

有  $n$  个学生站成一排，每个学生有一个能力值，牛牛想从这  $n$  个学生中按照顺序选取  $k$  名学生，要求相邻两个学生的位置编号的差不超过  $d$ ，使得这  $k$  个学生的能力值的乘积最大，你能返回最大的乘积吗？

```
1  import java.util.*;
2  public class Main{
3      public static void main(String[] args){
4          Scanner scan = new Scanner(System.in);
5          int n = scan.nextInt();
6          int[] nums = new int[n];
7          for(int i = 0; i < n; i++){
8              nums[i] = scan.nextInt();
9          }
10         int k = scan.nextInt();
11         int d = scan.nextInt();
12         long[][] max = new long[k][n];
13         long[][] min = new long[k][n];
14         for(int i = 0; i < k; i++){
15             for(int j = 0; j < n; j++){
16                 //min[i][j] = Integer.MAX_VALUE;
17                 max[i][j] = 1;
18                 if(i == 0){
19                     min[i][j] = nums[j];
20                     max[i][j] = nums[j];
21                 }
22             }
23         }
```

```
24         for(int i = 1; i < k; i++)
25             for(int j = 0; j < n; j++)
26                 for(int m = 1; m <= d; m++){
27                     if(j - m >= 0){
28                         if(nums[j] > 0){
29                             min[i][j] = Math.min(min[i][j], min[i - 1][j - m] * nums[j]);
30                             max[i][j] = Math.max(max[i][j], max[i - 1][j - m] * nums[j]);
31                         } else{
32                             min[i][j] = Math.min(min[i][j], max[i - 1][j - m] *
33 nums[j]);
34                             max[i][j] = Math.max(max[i][j], min[i - 1][j - m] *
35 nums[j]);
36                         }
37                     }
38                 }
39         long Max = 0;
40         for(int i = 0; i < n; i++)
41             Max = Math.max(Max, max[k - 1][i]);
42         System.out.println(Max);
43     }
}
```

给定一个  $n$  行  $m$  列的地牢，其中 '.' 表示可以通行的位置，'X' 表示不可通行的障碍，牛牛从  $(x_0, y_0)$  位置出发，遍历这个地牢，和一般的游戏所不同的是，他每一步只能按照一些指定的步长遍历地牢，要求每一步都不可以超过地牢的边界，也不能到达障碍上。地牢的出口可能在任意某个可以通行的位置上。牛牛想知道最坏情况下，他需要多少步才可以离开这个地牢。

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args){
5         Scanner in = new Scanner(System.in);
6
7         while (in.hasNext()) {//注意 while 处理多个 case
8             int x=in.nextInt();
9             int y=in.nextInt();
10
11             char[][] points=new char[x][y];
12             int[][] tar=new int[x][y];
13             for(int i=0;i<x;i++){
14                 String str=in.next();
15                 points[i]=str.toCharArray();
16             }
17             int startx=in.nextInt();
18             int starty=in.nextInt();
19             int k=in.nextInt();
20             int[] stepx=new int[k];
21             int[] stepy=new int[k];
22             for(int i=0;i<k;i++){
23                 stepx[i]=in.nextInt();
24                 stepy[i]=in.nextInt();
25             }
26             Queue<Integer> xqueue=new LinkedList<Integer>();
```

```
27 Queue<Integer> yqueue=new LinkedList<Integer>();
28 //引入队列是为了遍历到最后不能走为止
29
30 xqueue.add(startx);
31 yqueue.add(starty);
32
33 tar[startx][starty]=1;    //起始点访问标记；1 表示已经访问
34 while(!xqueue.isEmpty()&&!yqueue.isEmpty()){
35     startx=xqueue.remove();    //取队首
36     starty=yqueue.remove();
37     for(int i=0;i<k;i++){
38         if(startx+stepx[i]<x&&startx+stepx[i]>=0&&starty+stepy[i]<y&&starty+stepy[i]>=0)
39 不出界
40         if(tar[startx+stepx[i]][starty+stepy[i]]==0){
41             if(points[startx+stepx[i]][starty+stepy[i]]=='.') {
42                 tar[startx+stepx[i]][starty+stepy[i]]=tar[startx][starty]+1;
43                 xqueue.add(startx+stepx[i]);
44                 yqueue.add(starty+stepy[i]);
45             }
46             else
47                 tar[startx+stepx[i]][starty+stepy[i]]=-1;    //访问点为 X
48         }
49     }
50 }
51 int max=0;
52 int getRoad=1;
```

```
53         for(int i=0;i<x;i++)
54             for(int j=0;j<y;j++){
55                 if(points[i][j]=='.' && tar[i][j]==0){
56                     getRoad=0;           //有存在没有被访问的“.”说明不能遍历完全，有些出口到不了。
57                 }
58                 max=Math.max(max, tar[i][j]);
59             }
60         if(getRoad==0)
61             System.out.println(-1);
62         else
63             System.out.println(max-1);
64     }
65 }
66 }
```

蔚蓝想尝试一些新的料理，每个料理需要一些不同的材料，问完成所有的料理需要准备多少种不同的材料。

```
1  import java.util.HashSet;
2  import java.util.Scanner;
3
4  public class Main4 {
5
6      public static void main(String[] args){
7
8          Scanner cin = new Scanner(System.in);
9          HashSet<String> martic = new HashSet<String>();
10         while(cin.hasNext()){
```



```
11             martic.add(cin.next());
12         }
13         int Number = martic.size();
14         System.out.println(Number);
15     }
16 }
```

蔚蓝和 15 个朋友来玩打土豪分田地的游戏，蔚蓝决定让你来分田地，地主的田地可以看成是一个矩形，每个位置有一个价值。分割田地的方法是横竖各切三刀，分成 16 份，作为领导干部，蔚蓝总是会选择其中总价值最小的一份田地，作为蔚蓝最好的朋友，你希望蔚蓝取得的田地的价值和尽可能大，你知道这个值最大可以是多少吗？

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  int subValue(vector<vector<int>>& vecValue, size_t x1, size_t y1, size_t x2, size_t y2) {
6      return vecValue[x2][y2] - vecValue[x2][y1] - vecValue[x1][y2] + vecValue[x1][y1];
7  }
8
9  bool isMinimal(vector<vector<int>>& vecValue, int valToComp) {
10     size_t row = vecValue.size();
11     size_t col = vecValue[0].size();
12     for (size_t c1 = 1; c1 < col - 3; ++c1) {
13         for (size_t c2 = c1 + 1; c2 < col - 2; ++c2) {
14             for (size_t c3 = c2 + 1; c3 < col - 1; ++c3) {
```





```
15         int cutTimes = 0;
16         size_t lastRow = 0;
17         for (size_t r = 1; r < row; ++r) {
18             int s1 = subValue(vecValue, lastRow, 0, r, c1);
19             int s2 = subValue(vecValue, lastRow, c1, r, c2);
20             int s3 = subValue(vecValue, lastRow, c2, r, c3);
21             int s4 = subValue(vecValue, lastRow, c3, r, col - 1);
22             if (valToComp <= min(min(s1, s2), min(s3, s4))) {
23                 ++cutTimes;
24                 lastRow = r;
25             }
26         }
27         if (cutTimes >= 4) {
28             return true;
29         }
30     }
31 }
32
33 return false;
34 }
35
36 int main() {
37     size_t row, col;
38     cin >> row >> col;
39     vector<vector<int>> vecValue(row + 1, vector<int>(col + 1, 0));
40     for (size_t i = 1; i <= row; ++i) {
```

```
41         string str;
42         cin >> str;
43         for (size_t j = 1; j <= col; ++j) {
44             vecValue[i][j] = str[j - 1] - '0';
45             vecValue[i][j] += vecValue[i - 1][j] + vecValue[i][j - 1] - vecValue[i - 1][j
46 - 1];
47         }
48     }
49
50     int left = 0, right = vecValue[row][col];
51     int ret = 0;
52     while (left <= right) {
53         int valToComp = left + right >> 1;
54         if (isMinimal(vecValue, valToComp)) {
55             ret = valToComp;
56             left = valToComp + 1;
57         }
58         else {
59             right = valToComp - 1;
60         }
61     }
62     cout << ret << endl;
63     return 0;
}
```

$n$  只奶牛坐在一排，每个奶牛拥有  $a_i$  个苹果，现在你要在它们之间转移苹果，使得最后所有奶牛拥有的苹果数都相同，每一次，你只能从一只奶牛身上拿走恰好两个苹果到另一个奶牛上，问最少需要移动多少次可以平分苹果，如果方案不存在输出 -1。



```
1  #include<iostream>
2  #include<vector>
3  using namespace std;
4  int main() {
5      int n;
6      while (cin >> n){
7          vector<int>num(n);
8          int sum = 0;
9          for (vector<int>::iterator iter = num.begin(); iter != num.end(); iter++){
10             cin>>*iter;
11             sum = sum + *iter;
12         }
13         if (sum%n != 0){
14             cout << '-1' << endl;
15             return 0;
16         }
17         sum = sum / n;
18         int count = 0;
19         for (vector<int>::iterator iter = num.begin(); iter != num.end(); iter++){
20             int temp = *iter - sum;
21             if (temp % 2 != 0){
22                 cout << '-1' << endl;
23                 return 0;
24             }
25             if (temp > 0){
26                 count=count+temp/2;
```

```
27         }  
28     }  
29     cout << count << endl;  
30  
31  
32 }  
33 return 0;  
34 }
```

航天飞行器是一项复杂而又精密的仪器，飞行器的损耗主要集中在发射和降落的过程，科学家根据实验数据估计，如果在发射过程中，产生了  $x$  程度的损耗，那么在降落的过程中就会产生  $x^2$  程度的损耗，如果飞船的总损耗超过了它的耐久度，飞行器就会爆炸坠毁。问一艘耐久度为  $h$  的飞行器，假设在飞行过程中不产生损耗，那么为了保证其可以安全的到达目的地，只考虑整数解，至多发射过程中可以承受多少程度的损耗？

//注意是无符号长整形。因为最后有 **res-1**，如果为有符号，就回出错。

```
#include<iostream>
```

```
using namespace std;
```

```
intmain()  
{  
    unsigned longlong h;  
    cin>>h;
```

```
unsigned longlong res = 0;
unsigned longlong x=0;
unsigned longlongtemp = 0;
for(x = 0;temp<=h;x++)
{
    temp = x + x*x;
    res = x;
}
cout<<res-1<<endl;
return 0;
}
```

牛牛拿到了一个藏宝图，顺着藏宝图的指示，牛牛发现了一个藏宝盒，藏宝盒上有一个机关，机关每次会显示两个字符串 **s** 和 **t**，根据古老的传说，牛牛需要每次都回答 **t** 是否是 **s** 的子序列。注意，子序列不要求在原字符串中是连续的，例如串 **abc**，它的子序列就有 {空串, **a**, **b**, **c**, **ab**, **ac**, **bc**, **abc**} 8 种。

```
1  import java.util.*;
2  public class Main {
3
4      public static final void main(String[] args){
5          Scanner scan=new Scanner(System.in);
6          while(scan.hasNext()){
```



```
7         String str1=scan.nextLine();
8         String str2=scan.nextLine();
9         boolean result=isContain(str1,str2);
10        if(result){
11            System.out.println("Yes");
12        }else{
13            System.out.println("No");
14        }
15    }
16    scan.close();
17 }
18
19 public static boolean isContain(String str1,String str2){
20     for(int i=0,index=0;i<str1.length();i++){
21         if(str1.charAt(i)==str2.charAt(index)){
22             index++;
23             if(index==str2.length()){
24                 return true;
25             }
26         }
27     }
28     return false;
29 }
30
31
32 }
```

牛牛的作业簿上有一个长度为  $n$  的排列  $A$ ，这个排列包含了从 1 到  $n$  的  $n$  个数，但是因为一些原因，其中有一些位置（不超过 10 个）看不清了，但是牛牛记得这个数列顺序对的数量是  $k$ ，顺序对是指满足  $i < j$  且  $A[i] < A[j]$  的对数，请帮助牛牛计算出，符合这个要求的合法排列的数目。

```
1 //方法是 hofighter 提供的，个人只是加了一些注释，方便自己和大家理解
2 /**思路：首先将模糊的数字统计出来，并求出这些数字的全排列，然后对每个排列求顺序对
3 关键去求全排列，需要递归的求出所有排列。。。
4
5 ps：第一次做的时候没看清楚题，以为可以有重复数字，直接深搜计算了，结果。。。*/
6
7 import java.util.ArrayList;
8 import java.util.Arrays;
9 import java.util.Collections;
10 import java.util.List;
11 import java.util.Scanner;
12
13 public class Main{
14
15     /**
16      * @param args
17      */
18     public static void main(String[] args) {
19         // TODO Auto-generated method stub
20         Scanner sc = new Scanner(System.in);
21         while(sc.hasNext()){
22             int RES = 0;
```

```
23         int n = sc.nextInt();
24         int k = sc.nextInt();
25         int[] A = new int[n];
26         boolean[] flag = new boolean[n+1]; //为什么是 n+1?因为 n 是从 1 开始的, 必须有
27         flag[n]
28
29         //flag 标记哪些数字已经存在
30         for(int i=0;i<n;i++){
31             A[i] = sc.nextInt();
32             if(A[i] != 0){
33                 flag[A[i]] = true;
34             }
35
36         //统计排列中不存在的数字
37         ArrayList<Integer> list = new ArrayList<Integer>();
38         for(int i=1;i<=n;i++){
39             if(flag[i] == false)
40                 list.add(i);
41         }
42
43         //perm 用来存模糊数字的全排列
44         List<ArrayList<Integer>> perm = new ArrayList<ArrayList<Integer>>();
45
46         //计算 perm
47         calperm(perm, list, 0);
48
```

```
49         //统计已有的排列的顺序对
50         int cv = 0;
51         for(int i=0;i<n;i++){
52             if(A[i]!= 0){
53                 for(int j=i+1;j<n;j++){
54                     if(A[j] != 0 && A[i] < A[j])
55                         cv++;
56                 }
57             }
58         }
59
60         //计算每个模糊数字排列的顺序对，如果与 k 相等，则结果 RES 加一
61         for(ArrayList<Integer> tmp : perm){
62             int val = cv;
63             int[] tmpA = Arrays.copyOf(A, n);
64             val += calvalue(tmp, tmpA);
65             if(val == k)
66                 RES++;
67         }
68
69         System.out.println(RES);
70     }
71 }
72
73 /**
74  * 计算排列的顺序对
```

```
75      * @param list 模糊数列的某个排列
76      * @param A 最终的某个排列
77      */
78      public static int calvalue(List<Integer> list, int[] A){
79          int val = 0;
80          int j = 0;
81          for(int i=0;i<A.length;i++){
82              if(A[i] == 0){
83                  A[i] = list.get(j++); //每一个为0的位置 安插一个list中的数字
84                  for(int k = 0;k<i;k++){
85                      if(A[k]!=0 && A[k]<A[i])
86                          val++;
87                  }
88                  for(int k=i+1;k<A.length;k++){
89                      if(A[k]!=0 && A[k]>A[i])
90                          val++;
91                  }
92              }
93          }
94          return val; //最后返回时因为必须报list中的所有数据安插在为0的位置上
95      }
96
97      /**
98      * 计算全排列
99      * @param perm
100     * @param list 排列中不存在的数字
```



```
101         * @param n 初始为 0
102         */
103     public static void calperm(List<ArrayList<Integer>> perm, ArrayList<Integer> list, int n) {
104         if(n == list.size()) {
105             perm.add(new ArrayList<Integer>(list));
106         } else {
107             for(int i=n; i<list.size(); i++) {
108                 Collections.swap(list, i, n);
109                 calperm(perm, list, n+1);
110                 Collections.swap(list, i, n);
111             }
112         }
113     }
}
```