第 2 周

Day06

1. 扫雷游戏

> **题目来源：** 洛谷
>
> **题目链接：** P2670[NOIP 2015 普及组] 扫雷游戏

【题目描述】

> 扫雷游戏是一款十分经典的单机小游戏。在 n 行 m 列的雷区中有一些格子含有地雷（称之为地雷格），其他格子不含地雷（称之为非地雷格）。玩家翻开一个非地雷格时，该格将会出现一个数字——提示周围格子中有多少个是地雷格。游戏的目标是在不翻出任何地雷格的条件下，找出所有的非地雷格。
>
> 现在给出 n 行 m 列的雷区中的地雷分布，要求计算出每个非地雷格周围的地雷格数。
>
> 注：一个格子的周围格子包括其上、下、左、右、左上、右上、左下、右下八个方向上与之直接相邻的格子。

【输入描述】

> 第一行是用一个空格隔开的两个整数 n 和 m，分别表示雷区的行数和列数。
>
> 接下来 n 行，每行 m 个字符，描述了雷区中的地雷分布情况。字符 $*$ 表示相应格子是地雷格，字符 ？ 表示相应格子是非地雷格。相邻字符之间无分隔符。
>
> 数据描述:
>
> 对于 100% 的数据，$1 \leq n \leq 100, 1 \leq m \leq 100$。

【输出描述】

> 输出文件包含 n 行，每行 m 个字符，描述整个雷区。用 $*$ 表示地雷格，用周围的地雷个数表示非地雷格。相邻字符之间无分隔符。

【示例一】

> 输入：
>
> ```
> 3 3
> *??
> ???
> ?*?
> ```

> 输出：
>
> *10
>
> 221
>
> 1*1

【示例二】

> 输入：
>
> 2 3
>
> ?*?
>
> *??
>
> 输出：
>
> 2*1
>
> *21

【解法】

解法：简单模拟。

- 遍历矩阵，遇到 `'*'` 直接输出，遇到 `'?'` 统计周围 `'*'` 的个数。
- 因为是一个 8 联通，因此可以两层循环枚举到周围的一圈。

【参考代码】

```
#include <iostream>

using namespace std;

const int N = 110;

int n, m;
char a[N][N];

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            cin >> a[i][j];

    for(int i = 1; i <= n; i++)
    {
```

```
19          for(int j = 1; j <= m; j++)
20          {
21              if(a[i][j] == '*') cout << '*';
22              else
23              {
24                  int cnt = 0;
25                  for(int dx = -1; dx <= 1; dx++)
26                  {
27                      for(int dy = -1; dy <= 1; dy++)
28                      {
29                          if(!dx && !dy) continue;
30                          int x = i + dx, y = j + dy;
31                          if(a[x][y] == '*') cnt++;
32                      }
33                  }
34                  cout << cnt;
35              }
36          }
37          cout << endl;
38      }
39
40
41      return 0;
42  }
```

2. ISBN号码

题目来源： 洛谷

题目链接： P1055[NOIP 2008 普及组] ISBN 号码

【题目描述】

每一本正式出版的图书都有一个 ISBN 号码与之对应，ISBN 码包括 9 位数字、1 位识别码和 3 位分隔符，其规定格式如 `x-xxx-xxxxx-x`，其中符号 `-` 就是分隔符（键盘上的减号），最后一位是识别码，例如 `0-670-82162-4` 就是一个标准的 ISBN 码。ISBN 码的首位数字表示书籍的出版语言，例如 0 代表英语；第一个分隔符 `-` 之后的三位数字代表出版社，例如 670 代表维京出版社；第二个分隔符后的五位数字代表该书在该出版社的编号；最后一位为识别码。

识别码的计算方法如下：

首位数字乘以 1 加上次位数字乘以 2 ……以此类推，用所得的结果 $mod11$，所得的余数即为识别码，如果余数为 10，则识别码为大写字母 X。例如 ISBN 号码 `0-670-82162-4` 中的识别码 4 是这样得到的：对 `067082162` 这 9 个数字，从左至右，分别乘以 1,2,…,9 再求和，即 $0\times1+6\times2+\ldots+2\times9=158$，然后取 $158mod11$ 的结果 4 作为识别码。

> 你的任务是编写程序判断输入的 ISBN 号码中识别码是否正确，如果正确，则仅输出 `Right`；如果错误，则输出你认为是正确的 `ISBN` 号码。

【输入描述】

> 一个字符序列，表示一本书的 ISBN 号码（保证输入符合 `ISBN` 号码的格式要求）。

【输出描述】

> 一行，假如输入的 `ISBN` 号码的识别码正确，那么输出 `Right`，否则，按照规定的格式，输出正确的 ISBN 号码（包括分隔符 `-`）。

【示例一】

> 输入：
>
> 0-670-82162-4
>
> 输出：
>
> `Right`

【示例二】

> 输入：
>
> 0-670-82162-0
>
> 输出：
>
> 0-670-82162-4

【解法】

解法：直接模拟。

- 从前往后遍历 ISBN 号码，计算出所有数字乘上权值之和结果之后与识别码作比较。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

int main()
{
    string s; cin >> s;

    int sum = 0, cnt = 1;
    for(int i = 0; i < s.size() - 1; i++)
    {
        if(s[i] != '-')
```

```
        {
            sum += (s[i] - '0') * cnt;
            cnt++;
        }
    }

    sum %= 11;

    if(sum == 10 && s.back() == 'X') cout << "Right" << endl;
    else if(s.back() - '0' == sum) cout << "Right" << endl;
    else
    {
        if(sum == 10) s[12] = 'X';
        else s[12] = sum + '0';
        cout << s << endl;
    }

    return 0;
}
```

3. 奶酪

> **题目来源：** 洛谷
>
> **题目链接：** [NOIP2017]奶酪

【题目描述】

> 现有一块大奶酪，它的高度为 h，它的长度和宽度我们可以认为是无限大的，奶酪中间有许多半径相同的球形空洞。我们可以在这块奶酪中建立空间坐标系，在坐标系中，奶酪的下表面为 $z=0$，奶酪的上表面为 $z=h$。
>
> 现在，奶酪的下表面有一只小老鼠 Jerry，它知道奶酪中所有空洞的球心所在的坐标。如果两个空洞相切或是相交，则 Jerry 可以从其中一个空洞跑到另一个空洞，特别地，如果一个空洞与下表面相切或是相交，Jerry 则可以从奶酪下表面跑进空洞；如果一个空洞与上表面相切或是相交，Jerry 则可以从空洞跑到奶酪上表面。
>
> 位于奶酪下表面的 Jerry 想知道，在不破坏奶酪的情况下，能否利用已有的空洞跑 到奶酪的上表面去?
>
> 空间内两点 $P_1(x_1,y_1,z_1)$、$P_2(x_2,y_2,z_2)$ 的距离公式如下：
>
> $$dist(P_1,P_2)=(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2$$

【输入描述】

> 每个输入文件包含多组数据。

> 第一行，包含一个正整数 T ，代表该输入文件中所含的数据组数。
>
> 接下来是 T 组数据，每组数据的格式如下： 第一行包含三个正整数 n, h, r ，两个数之间以一个空格分开，分别代表奶酪中空洞的数量，奶酪的高度和空洞的半径。
>
> 接下来的 n 行，每行包含三个整数 x, y, z ，两个数之间以一个空格分开，表示空洞球心坐标为 (x, y, z) 。

【输出描述】

> T 行，分别对应 T 组数据的答案，如果在第 i 组数据中，Jerry 能从下表面跑到上表面，则输出 `Yes` ，如果不能，则输出 `No` 。

【示例一】

> 输入：
>
> ```
> 3
> 2 4 1
> 0 0 1
> 0 0 3
> 2 5 1
> 0 0 1
> 0 0 4
> 2 5 2
> 0 0 2
> 2 0 4
> ```
>
> 输出：
>
> ```
> Yes
> No
> Yes
> ```

【解法】

解法：并查集。

- 在最上方以及最下方加入两个编号为 0 和 n + 1 的球；
- 将所有能够联通的空洞合并在一起；
- 最后判断 0 和 n + 1 是否联通。

【参考代码】

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 1e3 + 10;

LL n, h, r;
LL x[N], y[N], z[N];

int fa[N]; // 并查集

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

void merge(int x, int y)
{
    fa[find(x)] = find(y);
}

bool check(int i, int j)
{
    LL d = (x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] - y[j]) + (z[i] - z[j]) * (z[i] - z[j]);

    return d <= 4 * r * r;
}

int main()
{
    int T; cin >> T;
    while(T--)
    {
        cin >> n >> h >> r;

        for(int i = 0; i <= n + 1; i++) fa[i] = i;

        for(int i = 1; i <= n; i++)
        {
            cin >> x[i] >> y[i] >> z[i];
            // 判断下表面
            if(z[i] <= r) merge(i, 0);
            // 判断上表面
            if(z[i] + r >= h) merge(i, n + 1);
```

```
47                 // 判断其余的球
48                 for(int j = 1; j < i; j++)
49                 {
50                     if(check(i, j)) merge(i, j);
51                 }
52             }
53 
54             if(find(0) == find(n + 1)) cout << "Yes" << endl;
55             else cout << "No" << endl;
56         }
57 
58         return 0;
59     }
```

4. 均分纸牌

题目来源： 洛谷

题目链接： P1031 [NOIP2002 提高组] 均分纸牌

【题目描述】

有 N 堆纸牌，编号分别为 $1, 2, ..., N$。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为 1 堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 $N-1$ 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。

例如 $N=4$ 时，4 堆纸牌数分别为 $9, 8, 17, 6$。

移动 3 次可达到目的：

- 从第三堆取 4 张牌放到第四堆，此时每堆纸牌数分别为 $9, 8, 13, 10$。
- 从第三堆取 3 张牌放到第二堆，此时每堆纸牌数分别为 $9, 11, 10, 10$。
- 从第二堆取 1 张牌放到第一堆，此时每堆纸牌数分别为 $10, 10, 10, 10$。

【输入描述】

第一行共一个整数 N，表示纸牌堆数。
第二行共 N 个整数 $A_1, A_2, ..., A_N$，表示每堆纸牌初始时的纸牌数。

对于 100% 的数据，$1 \le N \le 1001 \le N \le 100, 1 \le Ai \le 10000$。

【输出描述】

> 共一行，即所有堆均达到相等时的最少移动次数。

【示例一】

> 输入：
>
> 4
>
> 9 8 17 6
>
> 输出：
>
> 3

【解法】

解法：贪心。

先算出所有纸牌均分之后的平均数 x ，移动次数为 cnt 。然后针对每一堆分类讨论：

- 如果 $a[i]=x$ ：不需要调整，跳过；
- 如果 $a[i]<x$ ：需要补充 $x-a[i]$ 个。此时： $a[i+1]-=x-a[i]$ ， $cnt++$ ；
- 如果 $a[i]>x$ ：需要移走 $a[i]-x$ 个。此时： $a[i+1]+=a[i]-x$ ， $cnt++$ ；

遍历整个数组一遍之后，就可得最少移动次数。

善于思考的同学此时就会有大大的疑惑了，你这个策略为什么是正确的？而且在补充和移走的过程中，会让某一些位置的拍张变为负数。比如 $[0,0,0,4]$ ，按照上述过程模拟，第一步的时候，第二个元素就变成 -1 ，这样显然是不合法的。

因此我们需要证明两点：

1. 上述转移一定是最小转移次数；
2. 上述从左往右的转移过程虽然会出现负数，但是实际情况下一定存在一种合法的转移顺序。

上述转移一定是最小转移次数可以用代数法证明：

> 设 x_i 表示从 $a[i]$ 与 $a[i+1]$ 之间转移情况的汇总值。如果 x_i 是正数，表明虽然转移很多次，但是整体是 $a[i]$ 均给 $a[i+1]$ ；如果 x_i 是负数，表明虽然转移很多次，但是整体是 $a[i+1]$ 均给 $a[i]$ 。
>
> 设平均数是 x ，那么我们会得到下面的方程组：
>
> $$a_1+x_1=x$$
>
> $$a_2-x_1+x_2=x$$
>
> $$a_3-x_2+x_3=x$$

$$\dots$$

$$a_{n-1} - x_{n-2} + x_{n-1} = x$$

$$a_n - x_{n-1} = x$$

对这个方程组移项求解，我们可以得到每一个 x_i 的值：

$$x_1 = x - a_1$$

$$x_2 = x - a_2 + x_1$$

$$x_3 = x - a_3 + x_2$$

$$\dots$$

$$x_{n-1} = x - a_{n-1} + x_{n-2}$$

我们发现，每一个 x_i 其实都是固定的，那么我们仅需让任意两个数之间转移的时候，只用转移一次 x_i 即可。也就是所有 x_i 中非零元素的个数。

可以用构造法证明一定存在合法的转移顺序，使的每次转移都只用转移一次 x_i：

从前往后模拟整个过程，对于 $a[1]$，我们有以下三种情况：

1. $a[1] = x$：正好是平均数，不做处理，继续处理 $[2, n]$ 区间；
2. $a[1] > x$：这个操作一定不会造成不合法的情况，直接把 $a[1] - x$ 张牌转递给 $a[2]$；
3. $a[1] < x$：这个操作需要 $a[2]$ 补上 $a[1] - x$，此时为了避免出现负数，我们可以
   a. 先在 $a[2]$ 位置上打上欠债的标记；
   b. 然后去处理 $[2, n]$ 这段区间；
   c. 处理完毕之后，再来把欠的 $a[1] - x$ 补上。
   d. 此时要注意，处理 $a[2]$ 时，要用 $a[2]$ 与 $x + (a[1] - x)$ 作比较，表明我这个位置需要的更多。

经过上面的处理，我们就又把问题变成一个重复的子问题，这样下去一定可以构造出来一个合法的序列，其实这里蕴含了递归的思想。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 110;
```

```cpp

int n, x;
int a[N];

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i];
        x += a[i];
    }

    x /= n;

    int ret = 0;
    for(int i = 1; i <= n; i++)
    {
        if(a[i] == x) continue;

        ret++;
        a[i + 1] -= x - a[i];
    }

    cout << ret << endl;

    return 0;
}
```


Day07

1. 乘方

> **题目来源：** 洛谷
>
> **题目链接：** P8813[CSP-J 2022] 乘方

【题目描述】

> 小文同学刚刚接触了信息学竞赛，有一天她遇到了这样一个题：给定正整数 a 和 b ，求 a^b 的值是多少。
>
> a^b 即 b 个 a 相乘的值，例如 23 即为 3 个 2 相乘，结果为 2×2×2=8。
>
> “简单！”小文心想，同时很快就写出了一份程序，可是测试时却出现了错误。

> 小文很快意识到，她的程序里的变量都是 `int` 类型的。在大多数机器上， `int` 类型能表示的最大数为 $2^{32}-1$，因此只要计算结果超过这个数，她的程序就会出现错误。
>
> 由于小文刚刚学会编程，她担心使用 `int` 计算会出现问题。因此她希望你在 a^b 的值超过 10^9 时，输出一个 `-1` 进行警示，否则就输出正确的 a^b 的值。
>
> 然而小文还是不知道怎么实现这份程序，因此她想请你帮忙。

【输入描述】

> 输入共一行，两个正整数 a, b。
>
> 数据说明：
>
> 对于 10% 的数据，保证 $b=1$。
> 对于 30% 的数据，保证 $b \leq 2$。
> 对于 60% 的数据，保证 $b \leq 30, a^b \leq 10^{18}$。
> 对于 100% 的数据，保证 $1 \leq a, b \leq 10^9$。

【输出描述】

> 输出共一行，如果 a^b 的值不超过 10^9，则输出 a^b 的值，否则输出 `-1`。

【示例一】

> 输入：
>
> 10 9
>
> 输出：
>
> 1000000000

【示例二】

> 输入：
>
> 23333 66666
>
> 输出：
>
> -1

【解法】

解法：快速幂。

在快速幂计算的过程中，判断 a 以及 ret 是否会超出 1e9。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

LL qpow(LL a, LL b)
{
    LL ret = 1;
    while(b)
    {
        if(a > 1e9) return -1; // 防止 longlong 都存不下

        if(b & 1) ret = ret * a;

        if(ret > 1e9) return -1;

        b >>= 1;
        a = a * a;
    }
    return ret;
}

int main()
{
    LL a, b; cin >> a >> b;

    cout << qpow(a, b) << endl;

    return 0;
}
```


2. 领地选择

> **题目来源：** 洛谷
>
> **题目链接：** P2004 领地选择

【题目描述】

> 作为在虚拟世界里统帅千军万马的领袖，小 Z 认为天时、地利、人和三者是缺一不可的，所以，谨慎地选择首都的位置对于小 Z 来说是非常重要的。
>
> 首都被认为是一个占地 $C \times C$ 的正方形。小 Z 希望你寻找到一个合适的位置，使得首都所占领的位置的土地价值和最高。

【输入描述】

第一行三个整数 N, M, C ，表示地图的宽和长以及首都的边长。

接下来 N 行每行 M 个整数，表示了地图上每个地块的价值。价值可能为负数。

对于 60% 的数据， $N, M \leq 50$ 。

对于 90% 的数据， $N, M \leq 300$ 。

对于 90% 的数据， $1 \leq N, M \leq 10^3$ ， $1 \leq C \leq min(N, M)$ 。

【输出描述】

一行两个整数 X, Y ，表示首都左上角的坐标。

【示例一】

输入：

```
3 4 2
1 2 3 1
-1 9 0 2
2 0 1 1
```

输出：

```
1 2
```

【解法】

解法：二维前缀和。

- 先预处理出来前缀和矩阵；
- 然后枚举所有的 c*c 的矩阵，利用前缀和矩阵快速求和。

【参考代码】

```
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 1e3 + 10;

int n, m, c;
LL f[N][N];
```

```cpp

int main()
{
    cin >> n >> m >> c;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            int x; cin >> x;
            f[i][j] = f[i - 1][j] + f[i][j - 1] - f[i - 1][j - 1] + x;
        }
    }

    LL ret = -1e18;
    int x, y;

    for(int x1 = 1; x1 <= n - c + 1; x1++)
    {
        for(int y1 = 1; y1 <= m - c + 1; y1++)
        {
            int x2 = x1 + c - 1, y2 = y1 + c - 1;
            LL t = f[x2][y2] - f[x2][y1 - 1] - f[x1 - 1][y2] + f[x1 - 1][y1 - 1];
            if(t > ret)
            {
                ret = t;
                x = x1, y = y1;
            }
        }
    }

    cout << x << " " << y << endl;

    return 0;
}
```


3. 砝码称重

> 题目来源： 洛谷
>
> 题目链接： P2347 [NOIP1996 提高组] 砝码称重

【题目描述】

> 设有 $1g$、$2g$、$3g$、$5g$、$10g$、$20g$ 的砝码各若干枚（其总重 ≤ 1000），可以表示成多少种重量?

【输入描述】

> 输入方式： $a_1, a_2, a_3, a_4, a_5, a_6$
>
> （表示 $1g$ 砝码有 a_1 个， $2g$ 砝码有 a_2 个，……， $20g$ 砝码有 a_6 个）

【输出描述】

> 输出方式： `Total=N`
>
> （N 表示用这些砝码能称出的不同重量的个数，但不包括一个砝码也不用的情况）

【示例一】

> 输入：
>
> 1 1 0 0 0 0
>
> 输出：
>
> Total=3

【解法】

解法：动态规划 - 多重背包问题。

1. 状态表示：

   $dp[i][j]$ 表示：从前 i 个砝码中挑选，是否能凑成总质量为 j 。

   最终结果就是 dp 表里面最后一行为 $true$ 的个数。

2. 状态转移方程：

   根据第 i 个砝码选的个数，能分成 $a[i]+1$ 种情况，设选了 k 个：

   - 此时重量为 $w[i] \times k$，那么就应该去前面看看能不能凑成 $j - w[i] \times k$，即：$dp[i-1][j-w[i] \times k]$

   在所有的情况里面，只要有一个为真，$dp[i][j]$ 就为真。

3. 初始化：

   $dp[0][0] = true$，起始状态，也是为了后续填表有意义且正确。

4. 填表顺序：

   从上往下每一行，每一行从左往右。

【参考代码】

```cpp
#include <iostream>

using namespace std;

const int N = 10, M = 1010;

int n = 6, m = 1000;
int w[N] = {0, 1, 2, 3, 5, 10, 20};
int a[N];
int f[M];

int main()
{
    for(int i = 1; i <= n; i++) cin >> a[i];

    f[0] = true;
    for(int i = 1; i <= n; i++)
    {
        for(int j = m; j >= 0; j--)
        {
            for(int k = 0; k <= a[i] && k * w[i] <= j; k++)
            {
                f[j] = f[j] || f[j - k * w[i]];
            }
        }
    }

    int ret = 0;
    for(int i = 1; i <= m; i++)
        if(f[i])
            ret++;

    cout << "Total=" << ret << endl;

    return 0;
}
```


4. 奶牛晒衣服

> 题目来源： 洛谷

> 题目链接： P1843 奶牛晒衣服

【题目描述】

> 一件衣服在自然条件下用一秒的时间可以晒干 a 点湿度。抠门的熊大妈只买了一台烘衣机。使用用一秒烘衣机可以让一件衣服额外烘干 b 点湿度（一秒晒干 $a+b$ 湿度），但在同一时间内只能烘一件衣服。现在有 n 件衣服，第 i 衣服的湿度为 w_i（保证互不相同），要你求出弄干所有衣服的最少时间（湿度为 0 为干）。

【输入描述】

> 第一行三个整数，分别为 n,a,b
> 接下来 2 到 $n+1$ 行，第 i 行输入 w_i。
>
> $1 \le w_i,a,b,n \le 5 \times 10^5$

【输出描述】

> 一行，弄干所有衣服的最少时间。

【示例一】

> 输入：
>
> ```
> 3 2 1
> 1
> 2
> 3
> ```
>
> 输出：
>
> ```
> 1
> ```

【解法】

解法：二分答案。

设经过的时间是 x。根据题意，我们可以发现如下性质：

- 经过的时间如果是 x 的话，烘干机的「使用次数」最多也是 x；
- 当 x 在「增大」的时候，能弄干的衣服在「增多」；
- 当 x 在「减小」的时候，能弄干的衣服也在「减少」。

那么在整个「解空间」里面，设弄干所有衣服的最少时间是 ret，于是有：

- 当 $x \ge ret$ 时，我们「能弄干」所有衣服；
- 当 $x < ret$ 时，我们「不能弄干」所有衣服。

在解空间中，根据 ret 的位置，可以将解集分成两部分，具有「二段性」，那么我们就可以「二分答案」。

接下来的重点就是，给定一个时间 x ，判断「是否能把所有的衣服全部弄干」。当时间为 x 时，所有衣服能够自然蒸发 $a \cdot x$ 的湿度，于是：

- 如果 $w[i] \le a \cdot x$ ：让它自然蒸发；
- 如果 $w[i] > a \cdot x$ ：需要用烘干机烘干 $t = w[i] - a \cdot x$ 的湿度，次数为 $t \ / \ b + (t \ \% \ b == 0 \ ? \ 0 : 1)$ 。

那么我们可以遍历所有的衣服，计算烘干机的「使用次数」：

- 如果使用次数「大于」给定的时间 x ，说明不能全部弄干；
- 如果使用次数「小于等于」给定的时间 x ，说明能全部弄干。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 5e5 + 10;

LL n, a, b;
LL w[N];

bool check(LL x)
{
    int cnt = 0;
    for(int i = 1; i <= n; i++)
    {
        if(w[i] <= a * x) continue;

        LL d = w[i] - a * x;
        cnt = cnt + d / b + (d % b == 0 ? 0 : 1);
    }
    return cnt <= x;
}

int main()
{
```

```
27      cin >> n >> a >> b;
28      for(int i = 1; i <= n; i++) cin >> w[i];
29
30      LL l = 1, r = 5e5;
31      while(l < r)
32      {
33          LL mid = (l + r) / 2;
34          if(check(mid)) r = mid;
35          else l = mid + 1;
36      }
37
38      cout << l << endl;
39
40      return 0;
41  }
```

Day08

1. 糖果

> **题目来源：** 洛谷
>
> **题目链接：** P9226 糖果

【题目描述】

> 三年级七班共有 n 名同学。体育课开始，他们从左到右站成了一排，准备进行报数分组。
>
> 体育老师口袋里有很多袋装的糖果（一个袋子里有很多糖果），他准备在分组的过程中顺便将这些袋装糖果分给同学们。
>
> 具体地，在从左到右报数的过程中，每报数 k 名同学，体育老师就会将这 k 名同学划为一组，同时给予这 k 名同学中的最后一名一袋糖果，让这最后一名同学来负责分发给组内的同学。也就是说，体育老师会依次给从左到右第 $k, 2k, \cdots$ 名同学一袋糖果。
>
> 恰巧，三年级六班的同学听到了三年级七班发糖果的消息，于是他们打算混入队伍的末尾（即队伍最右侧），企图白嫖到一袋糖果。
>
> 三年级六班的同学想知道，他们**至少**需要向队伍末尾混入多少人。

【输入描述】

> 一行两个整数 n, k。
>
> 对于 100% 的数据，$1 \le n \le 10^9$，$2 \le k \le 10^9$。

【输出描述】

> 一行一个整数，表示答案。

【示例一】

> 输入：
>
> 10 3
>
> 输出：
>
> 2
>
> 说明：
>
> 此时每 3 人分一组。三年级六班只需要向队伍中混入 2 名同学，就可以和原来三年级七班的最后 1 名同学一同凑成一组。
>
> 由于混入的这 2 名同学在队伍的末尾，因此这一组中的最后一名同学一定是隔壁班的同学，因此隔壁班的同学可以白嫖到一袋糖果。

【示例二】

> 输入：
>
> 16 4
>
> 输出：
>
> 4
>
> 说明：
>
> 此时每 4 人分一组。三年级七班的所有同学都已经分好了组，因此三年级六班需要完整地向队伍中混入 4 名同学凑成一组，才能白嫖到一袋糖果。

【解法】

解法：模拟。

- 仅需读懂题意...
- 让模 k 之后的余数凑够 k 即可。

【参考代码】

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n, k; cin >> n >> k;
```

```
8
9       cout << (k - n % k) << endl;
10
11      return 0;
12  }
```

2. Fixed Points

> **题目来源：** 洛谷
>
> **题目链接：** CF347B Fixed Points

【题目描述】

> 给你一个 0 ~ n-1 的全排列，可以交换两个数的位置一次，问最多能有多少个数与自己所在的位置对应

【示例一】

> 输入：
>
> 5
>
> 0 1 3 4 2
>
> 输出：
>
> 3

【解法】

解法：分类讨论。

假设在正确位置上的数字个数为 p，扫描所有数：

1. 如果所有数字都在正确的位置上，也就是 p = n，可以直接输出 p；
2. 如果有 2 个数字交换位置，可以将这两个数字交换位置，所以答案为 p + 2；
3. 如果没有 2 个数字交换位置，最好的方案是让其中 1 个数回到正确位置，答案为 p + 1。

【参考代码】

代码块

```
1   #include <iostream>
2
3   using namespace std;
4
```

```cpp
const int N = 1e5 + 10;

int n;
int a[N];

int main()
{
    cin >> n;
    int p = 0;
    for(int i = 0; i < n; i++)
    {
        cin >> a[i];
        if(a[i] == i) p++;
    }

    if(p == n) cout << n << endl;
    else
    {
        for(int i = 0; i < n; i++)
        {
            if(i != a[i] && a[a[i]] == i)
            {
                cout << p + 2 << endl;
                return 0;
            }
        }

        cout << p + 1 << endl;
    }

    return 0;
}
```


3. 单词接龙

> **题目来源：** 洛谷
>
> **题目链接：** P1019 [NOIP 2000 提高组] 单词接龙

【题目描述】

> 单词接龙是一个与我们经常玩的成语接龙相类似的游戏，现在我们已知一组单词，且给定一个开头的字母，要求出以这个字母开头的最长的“龙”（每个单词都最多在“龙”中出现两次），在两个单词相连时，其重合部分合为一部分，例如 `beast` 和 `astonish`，如果接成一条龙则变为 `beastonish`，另外相邻的两部分不能存在包含关系，例如 `at` 和 `atide` 间不能相连。

【输入描述】

> 输入的第一行为一个单独的整数 n 表示单词数，以下 n 行每行有一个单词，输入的最后一行为一个单个字符，表示“龙”开头的字母。你可以假定以此字母开头的“龙”一定存在。
>
> $n \leq 20$

【输出描述】

> 只需输出以此字母开头的最长的“龙”的长度。

【示例一】

> 输入：
>
> 5
>
> at
>
> touch
>
> cheat
>
> choose
>
> tact
>
> a
>
> 输出：
>
> 23

【解法】

解法：暴搜。

从龙头开始，暴搜所有的接龙方案，找出其中最长的那个即可。

【参考代码】

```
#include <iostream>

using namespace std;

const int N = 30;

int n, ret;
string s[N];

```

```cpp
int cnt[N]; // 标记字符串用了几次

void dfs(string path)
{
    if(path.size() > ret) ret = path.size();

    // 枚举所有的字符串，看看能不能拼接
    for(int i = 1; i <= n; i++)
    {
        if(cnt[i] >= 2) continue;

        int cur1 = path.size() - 1, cur2 = 0;
        while(cur1 >= 1 && cur2 < s[i].size() - 1)
        {
            if(path.substr(cur1) == s[i].substr(0, cur2 + 1))
            {
                cnt[i]++;
                dfs(path + s[i].substr(cur2 + 1));
                cnt[i]--;
            }

            cur1--; cur2++;
        }
    }
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> s[i];

    char ch; cin >> ch;

    for(int i = 1; i <= n; i++)
    {
        if(s[i][0] == ch)
        {
            cnt[i]++;
            dfs(s[i]);
            cnt[i]--;
        }
    }

    cout << ret << endl;

    return 0;
}
```

4. 能量项链

> **题目来源：** 洛谷
>
> **题目链接：** P1063 [NOIP2006 提高组] 能量项链

【题目描述】

> 在 Mars 星球上，每个 Mars 人都随身佩带着一串能量项链。在项链上有 N 颗能量珠。能量珠是一颗有头标记与尾标记的珠子，这些标记对应着某个正整数。并且，对于相邻的两颗珠子，前一颗珠子的尾标记一定等于后一颗珠子的头标记。因为只有这样，通过吸盘（吸盘是 Mars 人吸收能量的一种器官）的作用，这两颗珠子才能聚合成一颗珠子，同时释放出可以被吸盘吸收的能量。如果前一颗能量珠的头标记为 m，尾标记为 r，后一颗能量珠的头标记为 r，尾标记为 n，则聚合后释放的能量为 $m \times r \times n$（Mars 单位），新产生的珠子的头标记为 m，尾标记为 n。
>
> 需要时，Mars 人就用吸盘夹住相邻的两颗珠子，通过聚合得到能量，直到项链上只剩下一颗珠子为止。显然，不同的聚合顺序得到的总能量是不同的，请你设计一个聚合顺序，使一串项链释放出的总能量最大。
>
> 例如：设 $N = 4$，4 颗珠子的头标记与尾标记依次为 $(2,3)(3,5)(5,10)(10,2)$。我们用记号 $\oplus$ 表示两颗珠子的聚合操作，$(j \oplus k)$ 表示第 j,k 两颗珠子聚合后所释放的能量。则第 $4,1$ 两颗珠子聚合后释放的能量为：$(4 \oplus 1) = 10 \times 2 \times 3 = 60$。
>
> 这一串项链可以得到最优值的一个聚合顺序所释放的总能量为：
>
> $(((4 \oplus 1) \oplus 2) \oplus 3) = 10 \times 2 \times 3 + 10 \times 3 \times 5 + 10 \times 5 \times 10 = 710$。

【输入描述】

> 第一行是一个正整数 $N(4 \le N \le 100)$，表示项链上珠子的个数。第二行是 N 个用空格隔开的正整数，所有的数均不超过 1000。第 i 个数为第 i 颗珠子的头标记 $(1 \le i \le N)$，当 $i < N$ 时，第 i 颗珠子的尾标记应该等于第 $i+1$ 颗珠子的头标记。第 N 颗珠子的尾标记应该等于第 1 颗珠子的头标记。
>
> 至于珠子的顺序，你可以这样确定：将项链放到桌面上，不要出现交叉，随意指定第一颗珠子，然后按顺时针方向确定其他珠子的顺序。

【输出描述】

> 一个正整数 $E(E \le 2.1 \times 10^9)$，为一个最优聚合顺序所释放的总能量。

【示例一】

> 输入：
>
> ```
> 4
> 2 3 5 10
> ```

> 输出：
>
> 710

【解法】

解法：区间 dp。

处理本道题的环形问题：倍增。

倍增之后，就是石子合并的变形。石子合并每次合并的是相邻两个元素，而这道题一次会在数组中选三个数合并。但是分析的方式是不变的。

1. 状态表示：

   $dp[i][j]$ 表示：合并区间 $[i,j]$ 的项链，最多能释放多少的能量。

   那么 $dp[i][n+i]$ 就是最终结果。

2. 状态转移方程：

   根据区间的分割点 $k(i<k<j)$，可以将区间分成 $[i,k]$ 以及 $[k,j]$，注意这道题 k 的取值范围以及分割的区间的结果。因为我们的项链每次需要数组中两个元素，所以区间长度最少是 2，因此 k 不能等于 i 或者 j。又因为左边区间合并之后，剩余的元素是 $a[i]$ 和 $a[k]$，右边的区间合并之后必须是 $a[k]$ 开头才能继续合并。这样的数据范围和区间分割的结果是根据题意来的。

   那么，分割之后再合并的最大能量为 $dp[i][k]+dp[k][j]+a[i]\times a[k]\times a[j]$。因为要的是最大能量，所以状态转移方程就是所有 k 变化过程中的最大值。

3. 初始化：

   长度为 1 不合法，但是里面的值是 0 就不影响结果。

   长度为 2 不能合并，里面的值是 0 即可。

4. 填表顺序：

   先枚举区间长度，再枚举左端点，右端点通过计算。

【参考代码】

代码块

```
#include <iostream>

```

```cpp
using namespace std;

const int N = 210;

int n;
int a[N];

int f[N][N];

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        cin >> a[i];
        a[i + n] = a[i]; // 倍增
    }

    for(int len = 3; len <= n + 1; len++)
    {
        for(int i = 1; i + len - 1 <= n + n; i++)
        {
            int j = i + len - 1;
            for(int k = i + 1; k < j; k++)
            {
                // [i, k] [k, j]
                f[i][j] = max(f[i][j], f[i][k] + f[k][j] + a[i] * a[k] * a[j]);
            }
        }
    }

    int ret = 0;
    for(int i = 1; i <= n; i++)
    {
        ret = max(ret, f[i][i + n]);
    }

    cout << ret << endl;

    return 0;
}
```


Day09

1. Popsicle

题目来源： 洛谷

题目链接： P9354 「SiR-1」Popsicle

【题目描述】

猫猫有若干个雪糕棒排成一排，每个雪糕棒上有一个 0~9 的数字，并且满足最左边的雪糕棒上写的数字不为 0。猫猫认为这一排雪糕棒从左到右依次构成了十进制正整数 n 。

猫猫认为 0 是美好的，所以她会尽可能把 n 变成 0，也就是把所有雪糕棒都拿走。

猫猫每次会进行一次操作。每次操作选择一个数字非 0 的雪糕棒，并将其减 1。这之后，如果最左边有连续的一些数字为 0 的雪糕棒（也即 n 出现了前导 0），猫猫会把这些雪糕棒拿走。

小老鼠会来捣乱，它会在某个时刻（可能是所有操作开始之前，也可能是猫猫任意一次操作之后）改变某个雪糕棒上的一个数字。小老鼠**总共只能改变一个数字**。

小老鼠希望操作次数尽量多，猫猫希望操作次数尽量少，所以她想知道二者都使用最优策略时，她的操作次数。

【输入描述】

本题单个测试点内有多组数据。

第一行，一个正整数 T 表示数据组数。对于每组数据：

- 仅一行，一个正整数 n 。

本题采用捆绑测试。

- Subtask 0（13 pts）：$n \leq 99$ 。
- Subtask 1（13 pts）：$n = 10^k$，k 为自然数。
- Subtask 2（13 pts）：$n = 10^{k-1}$，k 为正整数。
- Subtask 3（13 pts）：$n \leq 999999$ 。
- Subtask 4（48 pts）：无特殊限制。

对于所有数据，$1 \leq T \leq 3333$，$1 \leq n \leq 9,999,999,999,999 (= 10^{13} - 1)$ ，毕竟猫猫最多一捆只有 13 根雪糕嘛。

【输出描述】

共 T 行，每行一个整数，表示答案。

【示例一】

输入：

```
2
1100
```

> 11332132121
>
> 输出：
>
> 11
>
> 28
>
> 说明：
>
> 对于第一组数据，小老鼠可以一开始就将 1100 变为 1109，这样猫猫共需要 1+1+9 次操作把 n 变为 0。

【解法】

解法：贪心。

- 如果整个序列中初始状态没有 0，那么只能在某个数减为 1 的时候，变成 9。那就是数位之和 + 8；
- 如果整个序列中初始状态有 0，那么就把 0 变成 9。那就是位数之和 + 9。

【参考代码】

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

int main()
{
    int T; cin >> T;
    while(T--)
    {
        string s; cin >> s;
        int sum = 0;
        bool flag = false;

        for(auto ch : s)
        {
            sum += ch - '0';
            if(ch == '0') flag = true;
        }

        if(flag) cout << sum + 9 << endl;
        else cout << sum + 8 << endl;
    }
```

```
25
26        return 0;
27    }
```

2. Holidays

> **题目来源：** 洛谷
>
> **题目链接：** CF44C Holidays

【题目描述】

> n 天假期，安排 m 个人来浇花，第 i 个人负责 [a[i], b[i]] 天，问花是否可以每天都被浇水且不重复。可以的话输出“OK”，不可以的话输出最早出问题的那天的天号以及那天花被浇了多少次水。
>
> 1≤n,m≤100 1≤a[i]≤b[i]≤n b[i]≤a[i+1]

【示例一】

> 输入：
>
> 10 5
>
> 1 2
>
> 3 3
>
> 4 6
>
> 7 7
>
> 8 10
>
> 输出：
>
> OK

【解法】

解法：差分。

利用差分处理每一次区间操作，还原出原数组之后，找出第一个非 1 的位置。

【参考代码】

```
1   #include <iostream>
2
3   using namespace std;
```

```cpp

const int N = 110;

int n, m;
int f[N];

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= m; i++)
    {
        int l, r; cin >> l >> r;
        f[l]++; f[r + 1]--;
    }

    for(int i = 1; i <= n; i++)
    {
        f[i] = f[i - 1] + f[i];

        if(f[i] == 0 || f[i] > 1)
        {
            cout << i << " " << f[i] << endl;
            return 0;
        }
    }

    cout << "OK" << endl;

    return 0;
}
```


3. 螺旋矩阵

> **题目来源：** 洛谷
>
> **题目链接：** P2239 [NOIP 2014 普及组] 螺旋矩阵

【题目描述】

> 一个 n 行 n 列的螺旋矩阵可由如下方法生成：
>
> 从矩阵的左上角（第 1 行第 1 列）出发，初始时向右移动；如果前方是未曾经过的格子，则继续前进，否则右转；重复上述操作直至经过矩阵中所有格子。根据经过顺序，在格子中依次填入 $1, 2, 3, ..., n^2$ ，便构成了一个螺旋矩阵。
>
> 下图是一个 $n = 4$ 时的螺旋矩阵。

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 12 & 13 & 14 & 5 \\ 11 & 16 & 15 & 7 \\ 10 & 9 & 8 & 7 \end{pmatrix}$$

现给出矩阵大小 n 以及 i 和 j ，请你求出该矩阵中第 i 行第 j 列的数是多少。

【输入描述】

共一行，包含三个整数 n, i, j 每两个整数之间用一个空格隔开，分别表示矩阵大小、待求的数所在的行号和列号。

【数据说明】

对于 50% 的数据， $1 \leq n \leq 100$;

对于 100% 的数据， $1 \leq n \leq 30000, 1 \leq i \leq n, i \leq i \leq n$ 。

【输出描述】

一个整数，表示相应矩阵中第 *i* 行第 *j* 列的数。

【示例一】

输入：

```
4 2 3
```

输出：

```
14
```

【解法】

解法：规律 + 递归。

找相同子问题（递归）：

- 主问题：边长为 n 的矩形，从 0 开始累加，找出 (i, j) 位置的值；
- 当 (i, j) 的位置不在边缘的时候，接下来去：边长为 n - 2 的矩形，从 4 * (n - 1) 开始累加，找出 (i - 1, j - 1) 位置的值。

前后是相同的问题，因此可以用递归帮忙解决。

边界情况，当 (i, j) 在矩阵的边缘时，设 begin 为到达这一圈时，一共经过了多少个点。于是有：

1. 当 i = 1 时：要找的数就是 begin + j；
2. 当 j = n 时：要找的数就是 n + i - 1 + begin；
3. 当 i = n 时：要找的数就是 3 × n - j - 1 + begin；

4. 当 j = 1 时：要找的数就是 4 × n - i - 2 + begin。

【参考代码】

```cpp
#include <iostream>

using namespace std;

int dfs(int n, int begin, int i, int j)
{
    if(i == 1) return begin + j;
    else if(j == n) return begin + n + i - 1;
    else if(i == n) return begin + 3 * n - 1 - j;
    else if(j == 1) return begin + 4 * n - 2 - i;

    return dfs(n - 2, begin + 4 * (n - 1), i - 1, j - 1);
}

int main()
{
    int n, i, j; cin >> n >> i >> j;

    cout << dfs(n, 0, i, j) << endl;

    return 0;
}
```

4. 最长路

> **题目来源：** 洛谷
>
> **题目链接：** P1807 最长路

【题目描述】

> 设 G 为有 n 个顶点的带权有向无环图，G 中各顶点的编号为 1 到 n，请设计算法，计算图 G 中 1,n 间的最长路径。

【输入描述】

> 输入的第一行有两个整数，分别代表图的点数 n 和边数 m。
>
> 第 2 到第 (m+1) 行，每行 3 个整数 u,v,w（u<v），代表存在一条从 u 到 v 边权为 w 的边。

【输出描述】

> 输出一行一个整数，代表 1 到 n 的最长路。
>
> 若 1 无法到达 n，请输出 -1。

【示例一】

> 输入：
>
> ```
> 2 1
> 1 2 1
> ```
>
> 输出：
>
> ```
> 1
> ```

【解法】

解法一：转换为最短路。

问题转化：当所有边的权值改成相反数。

那么最长路问题就变成了最短路问题，找出最短路之后，输出相反数即可。

注意：这种方法是万能的，但是如果数据范围过大时，时间复杂度接受不了。

解法二：拓扑排序 + 动态规划。

在本题中有个重要的条件，对于任意一条边 u->v，u 都是小于 v 的。因此，1 绝对是入度为 0 的点，n 绝对是出度为 0 的点。并且图中没有环形结构，是一个拓扑图。在拓扑图中，可以看做 1 为起点，n 为终点。那么就可以利用拓扑排序的过程，更新出最长路。

细节问题：有可能存在别的入度为 0 的点，需要提前处理掉。

动态规划的逻辑：

1. 状态表示：`f[i]` 表示从 1 走到 i 位置的最长路。
2. 状态转移方程：`f[i] = max(f[prev] + w)`，其中 prev 为 i 结点的前驱结点，w 为 i->prev 的权值。

【参考代码】

```cpp
#include <iostream>
#include <queue>
#include <vector>
```

```cpp
#include <cstring>

using namespace std;

typedef pair<int, int> PII;

const int N = 1510;

int n, m;
vector<PII> edges[N];
int in[N];
int f[N];

int main()
{
    cin >> n >> m;

    for(int i = 1; i <= m; i++)
    {
        int u, v, w; cin >> u >> v >> w;
        edges[u].push_back({v, w});
        in[v]++;
    }

    queue<int> q;
    // 先处理入度为 0，但是不是 1 号结点的结点
    for(int i = 2; i <= n; i++)
    {
        if(in[i] == 0) q.push(i);
        f[i] = -1e9; // -0x3f3f3f3f
    }

    // memset(f, -0x3f, sizeof f);
    // f[1] = 0;

    while(q.size())
    {
        auto u = q.front(); q.pop();
        for(auto& t : edges[u])
        {
            int v = t.first;
            in[v]--;
            if(in[v] == 0) q.push(v);
        }
    }

    q.push(1);
```

```cpp
        while(q.size())
        {
            auto u = q.front(); q.pop();
            for(auto& t : edges[u])
            {
                int v = t.first, w = t.second;
                f[v] = max(f[v], f[u] + w);

                in[v]--;
                if(in[v] == 0) q.push(v);
            }
        }

        if(f[n] == -0x3f3f3f3f) cout << -1 << endl;
        else cout << f[n] << endl;

        return 0;
    }
```


Day10

5. 金盏花

> **题目来源：** 洛谷
>
> **题目链接：** P9390 金盏花

【题目描述】

> 有一个十二位十进制数 X，你只知道它的后六位构成的数是 Y。
>
> 另外再给出一个整数 Z，你需要求出所有可能的 X 中， X 与 Z 的差，即 $|X-Z|$ 的最小值。
>
> 注意， X,Y,Z 都没有前导零（即最高位不是 0）， X,Y 分别要有恰好十二位和六位。

【输入描述】

> 第一行：两个整数 Y,Z。
>
> 对于全部数据： $100,000 \le Y \le 999,999$，$0 \le Z \le 10^{12}$。

【输出描述】

> 第一行：一个整数表示 $|X-Z|$ 的最小值。

【示例一】

> 输入：

987654 123456123456

输出：

135802

说明：

令 $X = 123,455,987,654$，可以取到 $|X - Z|$ 的最小值 $135,802$。

【示例二】

输入：

428571 714285

输出：

99999714286

说明：

令 $X = 100,000,428,571$，可以取到 $|X - Z|$ 的最小值 $99,999,714,286$。

【解法】

解法：暴力枚举。

- 直接枚举 x 的所有可能的情况，计算最小值。

【参考代码】

代码块

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

int main()
{
    LL y, z; cin >> y >> z;

    LL ret = 1e12 + 10;
    for(LL k = 100000; k < 1000000; k++)
    {
        LL x = k * 1000000 + y;
        ret = min(ret, abs(x - z));
    }

```

```
18        cout << ret << endl;
19
20        return 0;
21    }
22
```

6. "非常男女"计划

> **题目来源：** 洛谷
>
> **题目链接：** P1114 “非常男女”计划

【题目描述】

> 近来，初一年的 XXX 小朋友致力于研究班上同学的配对问题（别想太多，仅是舞伴），通过各种推理和实验，他掌握了大量的实战经验。例如，据他观察，身高相近的人似乎比较合得来。
>
> 万圣节来临之际，XXX 准备在学校策划一次大型的 “非常男女” 配对活动。对于这次活动的参与者，XXX 有自己独特的选择方式。他希望能选择男女人数相等且身高都很接近的一些人。这种选择方式实现起来很简单。他让学校的所有人按照身高排成一排，然后从中选出连续的若干个人，使得这些人中男女人数相等。为了使活动更热闹，XXX 当然希望他能选出的人越多越好。请编写程序告诉他，他最多可以选出多少人来。

【输入描述】

> 第一行有一个正整数 $n(1 \leq n \leq 10^5)$，代表学校的人数。
>
> 第二行有 n 个用空格隔开的数，这些数只能是 0 或 1，其中，0 代表是一个女生，1 代表是一个男生。

【输出描述】

> 输出一个非负整数。这个数表示在输入数据中最长的一段男女人数相等的子区间的长度。
>
> 如果不存在男女人数相等的子区间，请输出 0。

【示例一】

> 输入：
>
> ```
> 9
> 010001100
> ```
>
> 输出：
>
> ```
> 6
> ```

【解法】

解法：正负抵消 + 前缀和 + 哈希表。

将女生看成 -1，男生看成 1。题目就变成：找出数组中最长的一个区间，使得这一段区间的和为 0。

针对某一个区间 $[1,r]$，记和为 $sum[r]$，只需找到这个区间内第一次出现区间和为 $sum[r]$ 的位置 l，$r-l$ 就是以 r 为结尾的最长的长度。如果不存在 l，那就不统计。从前往后遍历所有的位置，找出所有 $r-l$ 的最大值即可。

其中，为了快速找出第一次出现区间和 sum，可以用哈希表标记。

【参考代码】

代码块

```cpp
#include <iostream>
#include <unordered_map>

using namespace std;

int n;
unordered_map<int, int> mp;

int main()
{
    cin >> n;
    int sum = 0, ret = 0;
    mp[0] = 0;

    for(int i = 1; i <= n; i++)
    {
        int x; cin >> x;
        x = (x == 0 ? -1 : 1);
        sum += x;

        if(mp.count(sum)) ret = max(ret, i - mp[sum]);
        else mp[sum] = i;
    }

    cout << ret << endl;

    return 0;
}
```

7. 次大值

> **题目来源：** 洛谷
>
> **题目链接：** P5682 [CSP-J2019 江西] 次大值

【题目描述】

> `Alice` 有 n 个正整数，数字从 1~ n 编号，分别为 $a_1, a_2, ..., a_n$ 。
> `Bob` 刚学习取模运算，于是便拿这 n 个数进行练习，他写下了所有
>
> $$a_i \ mod \ a_j (1 \leq i, j \leq n \wedge i \neq j)$$
>
> 的值，其中 mod 表示取模运算。
>
> `Alice` 想知道所有的结果中，严格次大值是多少。将取模后得到的所有值进行去重，即相同的结果数值只保留一个，剩余数中第二大的值就称为严格次大值。

【输入描述】

> 第一行一个正整数 n ，表示数字个数。
> 第二行 n 个正整数表示 a_i 。
>
> 对于 40% 的数据， $1 \leq n, a_i \leq 100$;
> 对于 70% 的数据， $1 \leq n \leq 3000, 1 \leq a_i \leq 10^5$;
> 对于 100% 的数据， $3 \leq n \leq 2 \times 10^5, 1 \leq a_i \leq 10^9$ 。

【输出描述】

> 仅一行一个整数表示答案。
>
> 若取模结果去重后剩余数字不足两个，则输出 -1

【示例一】

> 输入：
>
> 4
>
> 4 5 5 6
>
> 输出：
>
> 4
>
> 说明：
>
> 所有取模的结果为 $\{ 4, 4, 4, 1, 0, 5, 1, 0, 5, 2, 1, 1 \}$。
>
> 去重后有：$\{ 0, 1, 2, 4, 5 \}$，结果为 4。

【示例二】

> 输入：

> 4
>
> 1 1 1 1
>
> 输出：
>
> -1

【示例三】

> 输入：
>
> 7
>
> 12 3 8 5 7 20 15
>
> 输出：
>
> 12

【解法】

解法：数学。

题目要的是严格次大值，因此可以对所有的数据排序 + 去重。设排序后的数组为 a，那么：

- 最大值一定是 `a[n - 1] % a[n]`

那么次大值可以分为两种情况：

1. 小数 模 大数：那就只能是 `a[n - 2] % a[n - 1]`；
2. 大数 模 小数：只能是 `a[n] % a[n - 1]`。

在这两种情况中取最大值即可。

【参考代码】

代码块

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

using namespace std;

const int N = 2e5 + 10;

int n;
int a[N];

int main()
```

```
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i];

    sort(a + 1, a + 1 + n);
    n = unique(a + 1, a + 1 + n) - (a + 1);

    if(n == 1) cout << -1 << endl;
    else if(n == 2) cout << a[2] % a[1] << endl;
    else cout << max(a[n - 2], a[n] % a[n - 1]) << endl;

    return 0;
}
```

8. 瑞士轮

题目来源： 洛谷

题目链接： P1309 [NOIP 2011 普及组] 瑞士轮

【题目描述】

$2 \times N$ 名编号为 $1 \sim 2N$ 的选手共进行R 轮比赛。每轮比赛开始前，以及所有比赛结束后，都会按照总分从高到低对选手进行一次排名。选手的总分为第一轮开始前的初始分数加上已参加过的所有比赛的得分和。总分相同的，约定编号较小的选手排名靠前。

每轮比赛的对阵安排与该轮比赛开始前的排名有关：第1 名和第2 名、第 3 名和第 4名、……、第 $2K-1$ 名和第 $2K$ 名、……、第 $2N-1$ 名和第 $2N$ 名，各进行一场比赛。每场比赛胜者得1分，负者得 0分。也就是说除了首轮以外，其它轮比赛的安排均不能事先确定，而是要取决于选手在之前比赛中的表现。

现给定每个选手的初始分数及其实力值，试计算在 R 轮比赛过后，排名第 Q 的选手编号是多少。我们假设选手的实力值两两不同，且每场比赛中实力值较高的总能获胜。

【输入描述】

第一行是三个正整数 N, R, Q 每两个数之间用一个空格隔开，表示有 $2 \times N$ 名选手、 R 轮比赛，以及我们关心的名次 Q。

第二行是 $2 \times N$ 个非负整数 $s_1, s_2, ..., s_{2N}$ ，每两个数之间用一个空格隔开，其中 s_i 表示编号为 i 的选手的初始分数。 第三行是 $2 \times N$ 个正整数 $w_1, w_2, w...w_{2N}$ ，每两个数之间用一个空格隔开，其中 w_i 表示编号为 i 的选手的实力值。

对于 30% 的数据， $1 \leq N \leq 100$;

对于 50% 的数据， $1 \leq N \leq 10,000$;

> 对于 100% 的数据，$1 \le N \le 100,000$，$1 \le R \le 50$，$1 \le Q \le 2N$，
> $0 \le s_1, s_2, ..., s2_2N \le 10^8$，$1 \le w_1, w_2, ..., w_2 2N \le 10^8$

【输出描述】

> 一个整数，即 R 轮比赛结束后，排名第 Q 的选手的编号。

【示例一】

> 输入：
>
> 2 4 2
>
> 7 6 6 7
>
> 10 5 20 15
>
> 输出：
>
> 1

【解法】

解法：归并排序。

前后两两比较的过程中，如果把获胜的人放在一组，失败的人放在一组，那么这两组人的比分也是从大到小排序。因此，可以在模拟的过程中，将每次比试的结果放在两个数组中，然后合并两个有序数组即可。

整个过程就类似于归并排序。

【参考代码】

代码块

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

using namespace std;

const int N = 2e5 + 10;

int n, r, q;
struct node
{
    int s, id, w;
}a[N];

node x[N], y[N]; // 胜利组 + 失败组

```

```cpp
bool cmp(node& x, node& y)
{
    if(x.s == y.s) return x.id < y.id;
    return x.s > y.s;
}

void merge()
{
    // 合并两个有序数组
    int cur1 = 1, cur2 = 1, i = 1;
    while(cur1 <= n && cur2 <= n)
    {
        if(x[cur1].s > y[cur2].s || (x[cur1].s == y[cur2].s && x[cur1].id < y[cur2].id))
        {
            a[i++] = x[cur1++];
        }
        else
        {
            a[i++] = y[cur2++];
        }
    }

    while(cur1 <= n) a[i++] = x[cur1++];
    while(cur2 <= n) a[i++] = y[cur2++];
}

int main()
{
    cin >> n >> r >> q;
    for(int i = 1; i <= n + n; i++)
    {
        cin >> a[i].s;
        a[i].id = i;
    }
    for(int i = 1; i <= n + n; i++)
    {
        cin >> a[i].w;
    }

    sort(a + 1, a + 1 + n + n, cmp);

    while(r--)
    {
        int pos = 1;
        for(int i = 1; i <= n + n; i += 2)
        {
```

```
62              if(a[i].w > a[i + 1].w)
63              {
64                  a[i].s++;
65                  x[pos] = a[i];
66                  y[pos] = a[i + 1];
67              }
68              else
69              {
70                  a[i + 1].s++;
71                  x[pos] = a[i + 1];
72                  y[pos] = a[i];
73              }
74              pos++;
75          }
76          merge();
77      }
78
79      cout << a[q].id << endl;
80
81      return 0;
82  }
```