**Problem Description**

Given a sequence of distinct numbers a1, a2, ….. an, an inversion occurs if there are indices i<j such that ai > aj .  
For example, in the sequence 2 1 4 3 there are 2 inversions (2 1) and (4 3).  
The input will be a main sequence of N positive integers. From this sequence, a **Derived Sequence**will be obtained using the following rule. The output is the number of inversions in the derived sequence.  
**Rule for forming derived sequence**  
An integer may be represented in base 6 notation. In base 6, 10305 is 1x64 + 3x62 + 5 =1409. Note that none of the digits in that representation will be more than 5.  
The sum of digits in a base 6 representation is the sum of all the digits at the various positions in the representation. Thus for the number 1409, the representation is 10305, and the sum of digits is 1+0+3+0+5=9. The sum of digits may be done in the decimal system, and does not need to be in base 6  
The derived sequence is the sum of the digits when the corresponding integer is represented in the base 6 form number will be expressed in base 6, and the derived sequence is the sum of the digits of the number in the base 6 representation.

**Constraints**

N <= 50   
Integers in sequence <= 107

**Input Format**

The first line of the input will have a single integer, which will give N.   
The next line will consist of a comma separated string of N integers, which is the main sequence

**Output**

The number of inversions in the derived sequence formed from the main sequence.

**Explanation**

**Example 1**   
Input   
5  
55, 53, 88, 27, 33  
Output   
2  
**Explanation**   
The number of integers is 5, as specified in the first line. The given sequence is 55, 53, 88, 27, 33.   
The base 6 representation is 131, 125, 224, 43, 53 The derived sequence is 5,8,8,7,8 (corresponding to the sum of digits). The number of inversions in this is 2, namely (8, 7), (8, 7)  
**Example 2**   
Input

|  |
| --- |
| 8 |
| 120,21,47,64,72,35,18,98 |

Output  
11  
  
**Explanation**   
The base 6 representation of this is 320,33,115,144,200,55,30,242, and the derived sequence (sum of digits) is 5,6,7,9,2,10,3,8. The number of inversions is 11 (5,2), (5,3),(6,2) (6,3), (7,2), (7,3) (9,2),(9,3) (9,8),(10,3), (10,8)

**Program:**

#include<stdio.h>

#include<math.h>

int sum(int a)

{

    int s=0,r;

    while(a>0)

    {

       r=a%10;

       s+=r;

       a=a/10;

    }

    return s;

}

int main()

{

    int a[100],b[100],c[100],i,n,j,rem,s=0,cn=0;

    char ch;

    scanf("%d",&n);

    for(i=0;i<n;i++)

    {

        scanf("%d",&a[i]);

        if(i<n-1)

        scanf("%c",&ch);

    }

    for(i=0;i<n;i++)

    {

        j=0;

        while(a[i]>0)

        {

            rem=a[i]%6;

            s=s+(rem\*pow(10,j));

            a[i]=a[i]/6;

            j++;

        }

        b[i]=s;

        s=0;

    }

    for(i=0;i<n;i++)

    {

    c[i]=sum(b[i]);

    }

    for(i=0;i<n;i++)

    {

        for(j=i+1;j<n;j++)

        {

            if(c[i] >c[j])

                cn++;

        }

    }

    printf("%d",cn);

    return 0;

}

**Problem Description**

A cube has its 6 faces numbered from 1-6 (three of its faces are shown in the diagram). There are six possible orientations on the cube (U,D,N,E,W,S) as shown in the diagram, of which only 4 are possible on any one face (for example, on face 1, N and S are impossible orientations)

An electronic remote control bug is on a face of a cube at some orientation. On your command it can crawl around the cube. The possible commands are F,B,L,R.

Command F : it moves to the adjacent face in the same direction it is facing

.Command B : it turns around by 180° and moves forward by one face.

Command L: it turns left and moves forward by one face

Command R: it turns right and moves forward by one face.

The faces 4,5,6 are opposite to faces 1,2,3 respectively.

For example, if it has orientation U on face 1, on command L, it goes to face 5 and has orientation N, and from there on command F , it will move to face 4 and will have orientation E.

Given a sequence of commands, and the final position (face and orientation) of the bug (after executing the commands in the sequence), we need to determine the initial position (face and orientation) of the bug.

**Constraints**

The length of the string of command letters <=50

**Input Format**

One string of 2 characters giving the face and orientation of the bug after it executes the instructions. The first character is a number between 1 and 6, denoting the face, and the second character is the orientation (from the set {U, D, N, S, E, W}) of the bug after executing the commands

One string of command letters. This is a sequence of letters from the set of valid commands {F, B, L, R}

**Output**

One string of two characters denoting the position of the bug before it executes the instructions. The first character gives the face number (1,2,3,...,6) and the second giving the orientation (E,W,U,D,N,S ) the bug was facing before it executed the instructions

**Explanation**

**Example 1**

Input

1U

FFF

Output

3N

Explanation

If the bug starts at 3N, it will move to 4D, 6S and 1U if a command F is given at each position. Hence, if it starts at 3N, after 3 consecutive F commands, it will be at 1U, which is the given final position. Hence the output is 3N.

**Example 2**

Input

4W

LRB

Output

3E

Explanation

If the bug starts at 3E, after the L command, it goes to 4D, and after the R command, it goes to 2S. From there, on the B command, it goes to 4W, which is the end position. Hence the output is 3E.

**Program:**

#include<stdio.h>  
#include<string.h>  
  
int main()  
{  
    char c1,c2,a[6],com[]={"WDSEUNWDSEUN"},s[51],b[10];  
    int i=0,j,k,t,n;  
    scanf("%c%c",&c1,&c2);  
    scanf("%s",s);  
    n=strlen(s);  
    t=c1-48;  
    for(i=n-1;i>=0;i--)  
    {  
        for(j=t-1,k=0;j<t+4;j++,k++)  
            a[k]=com[j];  
        for(j=0;j<5;j++)  
        {  
            if(c2==a[j])  
            {  
                c2=com[t+1];  
                if(s[i]=='B')  
                    c2=com[t+4];  
                t=t+j+1;  
                if(t>6)  
                    t-=6;  
                if(s[i]=='R' || s[i]=='L')  
                {  
                    if(s[i]=='R')  
                    {  
                        switch(c2)  
                        {  
                            case 'U':   strcpy(b,"WSUEND"); break;  
                            case 'D':   strcpy(b,"ENDWSU"); break;  
                            case 'E':   strcpy(b,"UENDWS"); break;  
                            case 'W':   strcpy(b,"DWSUEN"); break;  
                            case 'S':   strcpy(b,"SDENUW"); break;  
                            case 'N':   strcpy(b,"NUWSDE"); break;  
                        }  
                    }  
                    else if(s[i]=='L')  
                    {  
                        switch(c2)  
                        {  
                            case 'D':   strcpy(b,"WSUEND"); break;  
                            case 'U':   strcpy(b,"ENDWSU"); break;  
                            case 'W':   strcpy(b,"UENDWS"); break;  
                            case 'E':   strcpy(b,"DWSUEN"); break;  
                            case 'N':   strcpy(b,"SDENUW"); break;  
                            case 'S':   strcpy(b,"NUWSDE"); break;  
                        }  
                    }  
                c2=b[t-1];  
                }  
                break;     
            }  
        }  
    }  
    printf("%d%c\n",t,c2);  
    return 0;  
}

**Problem Description**

The Mathematics teacher wanted to introduce a new competition to the students to sharpen their skills in optimization. He drew a rectangular M ×N grid on the ground and filled it with some non-negative integers on each cell of the grid. The cell named (i,j) is at the intersection of i th row and j th column. He gave the following challenge to the students:

1. On each cell of the grid, you can pile any number of cube blocks.

2. Each layer must be rectangular (with no gaps) and rest p supported completely by the immediate below layer.

3. The number of blocks on each cell should not exceed the number written on the cell on the ground.

4. In each layer, the cell above (1,1) must be covered.

The challenge is to pile up the maximum number of blocks subject to the above conditions.

For example if the bottom grid was as follows:

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 0 |

the maximum number of blocks you can pile is 6 with one layer covering all the cells from (1,1) to (3,2).

**Constraints**

1 <= M,N <= 50

Maximum value in each cell is 50

**Input Format**

The first line will contain two comma separated integers M,N giving the size of the grid, where M is the number of rows and N is the number of columns.

The next M lines will each contain comma separated N non-negative integers giving the numbers in the grid cells.

**Output**

One line containing the number of blocks that can be piled according to the rules.

**Explanation**

**Example 1:**

Input:

3,4

5,4,9,3

4,3,5,6

2,2,1,1

Output:

32

Explanation:

One example of the maximum number of blocks that could be piled on the grid is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | 4 | 4 | 3 |
| 3 | 3 | 3 | 3 |
| 1 | 1 | 1 | 1 |

The total number of blocks is 32. Hence the output is 32.

**Example 2:**

Input

4,7

27,26,28,14,15,38,0

38,40,35,2,20,43,39

18,48,43,2,47,18,26

38,2,29,23,14,31,32

Output

242

Explanation

The number of rows is 4. The number of columns is 7. The values in the cells of the grid are

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 27 | 26 | 28 | 14 | 15 | 38 | 0 |
| 38 | 40 | 35 | 2 | 20 | 43 | 39 |
| 18 | 48 | 43 | 2 | 47 | 18 | 26 |
| 38 | 2 | 29 | 23 | 14 | 31 | 32 |

One possible maximal placing of the blocks is

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 27 | 26 | 26 | 2 | 2 | 2 | 0 |
| 27 | 26 | 26 | 2 | 2 | 2 | 0 |
| 18 | 18 | 18 | 2 | 2 | 2 | 0 |
| 2 | 2 | 2 | 2 | 2 | 2 | 0 |

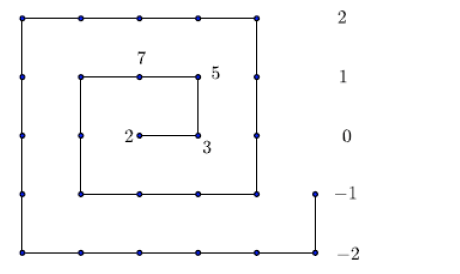
As there are 242 blocks in this maximal placing, the output is 242.

**Program:**

#include<stdio.h>  
int a[10][10];  
void del(int i,int j,int s,int f,int r,int c)  
{  
    int k,l;  
    if(s==0)  
    {  
        for(k=0;k<r;k++)  
            for(l=j;l<c;l++)  
                a[k][l]=f-1;  
    }     
    else  
    {  
        for(k=i;k<r;k++)  
            for(l=0;l<c;l++)  
                a[k][l]=f-1;  
    }  
}  
int rsum(int i,int j,int r,int c)  
{  
    int k,l,s=0;  
    for(k=0;k<i;k++)  
        for(l=0;l<c;l++)  
            s+=a[k][l];  
    return s;         
}  
int csum(int i,int j,int r,int c)  
{  
    int k,l,s=0;  
    for(k=0;k<r;k++)  
        for(l=0;l<j;l++)  
            s+=a[k][l];  
    return s;  
}  
int main()  
{  
    int i,j,k,r,c,max=0,r1,c1,s,t1,t2,sum=0;  
    char ch;  
    scanf("%d%c%d",&r,&ch,&c);  
    r1=r;  
    c1=c;  
    for(i=0;i<r;i++)  
        for(j=0;j<c;j++)  
        {  
            scanf("%d",&a[i][j]);  
            if(j<c-1)  
            scanf("%c",&ch);  
            if(a[i][j] > max)  
                max=a[i][j];  
        }  
    for(k=1;k<=max;k++)  
    {  
        for(i=0;i<r1;i++)  
            for(j=0;j<c1;j++)  
            {  
                if(a[i][j] < k)  
                {  
                    t1=rsum(i,j,r1,c1);  
                    t2=csum(i,j,r1,c1);  
                    s=(t1 > t2) ? 1 : 0;  
                    del(i,j,s,k,r1,c1);  
                    if(s==1)  
                        r1=i;  
                    else  
                        c1=j;  
                }  
            }  
    }  
    for(i=0;i<r;i++)  
    {  
        for(j=0;j<c;j++)  
        sum+=a[i][j];      
    }  
    printf("%d",sum);  
    return 0;  
}

**Spiral**

The prime numbers are written in a spiral form staring at (0,0) and moving as shown in the diagram below. The numbers  
shown on the right column and the bottom row are the column numbers and row numbers respectively (y and x coordinate  
frames).



   
The objective is to find the position (x and y coordinates) of a given prime.  
Input Format:  
The input consists of multiple lines.  
The first line gives the number of primes (N) in this test case.  
The next N lines contain one prime in each line.  
Output Format:  
The output consists of N lines.  
Each consists of a space separated pair of integers giving the x and y coordinates of the corresponding prime in the input.  
Constraints:  
N≤10  
Each prime < 1000000  
Example 1  
Input  
2  
3  
7  
Output  
1 0  
0 1  
Explanation  
There are 2 primes in this test case (N=2). The primes are 3 and 7. The coordinates of these in the spiral is (1,0) and (0,1).  
The output hence has these in space separated form.  
Example 2  
Input  
3  
5  
11  
13  
Output  
1 1  
1  
1  
1  
0  
Explanation  
There are 3 primes in this test case (N=2). The primes are 5, 11 and 13. The coordinates of these in the spiral is (1,1), (1,1)  
and (1,0).  
The output hence has these in space separated form.

**Program:**

#include<stdio.h>  
int prime[100];  
int a[9][10],num,search[100];  
void create()  
{  
    int i=4,j=4,n=3,m=5,n1=5,m1=3,k=0,dire=0;  
    //creating an array size of 9\*10 which is shown in program   
    while(i!=-1)  
    {  
        a[i][j]=prime[k];  
        k++;  
        if(dire==0)//Right direction set to 0  
        {  
            j++;  
            if(j==m)  
            {  
            m++;  
            dire=1;  
            }  
        }  
        else if(dire==1)//Up direction set to 1  
        {  
            i--;  
            if(i==n)  
            {  
                n--;  
                dire=2;  
            }  
        }  
        else if(dire==2)//Left direction set to 2  
        {  
            j--;  
            if(j==m1)  
            {  
                m1--;  
                dire=3;  
            }  
        }  
        else if(dire==3)//Down direction set to 3  
        {  
            i++;  
            if(i==n1)  
            {  
                n1++;  
                dire=0;  
            }  
        }  
    }  
    //Serahing the element postion in the Array  
    for(k=0;k<num;k++)  
    {  
    for(i=0;i<9;i++)  
    {  
        for(j=0;j<10;j++)  
        {  
        if(a[i][j]==search[k])  
        {  
            printf("%d %d\n",j-4,4-i);//printing the position as required  
        }  
        }  
    }  
    }  
      
}  
int main()  
{  
    int n, i = 3, count, c,j=0;  
    n=100;  
    //Reading Input  
    scanf("%d",&num);  
    for(j=0;j<num;j++)  
    scanf("%d",&search[j]);  
    //calculating 100 prime numbers and storing into array  
    j=0;  
        if ( n >= 1 )  
        {  
          prime[j]=2;  
          j++;  
        }  
        for ( count = 2 ; count <= n ;  )  
        {  
          for ( c = 2 ; c <= i - 1 ; c++ )  
          {  
             if ( i%c == 0 )  
                break;  
          }  
          if ( c == i )  
          {  
             prime[j]=i;  
             j++;  
             count++;  
          }  
          i++;  
        }   
        //Calling function to search and create the elements  
        create();  
return 0;  
}

In the Byteland country a string "S" is said to super ascii string if and only if count of each character in the string is equal to its ascii value.  
  
In the Byteland country ascii code of 'a' is 1, 'b' is 2 ...'z' is 26.  
  
Your task is to find out whether the given string is a super ascii string or not.

**Input Format:**  
  
First line contains number of test cases T, followed by T lines, each containing a string "S".

**Output Format:**  
  
For each test case print "Yes" if the String "S" is super ascii, else print "No"

**Constraints:**

**1<=T<=100**

**1<=|S|<=400, S will contains only lower case alphabets ('a'-'z').**

[**Sample Input and Output**](https://www.blogger.com/null)

|  |  |  |
| --- | --- | --- |
| **SNo.** | **Input** | **Output** |
| 1 | 2 bba scca | Yes No |

[**Explanation:**](https://www.blogger.com/null)  
 [In case 1, viz. String "bba" -](https://www.blogger.com/null)  
[The count of character 'b' is 2. Ascii value of 'b' is also 2.](https://www.blogger.com/null)  
[The count of character 'a' is 1. Ascii value of 'a' is also 1.](https://www.blogger.com/null)  
[Hence string "bba" is super ascii.](https://www.blogger.com/null)

**program code:**

#include<stdio.h>

#include<string.h>

#include<conio.h>

void main()

{

int a[26],l,i,k,j,c;

char s[50];

clrscr();

printf("enter the number of test cases\n");

scanf("%d",&j);

for(l=0;l<j;l++)

{

for(i=0;i<26;i++)

{

a[i]=0;

}

printf("enter the string\n");

scanf("%s",&s);

for(i=0;i<strlen(s);i++)

{

k=(int)s[i]-97;

a[k]++;

}

for(i=0;i<26;i++)

{

if(a[i]==i+1||a[i]==0)

c=1;

else

{

c=0;

break;

}

}

if(c!=0)

printf("\tyes\n");

else

printf("\tno\n");

}

getch();

}