<int> ans;
12     for (int i = m; i >= 1; i--) {
13         ans.push_back(t);
14         t = pre[t][i];
15     }
16     reverse(ans.begin() , ans.end());
17     return ans;
18 }
19 bool cmp(const vector<int> & a , const vector<int> & b) {
20     for (int i = 0; i < m; i++) {

```

```

21     if (a[i] < b[i]) return true;
22     else if (a[i] > b[i]) return false;
23 }
24 return true;
25 }
26 int main()
27 {
28     ios::sync_with_stdio(false);
29     cin.tie(0);
30     while (cin >> n >> m) {
31         for (int i = 1; i <= n; i++)
32             for (int j = 1; j <= m; j++) {
33                 cin >> a[i][j];
34                 f[i][j] = inf;
35                 pre[i][j] = inf;
36             }
37         for (int i = 1; i <= n; i++) {
38             f[i][1] = a[i][1];
39             pre[i][1] = i;
40         }
41         //转移顺序: 先列在行
42         function<void(int , int , int)> update = [&](int i , int j, int x) {
43             if (x == n + 1) x = 1;
44             if (x == 0) x = n;
45             if (f[i][j] > f[x][j - 1] + a[i][j]) {
46                 f[i][j] = f[x][j - 1] + a[i][j];
47                 pre[i][j] = x;
48             } else if (f[i][j] == f[x][j - 1] + a[i][j]) {
49                 pre[i][j] = min(pre[i][j] , 1LL * x);
50             }
51         };
52         for (int j = 2; j <= m; j++) {
53             for (int i = 1; i <= n; i++) {
54                 for (int k = -1; k <= 1; k++) {
55                     update(i , j , i + k);
56                 }
57             }
58         }
59         vector<int> ans;
60         ll mi = inf;
61         for (int i = 1; i <= n; i++) {
62             if (f[i][m] > mi) continue;
63             vector<int> temp = g(i);
64             if (f[i][m] < mi) {
65                 ans = temp;
66                 mi = f[i][m];
67             }
68             else if (cmp(temp , ans)) ans = temp;
69         }
70         // cout << "d : " << cunt++ << "\n";
71         // for (int i = 1; i <= n; i++) {
72         //     cout << f[i][m] << " \n"[i == m];
73         // }
74         for (int i = 1; i <= m; i++) {
75             cout << ans[i - 1] << " \n"[i == m];

```



```

76     }
77     cout << mi << "\n";
78 }
79 }
80
81 /* stuff you should look for
82  * int overflow, array bounds
83  * special cases (n=1?)
84  * do smth instead of nothing and stay organized
85  * WRITE STUFF DOWN
86  * DON'T GET STUCK ON ONE APPROACH
87  */

```

学霸题，与原神の终极之战

题目来源：

1. bzoj
2. 省赛筛选赛问题：

introduces

N 个高楼排成一排，第 i 个楼高度为 D_i 。

初始你位于 1 号楼，原神位于第 N 个楼。

你要从 1 号楼前往 N 号楼找到原神并与之 PK。

你一共有 M 次轮回的机会，第 i 次轮回你会得到 k_i 的祝福。

当你位于 i 号楼时，你可以跳到第 $i + 1, \dots, i + k_i$ 个楼。当你跳到一个楼高大于等于当前楼的楼，你的疲劳值会增加 1，否则不会。

为了节省体力与原神展开终极之战，你希望到达 N 号楼时疲劳值最小。

请输出每次轮回到达 N 号楼的最小疲劳值。

solve

状态定义：

f_i 表示，到达 i 点时 最低疲劳数。

转移方程：

$$f_i = \min(f_{i-k-1}, \dots, f_{i-1} + (a_j \leq a_i)) \quad (3)$$

初始化：

$$f_1 = 1$$

优化角度

1. 对转移方程优化一下：发现函数都是增加1的。因此最多只需要变化1即可。由于转移中只涉及1的常量。选最小值即可：两种选择在结果上是等效的：
2. 然后就可以转换成单调队列问题：
 1. 对于同大小的指标函数对应的位置，尽量维护最大值。
 2. 上面有个小贪心成分。

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  using ll = long long;
5  using i64 = long long;
6  using ull = unsigned long long;
7  using ld = long double;
8  using uint = unsigned int;
9  using pii = pair<int , int>;
10 using pli = pair<ll , int>;
11 using pll = pair<ll , ll>;
12
13 #define dbg(x) cerr << "[" << __LINE__ << "]" << ": " << x << "\n"
14
15 #define all(x) (x).begin(),(x).end()
16 #define sz(x) (int)(x).size()
17 #define pb push_back
18 #define fi first
19 #define se second
20
21 const int inf = 1 << 29;
22 const ll INF = 1LL << 60;
23 const int N = 1E6 + 10;
24 int a[N];
25 ll f[N];
26 int que[N] , low , high;
27 int n;
28 void work(int testNo) {
29     int k;
30     cin >> k;
31     fill(f , f + n + 1 , INF);
32     f[1] = 0;
33     low = 0;
34     high = -1;
35     for (int i = 2; i <= n; i++) {
36         while (low <= high && i - que[low] > k) low++;
37         while (low <= high) {
38             if (f[que[high]] > f[i - 1]) high--;
39             else if (f[que[high]] == f[i - 1] && a[que[high]] < a[i - 1])
40                 high--;
41             else break;
42         }
43         que[++high] = i - 1;
44         f[i] = f[que[low]] + (a[que[low]] <= a[i]);
45     }
46     cout << f[n] << "\n";

```

```

47 }
48 signed main()
49 {
50     ios::sync_with_stdio(false);
51     cin.tie(0);
52     cin >> n;
53     for (int i = 1; i <= n; i++) {
54         cin >> a[i];
55     }
56     int t; cin >> t;
57     for (int i = 1; i <= t; i++) work(i);
58 }
59 /* stuff you should look for
60 * int overflow, array bounds
61 * special cases (n=1?)
62 * do smth instead of nothing and stay organized
63 * WRITE STUFF DOWN
64 * DON'T GET STUCK ON ONE APPROACH
65 */

```

学霸题，帮原神分割布丁

题目描述

$N(1 \leq N \leq 10^5)$ 个布丁，第 i 个布丁的甜度为 $A_i (-10^9 \leq A_i \leq 10^9)$ 。

现在原神要​​从左往右依次吃掉一段连续的布丁。如果连续一段布丁的甜度总和为负数，那么原神就会痛苦。

请问使得原神享用完所有布丁并且不痛苦的方案数。输出方案数 $\bmod 1e9 + 9$ (注意!!!)。

输入格式

第一行：一个整数 N

第二行： N 个整数 A_i ，分别表示第 i 个布丁的甜度

输出格式

输出一个整数表示布丁的划分方案数 $\bmod 1e9 + 9$ 的结果

样例 #1

样例输入 #1

```

1 4
2 2
3 3
4 -3
5 1

```

样例输出 #1

1 | 4

提示

样例解释：一共 4 种分割方案:[2 3 -3 1]、[2] [3 -3 1]、[2 3 -3][1]、[2][3 -3] [1]。

时间限制：2s

空间限制：128MB

solve

状态定义：

f_i 以 i 为前缀和为 s_i 为前缀中，内部划分的方案数。

状态转移：

枚举到某一个位置：根据其前缀，更新其它 s_i 的答案。最终的解为。对小于等于 s_{last} 的状态做一次求和。

实现上：

1. 离散化。
2. 维护前缀和，可以用树状数组，线段树。

生长：

1. 离散化找到了一个更加好看的写法。

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  using ll = long long;
5  using i64 = long long;
6  using ld = long double;
7  using uint = unsigned int;
8  using pii = pair<int , int>;
9  using pli = pair<ll , int>;
10 using pll = pair<ll , ll>;
11
12 #define dbg(x) cerr << "[" << __LINE__ << "]" << ": " << x << "\n"
13
14 #define all(x) (x).begin(),(x).end()
15 #define sz(x) (int)(x).size()
16 #define pb push_back
17 #define fi first
18 #define se second
19
20 const int inf = 1 << 29;
21 const ll INF = 1LL << 60;

```

```

22  const int N = 1E6 + 10;
23  const int mod = 1E9 + 9;
24
25  int a[N] , id[N];
26  int n;
27  ll sum[N];
28  ll c[N];
29
30  void modify(int x , ll d) {
31      for (; x <= n; x += -x & x) {
32          c[x] = (c[x] + d) % mod;
33      }
34  }
35  ll query(int x) {
36      ll res = 0;
37      for (; x; x -= -x & x) {
38          res = (res + c[x]) % mod;
39      }
40      return res;
41  }
42  signed main()
43  {
44      ios::sync_with_stdio(false);
45      cin.tie(0);
46      cin >> n;
47      for (int i = 1; i <= n; i++)
48          cin >> a[i];
49      for (int i = 1; i <= n; i++) {
50          sum[i] = sum[i - 1] + a[i];
51      }
52      //然后做离散化。
53      sort(sum + 1 , sum + n + 1);
54      int last = unique(sum + 1 , sum + n + 1) - sum;
55      ll pre = 0;
56      for (int i = 1; i <= n; i++) {
57          pre += a[i];
58          id[i] = lower_bound(sum + 1, sum + last , pre) - sum;
59      }
60      for (int i = 1; i < n; i++) {
61          ll d = (sum[id[i]] >= 0) + query(id[i]);
62          modify(id[i] , d);
63      }
64      ll res = query(id[n]) + (sum[id[n]] >= 0);
65      cout << res % mod << "\n";
66  }
67  /* stuff you should look for
68   * int overflow, array bounds
69   * special cases (n=1?)
70   * do smth instead of nothing and stay organized
71   * WRITE STUFF DOWN
72   * DON'T GET STUCK ON ONE APPROACH
73  */

```

E. Removing Graph

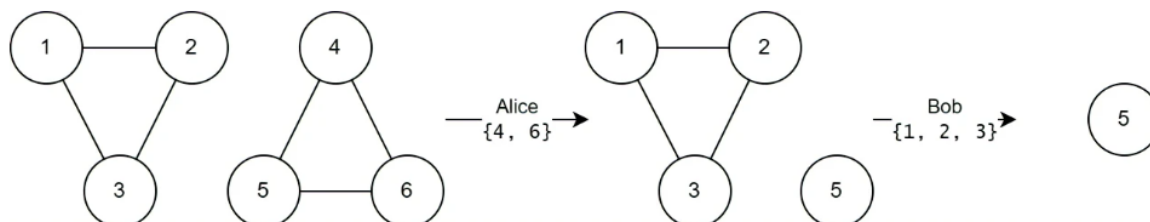
<https://codeforces.com/contest/1823/problem/E>

Alice and Bob are playing a game on a graph. They have an undirected graph without self-loops and multiple edges. All vertices of the graph have **degree equal to 2**. The graph may consist of several components. Note that if such graph has n vertices, it will have exactly n edges.

Alice and Bob take turn. Alice goes first. In each turn, the player can choose k ($l \leq k \leq r$; $l < r$) vertices that form a **connected subgraph** and erase these vertices from the graph, including all incident edges.

The player who can't make a step loses.

For example, suppose they are playing on the given graph with given $l = 2$ and $r = 3$:



A valid vertex set for Alice to choose at the first move is one of the following:

- {1, 2}
- {1, 3}
- {2, 3}
- {4, 5}
- {4, 6}
- {5, 6}
- {1, 2, 3}
- {4, 5, 6}

Suppose, Alice chooses subgraph {4, 6}.

Then a valid vertex set for Bob to choose at the first move is one of the following:

- {1, 2}
- {1, 3}
- {2, 3}
- {1, 2, 3}

Suppose, Bob chooses subgraph {1, 2, 3}.

Alice can't make a move, so she loses.

You are given a graph of size n and integers l and r . Who will win if both Alice and Bob play optimally.

code

嗨，比赛的时候没有看出它是环，导致以为这个图是一张普通的图。连打表都没有做出来：

1. 首先每一个节点的度数都为2.因此这个图是一个环组成的图。

sg定义：

sg_i 长度为i的长度的sg值。

计算sg函数：

就是根据sg定理求即可。求出每一个环的sg值。对这些图中的所有环的sg做一个异或和。

打表结果如下：

1	l : r : 3	4
2	1	0
3	2	0
4	3	1
5	4	1
6	5	1
7	6	2

```

8 7 0
9 8 0
10
11
12 l : r :2 8
13 1 0
14 2 1
15 3 1
16 4 2
17 5 2
18 6 3
19 7 3
20 8 4
21 9 4
22 10 0
23 11 0
24
25
26 l : r :5 10
27 1 0
28 2 0
29 3 0
30 4 0
31 5 1
32 6 1
33 7 1
34 8 1
35 9 1
36 10 2
37 11 2
38 12 2
39 13 2
40 14 2
41 15 0
42 16 0
43 17 0
44 18 0
45 19 0
46

```

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  using ll = long long;
5  using i64 = long long;
6  using ull = unsigned long long;
7  using ld = long double;
8  using uint = unsigned int;
9  using pii = pair<int , int>;
10 using pli = pair<ll , int>;
11 using pll = pair<ll , ll>;
12
13

```

```

14 #define dbg(x) cerr << "[" << __LINE__ << "]" << ": " << x << "\n"
15
16 #define all(x) (x).begin(),(x).end()
17 #define sz(x) (int)(x).size()
18 #define pb push_back
19 #define fi first
20 #define se second
21
22 const int inf = 1 << 29;
23 const ll INF = 1LL << 60;
24 const int N = 1E6 + 10;
25 struct DSU {
26     std::vector<int> f, siz;
27
28     DSU() {}
29     DSU(int n) {
30         init(n);
31     }
32
33     void init(int n) {
34         f.resize(n);
35         std::iota(f.begin(), f.end(), 0);
36         siz.assign(n, 1);
37     }
38
39     int find(int x) {
40         while (x != f[x]) {
41             x = f[x] = f[f[x]];
42         }
43         return x;
44     }
45
46     bool same(int x, int y) {
47         return find(x) == find(y);
48     }
49
50     bool merge(int x, int y) {
51         x = find(x);
52         y = find(y);
53         if (x == y) {
54             return false;
55         }
56         siz[x] += siz[y];
57         f[y] = x;
58         return true;
59     }
60
61     int size(int x) {
62         return siz[find(x)];
63     }
64 };
65 DSU dsu;
66
67 int sg[N];
68 int l = 3 , r = 4;

```



```

69 int calsg(int x) {
70     if (sg[x] != -1) return sg[x];
71     //然后变成了链式的情况。
72     if (x < 1) return sg[x] = 0;
73     //然后两个一起用。
74     sg[x] = 0;
75     set<int> rec;
76     for (int i = 1; i <= r; i++) {
77         for (int j = 0; j + i <= x; j++) {
78             rec.insert(calsg(j)^calsg(x - i - j));
79         }
80     }
81     while (rec.count(sg[x])) sg[x]++;
82     return sg[x];
83 }
84 void solve() {
85     cin >> l >> r;
86     memset(sg, -1, sizeof sg);
87     cout << "l : r :" << l << " " << r << "\n";
88     for (int i = 1; i <= 500; i++) {
89         int ans = 0;
90         set<int> rec;
91         for (int j = 1; j <= min(i, r); j++) {
92             rec.insert(calsg(i - j));
93         }
94         while (rec.count(ans)) ans++;
95         cout << i << "\t" << ans << "\n";
96     }
97 }
98
99 signed main()
100 {
101     ios::sync_with_stdio(false);
102     cin.tie(0);
103     // solve();
104     int n, l, r;
105     cin >> n >> l >> r;
106     dsu.init(n);
107     for (int i = 0; i < n; i++) {
108         int u, v;
109         cin >> u >> v;
110         u--, v--;
111         dsu.merge(u, v);
112     }
113     int res = 0;
114     for (int i = 0; i < n; i++) {
115         if (dsu.find(i) == i) {
116             int cunt = dsu.size(i);
117             res ^= cunt < 1 || cunt >= 1 + r ? 0 : cunt / 1;
118         }
119     }
120     if (res) cout << "Alice\n";
121     else cout << "Bob\n";
122 }
123 /* stuff you should look for

```

```
124 * int overflow, array bounds
125 * special cases (n=1?)
126 * do smth instead of nothing and stay organized
127 * WRITE STUFF DOWN
128 * DON'T GET STUCK ON ONE APPROACH
129 */
```

一个明智地追求快乐的人，除了培养生活赖以支撑的主要兴趣之外，总得设法培养其他许多闲情逸致。