# Question - 1
## Strokes to paint

Alex wants to paint a picture but hates taking the brush off the canvas. In one stroke, Alex can only paint the same colored cells which are joined via some edge.

Given the painting, determine the minimum number of strokes to completely paint the picture.

Take for example, the canvas with height given by *h = 3* and width given by *w = 5* is to be painted with picture *picture=["aabba", "aabba", "aaaca"]*, the diagram below shows the *4* strokes needed to paint the canvas.

```
        Strokes
Canvas  1   2   3 4
aabba   aa  bb    a
aabba   aa  bb    a
aaaca   aaa     c a
```

**Function Description**
Complete the function *strokesRequired* in the editor below. The function must return an integer, the minimum number of strokes required to paint the canvas.

strokesRequired has the following parameter(s):
*picture[picture[0],...picture[h-1]]:* an array of strings where each string represents one row of the picture to be painted

**Constraints**

- $1 \le h \le 10^5$
- $1 \le w \le 10^5$
- $1 \le h*w \le 10^5$
- *len(picture[i]) = w (where 0 ≤ i < h)*
- *picture[i][j] ∈ {'a', 'b', 'c'} (where 0 ≤ i < h and 0 ≤ j < w)*

**▼ Input Format For Custom Testing**

The first line contains an integer, *h*, that denotes the height of the picture and the number of elements in *picture*.

Each line *i* of the *h* subsequent lines (where *0 ≤ i < h*) contains a string that describes *picture[i]*.

**▼ Sample Case 0**

**Sample Input For Custom Testing**

Help

```
3
aaaba
ababa
aaaca
```

**Sample Output**

```
5
```

**Explanation**

The 'a's can be painted in *2* strokes, 'b's in *2* strokes and 'c' in *1* stroke, for a total of 5.

```
        Strokes
 Canvas  1   2 3 4 5
 aaaba  aaa    b   a
 ababa  a a  b b   a
 aaaca  aaa      c a
```

**▼ Sample Case 1**

**Sample Input For Custom Testing**

```
4
bbba
abba
acaa
aaac
```

**Sample Output**

```
4
```

**Explanation**

The 'a's can be painted in *1* stroke, the 'b's in *1* stroke and each 'c' requires *1* stroke.

```
        Strokes
 Canvas  1   2   3 4
 bbba   bbb    a
 abba    bb a  a
 acaa       a aa c
 aaac       aaa    c
```

## Question - 2

**Efficient Janitor**

The janitor at Hacker High School is insanely efficient. By the end of each day, all of the waste from the trash cans in the school has been shifted into plastic bags which can carry waste weighing between 1.01 pounds and 3.00 pounds. All of the plastic bags must be dumped into the trash cans outside the school. The janitor can carry at most 3.00 pounds at once. One trip is described as selecting a few bags which together don't weigh more than 3.00 pounds, dumping them in the outdoor trash can and returning to the school. The janitor wants to make minimum number of trips to the outdoor trash can. Given the number of plastic bags, *n,* and the weights of each bag, determine the minimum number of trips if the janitor selects bags in the optimal way.

For example, given *n = 6* plastic bags weighing *weight = [1.01, 1.99, 2.5, 1.5, 1.01],* the janitor can carry all of the trash out in *3* trips: *[1.01 + 1.99 , 2.5, 1.5 + 1.01]*.

**Function Description**
Complete the function *efficientJanitor* in the editor below. The function must return a single integer that represents the minimum number of trips to be made.

*efficientJanitor* has the following parameter(s):
*weight[weight[0],...weight[n-1]]:* an array of floating-point integers

**Constraints**

- *1 ≤ n ≤ 1000*
- *1.01 ≤ weight[i] ≤ 3.0*

**▼ Input Format For Custom Testing**

The first line contains an integer, *n*, that denotes the number of elements in *weight*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer that describes *weight[i]*.

**▼ Sample Case 0**

**Sample Input For Custom Testing**

```
5
1.50
1.50
1.50
1.50
1.50
```

**Sample Output**

```
3
```

**Explanation**
In this case, the janitor will carry the first *2* plastic bags together, the $3^{rd}$ and $4^{th}$ together and the last one alone to dispose of all of the trash in *3* trips.

**▼ Sample Case 1**

**Sample Input For Custom Testing**

```
4
1.50
1.50
1.50
1.50
```

**Sample Output**

```
2
```

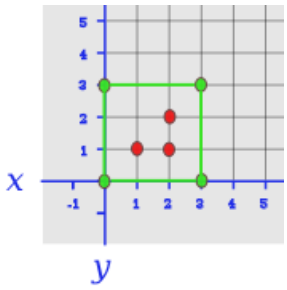
**Explanation**
In this case, the janitor will carry the first *2* plastics bags together and the $3^{rd}$ and $4^{th}$ together requiring only *2* trips.

# Question - 3
## Minimum Area

Given a list of points described by their *(x,y)* coordinates on a two dimensional plane, construct a square surrounding at least a given number of points within the area enclosed. That area should be minimal and the square must meet the following conditions:

- The *x-coordinates* and *y-coordinates* of the points should be *integers*.
- The *sides* of the square should be *parallel to coordinate axes*.
- At least *k* of the given *n* points should *lie strictly inside the square drawn. Strictly inside means that they cannot lie on a side of the square.*

For example, given *n=3* points *(1,1), (1,2) and (2,1)* and *k=3*, surround all three points. The minimum area square is *9* units, going from the origin *(0,0)*, to *(3,3)*.




**Function Description**

Complete the function *minArea* in the editor below. The function must return the *minimum possible area* of the square satisfying the constraints as an integer.

minArea has the following parameter(s):

*x[x[0],...x[n-1]]:* an array of integer *x* coordinates
*y[y[0],...y[n-1]]:* an array of integer *y* coordinates
*k:* an integer, the minimum number of points to surround

**Constraints**

- $2 \le n \le 100$
- $-10^9 \le x[i],y[i] \le 10^9$
- $1 \le k \le n$

**▼ Input Format for Custom Testing**

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *x*.
Each of the next *n* lines contains an integer *x[i]* where $0 \le i < n$.
The next line contains the integer *n*, the size of the array *y*.
Each of the next *n* lines contains an integer *y[i]* where $0 \le i < n$.
The last line contains the integer *k*.

## ▼ Sample Case 0

**Sample Input 0**

```
2
0
2
2
0
4
2
```

**Sample Output 0**

```
36
```

**Explanation 0**

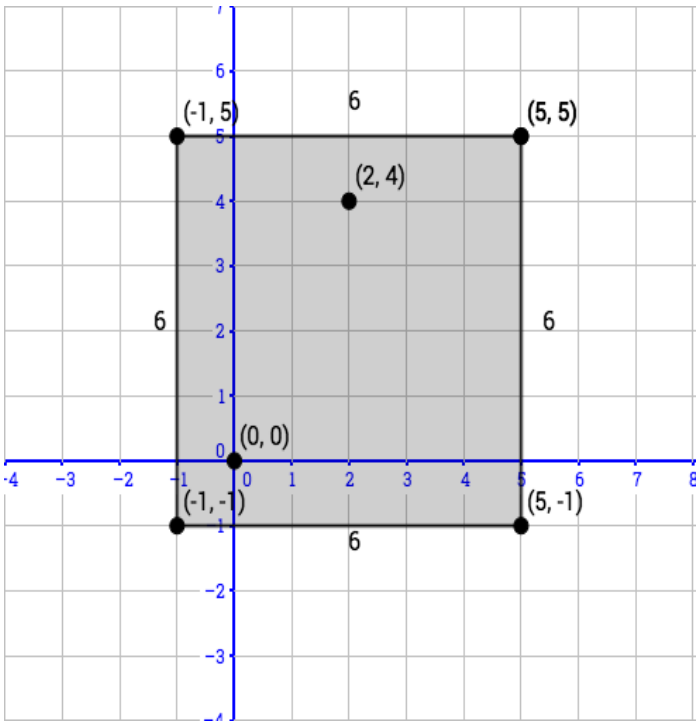
The given points are:

- *(0, 0)*
- *(2, 4)*

Choose following *four* points:

- *(-1, -1)*
- *(-1, 5)*
- *(5, 5)*
- *(5, -1)*

Draw a square of side *six*, satisfying the three constraints given in the problem statement and the area of the square is the minimum possible.




So, the function returns *36*, as the area of the square is *side x side (6 x 6 = 36)*.

## ▼ Sample Case 1

**Sample Input 1**

```
2
0
```

```
3
2
0
7
2
```

**Sample Output 1**

```
81
```

**Explanation 1**

The given points are:

- *(0, 0)*
- *(2, 7)*

Choose following *four* points:

- *(-1, -1)*
- *(-1, 8)*
- *(8, 8)*
- *(8, -1)*

Draw a square of side *nine* that satisfies the three constraints given in the problem statement and the area of the square is the minimum possible. The function returns *81 (9 x 9)*.

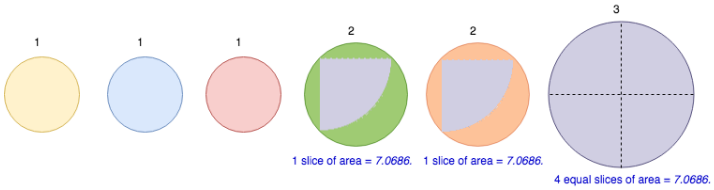
## Question - 4
**Office Party**

There is an office party to celebrate the last quarter's performance with cake for everyone! Many different circular cakes are ordered. Given the radii of the circular cakes and the number of guests, determine the largest piece that can be cut from the cakes such that every guest gets a piece with the same area. It is not possible that a single piece has some part of one cake and some part of another cake. To be fair, every guest is served only one piece of cake.

NOTE: Use *3.14159265359* as the value of pi and return the answer rounded to *4* places after the decimal. The answer is accepted if the absolute error is within $10^{-4}$ *i.e (1e-4)*.

For example, there are *6* cakes with *radii = [1, 1, 1, 2, 2, 3]* and there need to be *numberOfGuests = 6* equal size pieces. Area of a cake with radius *r* is calculated as (*pi * radius * radius)*.

- For radii *1*, *2* and *3*, the areas are *3.14159265359*, *12.5663706156*, and *28.2743338851 units*$^2$.
- It would be possible to serve everyone a piece that matches the area of the circle with radius *1*, but that would not be the biggest piece possible that can be served to everyone.
- The best way to cut the cakes is to cut the largest cake into *4* pieces (area = *28.743338851 / 4 = 7.068583471275)* and a similar size piece from each of the cakes with radius *2*.
- So, the answer is *7.0686*.




Determine the largest piece that can be cut given the above conditions.

**Function Description**

Complete the function *largestPiece* in the editor below. The function must return the maximum possible area of each piece of cake rounded to *4* decimals and cast as a string.

*largestPiece* has the following parameters:

*radii[radius[0],...radius[n-1]]:* an array of integers where $i^{th}$ element denotes the radius of the $i^{th}$ cake

*numberOfGuests*: an integer, the number of guests

**Constraints**

- $1 \le size\ of\ radii \le 10^5$
- $1 \le radii_i \le 10^4$ *(where* $0 \le i \le size\ of\ radii$*)*
- $1 \le numberOfGuests \le 10^5$

**▼ Input Format For Custom Testing**

The first line contains an integer, *n*, that denotes the number of elements in *radii*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes $radii_i$.

The last line contains an integer, *numberOfGuests*, that denotes the number of guests at the party.

**▼ Sample Case 0**

**Sample Input For Custom Testing**

```
3
4
3
3
3
```

**Sample Output**

```
28.2743
```

**Explanation**

There are *3* cakes of *radius = [4, 3, 3].* Each can have a piece as large as the smallest cake, $radius_1 = radius_2 = 3$. The area of each slice can be $pi * r * r = pi * 3 * 3 = 28.2743\ units^2$.

**▼ Sample Case 1**

**Sample Input For Custom Testing**

```
1
5
5
```

**Sample Output**

```
15.7079
```

**Explanation**

The area of the entire cake with $radii_0 = 5$ is $pi * 5 * 5 = 78.5398163475$. Divide that into $numberOfGuests = 5$ pieces of $15.7079\ units^2$

## Question - 5

**Minimum Unique Array Sum**

Given an array, you must increment any duplicate elements until all its elements are unique. In addition, the sum of its elements must be the minimum possible within the rules. For example, if *arr* = [3, 2, 1, 2, 7], then $arr_{unique}$ = [3, 2, 1, 4, 7] and its elements sum to a minimal value of 3 + 2 + 1 + 4 + 7 = 17.

**Function Description**

Complete the *getMinimumUniqueSum* function in the editor below to create an array of unique elements with a minimal sum. Return the integer sum of the resulting array.

*getMinimumUniqueSum* has the following parameter(s):

*arr:* an array of integers to process

**Constraints**

- $1 \le n \le 2000$
- $1 \le arr[i] \le 3000$ where $0 \le i < n$

**▼ Input Format For Custom Testing**

Input from stdin will be processed as follows and passed to the function:

The first line contains an integer, *n*, denoting size of array *arr*.
Each of the next *n* lines contains an integer describing element *arr[i]* where $0 \le i < n$.

**▼ Sample Case 0**

**Sample Input 0**

```
3
1
2
2
```

**Sample Output 0**

```
6
```

**Explanation 0**

*arr* = [1, 2, 2]
The duplicate array elements 2 must be addressed. The minimum unique array will be achieved by incrementing one of the twos by 1, creating the array [1, 2, 3]. The sum of elements in the new array is 1 + 2 + 3 = 6.

▼ Sample Case 1

**Sample Input 1**

```
3
1
2
3
```

**Sample Output 1**

```
6
```

**Explanation 1**

*arr* = [1, 2, 3]
Each number in *arr* is unique, so we do not need to modify any of its elements (i.e., $arr \equiv arr_{unique}$). We return the sum of all elements in the array, which is 1 + 2 + 3 = 6.

▼ Sample Case 2

**Sample Input 2**

```
4
2
2
4
5
```

**Sample Output 2**

```
14
```

**Explanation 2**

*arr* = [2, 2, 4, 5]
Because *arr[0]* and *arr[1]* are duplicates, we increment one of the two elements to *3*. When we do this, we get $arr_{unique}$ = [2, 3, 4, 5].
The sum of these elements is 2 + 3 + 4 + 5 = 14.

## Question - 6

**Casey's Image Editing**

Casey has a square image made up of black and white pixels represented as *0* and *1* respectively. As part of an image analysis process, Casey needs to determine the size of the largest square area of white pixels. Given a *2*-dimensional square matrix that represents the image, write a function to determine the length of a side of the largest square area made up of white pixels.

For example, the *n x n = 5 x 5* matrix of pixels is represented as *arr = [[1,1,1,1,1], [1,1,1,0,0], [1,1,1,0,0], [1,1,1,0,0], [1,1,1,1,1]*. A diagram of the matrix is:

```
1 1 1 1 1
1 1 1 0 0
1 1 1 0 0
```

```
1 1 1 0 0
1 1 1 1 1
```

The largest square sub-matrix is *3 x 3* in size starting at position *(0, 0)*, *(1, 0)* or *(2, 0)*. The expected return value is *3*.

**Function Description**

Complete the function *largestMatrix* in the editor below. The function must return width of the largest square sub-matrix of white pixels.

*largestMatrix* has the following parameter:

*arr[arr[0][0],...arr[n-1][n-1]]:* a 2D array of integers

**Constraints**

- *1 ≤ n ≤ 500*
- *arr[i][j] ∈ {0, 1} (0 denotes black pixel and 1 denotes white pixel)*

**▼ Input Format For Custom Testing**

The first line contains an integer, *n*, the number of rows.
The second line contains an integer, n, the number of columns.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains *n* space-separated integers that describe *arr[i]*.

**▼ Sample Case 0**

**Sample Input For Custom Testing**

```
3
3
1 1 1
1 1 0
1 0 1
```

**Sample Output**

```
2
```

**Explanation**

*(0, 0)* to *(1,1)* is the maximum square sub-matrix that contains all white pixels.

**▼ Sample Case 1**

**Sample Input For Custom Testing**

```
3
3
0 1 1
1 1 0
1 0 1
```

**Sample Output**

```
1
```

**Explanation**

There is no square sub-matrix of size greater than *1* that contains all white pixels.

## Question - 7

**Angry Animals**

Pi's father, Danny, runs the Hackerville Zoo. He is moving to Rookieville and wants to take all of the zoo animals with him via ship. He is confused about how to arrange them because a few of the species cannot be kept together in the same cabin.

There are *n* animals placed in a straight line. Each animal is identified by a unique number from *1* to *n* in order. There are *m* pairs *(a[i], b[i])* which imply that animals *a[i]* and *b[i]* are enemies and should not be kept in the same cabin. Pi is good at solving problems and he came up with following challenge: count the number of different groups that do not contain any pair such that they are enemies. A group is defined as an interval (x, y) such that all animals in the range from x to y form a group. Determine the number of groups that can be formed according to the Pi's challenge.

For example, given *n = 3* animals and *m = 3* pairs of enemies, *a = [1, 2, 3]* and *b = [3, 3, 1],* animal *1* is the enemy of animal *3,* and animal *3* is the enemy of animals *1* and *2*. Because *3* is an enemy of both *1* and *2*, it must be in its own cabin. Animals *1* and *2* can be roomed together or separately. There are four possible groupings meeting the constraints: *{1, 2} , {1}, {2}, {3}*. Note that the intervals are along the original line of animals numbered consecutively from *1* to *n,* i.e. *[1, 2, 3]* in this case. They cannot be reordered.

**Function Description**
Complete the function *angryAnimals* in the editor below. The function must return the number of groups that can be formed according to Pi's challenge.

*angryAnimals* has the following parameters:
*n*: an integer that denotes the number of unique animals
*a[a[0],...a[m-1]]:* an array of integers
*b[b[0],...b[m-1]]*: an array of integers

**Constraints**
- $1 \le n \le 10^5$
- $1 \le m \le 10^6$
- $1 \le a[i], b[i] \le n$

**▼ Input Format For Custom Testing**

The first line contains an integer, *n*.
The second line contains an integer, *m*, that denotes the number of elements in *a*.

Each line *i* of the *m* subsequent lines (where $0 \le i < m$) contains an integer that describes *a[i]*.
The next line again contains an integer, *m*, that denotes the number of elements in *b*.

Each line *i* of the *m* subsequent lines (where $0 \le i < m$) contains an integer that describes *b[i]*.

**▼ Sample Case 0**

**Sample Input For Custom Testing**

```
4
2
1
2
2
3
4
```

**Sample Output**

```
7
```

**Explanation**

*(1), (1,2), (2), (2,3), (3), (3,4), (4)* are the groups that be formed according to Pi's challenge.

**▼ Sample Case 1**

**Sample Input For Custom Testing**

```
5
2
1
2
2
3
5
```

**Sample Output**

```
11
```

**Explanation**

*(1), (1,2), (2), (2,3), (2,3,4), (3), (3,4), (3,4,5), (4), (4,5), (5)* are the groups that can be formed according to Pi's challenge.

## Question - 8

**Bob's and Alice's Chance**

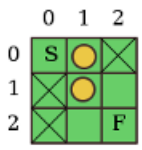
---

Bob and Alice have teamed up on a game show. They won the first round, allowing them access to a maze with hidden gold. If Bob can collect all the gold coins and deliver them to Alice's position, they can split the gold. Bob can move North⇆South or East⇆West as long as he stays in the maze and the cell is not blocked. The task is to determine the shortest path Bob can follow to collect all gold coins and deliver them to Alice. If it is not possible, return *-1*.

You will be given an *n × m* array where each of the values ∈ *{0, 1, 2}* representing *open, blocked* and *open with a gold coin*. Alice's position is given as *(x,y) = (row, column)*. Bob starts at the top left in cell *(0, 0)*.

For example, *maze = [[0,2,1],[1,2,0],[1,0,0]]* with Alice at *(2,2)* is represented as follows:

```
0 2 1
1 2 0 →
1 0 0
```




Alice's position is marked with an *F* for *Finish*. Tom, starting at *(0,0)*, has two paths to Alice of length *4*.

**Function Description**

Complete the function *minMoves* in the editor below. The function must return the integer length of Tom's shortest path or *-1* if it's not possible.

minMoves has the following parameter(s):

*maze[maze[0][0],...maze[n-1][m-1]]:* a 2D array of integers
*x:* an integer denoting Alice's row coordinate
*y:* an integer denoting Alice's column coordinate

**Constraints**

- *1 ≤ n, m ≤ 100*
- *0 ≤ the number of coins ≤ 10*
- *1 ≤ x < n*
- *1 ≤ y < m*

## ▼ Input Format For Custom Testing

The first line contains an integer n, the numbers of rows in *maze*.
The second line contains an integer m, the number of columns in *maze*.
Each of the next *n* lines contains *m* space-separated integers describing the cells of each row in *maze*.
The next line contains an integer *x*.
The next line contains an integer, *y*.

## ▼ Sample Case 0

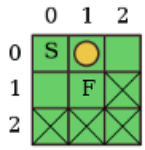
**Sample Input 0**

```
3
3
0 2 0
0 0 1
1 1 1
1
1
```

**Sample Output 0**

```
2
```

**Explanation 0**




The shortest path Bob can take is *(0, 0) → (0, 1) → (1, 1)*.

## ▼ Sample Case 1

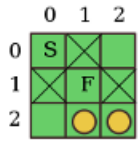
**Sample Input 1**

```
3
3
0 1 0
1 0 1
0 2 2
1
1
```

**Sample Output 1**

```
-1
```

**Explanation 1**




It is not possible for Bob to reach Alice, so we return *−1*.

## ▼ Sample Case 2

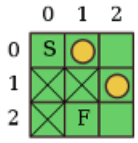
**Sample Input 2**

```
3
3
0 2 0
1 1 2
1 0 0
2
1
```

**Sample Output 2**

```
5
```

**Explanation 2**




The shortest path Bob can take is *(0, 0) → (0, 1) → (0, 2) → (1, 2) → (2, 2) → (2, 1)*.