import java.util.Scanner;

class Q4

{

static double a[][]=new double[7][7];

static boolean is\_done[][]=new boolean[7][7];

static boolean is\_max[][]=new boolean[7][7];

static boolean is\_min[][]=new boolean[7][7];

static int max\_count=0,min\_count=0;

static int area;

/\*

\* a[][]: is the height matrix

\* is\_done[][]: tells if the given entry is considered

\* for being a maxima or a minima or not

\* is\_max[][]: tells if maxima or not

\* is\_min[][]: tells if minima or not

\* max\_count: counts the number of maxima regions

\* min\_count: counts the number of minima regions

\* area: area of the peak or bottom

\*/

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

int i,j;//used in loops as counts and give the position of a point

System.out.println("Enter the 5 by 5 height matrix");

for(i=1;i<=5;i++)

{

for(j=1;j<=5;j++)

{

a[i][j]=sc.nextDouble();//input

}

}

//traversing through the non sea-level region of the map

for(i=1;i<=5;i++)

{

for(j=1;j<=5;j++)

{

if(!is\_done[i][j])//if the current element is not considered for being a maxima

{

check(i,j,0,0);//check if the element is a maxima

if(is\_max[i][j])

{

max\_count++;//increment count if maxima region is found

/\*

\* although the loop is checking each and every point but all points at the same level as

\* (i,j) have been considered and stated to be maxima, minima or neither hence they have been checked.

\* this makes assessing an element of the region equivalent to assessing the entire region

\*/

}

if(is\_min[i][j])

{

min\_count++;//description same as above just in case of a minima

}

}

}

}

if(max\_count==1 && min\_count==0)

//if there exists only 1 region of maxima and no other region of minima then the contours represent a mountain

{

System.out.print("H ");

//removing points that were initially considered as maxima but were not parts of maxima

for(i=1;i<=5;i++)

{

for(j=1;j<=5;j++)

{

if(is\_max[i][j])

confirmmax(i,j);

}

}

//finding area of maxima

for(i=0;i<=5;i++)

for(j=0;j<=5;j++)

if(is\_max[i][j])

{

find\_area(is\_max,i,j);

break;

}

System.out.print(area);

}

else if(min\_count==1 && max\_count==0)

//similar to the description of mountain

{

System.out.print("D ");

//removing points that were initially considered minimas but were not minimas

for(i=1;i<=5;i++)

{

for(j=1;j<=5;j++)

{

if(is\_max[i][j])

confirmmin(i,j);

}

}

//finding area of minima

for(i=0;i<=5;i++)

for(j=0;j<=5;j++)

if(is\_min[i][j])

{

find\_area(is\_min,i,j);

break;

}

System.out.print(area);

}

else

//if it is neither a mountain nor a valley

System.out.println("N");

}

static void confirmmax(int i,int j)

{

if(a[i-1][j-1]==a[i][j] && !is\_max[i-1][j-1])

removemax(i,j);

if(a[i-1][j]==a[i][j] && !is\_max[i-1][j])

removemax(i,j);

if(a[i-1][j+1]==a[i][j] && !is\_max[i-1][j+1])

removemax(i,j);

if(a[i][j-1]==a[i][j] && !is\_max[i][j-1])

removemax(i,j);

if(a[i][j+1]==a[i][j] && !is\_max[i][j+1])

removemax(i,j);

if(a[i+1][j-1]==a[i][j] && !is\_max[i+1][j-1])

removemax(i,j);

if(a[i+1][j]==a[i][j] && !is\_max[i+1][j])

removemax(i,j);

if(a[i+1][j+1]==a[i][j] && !is\_max[i+1][j+1])

removemax(i,j);

}

static void removemax(int i,int j)

{

is\_max[i][j]=false;

if(a[i-1][j-1]==a[i][j] && is\_max[i-1][j-1])

removemax(i-1,j-1);

if(a[i-1][j]==a[i][j] && is\_max[i-1][j])

removemax(i-1,j);

if(a[i-1][j+1]==a[i][j] && is\_max[i-1][j+1])

removemax(i-1,j+1);

if(a[i][j-1]==a[i][j] && is\_max[i][j-1])

removemax(i,j-1);

if(a[i][j+1]==a[i][j] && is\_max[i][j+1])

removemax(i,j+1);

if(a[i+1][j-1]==a[i][j] && is\_max[i+1][j-1])

removemax(i+1,j-1);

if(a[i+1][j]==a[i][j] && is\_max[i+1][j])

removemax(i+1,j);

if(a[i+1][j+1]==a[i][j] && is\_max[i+1][j+1])

removemax(i+1,j+1);

}

static void confirmmin(int i,int j)

{

if(a[i-1][j-1]==a[i][j] && !is\_min[i-1][j-1])

removemin(i,j);

if(a[i-1][j]==a[i][j] && !is\_min[i-1][j])

removemin(i,j);

if(a[i-1][j+1]==a[i][j] && !is\_min[i-1][j+1])

removemin(i,j);

if(a[i][j-1]==a[i][j] && !is\_min[i][j-1])

removemin(i,j);

if(a[i][j+1]==a[i][j] && !is\_min[i][j+1])

removemin(i,j);

if(a[i+1][j-1]==a[i][j] && !is\_min[i+1][j-1])

removemin(i,j);

if(a[i+1][j]==a[i][j] && !is\_min[i+1][j])

removemin(i,j);

if(a[i+1][j+1]==a[i][j] && !is\_min[i+1][j+1])

removemin(i,j);

}

static void removemin(int i,int j)

{

is\_max[i][j]=false;

if(a[i-1][j-1]==a[i][j] && is\_min[i-1][j-1])

removemin(i-1,j-1);

if(a[i-1][j]==a[i][j] && is\_min[i-1][j])

removemin(i-1,j);

if(a[i-1][j+1]==a[i][j] && is\_min[i-1][j+1])

removemin(i-1,j+1);

if(a[i][j-1]==a[i][j] && is\_min[i][j-1])

removemin(i,j-1);

if(a[i][j+1]==a[i][j] && is\_min[i][j+1])

removemin(i,j+1);

if(a[i+1][j-1]==a[i][j] && is\_min[i+1][j-1])

removemin(i+1,j-1);

if(a[i+1][j]==a[i][j] && is\_min[i+1][j])

removemin(i+1,j);

if(a[i+1][j+1]==a[i][j] && is\_min[i+1][j+1])

removemin(i+1,j+1);

}

static void find\_area(boolean a[][],int i,int j)

{

if(a[i][j] && is\_done[i][j]==true)

{

area++;

is\_done[i][j] = false;

find\_area(a,i-1,j-1);

find\_area(a,i-1,j);

find\_area(a,i-1,j+1);

find\_area(a,i,j-1);

find\_area(a,i,j+1);

find\_area(a,i+1,j-1);

find\_area(a,i+1,j);

find\_area(a,i+1,j+1);

}

}

static void check(int i,int j,int kp,int lp)

{

int k,l;

i=i+kp;

j=j+lp;

/\*

\* parameters i j are the co-ordinates of the calling point

\* kp,lp are the shift of current point with respect to calling point

\* i and j are adjusted to reffer to current point

\*/

if(!is\_done[i][j])

{

is\_done[i][j]=true;//marking as done

is\_max[i][j]=true;//let the current point be a maxima

is\_min[i][j]=false;//hence it is not a minima

//checking all neighbours of current point

for(k=-1;k<=1;k++)

{

for(l=-1;l<=1;l++)

{

if(i+k>=0 && j+l>=0 && i+k<=6 && j+l<=6)

{

//staying in array bounds

/\*

\* logic:

\* p: given point is a maxima

\* q: given point is greater than its neighbours

\* for all neighbours of different heights: p=p&q

\* for a neighbour of same height:

\* p(current)=p(neighbour)&q(neighbour)

\*/

if(a[i+k][j+l]>a[i][j])

is\_max[i][j]=false;

/\*

\* if there exists a point in vicinity of current point that has greater height than the current point

\* then the proposition of current point being a mxima becomes false and the failure of the test will

\* propagate throught the region of same height being a recursive call

\*/

else if(a[i+k][j+l]<a[i][j])

is\_max[i][j]=is\_max[i][j]&true;

/\*

\* if the current point is greater than a neighbour then the proposition of it being a maxima is sustained

\* in the given test

\*/

else if((k!=0 || l!=0) && (k!=-kp || l!=-lp))

//avoiding check with previous neighbour and the current element itself

{

check(i,j,k,l);

is\_max[i][j]=is\_max[i][j]&is\_max[i+k][j+l];

}

}

else

{

//calling the border as neither maxima nor a minima

is\_max[i][j]=false;

is\_min[i][j]=false;

}

}

}

if(!is\_max[i][j])

{

//if not maxima repeating the process for a mininma

is\_min[i][j]=true;

for(k=-1;k<=1;k++)

{

for(l=-1;l<=1;l++)

{

if(i+k>=0 && j+l>=0 && i+k<=6 && j+l<=6)

{

if(a[i+k][j+l]<a[i][j])

is\_min[i][j]=is\_min[i][j]&false;

else if(a[i+k][j+l]>a[i][j])

is\_min[i][j]=is\_min[i][j]&true;

else if((k!=0 || l!=0) && (k!=-kp || l!=-lp))

{

check(i,j,k,l);

is\_min[i][j]=is\_min[i][j]&is\_min[i+k][j+l];

}

}

else

{

is\_max[i][j]=false;

is\_min[i][j]=false;

}

}

}

}

}

}

}