现在有一张半径为 r 的圆桌，其中心位于(x,y)，现在他想把圆桌的中心移到(x1,y1)。每次移动一步，都必须在圆桌边缘固定一个点然后将圆桌绕这个点旋转。问最少需要移动几步。

```
public static void main(String args[]){
                Scanner reader=new Scanner(System.in);
                while(reader.hasNextInt()){
                        int r=reader.nextInt();
                        int x=reader.nextInt();
                        int y=reader.nextInt();
                        int x1=reader.nextInt();
                        int y1=reader.nextInt();
                        int lx = Math.abs(x-x1);
                        int ly = Math.abs(y-y1);
                        int len =
lx==0?ly:ly==0?lx:(int)Math.sqrt(lx*lx+ly*ly);
                        int l = len/(2*r);
                        int t = (int)l;
                        if (l - t >  0)
                                System.out.println(t+1);
                        else
                                System.out.println(t);
                }
    }
```

给定一个递增序列，$a_1 < a_2 < ... < a_n$ 。定义这个序列的最大间隔为 $d=max\{a_{i+1} - a_i\}(1 \le i < n)$,现在要从 $a_2, a_3 .. a_{n-1}$ 中删除一个元素。问剩余序列的最大间隔最小是多少？

解题思路：

1.先计算原始数组相邻间隔，并在计算的过程中记录最大相邻间隔 maxFull。

2.删除 ai(1≤i<n)后所得新数组的最大相邻间隔只会在 a[i+1]-a[i-1]与 maxFull 中取值，也就是 Math.max(arr[i+1]-arr[i-1], maxFull)。

3.记录每一次删除 ai(1≤i<n)后所得最大相邻间隔的最小值。

```
import java.util.*;
public class Main {
        public static void main(String[] args) {
                Scanner in =  new Scanner(System.in);
                while(in.hasNext()){
                        int n = in.nextInt(),i;
                        int arr[] =  new int[n];
                        for (i =  0; i < n; i++) {
                                arr[i] = in.nextInt();
                        }
                        int maxFull = Integer.MIN_VALUE,minMaxGap =
```

```
12  Integer.MAX_VALUE;
13                          for (i =  1; i < n; i++) {
14                                  maxFull = Math.max(maxFull,
15  arr[i]-arr[i-1]);
16                          }
17                          for (i =  1; i < n-1; i++) {
18                                  minMaxGap = Math.min(minMaxGap,
19  Math.max(arr[i+1]-arr[i-1], maxFull));
20                          }
21                          System.out.println(minMaxGap);
22                  }
                    in.close();
            }
    }
```

A 和 B 是好友，他们经常在空闲时间聊天，A 的空闲时间为$[a_1,b_1],[a_2,b_2]..[a_p,b_p]$。B 的空闲时间是$[c_1+t,d_1+t]..[c_q+t,d_q+t]$,这里 t 为 B 的起床时间。这些时间包括了边界点。B 的起床时间为[l,r]的一个时刻。若一个起床时间能使两人在任意时刻聊天，那么这个时间就是合适的，问有多少个合适的起床时间？

这道题目可以查看区间是否有重合，对于 A 的某个区间[a,b]，B 的某个区间[c+t,d+t]，如果(c+t<=b) && (d+t>=a)成立，就说明区间之间有重合，满足要求

```
1   #include <iostream>
2   #include <vector>
3   #include <utility>
4   using namespace std;
5
6   int p,q,l,r;
7   vector<pair<int,int>> A;
8   vector<pair<int,int>> B;
9   int ans;
10
11  bool isOK(int t){
12          for(int i=0; i<A.size();i++){
13                  for(int j=0; j<B.size();j++){
14                          if(B[j].second+t>=A[i].first &&
15  B[j].first+t<=A[i].second){
16                                  return true;
17                          }
18                  }
19          }
20          return false;
```

```
21  }
22
23  void solve(){
24          for(int t=l; t<=r; t++){
25                  if(isOK(t)){
26                          ans++;
27                  }
28          }
29  }
30
31  int main(){
32          while(cin>>p>>q>>l>>r){
33                  ans = 0;
34                  A = vector<pair<int,int>>(p);
35                  B = vector<pair<int,int>>(q);
36                  for(int i=0; i<p; i++){
37                          cin>>A[i].first>>A[i].second;
38                  }
39                  for(int i=0; i<q; i++){
40                          cin>>B[i].first>>B[i].second;
41                  }
42                  solve();
43                  cout<<ans<<endl;
44          }
45          return 0;
    }
```

有一个投篮游戏。球场有 p 个篮筐，编号为 0，1...，p-1。每个篮筐下有个袋子，每个袋子最多装一个篮球。有 n 个篮球，每个球编号 x_i 。规则是将数字为 x_i 的篮球投到 x_i 除 p 的余数为编号的袋里。若袋里已有篮球则球弹出游戏结束输出 i，否则重复至所有球都投完。输出-1。问游戏最终的输出是什么？

```
1   #include <iostream>
2   #include <string>
3   #include <iomanip>
4   #include <vector>
5   #include <algorithm>
6   #include <string.h>
7   #define REP(i,n) for(int i=0;i<(n);i++)
8   #define FOR(i,a,b) for(int i=(a);i<(b);i++)
9
10  #define MAXN 400
11  using namespace std;
12
```

```
int p,n;
int x[MAXN];
bool empty[MAXN];
int main(){
        ios::sync_with_stdio(0);
        while(cin>>p>>n){
                REP(i,n) cin>>x[i];
                memset(empty,0,sizeof(empty));
                int re=-1;
                REP(i,n)  if(empty[x[i]%p]){re=i+1;  break;}else
empty[x[i]%p]=1;
                cout<<re<<endl;
        }
        return 0;
}
```

给定一个字符串，问是否能通过添加一个字母将其变为回文串。

```
/**

*判断原字符串和翻转字符串的最长公共子序列长度是否比原字符串长度小 1 或相
等

*/

importjava.util.*;

publicclassMain

{

        publicstaticintlcs(String s, String s1)

        {

                if(s ==  null|| s1 ==  null){

                        return0;

                }

                intm = s.length();

                intn = s1.length();

```

```
28                  int[][] dp =  newint[m +  1][n +  1];
29
30                  dp[0][0] =  0;
31
32                  for(inti =  1; i < m; i++) {
33
34                          dp[0][i] =  0;
35
36                  }
37
38                  for(inti =  1; i < m; i++) {
39
40                          dp[i][0] =  0;
41
42                  }
43
44                  for(inti =  1;i < m +  1; i++) {
45
46                          for(intj =  1; j < n +  1; j++) {
47
48                                  if(s.charAt(i -  1) == s1.charAt(j -  1))
49  {
50
51                                          dp[i][j] = dp[i -  1][j -  1]
52  +  1;
53
54                                  }  else{
55
56                                          dp[i][j] = Math.max(dp[i
57  -  1][j], dp[i][j -  1]);
58
59                                  }
60
61                          }
62
63                  }
64
65                  returndp[m][n];
66
67          }
68
69
70
71          publicstaticvoidmain(String[] args) {
```

```
72
73                  Scanner scanner =  newScanner(System.in);
74
75                  while(scanner.hasNext()) {
76
77                          String s= scanner.nextLine();
78
79                          String s1
80  =  newStringBuilder(s).reverse().toString();
81
82                          intlen = lcs(s, s1);
83
84                          if(s.length() - len <=  1) {
85
86                                  System.out.println("YES");
87
88                          }  else{
89
90                                  System.out.println("NO");
91
92                          }
93
                    }

            }

    }
```