import java.util.\*;

class Main

{

public static void main (String[] args) {

Scanner sc= new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++)

arr[i] = sc.nextInt();

int max = 2;

for(int i=0;i<n-1;i++)

{

for(int j=i+1;j<n;j++)

{

int k = j+1,diff = arr[j] - arr[i], search = arr[j] + diff;

int count = 0;

while(k < n)

{

if(arr[k] == search)

{

count++;

search = arr[k]+diff;

}

k++;

}

max = Math.max(count+2, max);

}

}

System.out.println(max);

}

}

A forest is represented as a square grid consist of 0's an 1's only.

Few people stuck in the forest, where 1's are indicate people, and 0's are indicate as trees.

A person cannot move from the tree.

You want to save people, a person can be saved, if he/she connected to other person

who is at the boundary of forest

If 'a' is connected to 'b' and 'b' is connected to 'c',

then you can consider that 'a' is also connected to 'c'. They can connected in 4 directions only.

(up, right, left and down)

You need to find out the number of persons, who cannot be saved from thegiven forest grid.

Input Format:

-------------

Line-1 -> An integer N, size of the grid N\*N

next N lines -> N space separated integers(0's and 1's)

Output Format:

--------------

Print an integer as your result

Sample Input-1:

---------------

5

0 0 1 1 0

1 0 0 1 0

0 0 1 0 0

0 1 1 0 1

1 0 0 1 0

Sample Output-1:

----------------

3

Explanation:

------------

In the given grid, 1's at (2,2), (3,1), (3,2) positions are not connected to the boundary

So, number of people cannot be saved are 3

Sample Input-2:

---------------

5

0 0 1 1 0

1 0 0 1 0

0 0 1 1 0

0 1 1 0 1

1 0 0 1 0

Sample Output-2:

----------------

0

Explanation:

------------

In the given grid, each 1 is either at the boundary or connceted to 1 at boundary.

So, number of people cannot be saved are '0'

//backend testcases:

case = 1

input =

5

0 0 1 1 0

1 0 0 1 0

0 0 1 1 0

0 1 1 0 1

1 0 0 1 0

output =0

case = 2

input =

5

0 0 1 1 0

1 0 0 1 0

0 0 1 0 0

0 1 1 0 1

1 0 0 1 0

output =

3

case = 3

input =

3

0 0 0

0 1 0

0 0 1

output =1

import java.util.\*;

class Main

{

static int[] rowD = {-1, 0, 1, 0};

static int[] colD = {0, 1, 0, -1};

static LinkedList<int[]> q = new LinkedList<>();

static int getNonRescued(int[][] arr)

{

while(!q.isEmpty())

{

int[] cord = q.remove();

for(int i=0;i<4;i++)

{

int r = cord[0] + rowD[i];

int c = cord[1] + colD[i];

if(r<0 || r>=arr.length || c<0 || c>=arr.length || arr[r][c]!=1)

continue;

arr[r][c] = 0;

q.add(new int[]{r,c});

}

}

int non\_escaped = 0;

for(int i=0; i<arr.length;i++)

for(int j=0;j<arr.length;j++)

if(arr[i][j] == 1)

non\_escaped++;

return non\_escaped;

}

public static void main (String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[][] arr = new int[n][n];

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

{

arr[i][j] = sc.nextInt();

if((arr[i][j] == 1) && (i==0 || j==0 || i == arr.length-1 || j == arr.length-1))

{

q.add(new int[]{i, j});

arr[i][j] = 0;

}

}

System.out.println(getNonRescued(arr));

}

}