PROBLEM 1:

## Distinct Alphabets

### Problem Description

We are all familiar with alpha numeric keypad that was used for messaging in earlier days. Given sequence of numbers 2-9 (both inclusive), find out the number of distinct alphabets that can be formed.

The rules of interpreting keypad strokes are as follows

1) Let’'s understand with example. 25 can mean AJ but it can also mean pressing *button 5* two times. In that case it becomes K. See Examples section for better understanding.

2) Maximum number of distinct alphabets that can be formed cannot exceed 26

3) Alphanumeric keyboard used is as follows

· key 2 has letters "A B C"

· key 3 has letters "D E F"

· key 4 has letters "G H I"

· key 5 has letters "J K L"

· key 6 has letters "M N O"

· key 7 has letters "P Q R S"

· key 8 has letters "T U V"

· key 9 has letters "W X Y Z"

4) Input does not contain either 1 or 0 because no keypad buttons are associated with these numbers.

### Constraints

1 <= Length of Input Literals <= 40

### Input Format

Single Line contains a number with literals [2-9]

### Output

Number of distinct alphabets after factoring all possible interpretations of the input

### Explanation

**Example 1**

Input

253

Output

5

Explanation

It can be interpreted as AJD, KD , AE,  thus distinct alphabets formed- A , J , D ,K , E =**5**distinct alphabets.

**Example 2**

Input

294

Output

5

SOLUTION:

JAVA:

**import** java.util.\*;

**public** **class** DistinctAlhabates {

ArrayList<Integer> inputint = **new** ArrayList<>();

ArrayList<Character> characters = **new** ArrayList<>();

**public** DistinctAlhabates put(String str){

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

{

inputint.add(str.charAt(i) - '0');

}

**return** **this**;

}

**void** addChar(**char** c){

**int** index = 0;

**for**(**int** i=0; i<characters.size();i++){

**if**(characters.get(i)!=c)

index++;

}

**if**(index==characters.size()){

characters.add(c);

}

}

**public** **char** getKey(**int** a){

**switch**(a){

**case** 2:

**return** 'A';

**case** 3:

**return** 'D';

**case** 4:

**return** 'G';

**case** 5:

**return** 'J';

**case** 6:

**return** 'M';

**case** 7:

**return** 'P';

**case** 8:

**return** 'T';

**case** 9:

**return** 'W';

}

**return** '\0';

}

**public** **char** getKey(**int** a, **int** b){

**switch**(a){

**case** 2:

**switch**(b%3){

**case** 1:

**return** 'A';

**case** 2:

**return** 'B';

**case** 0:

**return** 'C';

}

**case** 3:

**switch**(b%3){

**case** 1:

**return** 'D';

**case** 2:

**return** 'E';

**case** 0:

**return** 'F';

}

**case** 4:

**switch**(b%3){

**case** 1:

**return** 'G';

**case** 2:

**return** 'H';

**case** 0:

**return** 'I';

}

**case** 5:

**switch**(b%3){

**case** 1:

**return** 'J';

**case** 2:

**return** 'K';

**case** 0:

**return** 'L';

}

**case** 6:

**switch**(b%3){

**case** 1:

**return** 'M';

**case** 2:

**return** 'N';

**case** 0:

**return** 'O';

}

**case** 7:

**switch**(b%4){

**case** 1:

**return** 'P';

**case** 2:

**return** 'Q';

**case** 3:

**return** 'R';

**case** 0:

**return** 'S';

}

**case** 8:

**switch**(b%3){

**case** 1:

**return** 'T';

**case** 2:

**return** 'U';

**case** 0:

**return** 'V';

}

**case** 9:

**switch**(b%4){

**case** 1:

**return** 'W';

**case** 2:

**return** 'X';

**case** 3:

**return** 'Y';

**case** 0:

**return** 'Z';

}

}

**return** '\0';

}

**public** **int** get(){

**for**(**int** i=0;i<inputint.size();i++){

**if**(i==0){

addChar(getKey(inputint.get(i)));

}**else**{

addChar(getKey(inputint.get(i)));

addChar(getKey(inputint.get(i), inputint.get(i-1)));

}

}

**return** **this**.characters.size();

}

**public** **static** **void** main(String[] args) {

Scanner s = **new** Scanner(System.***in***);

String str = s.nextLine();

DistinctAlhabates distinctAlhabates = **new** DistinctAlhabates();

distinctAlhabates.put(str);

System.***out***.print(distinctAlhabates.get());

}

}

C:

#include<stdio.h>

#include<string.h>

char ip[40];

int it[30];

char getKey(int a){

switch(a){

case 2:

return 'A';

case 3:

return 'D';

case 4:

return 'G';

case 5:

return 'J';

case 6:

return 'M';

case 7:

return 'P';

case 8:

return 'T';

case 9:

return 'W';

}

return '\0';

}

char getKeys(int a, int b){

switch(a){

case 2:

switch(b%3){

case 1:

return 'A';

case 2:

return 'B';

case 0:

return 'C';

}

case 3:

switch(b%3){

case 1:

return 'D';

case 2:

return 'E';

case 0:

return 'F';

}

case 4:

switch(b%3){

case 1:

return 'G';

case 2:

return 'H';

case 0:

return 'I';

}

case 5:

switch(b%3){

case 1:

return 'J';

case 2:

return 'K';

case 0:

return 'L';

}

case 6:

switch(b%3){

case 1:

return 'M';

case 2:

return 'N';

case 0:

return 'O';

}

case 7:

switch(b%4){

case 1:

return 'P';

case 2:

return 'Q';

case 3:

return 'R';

case 0:

return 'S';

}

case 8:

switch(b%3){

case 1:

return 'T';

case 2:

return 'U';

case 0:

return 'V';

}

case 9:

switch(b%4){

case 1:

return 'W';

case 2:

return 'X';

case 3:

return 'Y';

case 0:

return 'Z';

}

}

return 'A';

}

void addChar(char c){

int i,j;

int index = 0;

for(j = 0; ip[j] != '\0'; ++j);

for(i=0;i<j;i++){

if(ip[i]!=c){

index++;

}

}

if(index==j){

ip[j] = c;

}

}

int get(){

int i,j;

for(j = 0; it[j] != '\0'; ++j);

for(i=0;i<j;i++){

if(i==0){

addChar(getKey(it[i]));

}else{

addChar(getKey(it[i]));

addChar(getKeys(it[i], it[i-1]));

}

}

for(j = 0; ip[j] != '\0'; ++j);

return j;

}

int main(){

int i;

scanf("%s", ip);

for(i = 0; ip[i] != '\0'; i++){

it[i] = ip[i] - '0';

}

for(i=0;i<40;i++)

ip[i] = '\0';

printf("%d", get());

return 0;

}

PROBLEM 2:

Election in DisneyLand

Problem Description

Disney Land had an election on 1st April, 2018. Disney Land is divided into different constituencies.

Different parties who fought in the Election are as follows:-

Party 1 : AAA

Party 2 : BBB

Party 3 : Others

Disney Land is rolling out a new political process. Under this process, Disney Land will be divided into N \* M equal parts called constituencies. These constituencies will then have to be aggregated into Political seats. The winner of the maximum number of seats will be the winner of the elections and will come to power in Disney Land.

The Aggregation process will involve distributing N\*M constituencies into K Seats such that

Number of constituencies that are a part of a Electoral Seat is maximum i.e. of size B\*B

No constituency is left out

There is no overlap of constituencies across Seats

Distribution of constituencies in Seats is equal

The party which wins maximum constituencies in a given Seat wins that Seat. Winner of maximum Seats wins the election.

For representational purpose, N\*M constituencies of Disney Land are represented as a matrix. The winning political party in that constituency is represented with number as follows

Party AAA -> 1

Party BBB -> 2

Others -> 3

Given the matrix, find B\*B different constituencies in the Disney Land (B < N,M). Find B in such a way that, B is maximum.

Unfortunate reality of Elections in this day and age is, Horse Trading. Disney Land is no different. After the election, some party may bribe some other party to surrender a particular constituency to win that seat.

Your task is to find which party won the Election after due process, including Horse-trading. If no party has a majority, then print "**No Majority**".

Constraints

Both N and M are even numbers

2<=N , M <=10

1<=seats <= 10

1<= H <= 10

Input Format

First Line indicates the Row (N) and Column (M) of the total number of constituencies in Disney Land.

Next N lines contain the data about which party has won which constituency

Next Line (After the N Lines of Seats) contains number of Horse-trades (H) that took place during the elections.

Next B Lines contains a data tuple <X, Y, Z> where

X & Y are the row & column of the seat where bribe took place, and

Z is the Party who initiated the horse trading.

Output

First Line should contain the party who has the majority before horse-trading and the number of seats of that party, separated by a colon (:) or **No Majority**

Second Line should contain the party who has majority after horse-trading and the number of Seats of that party, separated by a colon (:) or **No Majority**

Explanation

Example 1

Input

4 4

1 1 2 2

2 3 2 3

1 1 3 3

2 1 3 3

1

1 1 2

Output

Initial Majority Party:Seats = AAA:2

Party Won Party:Seats = BBB:2

Explanation

From the input having 4\*4 matrix; wherein N = 4 and M = 4, it is divided into 4 equal constituency of 2\*2 matrix each.

First Seat:-

1 1

2 3

Second Seat:-

2 2

2 3

Third Seat:-

1 1

2 1

Fourth Seat:-

3 3

3 3

The Party who won the First Seat = AAA

The Party who won the Second Seat = BBB

The Party who won the Third Seat = AAA

The Party who won the Fourth Seat = Others

From the Seat wins, it is clear that the Party who has the initial majority = AAA

Now, the constituency present in Row = 1 and Column = 1 is bribed by Party BBB.

Hence, that Seat (First Seat) after the Bribing process will be:-

2 1

2 3

Hence, the Party who won the First Seat = BBB

Thus, after horse-trading activities complete, the Party who finally won the Election = BBB

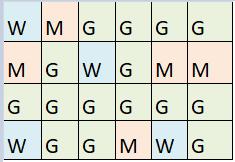
PROBLEM 1:

Jurrasic Park

Problem Description

Smilodon is a ferocious animal which used to live during the Pleistocene epoch (2.5 mya–10,000 years ago). Scientists successfully created few smilodons in an experimental DNA research. A park is established and those smilodons are kept in a cage for visitors.

This park consists of Grasslands(G), Mountains(M) and Waterbodies(W) and it has three gates (situated in grasslands only).Below is a sample layout.

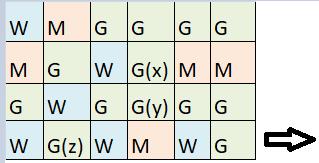


Before opening the park, club authority decides to calculate Safety index of the park. The procedure of the calculation is described below. Please help them to calculate.

Safety Index calculation

Assume a person stands on grassland(x) and a Smilodon escapes from the cage situated on grassland(y). If the person can escape from any of those three gates before the Smilodon able to catch him, then the grassland(x) is called safe else it is unsafe. A person and a Smilodon both takes 1 second to move from one area to another adjacent area(top, bottom, left or right) but a person can move only over grasslands though Smilodon can move over grasslands and mountains.

If any grassland is unreachable for Smilodon(maybe it is unreachable for any person also), to increase safe index value Club Authority use to mark those grasslands as safe land. Explained below



For the above layout, there is only one gate at (4,6)

Y is the position of Smilodon’s cage

X is not safe area

Z is a safe area as is it not possible for smilodon to reach z

Safety index=(total grassland areas which are safe\*100)/total grassland area

Constraints

3<= R,C <= 10^3

Gates are situated on grasslands only and at the edge of the park

The cage is also situated in grassland only

The position of the cage and the position of three gates are different

Input Format

The first line of the input contains two space-separated integers R and C, denoting the size of the park (R\*C)

The second line contains eight space-separated integers where

First two integers represent the position of the first gate

3rd and 4th integers represent the position of second gate

5th and 6th integers represent the position of third gate respectively

The last two integers represent the position of the cage

Next R lines, each contains space separated C number of characters. These R lines represent the park layout.

Output

Safety Index accurate up to two decimal places using Half-up Rounding method

Explanation

Example 1

Input

4 4

1 1 2 1 3 1 1 3

G G G G

G W W M

G G W W

M G M M

Output

75.00

Explanation

|  |  |  |  |
| --- | --- | --- | --- |
| G | G | G | G |
| G | W | W | M |
| G | G | W | W |
| M | G | M | M |
|  |  |  |  |
|  | Mountains | 4 |  |
|  | Gates- Safe Areas | 3 |  |
|  | Other Safe Areas | 3 |  |
|  | Waters | 4 |  |
|  | Cage Pos.-unsafe | 1 |  |
|  | Other unsafe areas | 1 |  |

Safety Index= (6\*100)/8

Example 2

Input

4 6

1 6 3 6 4 6 3 4

W M G G G G

M G W G M M

G W G G G G

W G W M W G

Output

69.23

PROBLEM 3:

## Bank Compare

### Problem Description

There are two banks; Bank A and Bank B. Their interest rates vary. You have received offers from both bank in terms of annual rate of interest, tenure and variations of rate of interest over the entire tenure.

You have to choose the offer which costs you least interest and reject the other.

Do the computation and make a wise choice.

The loan repayment happens at a monthly frequency and Equated Monthly Installment (EMI) is calculated using the formula given below :

EMI = loanAmount \* monthlyInterestRate /

( 1 - 1 / (1 + monthlyInterestRate)^(numberOfYears \* 12))

### Constraints

1 <= P <= 1000000

1 <=T <= 50

1<= N1 <= 30

1<= N2 <= 30

### Input Format

First line : P – principal (Loan Amount)

Second line : T – Total Tenure (in years).

Third Line : N1 is number of slabs of interest rates for a given period by Bank A. First slab starts from first year and second slab starts from end of first slab and so on.

Next N1 line will contain the interest rate and their period.

After N1 lines we will receive N2 viz. the number of slabs offered by second bank.

Next N2 lines are number of slabs of interest rates for a given period by Bank B. First slab starts from first year and second slab starts from end of first slab and so on.

The period and rate will be delimited by single white space.

### Output

Your decision – either Bank A or Bank B.

### Explanation

**Example 1**

Input

10000

20

3

5 9.5

10 9.6

5 8.5

3

10 6.9

5 8.5

5 7.9

Output

Bank B

**Example 2**

Input

500000

26

3

13 9.5

3 6.9

10 5.6

3

14 8.5

6 7.4

6 9.6

Output

Bank B

PROBLEM 4:

Cross Words

Problem Description

A crossword puzzle is a square grid with black and blank squares, containing clue numbers (according to a set of rules) on some of the squares. The puzzle is solved by obtaining the solutions to a set of clues corresponding to the clue numbers.

The solved puzzle has one letter in each of the blank square, which represent a sequence of letters (consisting of one or more words in English or occasionally other languages) running along the rows (called “Across”, or “A”) or along the columns (called “Down” or “D”). Each numbered square is the beginning of an Across solution or a Down solution. Some of the across and down solutions will intersect at a blank square, and if the solutions are consistent, both of them will have the same letter at the intersecting square.

In this problem, you will be given the specifications of the grid, and the solutions in some random order. The problem is to number the grid appropriately, and associate the answers consistently with the clue numbers on the grid, both as Across solutions and as Down solutions, so that the intersecting blank squares have the same letter in both solutions.

**Rules for Clue Numbering**

The clue numbers are given sequentially going row wise (Row 1 first, and then row2 and so on)

Only blank squares are given a clue number

A blank square is given a clue number if either of the following conditions exist (only one number is given even if both the conditions are satisfied)

It has a blank square to its right, and it has no blank square to its left (it has a black square to its left, or it is in the first column). This is the beginning of an Across solution with that number

It has a blank square below it, and no blank square above it (it has a black square above it or it is in the first row). This is the beginning of a Down solution with that number

Constraints

5<=N<=15

5<=M<=50

Input Format

The input consists of two parts, the grid part and the solution part

The first line of the grid part consists of a number, N, the size of the grid (the overall grid is N x N) squares. The next N lines correspond to the N rows of the grid. Each line is comma separated, and has number of pairs of numbers, the first giving the position (column) of the beginning of a black square block, and the next giving the length of the block. If there are no black squares in a row, the pair “0,0” will be specified. For example, if a line contains “2,3,7,1,14,2”, columns 2,3,4 (a block of 3 starting with 2), 7 (a block of 1 starting with 7) and 14,15 (a block of 2 starting with 14) are black in the corresponding row.

The solution part of the input appears after the grid part. The first line of the solution part contains M, the number of solutions. The M subsequent lines consist of a sequence of letters corresponding to a solution for one of the Across and Down clues. All solutions will be in upper case (Capital letters)

Output

The output is a set of M comma separated lines. Each line corresponds to a solution, and consists of three parts, the clue number, the letter A or D (corresponding to Across or Down) and the solution in to that clue (in upper case)

The output must be in increasing clue number order. Ifa clue number has both an Across and a Down solution, they must come in separate lines, with the Across solution coming before the Down solution.

Explanation

Example 1

Input

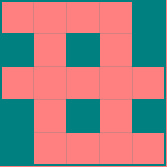
|  |
| --- |
| 5 |
| 5,1 |
| 1,1,3,1,5,1 |
| 0,0 |
| 1,1,3,1,5,1 |
| 1,1 |
| 5 |
| EVEN |
| ACNE |
| CALVE |
| PLEAS |
| EVADE |

Output

|  |
| --- |
| 1,A,ACNE |
| 2,D,CALVE |
| 3,D,EVADE |
| 4,A,PLEAS |
| 5,A,EVEN |

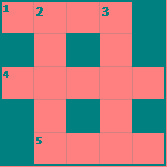
Explanation

N is 5, and the disposition of the black squares are given in the next 5 (N) lines. The grid looks like this

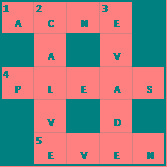


M=5, and there are 5 (M) solutions.

If the grid is numbered according to the rules, the numbered grid loos like this. Note that row 3 has no blanks, and the input line says “0,0”



The solutions are fitted to the grid so that they are consistent, and the result is shown below. Note that this is consistent, because the letter at each intersecting blank square in the Across solution and the Down solution.



Based on this the output is given in clue number order. 1 Across is ACNE, and hence the first line of the output is 1,A,ACNE. The same logic gives all the remaining solutions.

Example 2

Input

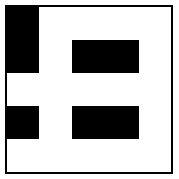
|  |
| --- |
| 5 |
| 1,1 |
| 1,1,3,2 |
| 0,0 |
| 1,1,3,2 |
| 0,0 |
| 5 |
| ASIAN |
| RISEN |
| FEAR |
| CLAWS |
| FALLS |

Output

|  |
| --- |
| 1,A,FEAR |
| 1,D,FALLS |
| 2,D,RISEN |
| 3,A,CLAWS |
| 4,A,ASIAN |

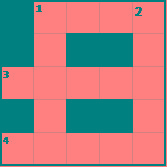
Explanation

N=5, and the grid looks like this

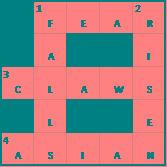


M=5, and the 5 solutions are given

The numbered grid looks like this



The consistently populated grid (with the solutions) look like this



The output can be easily given from this. Note that clue number 1 has both an Across solution (FEAR) and a DOWN solution (FALLS). The Across solution must precede the Down solution in the output.

PROBLEM 5:

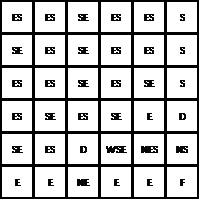
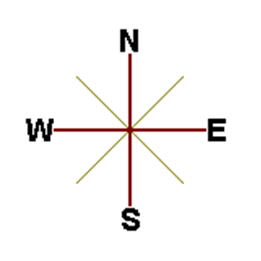
## Skateboard

### Problem Description

The amusement park at Patagonia has introduced a new skateboard competition. The skating surface is a grid of N x N squares. Most squares are so constructed with slopes that it is possible to direct the skateboard in any of up to three directions of the possible four (North ,East, South or West, represented by the letters N, E, S and W respectively). Some squares however have a deep drop from the adjacent square from which it is impossible to go to any adjacent square. These are represented by D (for Drop) in that square. The objective is to maneuver the skateboard to reach the South East corner of the grid, marked F.

Each contestant is given a map of the grid, which shows where the Drop squares are (marked D), where the Final destination is (marked F), and, for each other square, the directions it is possible to maneuver the skateboard in that square.

The contestant draws lots to determine which of the squares on the boundaries of the grid on the North or the West of the grid (the top or the left in the diagram) he or she should start in. Then, using a map of the grid, he or she needs to try to reach the South East corner destination by maneuvering the skateboard.

. 

In some cases, it is impossible to reach the destination. For example, in the diagram above, if one starts at the North East corner (top right in the diagram), the only way is to go is South, until the Drop square is reached (three squares South), and the contestant is stuck there.

A contestant asks you to figure out the number of squares at the North or West boundary (top or left boundary in the map) from which it is feasible to reach the destination.

### Constraints

5<=N<=50

### Input Format

The first line of the input is a positive integer N, which is the number of squares in each side of the grid.

The next N lines have a N strings of characters representing the contents of the map for that corresponding row. Each string may be F, representing the Final destination, D, representing a drop square, or a set of up to three of the possible four directions (N,E,S,W) in some random order. These represent the directions in which the contestant can maneuver the skateboard when in that square.

### Output

The output is one line with the number of North or West border squares from which there is a safe way to maneuver the skateboard to the final destination.

### Explanation

**Example 1**

Input

6

ES,ES,SE,ES,ES,S

SE,ES,SE,ES,ES,S

ES,ES,SE,ES,SE,S

ES,SE,ES,SE,E,D

SE,ES,D,WSE,NES,NS

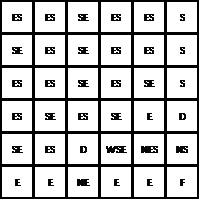
E,E,NE,E,E,F

Output

9

Explanation

N =6, and the size of the grid is 6x6. The map of the grid is as below.



There are many ways to maneuver the skateboard. For example, if the contestant starts from the North West corner (top left in the map) and goes East all the way until he reaches the top right corner in the map, and then goes South, he will reach a Drop square. But if he goes South all the way from the same square until he reaches the bottom left square on the map, and keeps going East from there, the Final destination will be reached. Hence the top left square (1,1) is safe.

Similarly, from the square (1,5), all the paths lead to a drop square., The other 9 North and West border squares have ways skateboard to get to the final destination. The output is 9

**Example 2**

Input

5

ES,SE,ES,SE,S

S,EWS,SE,E,S

E,D,SEW,NSE,S

NE,N,SEW,D,W

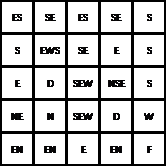
EN,EN,E,EN,F

Output

4

Explanation

N=5, and the grid is 5 x 5 squares. The map of the grid looks like this.



It can be seen that from squares (1,4) and (1,5), there is no way to maneuver the skateboard to the Final destination, and hence are not safe starting points.. Similarly, squares (2,1),(3,1), and (4,1) are not safe starting points. The only safe starting points on the North and West borders are (1,1),(1,2),(1,3),(5,1). Hence the output is 4

ESESSEESESS

SEESSEESESS

ESESSEESSES

ESSEESSEED

SEESDWSENESNS

EENEEEF

ESSEESSES

SEWSSEES

EDSEWNSES

NENSEWDW

ENENEENF

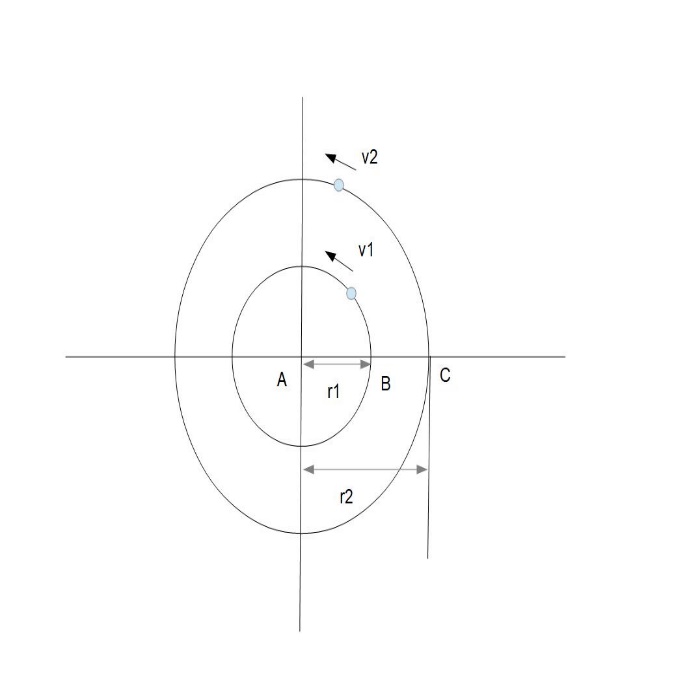
PROBLEM 6:

## Circles and Distances

### Problem Description

Task is to calculate the straight line distance between the two objects moving in a circular path. They may move at different velocities. The distance has to be calculated after N seconds.

The figure and commentary below it, explains the problem in detail.



We have two point objects B and C at rest on a straight line at a distance r1 and r2 units from a point A. At time t=0 seconds, the objects start moving in a circular path with A at the center with velocity v1 and v2 degrees per second.

Given inputs v1, v2, r1 and r2, calculate the distance between the B and C after N seconds.

Distance should be printed up to an accuracy of 2 places after the decimal point. Use [Rounding Half-up](https://simple.wikipedia.org/wiki/Rounding) semantics.

### Input Format

First line contains velocity of object B in degrees per second (v1)

Second line contains distance of object B from A (r1)

Third line contains velocity of object C in degrees per second (v2)

Fourth line contains distance of object C from A (r2)

Fifth line contains time in seconds after which the distance between B and C, is to be measured (N)

### Output

The distance between B and C, N seconds after they are set in motion

Constraints

v1, v2, r1, r2 > 0 and all are integer values.

r2 > r1

0 < n <= 100

The objects move in anticlockwise direction

v1, v2 <=360

r2 <= 100

### Explanation

Example 1

Input

90

5

270

10

1

Output

15.00

Explanation

After 1 second, the object at B would cover 90 degrees and the object at C would cover 270 degrees. Both the objects would be vertically opposite to each other and would lie in a straight line.

So the distance between them would be equal to the sum of their distance from the origin A=5+10= 15 units

JAVA:

**import** java.util.\*;

**public** **class** CircleDistance {

**float** s1,s2, angle;

Integer N;

**public** **static** **class** Obj {

**float** length;

**float** velocity;

Obj(**float** x, **float** y) {

**this**.velocity = x;

**this**.length = y;

}

}

**void** calculateAngle(Obj obj1 , Obj obj2, Integer N){

**this**.N = N;

angle = Math.*abs*((obj1.velocity\***this**.N)%360-(obj2.velocity\***this**.N)%360);

s1 = obj1.length;

s2=obj2.length;

}

**float** getResult(){

**return** (**float**) Math.*sqrt*((s1\*s1+s2\*s2)-2\*s1\*s2\*Math.*cos*(Math.*toRadians*(angle)));

}

**public** **static** **void** main(String[] args) {

Scanner s = **new** Scanner(System.***in***);

Obj obj1 = **new** Obj(s.nextFloat(),s.nextFloat());

Obj obj2 = **new** Obj(s.nextFloat(),s.nextFloat());

CircleDistance circleDistance = **new** CircleDistance();

circleDistance.calculateAngle(obj1, obj2, s.nextInt());

System.***out***.println(String.*format*("%.2f",circleDistance.getResult()));

}

}

C:

#include<stdio.h>

#include<math.h>

int main(){

float v1,v2,r1,r2,n;

scanf("%f%f%f%f%f",&v1,&r1,&v2,&r2,&n);

printf("%0.2f",sqrt(r1\*r1+r2\*r2-2\*r1\*r2\*cos((fabs(v1\*n-v2\*n)\*M\_PI/180))));

return 0;

}

Core JAVA:

import java.util.\*;

public class Name{

public static void main(String []args){

Scanner s = new Scanner(System.in):

float v1 = s.nextFloat();

float r1 = s.nextFloat();

float v2 = s.nextFloat();

float r2 = s.nextFloat();

int n = s.nextInt();

System.out.print("%0.2f",sqrt(r1\*r1+r2\*r2-2\*r1\*r2\*cos((fabs(v1\*n-v2\*n)\*M\_PI/180))));

}

}