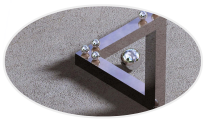


星 从未停歇

梦 永无止境

24 9 0

A. Line Up

2019-05-05 / 53 / 0

A. Line Up

It is well known that if a team has strong cohesion, it tends to burst into more powerful energy at critical moments. Thus, while choosing the P.E. club, Zzz wants to participate in a team, which has the top level of cohesion. (If a team has more people who have connection of friendship, this team will have more cohesion)

One day, Zzz needs to choose a team, but he can’t calculate the best answer by himself. So, he tells you the total numbers of people except himself, some connections between them and everyone’s capability value. Please search a team whose condition can satisfy above information. Moreover, as a curious boy, Zzz wants to know a value which is the sum of people’s capability value in intervals L to R including the boundary but the chosen people can’t exceed the number t which Zzz gives you at the beginning. Of course, when you selecting the section, teammates will stand in a row according to the order of the beginning. Now, please draw a conclusion of the maximum value within two seconds to show your wisdom!

Input:

The first row of input data includes three integers n, m, t, separated by spaces, and people are numbered 1 to n. ($1 \le n \le 10^6$ $0 \le m \le 2 \times n$ $1 \le t \le n$)
The second row of input data has n integers w_1, w_2 … w_n ($-1000 \le w \le 1000$), which is represented person's capability value.
For the following m rows, each row has two integers u and v means u and v are good friends.
($1 \le u, v \le n$)

Output:

Output tow integer separated by space.
The first integer represents the minimum number of people in the largest team (Zzz wishes you to choose the team which has the real minimum number when two or more teams' size is equal).
The second integer is the largest sum which satisfies the description.

Sample Input:

```
4 2 2
-2 -1 0 2
2 3
4 3
```

Sample Output:

```
2 2
```

Sample Input:

```
5 4 4
-3 5 1 -2 3
1 2
2 3
3 4
4 5
```

Sample Output:

```
1 7
```

并查集+前缀和+单调队列

根据题意很容易得知，第一步用并查集找出人数最多的那个队伍，为了更方便得出人数同时找出队伍中哪个人的编号最小，在合并的时候，根节点编号较小的那个点，依旧作为根节点，让另一个点的根节点的父节点等于这个根节点即可，同时这个根节点上的人数再加上另一点的根节点上的人数。

一段区间求和，很明显就是用前缀和预处理一下。而对于怎么去寻找这个区间，如果暴力跑一遍的话，显然会超时，所以就需要用一些数据结构来进行优化。

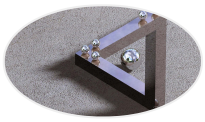
若该区间的右端点固定，那么它的左端点肯定会在人数要求内选择前缀和值最小的，因为这样相减后的结果才会最优，此时，对于左端点的选取，首先要排除超过了人数限制的编号，其次，我们也只关心在这段人数范围内的前缀和最小值，所以在该值之前比它大的任何一个编号都可以舍弃，而对于两个前缀和的值相等的编号，也只需要保留编号较大的，因为编号较大，右端点的选取范围就会更大，也更有可能成为最终答案所在的区间段的点。

也就是说，我们需要维护一个严格单调递增的序列，这个序列的人数要不超过限制，每次加入一个人，简单地更新一下答案就可以，显然这就是一个单调队列的应用。

C++

```
# include <iostream>
# include <cstdio>
# include <climits>
# include <algorithm>

using namespace std;

const int maxn = 1e6 + 5;
int people[maxn], pre[maxn], w[maxn], F[maxn], q[maxn];

int find(int x)
{
    return x == pre[x] ? x : pre[x] = find(pre[x]);
}

void merge(int a, int b)
{
    people[find(a)] += people[find(b)];
    pre[find(b)] = find(a);
    return;
}

void pre_sum(int n)
{
    for (int i = 1; i <= n; i++)
        F[i] += F[i - 1];
    return;
}

int search(int n, int t)
{
    pre_sum(n);
    int hh = 0, tt = 0, ans = INT_MIN;
    for (int i = 1; i <= n; i++) {
        if (hh <= tt && i - q[hh] > t)
            hh++;
        ans = max(ans, F[i] - F[q[hh]]);
        while (hh <= tt && F[i] <= F[q[tt]])
            tt--;
        q[++tt] = i;
    }
    return ans;
}

int main()
{
    int n, m, t;
    scanf("%d %d %d", &n, &m, &t);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &w[i]);
        pre[i] = i;
        people[i] = 1;
```

```
    }
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        if (find(u) != find(v)) {
            if (find(u) > find(v))
                swap(u, v);
            merge(u, v);
        }
    }
    int pos = 0;
    pair<int, int> now_max;
    now_max = { 0, 0 };
    for (int i = 1; i <= n; i++) {
        if (people[i] > now_max.first) {
            now_max = { people[i], i };
            pos = 0;
            F[++pos] = w[i];
        }
        else if (find(now_max.second) == find(i))
            F[++pos] = w[i];
    }
    printf("%d %d\n", find(now_max.second), search(pos, t));
    return 0;
}
```



星 从未停歇

梦 永无止境

24 9 0

B. Badminton

2019-05-05 / 92 / 0

B. Badminton

Zzz chose two badminton lessons before he realized the consequences that such option would cause a tired body. But Zzz could not change his selections. Having taken above into account, Zzz determined to practice his badminton’s skill with the help of new machine called NQ.

NQ is an intelligent robot invented by IvyHole, it can throw the ball across the net. Thus, Zzz wants to use it to advance his understanding of hitting badminton. According to the settings inside the NQ, it will throw out badminton of different sizes, and each badminton has a certain score. In order to hit the ball back, Zzz will lose some energy value which is equal to the volume of each badminton. At the same time, Zzz has to guarantee that his energy value could not less than zero! To gain the better training effect, Zzz wants to know the maximum score he can get before he runs out of his energy to decide whether to hit the current badminton or not.

Input:

The first row of input data includes two integer n, h represented the number of badmintons which will be thrown by NQ and the maximum value of Zzz's energy. (1 ≤ n, h ≤ 1000)

Output:

For the following n rows, each row has two integers v, s, represented every badminton's volume and score. (1 ≤ v ≤ h 1 ≤ s ≤ 10^7)

Sample Input:

```
4 5
1 2
2 4
3 4
4 5
```

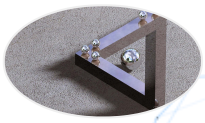
Sample Output:

```
8
```

能耐心读完题意的应该都知道，这就是一道01背包，直接默模板就行（为直接跳过这题的选手默哀一秒钟）。

C++

```cpp
# include <iostream>
# include <algorithm>

using namespace std;

typedef long long LL;
const int maxn = 1010;
LL F[maxn];

int main()
{
    LL n, h;
    cin >> n >> h;
    for (int i = 0; i < n; i++) {
        LL v, s;
        cin >> v >> s;
        for (int j = h; j >= v; j--)
            F[j] = max(F[j], F[j - v] + s);
    }
    cout << F[h] << endl;
    return 0;
}
```



星 从未停歇

梦 永无止境

24 9 0

C. 我们无法一起学习

2019-05-05 / 218 / 3

C. 我们无法一起学习

刻苦学习的高中3年生·唯我成幸，为了获得免除大学学费的“特别VIP推荐”资格，而要去担任为备考而苦战的同级生们的教育指导员。教导的对象是“文学之森的睡美人”古桥文乃和“机关精巧的拇指姑娘”绪方理珠这两位学园顶尖的天才美少女！原本以为她们的学习能力完美无缺，没想到对于不擅长

的学科却完全无能……！？成幸一边被充满个性的“学不来女孩”们玩弄于股掌之间，一边为了让她们努力通过大学考试！无论学习还是恋爱都“学不来”的天才们的恋爱喜剧，就此开幕！！

有一天，古桥文乃拿着如下问题请教唯我成幸：“有n个数字”1″，现将用这些数字”1″通过加法、乘法和括号运算符求得正整数s，求n的最小值。”

唯我成幸应该怎么帮助她呢？

输入描述：

第一行输入T，表示接下来有T组测试样例。$1 \le T \le 2 \times 10^5$
接下来，对于每一组测试样例，在一行内输入一个数字s。$1 \le s \le 10^4$

输出描述：

对于每一组测试样例，输出最小值n并换行。

输入样例：

2
3
4

输出样例：

3
4

动态规划

没想到好好的一道签到题，居然在好雨学姐的助攻下检查出了数据问题，改来改去，最后变成了一道dp~~

（当然，为了弥补出题人的疏忽，赛场上这题的测试数据很水很水很水~~，当然赛后更新了题面也加强了数据！！）

更改后的题目是只能用加法以及乘法，所以我们可以从小到大遍历，对于每一个数字i，只能由比它小的数字相加或者相乘得出结果，由此也就得到了下述dp转移方程：

```C++
for (int i = 1; i < maxn; i++) {
    dp[i] = i;
    for (int j = 1; j < i - j; j++)
        dp[i] = min(dp[i], dp[j] + dp[i - j]); //由j + (i - j) 得到i
    for (int j = 1; j * j <= i; j++)
        if (i % j == 0)
            dp[i] = min(dp[i], dp[i / j] + dp[j]); //由 (i / j) * j得到i
}
```

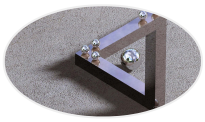
对于每一个数字i，从在它之前的数字中去暴力找一遍最优解。

因为这题数据组数要比数据范围大得多，也就必然会存在重复求答案的数字，所以如果每读入一个数字都进行一次上述操作就很有可能TLE，所以我们需要将数据范围内的数字的答案预处理出来，最后每读入一个数，就能在O(1)的复杂度内得出结果！

特别鸣谢：610给出的正确的思路但却不怎么正确的代码，算协各群友的不懈努力以及lzh最后给出的标程！！！

C++

```cpp
# include <iostream>
# include <algorithm>
using namespace std;

const int maxn = 10010;
int dp[maxn];

void pre_work()
{
    for (int i = 1; i < maxn; i++) {
        dp[i] = i;
        for (int j = 1; j < i - j; j++)
            dp[i] = min(dp[i], dp[j] + dp[i - j]);
        for (int j = 1; j * j <= i; j++)
            if (i % j == 0)
                dp[i] = min(dp[i], dp[i / j] + dp[j]);
    }
    return;
}

int main()
{
    ios::sync_with_stdio(false);
    pre_work();
    int T;
    cin >> T;
    while (T--) {
        int n;
        cin >> n;
        cout << dp[n] << endl;
    }
    return 0;
}
```



星 从未停歇

梦 永无止境

24 9 0

D. 埃罗芒阿老师

2019-05-05 / 110 / 1

D. 埃罗芒阿老师

《埃罗芒阿老师》中，和泉纱雾邀请山田妖精和千寿村征玩一个小游戏——扭扭乐。《老友记》、《生活大爆炸》里曾经出现过这个“扭扭乐”游戏，早已在美国流行多年。“扭扭乐”是一项非常好玩的多人游戏，可以让2~4人同时一起参加。顾名思义，扭扭乐游戏的最终目的就是要让大家扭成一

团。游戏套装中附有一张印有各种颜色的塑胶板及指针轮盘，由裁判负责转动指针，对比赛者发号施令，当指针指到哪一只手、脚要压在哪一个颜色上，参加者就必须依照指定动作做到，谁能够坚持到底不倒下，或成为唯一一个离开游戏圈的人，就是最后的胜利者。

“扭扭乐”的难点不仅在于玩家得按照指令做出平常难以做的姿势，多名玩家的同时参加也给游戏增加了难度。

假设有两个人玩“扭扭乐”，每个人可以抽象为一条线段。那么游戏的难度就仅由两条线段是否有公共点有关了。你要判断的是这两个人按照规定指令摆姿势的话，难度如何。

输入描述：

测试分多组测试样例。第一行输入T，表示有T组测试样例。$1 \le T \le 100$
接下来，对于第i组测试样例，第一行输入$x_{i1},y_{i1},x_{i2},y_{i2}$，表示第一条线段的端点坐标分别为$x_{i1},y_{i1}$和$x_{i2},y_{i2}$，同理第二行输入$x_{i3},y_{i3},x_{i4},y_{i4}$.
$1 \le i \le T \quad 0 \le x_{i1},y_{i1},x_{i2},y_{i2} \le 10^5$ 且 $x_{i1},y_{i1},x_{i2},y_{i2}$ 都是整数。
输入保证对于同一条线段，输入的两个点坐标不同。

输出描述：

对于每一组测试样例，如果两条线段有公共点，输出”Hard”,否则输出”Easy”。

输入样例：

```
2
0 0 4 6
0 2 5 0
0 0 1 2
3 0 1 2
```

输出样例：

```
Hard
Hard
```

题意不难懂，就是给出两条线段，判断这两条线段是否有交点。对于这个问题，相信大家都很容易想到用数学知识算出两条线段的表达式，得出公共交点后再进行判断这个交点是否落在两条线段上就可以（这么简单的推导，就不贴过程了吧，大家草稿纸上写写画画就能得出来的结论）。

这里默认大家都已经推导完了，那么也不难发现如果直接套公式的话可能发生除零错误，而导致发生除零错误的情况不外乎这两类：

1. 存在至少一条线段垂直于x轴
2. 两条线段的斜率相等

对于这两种特殊情况，特判一下就行了：

1. 一条线段垂直于x轴的话，将这个x坐标代入另一个线段表达式中算出y，再判断这个y是否在垂直于x轴的线段上即可。
2. 两条线段垂直于x轴的话，首先比较两条线段的x轴是否相等，如果相等的话，想让这两条线段有交点，需要满足的条件就是两条线段中较大的y轴坐标的较小者大于等于两条线段中较小的y轴坐标的较大者（随便画个图看一下就能理解）。
3. 对于斜率相同的情况则先判断在y轴上的截距是否相同，相同的话再用2中的方法判定即可，因为相对位置是不变的，所以此处不论挑x坐标还是挑y坐标进行2中的判断都是可行的。
4. 如果非上述条件的线段，则直接代入推导出来的公式算交点。

这里弱弱的说一句，我在验题的时候没注意题目说了保证线段的两个点的坐标不同，所以下面的代码里还特判了线段为点的情况~~ 大家选择性忽视吧。

C++

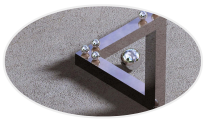
```
# include <iostream>
# include <algorithm>

using namespace std;

bool point_line(double x, double y, double a, double b, double c, double d)
{
    if (a == c)
        return x == a && y >= min(b, d) && y <= max(b, d);
    return (y - b == (d - b) / (c - a) * (x - a)) && x >= min(a, c) && x <= max(a, c) && y >= min(b, d) && y <= max(b, d);
}

bool x_line(double x, double y1, double y2, double a, double b, double c, double d)
{
    double y = (d - b) / (c - a) * (x - a) + b;
    return x >= min(a, c) && x <= max(a, c) && y >= min(y1, y2) && y <= max(y1, y2) && y >= min(b, d) && y <= max(b, d);
}

int main()
{
    int T;
    cin >> T;
    while (T--) {
        double a1, b1, c1, d1, a2, b2, c2, d2;
        cin >> a1 >> b1 >> c1 >> d1 >> a2 >> b2 >> c2 >> d2;
        bool success;
        // 输入的两组坐标完全相同
        if (a1 == a2 && b1 == b2 && c1 == c2 && d1 == d2 || a1 == c2 && b1 == d2 && c1 == a2 && d1 == b2)
            success = true;
        // 存在点的情况
        else if (a1 == c1 && b1 == d1 && a2 == c2 && b2 == d2)  // 两点坐标相同的判断可以也归入上面一个if
            success = a1 == a2 && b1 == b2;
        else if (a1 == c1 && b1 == d1 && (a2 != c2 || b2 != d2))
            success = point_line(a1, b1, a2, b2, c2, d2);
        else if ((a1 != c1 || b1 != d1) && a2 == c2 && b2 == d2)
            success = point_line(a2, b2, a1, b1, c1, d1);
        else {
            // 不存在点的情况
            if (a1 == c1 && a2 == c2)
                success = a1 == a2 && min(max(b1, d1), max(b2, d2)) >= max(min(b1, d1), min(b2, d2));
            else if (a1 != c1 && a2 == c2)
                success = x_line(a2, b2, d2, a1, b1, c1, d1);
            else if (a1 == c1 && a2 != c2)
                success = x_line(a1, b1, d1, a2, b2, c2, d2);
            else {
```

```
45                  // 两条均不垂直于x轴的线段
46                  if (b1 == d1 && b2 == d2)
47                      success = b1 == b2 && min(max(a1, c1), max(a2, c2)) >= max(min(a1, c1),
48  min(a2, c2));
                    else if ((d1 - b1) / (c1 - a1) == (d2 - b2) / (c2 - a2))
49                      success = (b1 - (a1 * (d1 - b1) / (c1 - a1)) == b2 - (a2 * (d2 - b2) /
50 (c2 - a2))) && min(max(a1, c1), max(a2, c2)) >= max(min(a1, c1), min(a2, c2));
                    else {
                        double x = (b2 - b1 + a1 * (d1 - b1) / (c1 - a1) - a2 * (d2 - b2) / (c2
51  - a2)) / ((d1 - b1) / (c1 - a1) - (d2 - b2) / (c2 - a2));
52                      double y = (d1 - b1) / (c1 - a1) * x - a1 * (d1 - b1) / (c1 - a1) + b1;
                        success = x >= min(a1, c1) && x <= max(a1, c1) && x >= min(a2, c2) && x
53  <= max(a2, c2) && y >= min(b1, d1) && y <= max(b1, d1) && y >= min(b2, d2) && y <= max(b2,
    d2);
54                  }
                }
            }
            if (success)
                cout << "Hard" << endl;
            else
                cout << "Easy" << endl;
        }
        return 0;
}
62
63
64
```

评论

Enal 2019-07-30 21:01

可以考虑用叉积做。 💡



星 从未停歇

梦 永无止境

24 9 0

E. 选课 & 学分

2019-05-05 / 76 / 0

E. 选课 & 学分

没错，又是选课，如果你看过这次比赛的另外两道英文题，就知道为什么要用 “又” 了。多来一题不多，少出一题也不少，那就来咯（这题题意十分简单明了，只作为各位神仙选手看不懂英语题目的补偿）。

SZ市某大学某年级共有m类选修课程，每类选修课程设有n种分支，每个分支单独为学生打分。特别的是，该学校学生对于任何选修课都没有明显的短板，所以校长特意度身定做了一个打分制度：

首先由各个老师给出成绩，每类选修课程的每个分支都需要单独打分，其次，从每类选修课程的分支中都任选出一个分数，相加，共有n^m种情况，也就有n^m个值（这个值可以相等），为了不让某位学生的特长拉高太多分数，需要再从这么多数中挑出最小的n个，最后在实数范围内任选一个数字，将这个数字分别与挑出的n个数字相减后取绝对值的结果再求和，最小的和即为该学生选修课的最终成绩（因为这个特殊的计算成绩的方式，所以各个分支的成绩可以为负数）。

现在，zzz只知道自己各类选修课程的各个分支的成绩，想提前知道自己选修课的成绩，但自己什么都不会，只好向你求救！

输入描述：

第一行输入m, n. (1 ≤ m ≤ 1000 1 ≤ n ≤ 2000)
接下来m行，每行n个整数表示某类选修课的某个分支的成绩x. (题目保证x的绝对值的int范围内)

输出描述：

第一行升序输出挑出的n的数字，数字之间用空格隔开，行末没有多余的空格。
第二行输出最终成绩。

输入样例：

2 3
1 2 3
2 2 3

输出样例：

3 3 4
1

分治+n路归并+中位数

直接考虑的话似乎怎么优化都不能达到一个让自己满意的时间复杂度，那就换个想法，分而治之，从小处入手，先来考虑只有两行数据的情况。

两行数字各挑选一个最小值，相加就是第一个结果，然后比较遗弃第一行选出的那个数字加入第一行第二小的数字以及遗弃第二行选出的那个数字加入第二行第二小的数字，哪个小则为第二个结果，以此类推，就可以得出最小的n个。

再想一想，如果第一行数字已经是升序排列好了，第二行的每一个数字都与第一行相加，得出如下结果（假设第一行为a数组，第二行为b数组）：

$b_1 + a_1, b_1 + a_2, \ldots, b_1 + a_n$
$b_2 + a_1, b_2 + a_2, \ldots, b_2 + a_n$
…
$b_n + a_1, b_n + a_2, \ldots, b_n + a_n$

因为第一行是排好序的，所以上述的n行都是有序的，所以第一个最小值只有可能从每行第一列中产生，而选出这个最小值之后，加入这一行的下一个值，第二个最小值也就在这几个数中产生，以此类推，不难得出前n个数（可以类比于归并排序的合并的过程，只不过这里是n路归并）。

对于最小值的候选项，只需要建立一个小根堆，每次弹出堆顶就是答案。而对于每次新加入的值，也不难发现，它跟弹出的堆顶之间存在着某种与a数组有关的联系：

堆中最小值为 $b_2 + a_1$ 时，弹出这个元素，需要加入 $b_2 + a_2$
堆中最小值为 $b_n + a_3$ 时，弹出这个元素，需要加入 $b_n + a_4$
不难发现：
当堆中最小值为 $b_i + a_j$ 时， 弹出这个元素，需要加入 $b_i + a_{j+1}$, 也就是堆顶的这个值 $- a_j + a_{j+1}$

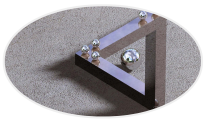
所以，对于堆中的每一个元素，不仅要记录下它的值，还需要记录下它是加了哪个a数组元素转移过来的，也就是说还需要记录一个下标。

最后，将两组数据得出的结果与下一组数进行相同的过程，即将上述过程重复m – 1次，就可得出想要的n个数字。

选出n个数字之后，接下来就又回归到了一个基本的问题，选择序列中的中位数作为基准，再与每个数相减，取绝对值，求和即可，至于为什么选中位数，这里也就不给出证明，简单点，画个数轴看一眼就行。

C++

```cpp
# include <iostream>
# include <cstdio>
# include <cmath>
# include <queue>
# include <vector>
# include <algorithm>

using namespace std;

typedef long long LL;
typedef pair<LL, LL> PLL;
const int maxn = 2010;
LL m, n;
LL a[maxn], b[maxn], c[maxn];

void merge()
{
    priority_queue<PLL, vector<PLL>, greater<PLL>> heap;
    for (int i = 0; i < n; i++)
        heap.push({ a[0] + b[i], 0 });
    for (int i = 0; i < n; i++) {
        auto t = heap.top();
        heap.pop();
        c[i] = t.first;
        heap.push({ t.first - a[t.second] + a[t.second + 1], t.second + 1 });
    }
    for (int i = 0; i < n; i++)
        a[i] = c[i];
    return;
}

int main()
{
    scanf("%lld %lld", &m, &n);
    for (int i = 0; i < n; i++)
        scanf("%lld", &a[i]);
    sort(a, a + n);
    for (int i = 0; i < m - 1; i++) {
        for (int j = 0; j < n; j++)
            scanf("%lld", &b[j]);
        merge();
    }
    for (int i = 0; i < n; i++)
        printf("%lld%c", a[i], " \n"[i + 1 == n]);
    LL sum = 0, mid = a[n >> 1];
    for (int i = 0; i < n; i++)
        sum += abs(mid - a[i]);
    printf("%lld\n", sum);
    return 0;
}
```



星 从未停歇

梦 永无止境

24 9 0

F. 乐道跃迁

2019-05-05 / 79 / 0

F. 乐道跃迁

作为当代大学生，不仅要学好专业知识，还要有强健的体魄来支撑每天高强度的学习与工作（~~只是不知道能不能防脱发啊QAQ~~）。而跑步可能是大多数人的选择，它有……（此处省略）等好处。

zzz也喜欢利用每天的空闲时间跑个步（~~说的我差点信了QAQ~~），但这一次，zzz因为准备这一次程序设计竞赛废寝忘食，等他回过神来，发现离比赛开始只剩下了1分钟，哪怕他以最快速度飞奔去机房也来不及，不过幸运的是，他召唤出了Enal大魔法师。

Enal大魔法师在学校随机生成n个传送点，同时也构建了m道单向超光速通道（允许两个传送点之间存在多条通道，但反向通道不通）用以连接两个固定的传送点，而通道中也因为强大的魔法而散落有能量点。如果两个传送点之间存在通道，则zzz可以在这条通道的起点毫不费时也毫不费力地传送到这条通道的终点。

Enal大魔法师承诺可以让zzz按时参加比赛，但作为释放“传送”技能的补偿，zzz需要在至少一个传送点搜集足够多的能量值（使之能够满足Enal的最低要求能量值t），否则，Enal也无能为力，而一个传送点的能量在离开了这个传送点之后便会消散，所以不能叠加，zzz本就焦头烂额，所以只好向你求救！

一个传送点可能的能量值为：从与该点直接相连且能过通过的所有通道的散落能量点值中任意挑选两个值进行异或运算后的结果（如果非要自己异或自己也是可以的哈）。（特别的，若该点没有通道，则该点能量值为0）

（此处为对于题中异或运算的简单解释：将两个数均写成二进制形式，可适当补前导零，对于同一位上的二进制，如果两数不同，则最后结果中这一位上为1，否则为0；再简单点说，就是二进制下的不进位加法！）

输入描述：

第一行输入n, m, t。 $1 \le n \le 10^5$ $0 \le m \le 2 \times n$ $0 \le t < 2^{31}$ （题目保证不存在自己到自己的通道）
接下来m行，每行一个整数u, v, w，表示从u号传送点到v号传送点有一条通道，通道中散落的能量点值为w。$1 \le u, v \le n$ 且 u != v $0 \le w < 2^{31}$

输出描述：

如果zzz能够按时参加比赛，则输出他在传送点可能搜集到的能量值的最大值，否则输出“QAQ”（输出不含有引号）

输入样例：

```
3 2 1
1 2 3
1 3 1
```

输出样例：

```
2
```

输入样例：

```
3 4 3
1 3 1
1 2 10
1 3 3
1 3 2
```

输出样例：

```
11
```

字典树

对于一个起点来说，我们只在乎它直接相连的且能够通过的通道中的能量点值，所以可以直接在map里套个vector把我们想要数据都存下来。

对于每一个起点，要在它所对应的vector中任意挑选两个进行异或运算，使计算结果最大。根据题中所规定的范围，极端情况下用暴力枚举是会TLE的，所以我们来探究一下规律。

对于一个确定的数，它的二进制位也是确定的，而对于一个二进制数来说，如果它的某一个高位为1，那么这个数肯定比这一位为0，低位上全为1的数字还要大，所以，优先考虑高位，如果在待挑选的数字中，存在数字能够使异或运算后这一位为1，那么对于低位的选择，数字的挑选范围就缩小到了这些数里。不断循环这一过程，最终就可以知道对于该数字，应该挑选哪一个数能够使异或运算后的结果最大。

上述思想的实现，相信大佬已经非常有感觉了，就是字典树。对于题中的每一个起点，将其相连的可通过的通道的能量点值根据二进制位由高到低建立一棵字典树，然后枚举其中的每一个数，在树上进行异或走路。

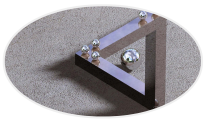
如果一个数字的某一位为1，那么优先在字典树中寻找该位为0的结点是否存在，如果存在，则走向这个结点，同时将答案的这一位置成1；如果这样的结点不存在，那么只好走向这一位为1的结点，此时答案不需要变化。

上述的所有答案取一个max即为一个起点可能的能量值的最大值，最后在所有起点中取一个max即为最终答案。

C++

```cpp
# include <iostream>
# include <cstdio>
# include <cstring>
# include <vector>
# include <unordered_map>
# include <algorithm>

using namespace std;

typedef pair<int, int> PII;
const int maxn = 1e5 + 5;
unordered_map<int, vector<PII>> M; // 某一个点能够直接到达的其它所有点 （也可以不存终点，只存由该点出发的每条通道的能量）
int F[maxn * 30][2], pos;

void build(vector<PII> &w)
{
    for (auto x : w) {
        int p = 0;
        for (int i = 30; ~i; i--) {
            if (!F[p][x.second >> i & 1])
                F[p][x.second >> i & 1] = ++pos;
            p = F[p][x.second >> i & 1];
        }
    }
    return;
}

int serch(vector<PII> &w)
{
    int ans = 0;
    for (auto x : w) {
        int res = 0, p = 0;
        for (int i = 30; ~i; i--) {
            if (F[p][x.second >> i & 1 ^ 1]) {
                res += 1 << i;
                p = F[p][x.second >> i & 1 ^ 1];
            }
```

```
38              else
39                  p = F[p][x.second >> i & 1];
40          }
41          ans = max(ans, res);
42      }
43      return ans;
44  }
45
46  int main()
47  {
48      int n, m, t;
49      scanf("%d %d %d", &n, &m, &t);
50      for (int i = 0; i < m; i++) {
51          int u, v, w;
52          scanf("%d %d %d", &u, &v, &w);
53          M[u].push_back({ v, w });
54      }
55      int ans = 0;
56      for (auto x : M) {
57          if (x.second.size() >= 2) {
58              memset(F, 0, sizeof(F));
59              pos = 0;
60              build(x.second);
61              ans = max(ans, serch(x.second));
62          }
63      }
64      if (ans >= t)
65          printf("%d\n", ans);
66      else
67          printf("QAQ\n");
68      return 0;
69  }
```



星 从未停歇

梦 永无止境

24 9 0

G. 游戏

2019-05-05 / 61 / 0

G. 游戏

敲敲敲，敲敲敲，是作为一个码农的日常，但是，码农也并非是神，偶尔玩玩游戏娱乐一下也是可以的，况且，当初很多人学计算机的目的也就是为了做游戏，那么，现在就请学编程的你来判断一下，zzz新创办的一款游戏英雄的大招是否存在BUG！

六芒星阵：

天地色变，星空浮现，剑阵平地而起，无数六芒星璀璨夺日，两者交相呼应，呼啸而下，直指敌方要害！

现在，为平衡该技能伤害，加入伤害浮动机制，即：六芒星的六个角内，含有6个伤害值（攻击敌人时，随机该6个伤害的任意一个作为真实伤害），要使每一个六芒星伤害尽可能不同，即要求使一个六芒星内部的六个角之间的6个伤害值各不相同，且不含有完全相同的六芒星。

为方便判定，角与角之间伤害值相同定义为：

一个角中的6个伤害值能够与另一个角中的6个伤害值忽略排序后一一对应。

例如：

1 2 3 4 5 6 与 5 6 3 1 2 4 这两个角的伤害值是相同的

1 2 3 4 5 6 与 1 2 3 4 5 0 这两个角的伤害值不同，因为6 != 0

六芒星与六芒星相同定义为：

一个六芒星中的6个角，能够跟另一个六芒星的6个角忽略排序后一一对应。

如果存在两个伤害可能相同的六芒星，输出“BUG!”，否则输出“SUCCESS!”.所有输出均不含有引号。

输入描述：

输入一个数n，代表六芒星的个数，接下去对于每一个六芒星，输入6行，每行6个数值x，代表六芒星每个角内的伤害浮动值。即一共需要输入6 × n + 1 行数据。$1 \le n \le 5 \times 10^4$　　$0 \le x \le 10^7$

输出描述：

如题所述。

输入样例：

```
1
474 383 236 466 391 533
929 770 991 402 340 744
311 931 880 594 124 480
307 182 534 893 563 414
539 68 152 387 252 347
182 495 433 55 930 94
```

输出样例：

SUCCESS!

输入样例：

```
1
182 495 433 94 930 55
929 770 991 402 340 744
311 931 880 594 124 480
307 182 534 893 563 414
539 68 152 387 252 347
182 495 433 55 930 94
```

输出样例：

BUG!

Hash + 暴力

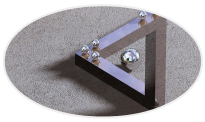
六芒星的两个角，如果它是相同的，那么，角内6个数字之和一定也是相同的，所以，可以以这个和为标准，如果和相同，才进行数字与数字之间的比较。当这个和比较大的时候，可以选取一个较大的质数，将这个和对这个质数取模后的结果作为标准进行判断。

对于六芒星内部角与角之间的判定，首先比较上述评判标准，相同的情况下再进行判断6个数值，而对于数值的判断，最直接的想法就是sort一遍暴力跑，毕竟一个角内只有6个数，哪怕是n^2的复杂度也可以接受。

在六芒星内部无相同角的前提下，进行星与星之间的判断，因为内部无相同角，所以，一个六芒星内部的一个角在另一个六芒星内部只可能存在最多一个角与其相同（此处可用反证法证明），所以对于Hash值相同的六芒星，也是简单粗暴的用暴力跑一遍判断一下就行。

```C++
# include <iostream>
# include <cstdio>
# include <vector>
# include <unordered_map>
# include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;
const int Hash = 99991;
unordered_map<int, vector<int>> v;  // 以第一关键字为整个六芒星的hash值的所有六芒星的编号

struct Star {
    int nums[10][10]; // nums[i][j] : 第i个角的6个伤害浮动值    nums[i][0] : 第i个角的hash值
}s[maxn];

bool check_indoor(int *a, int *b)
{
    sort(a + 1, a + 1 + 6);
    sort(b + 1, b + 1 + 6);
    for (int i = 1; i <= 6; i++)
        if (a[i] != b[i])
            return false;
    return true;
}

bool check_outdoor(Star a, Star b)
{
    int cnt = 0;
    for (int i = 1; i <= 6; i++) {
        for (int j = 1; j <= 6; j++) {
            if (a.nums[i][0] == b.nums[j][0] && check_indoor(a.nums[i], b.nums[j]))
                cnt++;
        }
    }
    return cnt == 6;
}

int main()
{
    int n;
```

```
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        int Sum = 0;
        for (int j = 1; j <= 6; j++) {
            int sum = 0;
            for (int k = 1; k <= 6; k++) {
                scanf("%d", &s[i].nums[j][k]);
                sum = (sum + s[i].nums[j][k]) % Hash;
            }
            s[i].nums[j][0] = sum;
            Sum = (Sum + sum) % Hash;
        }
        v[Sum].push_back(i);
    }
    bool success = true;
    for (int i = 1; i <= n; i++) {
        // 每个六芒星内部是否满足要求
        for (int j = 1; j < 6; j++) {
            for (int k = j + 1; k <= 6; k++)
                if (s[i].nums[j][0] == s[i].nums[k][0] && check_indoor(s[i].nums[j], s[i].n
ums[k])) {
                    success = false;
                    break;
                }
            if (!success)
                break;
        }
        if (!success)
            break;
    }
    if (success)
        // 六芒星之间是否满足要求
        for (auto x : v) {
            if (x.second.size() > 1) {
                for (int i = 0; i < x.second.size() - 1; i++) {
                    for (int j = i + 1; j < x.second.size(); j++)
                        if (check_outdoor(s[x.second[i]], s[x.second[j]])) {
                            success = false;
                            break;
                        }
                    if (!success)
                        break;
                }
            }
            if (!success)
                break;
        }
    if (success)
        printf("SUCCESS!\n");
    else
        printf("BUG!\n");
    return 0;
}
```



星 从未停歇

梦 永无止境

24 9 0

H. 笨拙之极的上野

2019-05-05 / 154 / 0

H. 笨拙之极的上野

《笨拙之极的上野》中，科学部部长上野每天都搞出千奇百怪的发明，并让她的部员田中来测试。这天，她又发明了一样东西，不过，这次是发明了一种语言——“Ueno语言”。

“Ueno语言”是一种处理数学上的映射关系的语言。它的语法如下：

addA(x)：往集合A中添加x，如果集合中已经有该元素，则忽略该命令。

addB(y)：往集合B中添加y，如果集合中已经有该元素，则忽略该命令。

buildmap(x,y)：建立集合A中的元素x到集合B中的元素y的映射。如果集合A中不存在x，但是集合B中存在y，则输出”Invalid!Can’t find x.”（输出不包括双引号，下同）。如果集合B中不存在y，但是集合A中存在x，则输出”Invalid!Can’t find y.”。如果集合A中不存在x，且集合B中不存在y，输出“Invalid!Can’t find both x and y.”。特别地，如果x元素之前已经建立了映射，则输出”Invalid!There is already a map from x.”，而无需考虑y是否存在的问题。

findy(x)：输出x映射到集合B的像。如果没有此映射，输出”Invalid!No such map.”。

findx(y)：升序输出映射到y的原像。每个元素后面有一个空格。如果没有此映射，输出”Invalid!No such map.”。

这种语言是怎么实现的呢？问题的答案就交给你了。

输入描述：

第一行，输入正整数m，表示有m条语句。$1 \le m \le 10^5$
接下来m行，每行输入一句语句。输入的语句保证符合“Ueno语言”的语法。
假定这些语句处理的数只可能是整数且绝对值不超过10^9（即上文中的$-10^9 \le x, y \le 10^9$ 且 x, y 是整数）。

输出描述：

根据对应的语句输出, 值得注意的是，每次输出后面都要有一个换行符。

样例输入：

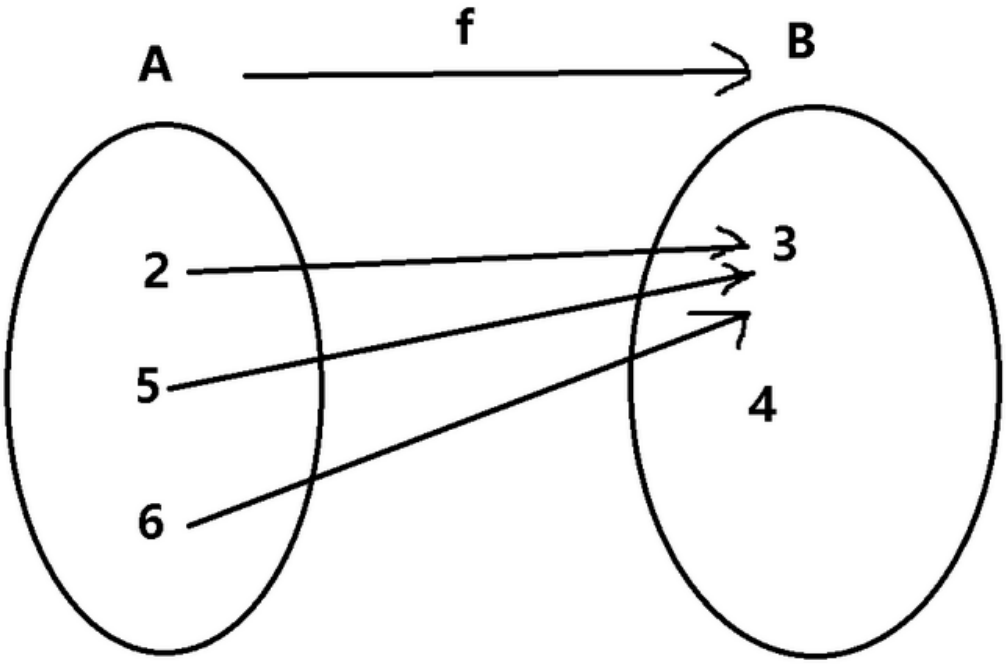
```
12
addA(2)
addA(5)
addB(3)
addB(4)
buildmap(2,3)
buildmap(2,5)
findy(2)
addA(6)
buildmap(6,3)
addA(6)
buildmap(5,3)
findx(3)
```

样例输出：

```
Invalid!There is already a map from 2.
3
```

2 5 6

如图所示为处理了全部样例输入后的映射图：



将每条命令读入，再根据题意模拟一下就行了。 对于输入x找y的情况，只需要一个原生的map；而对于输入y找x的情况，也只要在map里面套一个vector即可，输出前sort一下就能得到升序（有时候STL真的很方便）。

此代码来源于Enal，既然发给我了，那我也懒得再写一次了~~

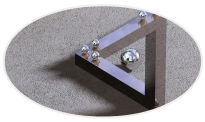
```C++
#include<iostream>
#include<cstdio>
#include<string>
#include<vector>
#include<set>
#include<map>
#include<algorithm>

using namespace std;
typedef long long ll;

int main()
{
    set<ll>a, b;
    map<ll, ll>mp_x;
    map<ll, vector<ll>>mp_y;
    ll m;
    cin >> m;
    while (m--) {
        string op;
        cin >> op;
        if (op[0] == 'a' && op[3] == 'A') {
            ll x = 0;
            if(op[5]=='-'){
                for (int i = 6; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
```

```
                a.insert(-x);
            }else{
                for (int i = 5; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
                a.insert(x);
            }
        }
        else if (op[0] == 'a' && op[3] == 'B') {
            ll x = 0;
            if(op[5]=='-'){
                for (int i = 6; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
                b.insert(-x);
            }else{
                for (int i = 5; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
                b.insert(x);
            }
        }
        else if (op[0] == 'b') {
            int p = 9;
            int sign=1;
            ll x = 0, y = 0;
            if(op[p]=='-'){
                sign=-1;
                ++p;
            }
            while (op[p] != ',') {
                ll digit = op[p] - '0';
                x = x * 10 + digit;
                ++p;
            }
            x=sign*x;
            if(op[p+1]=='-'){
                for (int i = p+2; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    y = y * 10 + digit;
                }
                y=-y;
            }else{
                for (int i = p+1; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    y = y * 10 + digit;
                }
            }
            if (a.count(x) == 0 && b.count(y) == 1) {
                cout << "Invalid!Can't find x." << endl;
            }
            else if (a.count(x) == 1 && b.count(y) == 0) {
                if (!mp_x.count(x))
                    cout << "Invalid!Can't find y." << endl;
                else
                    printf("Invalid!There is already a map from %lld.\n", x);
            }
            else if (a.count(x) == 0 && b.count(y) == 0) {
                cout << "Invalid!Can't find both x and y." << endl;
            }
            else {
                if (mp_x.count(x))
                    printf("Invalid!There is already a map from %lld.\n", x);
                else {
                    mp_x[x] = y;
```

```
                    mp_y[y].push_back(x);
                }
            }
        }
        else if (op[0] == 'f' && op[4] == 'y') {
            ll x = 0;
            if(op[6]=='-'){
                for (int i = 7; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
                x=-x;
            }else{
                for (int i = 6; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
            }
            auto search = mp_x.find(x);
            if (search != mp_x.end()) {
                cout << search->second << endl;
            }
            else {
                cout << "Invalid!No such map." << endl;
            }
        }
        else if (op[0] == 'f' && op[4] == 'x') {
            ll x = 0;
            if(op[6]=='-'){
                for (int i = 7; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
                x=-x;
            }else{
                for (int i = 6; i < op.size() - 1; ++i) {
                    int digit = op[i] - '0';
                    x = x * 10 + digit;
                }
            }
            auto search = mp_y.find(x);
            if (search != mp_y.end()) {
                sort(search->second.begin(), search->second.end());
                for (int i = 0; i < search->second.size(); i++)
                    printf("%lld%c", search->second[i], " \n"[i + 1 == search->second.size
()]);
            }
            else {
                cout << "Invalid!No such map." << endl;
            }
        }
    }
    return 0;
}
```

特别鸣谢：lzh大佬提供的~~开挂~~做法（具体sscanf语法请自行百度~~）

```cpp
# include <iostream>
# include <cstdio>
# include <string>
# include <set>
# include <map>

using namespace std;

set<int> a, b;
map<int, int> map_x;
map<int, set<int>> map_y;

int main()
{
    ios::sync_with_stdio(false);
    int T;
    cin >> T;
    while (T--) {
        int x, y;
        char str[100];
        cin >> str;
        if (str[3] == 'A') {
            sscanf(str, "addA(%d)", &x);
            if (!a.count(x))
                a.insert(x);
        }
        else if (str[3] == 'B') {
            sscanf(str, "addB(%d)", &y);
            if (!b.count(y))
                b.insert(y);
        }
        else if (str[0] == 'b') {
            sscanf(str, "buildmap(%d,%d)", &x, &y);
            if (map_x.count(x))
                cout << "Invalid!There is already a map from " << x << "." << endl;
            else if (!a.count(x) && b.count(y))
                cout << "Invalid!Can't find x." << endl;
            else if (a.count(x) && !b.count(y))
                cout << "Invalid!Can't find y." << endl;
            else if (!a.count(x) && !b.count(y))
                cout << "Invalid!Can't find both x and y." << endl;
            else {
                map_x[x] = y;
                map_y[y].insert(x);
            }
        }
        else if (str[4] == 'y') {
            sscanf(str, "findy(%d)", &x);
            if (!map_x.count(x))
                cout << "Invalid!No such map." << endl;
            else
                cout << map_x[x] << endl;
        }
        else {
            sscanf(str, "findx(%d)", &y);
            if (!map_y.count(y))
                cout << "Invalid!No such map." << endl;
            else {
                for (auto x : map_y[y])
                    cout << x << ' ';
                cout << endl;
            }
        }
    }
    return 0;
}

```



星 从未停歇

梦 永无止境

24 9 0

I. 买面包

2019-05-05 / 98 / 0

I. 买面包

不知从何时开始，zzz迷上了面包这一小食品，不仅便宜便捷，随时随地饿了就吃，而且也是ACM赛程中不可少的点心之一（特别是从早上打到下午，莫的吃午饭的那种比赛~~）！

好了，言归正传，在SZ市有一家IT牌面包店，种类繁多，回味无穷，但也有一个缺点，就是过道狭窄，而且入口与出口是同一个。也就是说，当你进入这家面包店挑选完面包之后，如果发现后面有人也进入了这家面包店，那么，你是无论如何也不可能越过这个人提前付款离开这家店的，因为走廊实在太窄了，所以你只能进入VIP席等待，只有抓住走廊中无人的这个时机才有可能出门结账！

由于这家店面的特殊性以及火爆程度，店长也很应景的提出一个问题，能回答者，将终身免单且享受绿色通道进出店面，zzz想要拿到这个至尊权限但又不会做这一题，只好抛给了学编程的你：

假设这家面包店一天内一共有n个顾客，按进入店内的顺序从1~n标号，而每当一位顾客出门结账时，他的编号也会被记录，问这一天内一共会有多少种不同的结账顺序？判断一个顾客是否能结账的条件如上所述，为了更加方便统计，现规定，如果店内存在编号大于自己的顾客，则自己也将受其约束，不能结账！

输入描述：

一行一个整数n，代表一天内一共有n个顾客进入。 $1 \le n \le 2 \times 10^5$

输出描述：

一行一个整数表示结果。

输入样例：

3

输出样例：

5

输入样例：

10

输出样例：

16796

大数模拟 + 简单的数学知识

将题意抽象一下就是火车进栈问题，问出栈的不同方案总数。

此问题也就是卡特兰数的结果。具体证明，蒟蒻就不给了，简单来说，就是火车进栈用+表示，栈顶火车出栈用-表示，问一共有多少种组合结果（显然，在任意一个时刻，-的数量不能多于+的数量）。

当然，如果只是照着卡特兰数的原始公式模拟大数运算的话这题是会超时的，所以我们还需要在这个公式的基础上进行优化。

我们可以发现，这个公式的结果一定为整数，而且先乘法后除法的模拟会让答案的结果先变长再变短，此处浪费了时间，所以，我们可以将公式的分子中的各个质因子的个数统计出来，再减去分母中相应的质因子的个数（当然了，要事先把素数筛出来），因为最后结果为整数，所以不存在相减后为负数的结果。然后，将这些统计出来的质因子进行乘法模拟，就可以保证我们的答案由短变长，直接得出结果。

对于一个数中某个质因子的个数，有这样一个结论：

$sum = \lfloor n / p \rfloor + \lfloor n / (p^2) \rfloor + \lfloor n / (p^3) \rfloor + \dots$ 直到分子小于分母。

下面为该结论的简单证明：

⌊ n / p ⌋ 为n中含有几个p，由于p^2中有两个p，所以这个个数也要加上，也就有了结论中的第二项 ⌊ n / (p^2) ⌋，依次类推可以得出上述结论。

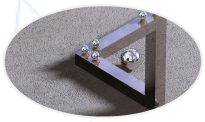
OJ评测姬仍然嫌弃这个代码，那就继续优化吧，~~如果神仙选手被这题搞崩了，就当作防AK的吧~~（也可能是因为我用了vector，常数比较大，各位神仙选手可以试试直接开数组），在模拟大数运算的时候，对于同一位，可以不止存一位数字，它可以是两位，三位...，在开了long long的情况下可以压9位，这样整个储存的长度又大大缩短，每次模拟的运算时间也大大缩短，只是在这里需要注意一下输出的格式，除了第一个数之外，其它的数字如果不足位数需要补前导零。

接下来就是激动人心的代码：

C++

```cpp
# include <iostream>
# include <cstdio>
# include <vector>
# include <algorithm>

using namespace std;

typedef long long LL;
const int maxn = (2e5 + 5) * 2;
int pos;
int cnt_prime_numbers[maxn], prime[maxn];
bool vis[maxn];

void get_prime()
{
     for (int i = 2; i < maxn; i++) {
        if (!vis[i]) {
            prime[pos++] = i;
        }
        for (int j = 0; j < pos && prime[j] <= maxn / i; j++) {
            vis[i * prime[j]] = true;
            if (i % prime[j] == 0)
                break;
        }
    }
    return;
}

int get_nums(int n, int p)
{
    int s = 0;
    while (n) {
        s += n / p;
        n /= p;
    }
    return s;
}

void multi(vector<LL> &a, int b)
{
    LL t = 0;
    for (int i = 0; i < a.size(); i++) {
        a[i] = a[i] * b + t;
        t = a[i] / 1000000000;
        a[i] %= 1000000000;
    }
    while (t) {
        a.push_back(t % 1000000000);
        t /= 1000000000;
```

```
    }
    return;
}

int main()
{
    get_prime();
    int n;
    cin >> n;
    for (int i = 2; i <= 2 * n; i++)
        if (!vis[i])
            cnt_prime_numbers[i] = get_nums(2 * n, i) - get_nums(n, i) * 2;
    for (int i = 2, k = n + 1; i <= k; i++)
        while (k % i == 0)
            cnt_prime_numbers[i]--, k /= i;
    vector<LL> ans;
    ans.push_back(1);
    for (int i = 2; i <= 2 * n; i++)
        while (cnt_prime_numbers[i]--)
            multi(ans, i);
    reverse(ans.begin(), ans.end());
    cout << ans[0];
    for (int i = 1; i < ans.size(); i++)
        printf("%09lld", ans[i]);
    cout << endl;
    return 0;
}
```



星 从未停歇

梦 永无止境

24 9 0

J. 漂泊

2019-05-05 / 153 / 0

J. 漂泊

这次的主人公不再是zzz，因为背景略显阴暗，引人深思，故仅当做一个小故事，让已经自闭的选手能够有事可做！（如有雷同，纯属巧合）

公元前9102年的一个夜晚，狂风呼啸，雷雨交加，一道道闪电划破天际，宛如一条条雷龙盘踞于乌云之上，时不时露个头，让世人在其无尽的威压下瑟瑟发抖。此时此刻，一望无垠的大海上漂泊着一艘小船，船上一共有n个人，闪烁不断的雷光映照出每个人阴沉的脸庞，忽有一道无奈的声音响起：“再按规矩来吧……”

此处画面暂停，引入旁白：

这艘小船已经不知道在这片大海上漂泊了多久，船员原本意气风发，风华正茂，自信满满的向往着前往开辟出一片属于自己的新天地。但却渐渐的发现，这片海，只有水，根本没有任何值得开发的东西。随着时间的推移，很多人都想过原路返回，但原路在哪？现在位于什么方位？早就迷失了方向，也没有人知道自己该去向何方。粮食也逐渐满足不了那么多人的需求，于是他们共同制定了一个规矩：每隔10天，这n个船员每个人抽取一个非负数字，然后围成一圈，编号从1~n，按编号升序 方向从1开始报数，当报到k时，这个人出列，下一个人重新从1开始报数（为了方便，同时也为了节省挑人的时间，k值不会太大，也就不再给出k的具体范围）……如此循环，直到选出n/2个幸运

儿。至此，n个船员分成了两组，每组再按初始编号顺序站成一排，两组需要各给出一个数字，这个数字代表自己组内一段最小的连续的成员个数，并且这些成员手中的数字的总和能够不小于S（若不存在这种情况，则给出的数字为0）。最后哪组给出的满足条件的数字比较大，则将该组成员全员丢进海里（特别的，如果有一组给出的是0，那么当然是丢这一组）， 如果两个数字相等，则这次没有人会被遗弃。

……

“又少了一半的人啊~”一位幸运儿说着，无助的眼神望向前方。因为剩下的人谁也不知道，自己究竟还能活过多少个10天，谁也不知道风暴究竟有没有尽头，谁也无法想象这场浩劫的终点到底是深渊还是所谓的 荣耀！

输入描述：

第一行输入三个整数n, k, S ($1 \le n \le 10^6$　$1 \le S \le 10^{10}$)
第二行输入n个整数x，代表每个人手中的数字。($0 \le x \le 2 \times 10^4$)

输出描述:

输出两个整数，中间用空格隔开。
第一个整数代表因报到k而出列的人的小组所给出的数字（该小组的人数需要小于等于另一个小组）。
第二个整数代表另一个小组给出的数字。

输入样例：

6 3 8
1 2 3 4 5 6

输出样例：

2 3

暴力 + 前缀和 + 二分

题目保证k不会太大，所以对于选出一半的人这一个操作，直接暴力找出来就行。

区间内求和，如果是按字典序看题的话，这是本场比赛出现的第二次，没错，又是前缀和，至于原因，往下看。

现在问题来到了怎么找一段区间，如果暴力枚举的话时间复杂度是$O(n^2)$的，并不可取。再看看处理完前缀和的序列，因为每个人手中的数字是非负数的，所以前缀和序列是升序，也就是说，如果有一段区间L ~ R满足要求，那么，L ~ R之后的任意一个位置都会满足要求，显然，对于一个确定的左端点，在选取右端点的时候可以采取二分的方法，规定二分的左边界为0，右边界为囊括这个左端

点开始，右边的所有人。二分结束后，如果存在满足条件的边界，则直接更新一下答案取min即可。值得注意的是，最后用于更新答案的数字是二分的结果+1，因为在一段包含两个端点区间内，数字总个数是r – l + 1，这里对于人数的更新也是同理。

而对于将这些人手中的数字全部加起来都不能满足条件的情况，答案是一次都不会得到更新的，所以可以在一开始将答案初始化为一个很大的数字（题目要求的区间范围到达不了即可），遍历完所有的点之后，如果仍然等于这个很大的数字，说明不能满足大于等于S的这个要求，那就直接输出0。

```cpp
# include <iostream>
# include <cstdio>
# include <climits>
# include <vector>
# include <algorithm>

using namespace std;

typedef long long LL;
typedef pair<LL, LL> PLL;
const int maxn = 1e6 + 5;
LL n, k, S, a[maxn], sum[maxn];
vector<PLL> F, G;
bool vis[maxn];

void search(LL &team, vector<PLL> &T)
{
    for (int i = 0; i < T.size(); i++)
        sum[i + 1] = sum[i] + T[i].second;
    for (int i = 1; i <= T.size(); i++) {
        LL l = 0, r = T.size() - i;
        while (l < r) {
            LL mid = l + r >> 1;
            if (sum[i + mid] - sum[i - 1] >= S)
                r = mid;
            else
                l = mid + 1;
        }
        if (sum[i + l] - sum[i - 1] >= S)
            team = min(team, l + 1);
    }
    return;
}

int main()
{
    scanf("%lld %lld %lld", &n, &k, &S);
    for (int i = 1; i <= n; i++)
        scanf("%lld", &a[i]);
    LL cnt = 0;
    for (int i = 1; F.size() < n / 2; i = i + 1 <= n ? i + 1 : 1) {
        if (vis[i])
            continue;
        cnt++;
        if (cnt == k) {
            cnt = 0;
            vis[i] = true;
            F.push_back({ i, a[i] });
        }
    }
    sort(F.begin(), F.end());
    for (int i = 1; i <= n; i++)
        if (!vis[i])
            G.push_back({ i, a[i] });
    LL team1, team2;
    team1 = team2 = INT_MAX;
    search(team1, F);
    search(team2, G);
    printf("%lld %lld\n", team1 == INT_MAX ? 0 : team1, team2 == INT_MAX ? 0 : team2);
    return 0;
}

```

至于为什么再次出现了前缀和，因为后续要用二分呐，大一这学期不是刚学嘛，一道送分题！！