Lab 1

Odd One Out

[40 Points] --- **Automated Grading Scheme**: **Public Test Cases (1 point each. 1*3 = 3 points) Hidden Test Cases (37 points)**

Test Case number 1 to 3 are of 1 marks each Test Case number 4 to 8 are of 4 marks each Test Case number 9 is of 6 marks Test case number 10 is of 7 marks Test case number 11 is of 4 marks **Manual Grading Scheme Note:** Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** 15 marks to be deducted if any future concept like arrays is used or any other header file apart from stdio.h is used for solving the problem. --
You are a new intern at a famous multinational company. Your first task given by your manager is as follows.

You are provided with 4 positive integers: three numbers are the same while one is different. You need to find the number that is not equal to the others. Furthermore, you also need to check whether the sum of the three equal numbers (denoted by `S`) is divisible by `6` and `9`.

Input: 4 integers

All the numbers are positive integers and all numbers except one are equal.

Output Format: The output should be across 2 lines. In the first line, print the number that is not equal to the others. In the second line, print "YES BOTH" if `S` is divisible by `6` and `9`, print "YES SIX" if `S` is divisible by `6` and not by `9`, print "YES NINE" if `S` is divisible by `9` and not by `6`, print "NO NONE" if `S` is divisible by neither `6` nor `9`. The quotes are for emphasis.

Example Input
10 6 10 10

Example Output
6
YES SIX

2	`232 323 323 323`	`232 NO NONE`	`232 NO NONE`
3	`36 7 36 36`	`7 YES BOTH`	`7 YES BOTH`
4	`1 1 1 2`	`2 NO NONE`	`2 NO NONE`
5	`5 6 5 5`	`6 NO NONE`	`6 NO NONE`
6	`1 270 270 270`	`1 YES BOTH`	`1 YES BOTH`
7	`27 27 27 1`	`1 YES NINE`	`1 YES NINE`
8	`20 20 10 20`	`10 YES SIX`	`10 YES SIX`

9	12345 12345 54321 12345	54321 YES NINE	54321 YES NINE
10	8968 8968 8968 889688	889688 YES SIX	889688 YES SIX
11	66 25 25 25	66 NO NONE	66 NO NONE

```
#include <stdio.h>


int main() {
  int a, b, c, d;
  scanf("%d %d %d %d", &a, &b, &c, &d);


  if (a == b && b == c) {
    printf("%d \n", d);
    int sum = a + b + c;
    if (sum % 6 == 0 && sum % 9 == 0) {
      printf("YES BOTH");
    } else if (sum % 6 == 0) {
      printf("YES SIX");
    } else if (sum % 9 == 0) {
      printf("YES NINE");
    } else {
      printf("NO NONE");
    }
  } else if (a == b && b == d) {
    printf("%d \n", c);
    int sum = a + b + d;
    if (sum % 6 == 0 && sum % 9 == 0) {
      printf("YES BOTH");
    } else if (sum % 6 == 0) {
```

```
    printf("YES SIX");
  } else if (sum % 9 == 0) {
    printf("YES NINE");
  } else {
    printf("NO NONE");
  }
} else if (b == c && c == d) {
  printf("%d \n", a);
  int sum = b + c + d;
  if (sum % 6 == 0 && sum % 9 == 0) {
    printf("YES BOTH");
  } else if (sum % 6 == 0) {
    printf("YES SIX");
  } else if (sum % 9 == 0) {
    printf("YES NINE");
  } else {
    printf("NO NONE");
  }
} else if (a == c && c == d) {
  printf("%d \n", b);
  int sum = a + c + d;
  if (sum % 6 == 0 && sum % 9 == 0) {
    printf("YES BOTH");
  } else if (sum % 6 == 0) {
    printf("YES SIX");
  } else if (sum % 9 == 0) {
    printf("YES NINE");
  } else {
    printf("NO NONE");
  }
}
return 0;
```

A Two Player Game

[25 Points] --- **Automated Grading Scheme**: **Public Test Cases (1 point each. 1*3 = 3 points) Hidden Test Cases (2 point each. 2*11 = 22 points)**

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** 15 marks to be deducted if any future concept like arrays is used or any other header file apart from stdio.h is used for solving the problem. --
Alice and Bob are playing a game using a pair of normal six-faced dice.

First, Alice rolls the pair of dice. If the two die show `a1` and `a2` respectively, then Alice's score is calculated as `a1 + a2`. Next, Bob rolls the pair of dice. If the two die show `b1` and `b2` respectively, then Bob's score is calculated as `b1 + b2`. The player with a higher score wins the game. It is a draw if the scores of both the players are equal.

Given `a1`, `a2`, `b1`, and `b2` as input (positive integers between 1 and 6, both inclusive), print "Alice" (without quotes) if Alice wins, "Bob" (without quotes) if Bob wins and "Draw" (without quotes) if the game ends in a draw.

Input Format: 4 integers

The input will be given in the order `a1`, `a2`, `b1`, and `b2`.

Output Format: One of the following - Alice/Bob/Draw

Example Input
1 2 3 5

Example Output
Bob

```
#include <stdio.h>

int main() {

  int a1, a2, b1, b2;

  scanf("%d %d %d %d", &a1, &a2, &b1, &b2);

  int score1 = a1 + a2;

  int score2 = b1 + b2;

  if (score1 > score2) {

    printf("Alice");

  } else if (score2 > score1) {

    printf("Bob");

  } else {
    printf("Draw");

  }

  return 0;

}
```

Power of Two

[40 Points] --- **Automated Grading Scheme**: **Public Test Cases (1 point each. 1*4= 4 points) Hidden Test Cases (4 point each. 4*9 = 36 points)**

Manual Grading Scheme Note: Solution to this question is itself a hard-coding approach, so no need to check for hard coding for this question. **Penalty:** 15 marks to be deducted if any future concept like arrays is used or any other header file apart from stdio.h is used for solving the problem. ---
You are a new intern at a famous multinational company. Your first task given by your manager is as follows.

You are provided with a number `N` where `N` is an integer strictly smaller than 128. First, you have to check if `N` is positive. If `N` is a positive integer, then find the count of number of positive powers of 2 which are smaller than `N`. Print all the positive powers of 2 which are less than `N`.

Input Format: `N`

The number `N` is an integer and is strictly smaller than 128.

Output Format:

If `N` is not a positive integer, then print "Invalid Input" (without quotes, the quotes are for emphasis). If `N` is a positive integer, the output may span upto 2 lines. In the first line, print the count of positive powers of 2 which are less than `N`. In the following line, print all the powers of 2 which are less than `N`.

Example Input 10 **Example Output:** 3 2 4 8

1	-1	Invalid Input	Invalid Input
2	1	0	0
3	31	4 2 4 8 16	4 2 4 8 16
4	0	Invalid Input	Invalid Input
5	5	2 2 4	2 2 4
6	-100	Invalid Input	Invalid Input

```
#include <stdio.h>

int main() {

  int n;

  scanf("%d", &n);

  if (n <= 0) {

    printf("Invalid Input\n");

  } else if (n <= 2) {

    printf("0\n");

  } else if (n <= 4) {

    printf("1\n2\n");

  } else if (n <= 8) {

    printf("2\n2 4\n");

  } else if (n <= 16) {

    printf("3\n2 4 8\n");

  } else if (n <= 32) {

    printf("4\n2 4 8 16\n");

  } else if (n <= 64) {

    printf("5\n2 4 8 16 32\n");

  } else if (n <= 128) {

    printf("6\n2 4 8 16 32 64\n");

  }

  return 0;

}
```

Sachin vs Virat

[34 Points] --- **Automated Grading Scheme**:

Public Test Cases (1 point each. 1*2= 2 points) Hidden Test Cases (32 points) Test Cases 1 to 2 are of 1 marks each. Test Cases 3 to 10 are of 3 marks each. Test Cases 11 and 12 are of 4 marks each.

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** 15 marks to be deducted if any future concept like arrays is used or any other header file apart from stdio.h is used for solving the problem. --

Cricket is the most popular sport in India. The Indian cricket team has been lucky to have great world-class talents in their side. Today we have Virat Kohli, but back in the day, we had the God of cricket, Sachin Tendulkar. Both being such high-class talents, it is impossible for the fans to not compare them.

Suppose you are a data analyst at a sports analytics firm. You have been assigned the duty to find out what minimum runs Virat Kohli should score in the remaining matches, so that he overtakes Sachin as the best batsman in the history of the sport. As a newbie, you only like to work with integers. So, you need to first check if the average of Sachin is an integer or not. If it is not an integer, simply print "ERROR" (without quotes), other wise print "INTEGER". If it is an integer you need to calculate the runs Virat needs to score in order to overtake Sachin. Virat will overtake Sachin only if his total runs scored and his net average are both higher than that of Sachin's. In case of any equality, Sachin will be considered the better batsman.

Note: The average is calculated as the number of runs scored per match played.

Input Format:

The first line contains two numbers which represent the number of innings played by Sachin (`M1`) and the total runs scored by him (`R1`), respectively. The next line contains three numbers, the number of innings played by Virat Kohli (`M2`), total runs scored by him (`R2`) and the remaining matches he is yet to play(`K`).

All the inputs are positive integers.

Output Format:

If the average of Sachin is not an integer, simply print "ERROR" without quotes, else print "INTEGER". In case of integer average, you need to output the minimum runs (an integer) that Virat needs to score to overtake Sachin at the end of his career in the next line.

Example Test Cases

Input	Output
400 16000 300 12000 50	INTEGER 4001
----------	
350 17500 300 16500 100	INTEGER 3501
----------	
404 17300 380 16300 40	ERROR

Explanation

- In first sample test case, Virat needs to score atleast 4001 runs more in remaining 50 matches. Doing so, his total runs (16001) and average(~45.71) will be more than Sachin's runs(16000) and average(40).

- In second test case, Sachin's runs are 17500 and his average is 50. Virat will need to score to atleast 3501 runs in remaining 100 matches. Doing so, his total runs will be 20001 and the average will be 50.0025, helping him overtake Sachin.
- In third test case, average of Sachin is not an integer. Thus the corresponding output is 'ERROR'.

```
#include <stdio.h>

int main() {

  int r1, m1, r2, m2, k;

  // Scan integers

  scanf("%d %d %d %d %d", &m1, &r1, &m2, &r2, &k);

  // if Sachin avg is integer

  if (r1 % m1 != 0) {

    printf("ERROR");

  } else {

    printf("INTEGER\n");

    int temp, ans;

    // no. of runs Virat Kohli must have in total to get equal average as SRT

    temp = r1 / m1 * (m2 + k);

    // check if runs criteria is bigger or avg criteria

    if (temp > r1) {

      ans = temp + 1 - r2;

    } else {

      ans = r1 + 1 - r2;

    }

    // check if ans < 0

    if (ans < 0) {

      printf("%d", 0);

    } else {

      printf("%d", ans);

    }

  }
  return 0;

}
```

Lab 2

Minimums and Maximums

One day, Alice went treasure hunting. She had to surpass various obstacles to get to the treasure. Upon reaching the door of the treasure, she had a complicated puzzle to solve. She was given an integer `n` and some sequence of non-negative integers delimited by `-1`. Now, she needs to figure out the value of `a+b-c`, where `a`,`b` and `c` are defined as follows:

- `a` is the maximum of the minimums of even integers for every `n` integers in the sequence.
- `b` is the minimum of the maximums of odd integers for every `n` integers in the sequence.
- `c` is defined as the maximum of numbers that are divisible by `13` and not divisible by `37` over all the numbers. If there are no such numbers, `c` is `0`.

Input Format: The first integer is `n`. It is followed by the sequence of numbers delimited by `-1` (i.e., the last number in the sequence is `-1`).

The integer `n` is at least `2`. For every `n` integers, there will be at least one even integer and one odd integer. The total numbers input after the first number are divisible by `n`.

Output Format: Print a single integer, `a+b-c`.

Public Test Cases

Input	Output
Input1	Output1

6
4 5 3 63 2 1
481 23 13 37 99 0
22 147 111 22 22 22 -1
	72

Input2	Output2

5
244 27 46 173 33
144 162 70 263 350
67 42 167 249 50 -1
	243

Note: In the above examples the inputs are in diffierent lines its only to help in undustanding. Actual inputs will be given as described.

Explanation (Case 1):

- Value of `a`: Minimum values of even integers in the three consecutive sequences of length `6` are `2`, `0`, `22` The maximum value among the minimums is : `22`.

- Value of `b`: Maximum values of odd integers in the three consecutive sequences of length `6` are `63`, `481`,`147`. The minimum value among the maximums is `63`.
- Value of `c`: only one number, i.e., `13` is divisible by `13` and not `37`, therefore c is `13`
- So the output is `a+b-c = 72`.

```
#include <stdio.h>

int main() {

  int n;

  int counter = 0;

  int min_even = -1, a = 0;

  int max_odd = 0, b = -1;

  int curr;

  int c = 0;

  scanf("%d", &n);

  scanf("%d", &curr);

  while (!(curr == -1)) {

    if (curr % 2 == 0) {

      if (min_even == -1)

        min_even = curr;

      else if (min_even > curr)

        min_even = curr;

    } else {

      if (max_odd < curr)

        max_odd = curr;

    }

    if ((curr % 13 == 0) && (curr % 37 != 0) && (c < curr))

      c = curr;

    scanf("%d", &curr);

    counter = counter + 1;
    if (counter == n) {

      if (a < min_even)

        a = min_even;

      if (b == -1)

        b = max_odd;

      else if (b > max_odd)

        b = max_odd;

      counter = 0;

      min_even = -1;

      max_odd = 0;

    }

  }

  printf("%d", a + b - c);

  return 0;

}
```

Run a Series

DK and AK are preparing for IIT exam but series is a weak topic for them. They need your help to calculate Arithmetico-Geometric Series so that they can verify their results.

Arithmetico-Geometric Series is given by: a, (a+d)*r, (a+2d)*r^2, (a+3d)*r^3, ...

You will be given the following four inputs in a single line separated by space.

1. The first term of the series (a): (real number)
2. The number for terms in the series(n): (natural number)
3. The common ratio of series(r): (real number)
4. The common difference of the series(d): (real number)

First, you will be required to give the terms of the series starting from first term separated by new line Secondly, you will be required to give average of the series in last line. You are not allowed to use pow() functionality. All results must be printed with 3 places of decimal.

Input Format:

First line contains input values separated by space.

Output Format:

Output will contain the terms of the series separated by new line and average of series in lastline.

Marking Scheme: Visible Test Case: 4 marks Hidden Test Case: 6*6 = 36 marks Total: 40 Marks.

Public Test Cases

Test Case: 1 3 2 1 Output 1.000
4.000
12.000
5.667

1	`1 3 2 1`	`1.000 4.000 12.000 5.667`	`1.000 4.000 12.000 5.667`
2	`-1.2 4 5.4 6.8`	`-1.200 30.240 361.584 3023.309 853.483`	`-1.200 30.240 361.584 3023.309 853.483`
3	`1 1 1 1`	`1.000 1.000`	`1.000 1.000`

```
#include <stdio.h>

int main()

{   float a,r,d;

    int n,i=1;

    scanf("%f %d %f %f", &a,&n,&r,&d);



    float sum=a;

    float term=0;

    float s=a;

    float factor=1;

    printf("%0.3f\n",s);

    i=2;

    while(i<=n)

    {s+=d;

    factor*=r;

    term=s*factor;
    sum+=term;

    printf("%0.3f\n",term);

    i++;

    }

    printf("%0.3f",(sum/n));

    return 0;

}
```

Third Largest Number

Given a stream of any number of non-negative integers separated by spaces, find and print the third largest number in the stream. The end of the stream is indicated by `-1` and does not belong in the sequence of numbers. If such a number with the third largest value does not exist in the input stream, then print `-1`.

The largest number is *strictly greater* than the second largest number which in turn is *strictly greater* than the third largest number and so on.

Input Format:
Sequence of non-negative integers separated by spaces, the sequence ends with `-1`.

Output Format:
The third largest non-negative integer in the sequence or `-1`.

Example 1

Input:
91 27 78 42 -1

Output:
42

Explanation:
`91 > 78 > 42 > 27`

Example 2

Input:
34 49 34 -1
Output:
-1

Explanation:
`49 > 34`, so a third largest number does not exist in the stream.

```
#include <stdio.h>

int main() {
  int i = 0, first = -1, second = -1, third = -1;

  scanf("%d", &i);

  while (i != -1) {

    if (i > first) {

      third = second;

      second = first;

      first = i;

    }

    if (i > second && i < first) {

      third = second;

      second = i;

    }

    if (i > third && i < second) {

      third = i;

    }

    scanf("%d", &i);

  }

  printf("%d", third);

  return 0;

}
```

Reverse a Number (25 Marks)

You are given `2` non-negative integers. Your task is to reverse these numbers (ignore leading zeros for reversed numbers), and then print their sum. In case of any invalid input, output "INVALID INPUT" (without quotes).

Examples:

Reversing a number can be understood by following examples:

- 468 -> 864
- 100 -> 1
- 0 -> 0

Input Format:

First line contains two space separated integers.

Output Format:

Output will contain either the sum or "INVALID INPUT" (without quotes) in case of invalid input.

Public Test Cases

Input 44 78 Output 131

```
#include <stdio.h>

int main() {

   int n1,n2,r1,r2,s1=0,s2=0;

   scanf("%d %d",&n1,&n2);

   if(n1<0 || n2<0){

      printf("INVALID INPUT");

   }

   else if(n1>=0 && n2>=0){

       while(n1){

      r1=n1%10;

      s1=s1*10+r1;

      n1=n1/10;

   }

   while(n2){

      r2=n2%10;

      s2=s2*10+r2;

      n2=n2/10;

   }

   printf("%d",s1+s2);

   }

   return 0;

}
```

Lab 3

Finding Pretty Numbers

Your friend Naruto loves numbers with some unique qualities. He calls them pretty numbers. You, being his only friend, decide to go on a quest to find pretty numbers. While on the quest, you will find one number each day for `N` days. Pretty numbers can be described as:

1. All prime numbers are pretty
2. All numbers of the form $2^a x 3^b$, `a>0` and `b>0` are also pretty. For example, `6`, `12`, `18` are pretty. *Note that `1`, `2`, and `3` do not satisfy this condition.*

Naruto also hates negative numbers and considers them ugly. He wants you to count the number of pretty numbers and ugly numbers you encounter in your quest.

A *prime* number is a number `n` (`n not equal to 1`) with exactly two factors which are `1` and `n` itself. For example, `2`, `13`, and `17`.

Input

A sequence of integers separated by spaces, the last integer in the sequence is `0`.(i.e. sequence ends with `0`)

Output

Two integers that are separated by ONE space. The first is the number of *pretty* integers and the second is the number of *ugly* numbers encountered during the quest.

Sample Test Cases

Input:

```
1 2 3 -4 121 0
```

Output:

`2 1`

Input:

```
-100 81 96 37 23 17 -999 10 512 243 0
```

Output:

`4 2`

```
#include <stdio.h>
int main() {
  int pretty_cnt = 0, ugly_cnt = 0;
  while (1) {
    int x;
    scanf("%d", &x);
    if (x == 0)
      break;
    if (x < 0) {
      ugly_cnt++;
      continue;
    }
    if (x == 1)
      continue;
    int isprime = 1;
    for (int i = 2; i < x; i++) {
      if (x % i == 0) {
        isprime = 0;
        break;
      }
    }
    if (isprime) {
      pretty_cnt++;
      continue;
    }
    int a = 0, b = 0, rx = x;
```

```
   while (rx % 2 == 0) {
     rx /= 2;
     a++;
   }
   while (rx % 3 == 0) {
     rx /= 3;
     b++;
   }
   if (rx == 1 && a > 0 && b > 0) {
     pretty_cnt++;
   }
 }
 printf("%d %d", pretty_cnt, ugly_cnt);
 return 0;
}
```

Trial of the Troll

Lyza is an esteemed adventurer who loves exploration. Once during another one of her escapades, she came across a troll blocking her way. The troll presents her with the following riddle and refuses passage until the riddle is solved.

You are given two square Matrices the same size. You have to find whether the trace of the first matrix is equal to the product of each element along the “anti-diagonal” of the second matrix.

- Anti-diagonals are defined for square matrices only.
- For a square matrix of size `N`, the anti-diagonal consists of the `(N+1-i)-th` element of the `i-th` row for each `i` between `1` and `N`.

Despite being adept at conducting expeditions, Lyza is not very good at solving Math problems and hence asks for your help to overcome this challenge.

Input Format

The 1st line contains an integer `N`. (Both matrices are of size `N x N`). The next `N` lines contain `N` space separated integers each where the `j-th` integer at the `i-th` row corresponds to the `(i,j)-th` element of the **first matrix**. The next `N` lines contain `N` space separated integers each where the `j-th` integer at the `i-th` row corresponds to the `(i,j)-th` element of the **second matrix**

Output Format:

If the required condition holds, print `YES` (in capital letters) in the first line. In the next line, print the trace of the first matrix, which is also equal to the product found for the second matrix. If the required condition does not hold, print `NO` (in capital letters) in the first line. In the next line, print the trace of the first matrix, followed by the product found for the second matrix separated by a space.

Sample Test Cases

Input 1

```
4

2 3 4 1

5 3 2 1

6 4 3 3

2 3 1 7

2 6 -3 4

-2 -12 12 0

3 8 -5 4

1 3 2 4
```

Output 1:

```
NO

15 384
```

Input 2

```
3

5 3 1

5 3 2

6 1 4

2 -8 4

11 3 4

1 7 5
```

Output 2:

```
YES
12
```

```
#include <stdio.h>

int main() {

  int N;

  scanf("%d", &N);


  int mat1_sum = 0;

  int mat2_prod = 1;


  for (int i = 0; i < N; i++) {

    for (int j = 0; j < N; j++) {

      int aval;

      scanf("%d", &aval);

      if (i == j) {

        mat1_sum += aval;

      }

    }

  }


  for (int i = 0; i < N; i++) {

    for (int j = 0; j < N; j++) {

      int bval;

      scanf("%d", &bval);

      if (i + j == N - 1) {

        mat2_prod *= bval;
      }
    }
  }


  if (mat1_sum == mat2_prod) {
    printf("YES\n");
    printf("%d", mat1_sum);
  } else {
    printf("NO\n");
    printf("%d %d", mat1_sum, mat2_prod);
  }


  return 0;
}
```

Strange Sum

[100 Points] -- **Automated Grading Scheme**: **Public Test Cases (5 point each. 2*5 = 10 points) Hidden Test Cases (90 points)** Test Case numbers 1 and 2 are of 15 marks each Test Case numbers 3 and 4 are of 10 marks each Test Case number 5 is of 20 marks Test case number 6 is of 15 marks Test case number 7 is of 5 marks **Manual Grading Scheme Note:** Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** -20% for using array and any other future concept **Penalty:** -20% each for using any library function other than printf and scanf **Penalty:** -20% each for using any library other than stdio.h ---

Your genius little cousin has just learned the concept of divisors and multiples. To play with him, you give him a sequence of positive integers and ask him to find the sum of divisors of each number `x` in the sequence. Finally, you ask him to add up the sum of digits in each of these sums and tell you the answer.

For example, for the sequence `10, 12, 6`, you expect the sum of divisors being `18` for `10 (1+2+5+10)`, `28` for `12 (1+2+3+4+6+12)` and `12` for `6 (1+2+3+6)`. So, what you want is `(1+8) + (2+8) + (1+2) = 22` as the sum of the sum of digits.

However, the genius that he is, he is not interested in this easy task. For each number `x`, he finds the sum of divisors of `x` (let us call this `sum_x`; this is in `base 10`), and finally, for all these `sum_x`, he converts them to base `k` (`k` being his favorite number). He tells you the sum of the sum of digits of all these base `k` numbers, instead of what you originally wanted.

For example, let `k=3`.

1. For `10`, we had the sum of divisors as `18`, `18` in base 3 notation is `200`
2. For `12`, we had the sum of divisors as `28`, `28` in base 3 notation is `1001`
3. For `6`, we had the sum of divisors as `12`, `12` in base 3 notation is `110`

So, he tells you the answer as `(2+0+0) + (1+0+0+1) + (1+1+0) = 6`

You happen to know his favorite number `k`. Given the sequence of numbers and `k`, write a program that prints the answer you wanted and the answer given by your cousin.

Note : In this problem, we do not necessarily have `k <= 10`. Assume there exists some number system where a single digit represents any arbitrarily large number. For example, we can write `142` in base `12` as `AB` where `A` represents `11` and `B` represents `10`. So the sum of digits in base `12` in this case will be `11 + 10 = 21`

Input

The first line contains 2 integers, `n` (number of integers in the list you have) and `k` (`n > 0`, `k > 1`). The second line contains `n` space-separated `positive` integers.

Output

Two integers separated by a space. The first one is the answer you desired. The second is the answer given by your cousin.

Sample Test Cases

Input:

```
3 3

10 12 6
```

Output:

`22 6`

Input:

```
2 2

5 7
```

#include <stdio.h>

```
int main() {
  int n, k;
  // Take n and k as input
  scanf("%d %d", &n, &k);
  // ans10 : desired answer in base 10
  // ansk : expected answer in base k
  int ans10 = 0, ansk = 0;
  for (int i = 0; i < n; i++) {
    int element;
    // Take each element as input
    scanf("%d", &element);
    // Sum of divisors
    int sm = 0;
    for (int j = 1; j <= element; j++) {
      if (element % j == 0)
        sm += j;
    }

    int base10 = sm;
    int basek = sm;

    // Compute base k sum of digits
    while (basek > 0) {
      ansk += (basek % k);
      basek /= k;
    }
```

```
   // Compute base 10 sum of digits

   while (base10 > 0) {

     ans10 += (base10 % 10);

     base10 /= 10;

   }

  }



  // Print <desired answer> <expected answer>

  printf("%d %d", ans10, ansk);



  return 0;

}
```

DINO WALK

Dino loves to walk during lab hours and visits various places. Each location has a unique code `place@x@y` where `x,y` are positive integers. There are some constraints that Dino has to respect in a **walk**.

- He can neither visit nor cross barriers
- He can only move to a location that is to his right, i.e, from `place@i@j` to `place@i@j+1`
- He can only move if the location to his right has an equal or greater movability index than the previous location

Each location has a movability index which is a positive integer. If the movability index for a location is prime, then the place acts as a barrier.

A **walk** is represented as `place@k@l` → `place@k@l+1` → .... → `place@k@m` , where none of the places lying in this walk acts as a barrier and movability index of places are in non-decreasing order. The length of a walk is defined as the number of places in the walk (it is `m-l+1` for the above example).

Dino is provided with a map which is represented as a matrix `(MxN)`, where each entry in `i` row and `j` column of a matrix denotes the movability index of `place@i@j`.

Dino is interested in finding the walk with the maximum length for every row. The sum of lengths of these maximum length walks represents a HIGH score. As a pro-gamer (programmer) of the Dino Game, you have to calculate the HIGH score.

Input Format: The first line will contain `M` and `N` followed by `M` lines, where each line contains `N` integers.

Output format: A single integer representing the sum of the lengths of the maximum walk for each row.

Example Input:

3 5

1 2 4 8 16
1 3 7 15 31
2 3 5 17 11

Example Output:

4

Explanation:

Row ↓ /Col →	**1**	**2**	**3**	**4**	**5**
1	1	**2**	4	8	16
2	1	**3**	**7**	15	**31**
3	**2**	**3**	**5**	**17**	**11**

A bold places indicates a barrier

In Row 1:

`place@1@2` will acts as a barrier. In this row, there are 7 walks possible.

Walk 1 : `place@1@1` (length : 1)
Walk 2 : `place@1@3` (length : 1)
Walk 3 : `place@1@4` (length : 1)
Walk 4 : `place@1@5` (length : 1)
Walk 5 : `place@1@3` → `place@1@4` (length : 2)
Walk 6 : `place@1@4` → `place@1@5` (length : 2)
Walk 7 : `place@1@3` → `place@1@4` → `place@1@5` (length : 3)

Walk 7 has maximum length which is 3.

In Row 2:

Two walks are possible:

Walk 1: `place@2@1`
Walk 2: `place@2@4`

Both walks have length 1. So, walk with maximum length is 1.

In Row 3:

There are no walks in Row3 since all places have prime movability index.

Answer:

So, Higher Score = 3(for row 1) + 1 (for row 2) + 0 = 4

```c
#include <stdio.h>

int main() {

  int m, n, ans = 0;

  scanf("%d %d", &m, &n);

  for (int i = 0; i < m; i++) {

    int max_path_length = 0;

    int curr_path_length = 0, prev_idx = 0;


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

      int mov_idx;

      scanf("%d", &mov_idx);


      int is_prime = 1;

      for (int k = 2; k * k <= mov_idx; k++) {

        if (mov_idx % k == 0)

          is_prime = 0;

      }

      if (mov_idx == 1)

        is_prime = 0;


      if (is_prime) {

        curr_path_length = prev_idx = 0;

        continue;

      }

      if (mov_idx < prev_idx) {

        curr_path_length = 1;

      } else {
```

```
      curr_path_length++;

    }



    prev_idx = mov_idx;

    if (curr_path_length > max_path_length)

      max_path_length = curr_path_length;

  }

  ans += max_path_length;

 }



 printf("%d", ans);



 return 0;

}
```

Lab 4

Decode Numbers

You were taught about the binary and hexadecimal representations in the lectures about the ASCII codes. It is also possible to have numbers represented with bases other than the powers of two.

Let us say that you want to represent a number with the base 17. Any integer can be uniquely represented as: $a_n 17^{n-1} + a_{n-1} 17^{n-2} + ... + a_1 * 17^0$, where for each a_i, $0 <= a_i < 17$

In the hexadecimal representation, we represented 10 by A, 11 by B, ..., and 15 by F. Similarly, in our base 17 representation we can represent 1-15 as we did in the hexadecimal representation, and represent 16 by G.

Using this scheme we can represent any number to any base in the range 2-20 (even larger bases are possible, but we are going to restrict to 20). You are given a base and a number represented with the base. Your job is to convert the number to its decimal representation or print "INVALID" if it is not possible to do the same.

It may not always be possible to construct the decimal representation of a given number as it may contain characters, which should not be part of the representation. For instance, let us say that you are asked to convert "1012" base 2 to its decimal representation. It is not possible to do so as 2 is >= 2, and the representation of a number base k should contain characters from 0 to (k-1) only. Similarly, if you are asked to convert "ABC" base 12 to its decimal representation, it is not possible to do so as it contains C which represents 12, which is equal to the base. For such cases, you should print "INVALID".

You need not worry about integer overflow issues as the inputs are small enough to fit inside the integer datatype.

Constraints

You are not allowed to use strings, arrays, or array indexing to solve this problem, you can only scan the input character by character.

Input:

- The first line contains an integer from 2-20 which is to be interpreted as the base of the number.
- The second line contains the number of characters in the following representation.
- The third line contains as many characters as given in the previous input. Each of these characters can be either uppercase English letters or numbers from 0-9.

Output:

- Your program should print the decimal representation of the number given in the third line

Sample TestCases

Test Case 1

- Input:

  - `2`
  - `4`
  - `1010`

- Output:

  `10`

- Explanation: "1010" can be converted to its decimal representation as 1*(2^3) + 0*(2^2) + 1*(2^1) + 0*(2^0) = 8 + 0 + 2 + 0 = 10

Test Case 2

- Input:

  - `12`
  - `4`
  - `AB10`

- Output:

  `18876`

- Explanation: "AB10" can be converted to its decimal representation as 10*(12^3) + 11*(12^2) + 1*(12^1) + 0*(12^0) = 17280 + 1584 + 12 + 0 = 18876

Test Case 3

- Input:

```
17
4
ABHE
```

- Output:

  `INVALID`

- Explanation: "ABHE" contains the character H, which expands to 17 in the scheme described above, and it cannot be in a number represented to the base 17.

```
#include <stdio.h>

int main() {

 int base;

 scanf("%d", &base);

 // we just need to digest the newline

 char newLine;

 scanf("%c", &newLine);

 int numDigits;

 scanf("%d", &numDigits);

 // we just need to digest the newline

 scanf("%c", &newLine);

 int power = 1;

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

  power = power * base;

 }

 int ans = 0;

 char maxNumAllowed;

 if (base <= 10)

  maxNumAllowed = '0' + (base - 1);

 else
```

```
  maxNumAllowed = 'A' + (base - 11);
char currDig;
int isInvalid = 0;
for (int i = 0; i < numDigits; i++) {
  scanf("%c", &currDig);
  if (currDig - '0' >= 0 && currDig - '0' < 10) {
    // currDig is a number
    int intEquivalent = currDig - '0';
    if (intEquivalent >= base) {
      isInvalid = 1;
      break;
    }
    ans = ans + power * intEquivalent;
  } else {
    // currDig is a character
    if (base <= 10) {
      // currDig should not have been a character
      isInvalid = 1;
      break;
    } else {
      if (currDig > maxNumAllowed) {
        isInvalid = 1;
        break;
      }
    }
    int intEquivalent = 10 + (currDig - 'A');
    ans = ans + power * intEquivalent;
  }

  power = power / base;

 }



 if (isInvalid)

  printf("INVALID");

 else

  printf("%d", ans);



 return 0;

}
```

Count Operators

Given two integers A and B. Your task is to convert A to B. You are allow to double the value (multiply by 2) or half it (integer division by 2). These operations can be performed any number of times.

The program should print the minimum number of multiplication and integer division required to convert A to B.

Constraint:

1. The integer B is always in power of 2, i.e 2^n where n can be 0,1,2,3,4,....
2. Refrain from using floating point variables for storing A, B and number of operations.

Input:

- Each test case contains two space separated integer. (The program should handle long integer too)

Output:

- The program should print the minimum number of operations required for each operator separated by space to convert initial value into final value for each test case.
- If number of operations for both multiplication and integer division are 0. Print `No operation required.`

Visible TestCases:

- TestCase1: 2 marks Input

```
1 1
```

Output:

```
No operation required.
```

- TestCase2: 2 marks Input

```
4 16
```

Output:

```
2 0
```

- TestCase3: 2 marks Input

```
5 32
```

Output:

```
4 1
```

- Explanation: TEST CASE1: Initial value and final value are same. No operation required.

  TEST CASE2: Multiplying 4 two times by 2 gives 16, i.e., 4 * 2 = 8, 8 * 2= 16. No need to half the value.

  TEST CASE3: Division of 5 once gives 2. Multiply 2 four times by 2 gives 32.

```
#include <stdio.h>

int main(void) {

 long initial_value, final_value;

 int num_mul = 0, num_div = 0;

 scanf("%ld %ld", &initial_value, &final_value);

 // count the number of division

 while (final_value % initial_value != 0) {

  initial_value = initial_value / 2;

  num_div++;

 }

 // count number of multiplication

 while (initial_value != final_value) {
    initial_value = initial_value * 2;

    num_mul++;

  }

  if (num_div == 0 && num_mul == 0) {

    printf("No operation required.");

  } else {

    printf("%d %d", num_mul, num_div);

  }

  return 0;

}
```

Problem Description

You are given a passage. There are mistakes regarding which words should begin with capital letters in the passage. For instance, in the following passage, the first letter of the second sentence begins with a lowercase letter: "This course is Fundamentals of Computing. we hope you enjoy it!" You have to write to a program to take a passage as input and print the corrected version, with the following rule:

1. The first letter of each sentence in the corrected version begins with a capital letter.

Note that the input will obey the following rules:

1. The first letter of each sentence will always begin with either a lowercase or uppercase letter.
2. Each sentence can end with either of the following 3 punctuation marks: '.', '?', '!'.
3. There is a whitespace character after each sentence, except the last one.

Constraints

- You are not allowed to use strings, arrays, or array indexing to solve this problem, you can only scan the input character by character. You are not allowed to use any string library functions for checking if a character is lowercase or uppercase.

Input:

- The first line contains an integer specifying the number of the sentence in the input.
- The second line contains the passage. Note that there are new newline characters in the passage.

Output:

- Your program should print the correct version of the passage according to the rules specified above.

Sample Test Cases

Test Case 1

- Input:

```
2
This is already a correct sentence. No change is required.
```

- Output:

```
This is already a correct sentence. No change is required.
```

- Explanation: The above input already obeys the rule we had, hence the output is the same as the input.

Test Case 2

- Input:

```
2
the first sentence itself begins with a lowercase letter. It should be
capitalized.
```

- Output:

```
The first sentence itself begins with a lowercase letter. It should be
capitalized.
```

- Explanation: As mentioned in the rules the first letter of each sentence should begin with an uppercase letter. Hence, "the" should be transformed to the "The".

```
#include <stdio.h>

int main() {

 int numSentences;

 scanf("%d", &numSentences);



 // we just need to digest the newline

 char newLine;

 scanf("%c", &newLine);
```

```
  while (numSentences) {
    char currChar;
    scanf("%c", &currChar);
    // the first char of the sentence
    if (currChar >= 'a' && currChar <= 'z') {
      char capitalChar = currChar + 'A' - 'a';
      printf("%c", capitalChar);
    } else {
      printf("%c", currChar);
    }
    // keep scanning and printint till you get end of sentence
    scanf("%c", &currChar);
    while (currChar != '!' && currChar != '.' && currChar != '?') {
      printf("%c", currChar);
      scanf("%c", &currChar);
    }
    printf("%c", currChar);
    // ingest and print the space at the beginning of the next sentence,
    // if there is a next sentence
    if (numSentences > 1) {
      scanf("%c", &currChar);
      printf("%c", currChar);
    }
    numSentences--;
  }
  return 0;
}
```

Secret Message

Sherlock Holmes has to send a letter to Dr. Watson providing him the address of a thief in hiding. To stop the thief's friends from intercepting the message, they decide to use a special cipher to encode the message that requires a *key* to decode it. The key is a positive integer. The message is a sentence composed *only of uppercase (A-Z) and lowercase (a-z) letters*, along with spaces separating the words. The encoding is described below.

Suppose `k` is the key for the cipher. Then, shift the first letter in the message by `k` letters, the second letter by `k+1` letters, the third letter by `k+2` letters, and so on. The white spaces in the sentence are *not* changed and should be printed as it is, and ignored while incrementing the key. Moreover, the case (upper or lower) of each letter remains the same.

Consider the encoding to cycle. So shifting one letter beyond `Z` will take it back to `A`, and `z` to `a`. Consider the following example. Suppose key is `2` and the message is `HappY Boy`.

- `H` is shifted by 2 to `J` (`H` -> `I` -> `J`)
- `a` is shifted by 3 to `d` (`a` -> `b` -> `c` -> `d`)
- `p` is shifted by 4 to `t` (`p` -> `q` -> `r` -> `s` -> `t`)
- `p` is shifted by 5 to `u` (`p` -> `q` -> `r` -> `s` -> `t` -> `u`)
- `Y` is shifted by 6 to `E` (`Y` -> `Z` -> `A` -> `B` -> `C` -> `D` -> `E`)
- Space remains the same.
- `B` is shifted by 7 to `I`
- `o` is shifted by 8 to `w`
- `y` is shifted by 9 to `h`

So the output would become `JdtuE Iwh`.

Sherlock wants you to create a program that takes the key and the message as the input, and prints the encoded message.

Input Format

The input contains a single line. The line first has an integer giving the key `k`. This is followed by a single space. Then a sentence of uppercase/lowercase letters follows with words separated by a single space. The final character is `?` which marks the end of the input.

1. Ignore the first space after the key. The sentence will start with a letter.
2. The sentence will have no trailing spaces.
3. DO NOT print the final `?` in the output. It is only to mark the end of the input sentence.

Output Format

Print the encoded message. Do not print any leading/trailing white spaces or new lines.

Sample Test Cases

Sample Case 1

- Input:

```
• 2 HappY Boy?
```

- Output:

  `JdtuE Iwh`

Sample Case 2

- Input:

```
• 5 aaaaaaaa?
```

- Output:

  `fghijklm`

```
#include <stdio.h>

int main() {

 int k;

 char c;

 scanf("%d", &k); // Input for key

 scanf("%c", &c); // Dummy input to ignore space

 scanf("%c", &c); // Input first letter

 while (c != '?') { // Check for end of input

  if (c == ' ') {

   printf("%c", c);                     // Print space as is

  } else if (c >= 'A' && c <= 'Z') {      // Uppercase letter

   char c1 = ((c - 'A') + k) % 26 + 'A'; // Shifted letter

   printf("%c", c1);

   k++;                              // Increment the key for next letter

  } else {                           // Lowercase letter

   char c1 = ((c - 'a') + k) % 26 + 'a'; // Shifted letter

   printf("%c", c1);

   k++; // Increment the key for next letter

  }
```

```
    scanf("%c", &c); // Input next letter
  }
}
```

Lab 5

Char Array Range Maximum

[20 Points] --- **Automated Grading Scheme**: **Public Test Cases (1 point each. 1*2 = 2 points) Hidden Test Cases (2 point each. 2*9 = 18 points)**

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** Solutions using functions to answer the queries should be awarded 0 points. --
You are given a character array (called `str`) of length `n`. The array elements will only be uppercase letters. A total of `q` queries will be made. Each query will be of the form `[l, r]`, where `[0 <= l <= r <= n-1]`. For each query, you have to print the maximum array element between indices `l` and `r` (both inclusive).

Any solution that uses functions (for answering the queries) will be awarded **ZERO** points.

Input Format

The first line contains an integer `n` that represents the length of the `str` array. The next line contains `n` uppercase letters that constitute `str` (**NOT** space separated). The third line contains an integer `q`, which is the total number of queries that will be made). The next `q` lines contain `2` integers `l` and `r` each, representing the query's range.

Output Format

Print a total of `q` characters (**NOT** space separated). The `i`th character of the output corresponds to the answer of the `i`th query.

Sample Input 1

```
10
DCBAEFGHJI
5
0 4
5 9
0 9
3 6
2 7
```

Sample Output 1

```
EJJGH
```

Explanation 1

- The character array between indices `[0, 4]` is `DCBAE`, so the answer is `E`
- The character array between indices `[5, 9]` is `FGHJI`, so the answer is `J`
- The character array between indices `[0, 9]` is `DCBAEFGHJI`, so the answer is `J`
- The character array between indices `[3, 6]` is `AEFG`, so the answer is `G`
- The character array between indices `[2, 7]` is `BAEFGH`, so the answer is `H`

Sample Input 2

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

Sample Output 2

```
LL
```

Explanation 2

- The character array between indices `[0, 1]` is `JL`, so answer is `L`
- The character array between indices `[0, 2]` is `JLE`, so answer is `L`

```
#include<stdio.h>

int main(){

        int n;

        scanf("%d", &n);

        char dummy;

        scanf("%c", &dummy);

        char str[n+1];

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

                scanf("%c", &str[i]);

        }

        str[n] = '\0';

        int q;

        scanf("%d", &q);

        while(q){

                q = q-1;

                int l, r;

                scanf("%d %d", &l, &r);

                int run_max = -1;

                for(int i=l; i<=r; i++){

                        if((str[i]-'A')>run_max){
```

```
                                        run_max = str[i]-'A';
                              }
                    }
                    printf("%c", ('A'+run_max));
          }
          return 0;
}
```

Sticky Note

[50 Points] -- **Automated Grading Scheme**: **Public Test Cases (2 point each. 2*2= 4 points) Hidden Test Cases (46 points)** Test Cases number 1 to 2 are of 2 marks each. Test Cases number 3 to 6 are of 5 marks each. Test Case number 7 is of 6 marks. Test Cases 8 and 9 are of 10 marks each.

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); --
You are a professional typist. One morning as you were finishing typing out a document, you notice a sticky note at the back of the document which states "Could you please replace all the occurences of alternating sequence of characters `c1` and `c2` with length greater than `1` in the document with a sentence `s_new`?"

Apart from being skilled in typing, you are also a skilled programmer. So to avoid the hassle of editing the document you decide to create a program for it. Suppose, you break the task into the following points for better understanding.

- Replace a substring `s1` in the original sentence `S` with sentence `s_new`
- `s1` consists of an alternating sequence of `c1` and `c2`, i.e. of the form `c1c2c1c2...` or `c2c1c2c1...`
- Length of `s1` should be greater than or equal to `2`

Input:

The input will be across `5` lines. The first line contains the original sentence `S`. The second line contains the character `c1` and the third line contains the character `c2`. The fourth line contains an integer denoting the length of the string `s_new`. The last line contains the sentence `s_new`.

You can assume that the length of `S` is `<= 600` and the length of `s_new` is `<= 50`.

Output Format

The final sentence `S'` after replacing all the occurrences of an alternating sequence of characters `c1` and `c2` in `S` with a sentence `s_new`.

Example Input
I did take a ride in Trinidad alongside Picadili Idaho
i
d
3
esc

Example Output

I esc take a resce in Trinescad alongsesce Picaescli Idaho

Brief Explanation

All the substitutions are marked in bold for clarity.

I **esc** take a r**esc**e in Trin**esc**ad alongs**esc**e Pica**esc**li Idaho

```
#include <stdbool.h>

#include <stdio.h>



int main() {

  char str[20000];



  int start = 0, end = 0;



  for (int i = 0; i < 600; i++) {

    char c;

    scanf("%c", &c);

    if (c == '\n')

      break;

    str[i] = c;

  }



  char c1;

  scanf("\n%c", &c1);

  char c2;

  scanf("\n%c", &c2);



  int s_n_len;

  scanf("%d", &s_n_len);
```

```
char dummy;
scanf("%c", &dummy);

char subs1[s_n_len];
for (int i = 0; i < s_n_len; i++) {
  char c;
  scanf("%c", &c);
  subs1[i] = c;
}
int i = -1;
int length_str = 0;
while (str[length_str] != '\0') {
  length_str++;
}
for (i = 0; i < length_str; i++) {

  int val;
  int seq_len = 0;
  int i2 = i;
  bool flagA = true, flagB = true; // Which char is allowed
  if (str[i2] != c1 && str[i2] != c2)
      val = -1;
  while ((str[i2] == c1 && flagA) || (str[i2] == c2 && flagB)) {
      flagA = true, flagB = true;
      if (str[i2] == c1)
        flagA = false;
      else
        flagB = false;

      i2++;

      seq_len++;

  }

  if (seq_len > 1)

      val = i2 - 1;

  else

      val = -1;



  if (val >= 0) {

    start = i;

    end = val;


    int i3 = -1;

    int len = length_str;

    char temp[20000];

    do {

      i3++;

      temp[i3] = str[i3];

    } while (i3 != len);


    int j = start;


    for (i3 = 0; i3 < s_n_len; i3++) {

      str[j++] = subs1[i3];

    }
```

```
      for (i3 = end + 1; i3 <= len; i3++) {

        str[j++] = temp[i3];

      }




      i = start + s_n_len;

    }

    length_str = 0;

    while (str[length_str] != '\0') {

        length_str++;

  }

  }


  printf("%s", str);


  return 0;

}
```

Training Day at IITK FC

[30 Points] -- Automated Grading Scheme: Public Test Cases (2.5 point each. 2.5*2 = 5 points) Hidden Test Cases (5 point each. 5*5 = 25 points) Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); Penalty: -15 marks for use of any inbuilt function to calculate GCD or LCM. --- Suppose you are a part of a 22 member squad of IITK Football team. Inter IIT is coming around, so it is important for the team to start training. The first day of training is simply jogging. Since it is the first day, not all team members need to be present. Every team member jogs around the field at their own pace and hence takes a different time to complete a single round. To track the time, the coach uses a stopwatch. All of them start from the same starting point. The stopwatch reads 00:00:00 at the time of start. The coach wants to know how many times any two players meet each other at the starting point in the whole jogging period. He would also like to know how many times will the whole team meet at the starting point together. The final reading of the stopwatch after training is hh:mm:ss. In case multiple team members are at the starting point simultaneously, each pair is counted individually towards the result of the first part. Input Format The first line contains the number of team members (n) out of 22 that showed up at training that day. The second line contains the final reading of the stopwatch. It is provided in a character array format. The number of characters is 8. The character array looks like hh:mm:ss, where hh represents the hours, mm represents the minutes, and ss represents the seconds. The third and the last line contains n positive integers ai, representing the time taken (in seconds) by the n team members to complete a single round of the field. Output Format The first line of the output should represent the number of times any two players meet at the starting point during the total training. The second line of the output must represent the number of times the whole squad meets together at the starting point. Constraints: 1 < n <= 22 0 < ai <= 1000 for i ? [0,n-1] The total jogging duration will not be greater than 12 hours. Hours (hh) <= 12, Minutes (mm) < 60, Seconds (ss) < 60 You are only allowed to use Standard Input-Output library. Public Test Cases Sample Input 1 3 00:10:00 50 100 150 Sample Ouput 1 12 2 Sample Input 2 6 02:00:00 40 40 40 40 40 40 Sample Ouput 2 2700 180 Explanation In the first test case, only 3 players showed up at training, and the training was only for 10 minutes. The players took 50, 100, and 150 seconds to run around the field. Players 1 and 2 will meet at the starting point after every 100s, thus meeting 6 times in total. Players 2 and 3 meet after every 300s at the starting point, making them meet 2 times during the 10 minutes. Players 1 and 3 meet after 150s, thus meeting 4 times in total. Hence, the total number of times the players meet are 6+4+2 = 12. All three players will meet together at every 300s, hence making them meet 2 times in 10 minutes.

```
#include <stdio.h>

int GCD(int a,int b){

        // Iterative Euclidean GCD

        int r;



        while(b > 0){

                r = a%b;

                a=b;

                b=r;

        }

        return a;

}



int LCM(int a, int b){

        int gcd =  GCD(a,b);



        int lcm = (a/gcd)*(b) ;
```

```
            return lcm;

}

int main() {

            int n; // no. of players training

    int array[22]; // n has a maximum value of 22

            int lcm_array[(22*21)/2]; // array to store all pairwise lcms : Max_size = nC2, where n = 22

            int lcm_all; // to store lcm of all the numbers

            int idx = 0,i,ans1,ans2;

            char time[9],c;

            int total_time = 0;


            scanf("%d\n",&n);


            for(int i=0;i<8;i++){

                        scanf("%c",&c);

                        time[i]=c;

            }

            time[8]='\0';

            int n_hours = (time[0]-'0')*10+ (time[1]-'0');

            int n_mins = (time[3]-'0')*10+ (time[4]-'0');

            int n_secs = (time[6]-'0')*10+ (time[7]-'0');


            total_time = n_hours * 3600 + n_mins * 60 + n_secs ;


            // printf("%d\n",total_time);


            // Now scan the times taken by players to complete the field
```

```
for (int i=0;i<n;i++){
        scanf("%d",&array[i]);
}
// two players will meet again at starting point after every LCM of their times
for(i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
                lcm_array[idx]=LCM(array[i],array[j]);
                idx = idx + 1;
        }
}

// here idx = (n*n-1)/2

ans1=0; // store the number of times any two players meet


for(i=0;i<idx;i++){
        ans1 += (total_time/lcm_array[i]);
}
// All of the teammates will meet together after every LCM second
ans2 = -1;
lcm_all = LCM(array[0],array[1]);
for(i=2;i<n;i++){
        lcm_all = LCM(lcm_all,array[i]);
        if(lcm_all > total_time){
                // in this case they will never be together
                ans2 = 0;
                lcm_all = -1;
```

```
                        break;

                }

        }

        // printf("%d",lcm_all);

        if(lcm_all !=-1){

                ans2 = total_time/lcm_all;

        }

        // printing the answers

        printf("%d\n",ans1);

        printf("%d",ans2);


  return 0;

}
```

Cyclic Shifting Problem

[40 Points] -- Automated Grading Scheme: Public Test Cases (2.5 point each. 2.5*2 = 5 points) Hidden Test Cases (5 point each. 5*7 = 35 points) Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); -- Dragnoid is the emperor of Dragon City. One day, Tourist visited Dragon city. Dragnoid offered him gold coins that would make Tourist rich if he could solve a problem. Unfortunately, Tourist cannot solve the problem, so he asks you to help him solve the problem. The problem goes as follows. You are given an array consisting of n integers. You have to process some queries which will modify the array and print the resulting array after each query. In each query, two numbers l and r are provided such that l>=0, r>=l, and r. In each query, two types of cyclic shifts are involved. The elements that are at a position between l and r (inclusive) and at an even distance from l, i.e. positions {x | x>=l and x<=r and (x-l)%2 = 0}, are shifted cyclically left by an offset of 1 The elements that are at a position between l and r (inclusive) and at an odd distance from l, i.e. positions {x | x>=l and x<=r and (x-l)%2 = 1}, are shifted cyclically right by an offset of 1 A left cyclic shift by an offset 1 of an array A = {a[l], a[l+1], a[l+2], a[l+3], … a[r-1], a[r]} results in the array {a[l+1], a[l+2], a[l+3], … a[r], a[l]}. A right cyclic shift by an offset 1 of the array A results in the array {a[r], a[l+1], a[l+2], … a[r-2], a[r-1]}. After each query, you will have to print the resulting array in a new line and use the new array as the initial array for the next input. Note the original array gets modified after a query, and the next query will be executed on the transformed array. Input Format The first line contains two integers, the number of elements in the array n and the number of queries q. The second line contains the array arr. The following q lines contain two integers, l and r each, representing one query. Output Format The output will contain a total of q lines, ith linw will contain the resulting array after ith query. Example Input 6 3 1 2 3 4 5 6 0 5 2 2 3 5 Example Output 3 6 5 2 1 4 3 6 5 2 1 4 3 6 5 4 1 2 Explanation In the first query, we have cyclically left-shifted the numbers at positions {0,2,4} by an offset 1, and cyclically right-shifted the numbers at positions {1,3,5} by an offset 1. So, the resulting array will be {3,6,5,2,1,4}. Primary Solution Code editor

1	3 3 19 12 8 1 1 1 2 0 2	19 12 8 19 12 8 8 12 19	19 12 8 19 12 8 8 12 19	
2	4 6 10 16 18 11 3 3 2 3 0 2 1 3 0 1 0 3	10 16 18 11 10 16 18 11 18 16 10 11 18 11 10 16 18 11 10 16 10 16 18 11	10 16 18 11 10 16 18 11 18 16 10 11 18 11 10 16 18 11 10 16 10 16 18 11	
3	1 1 14 0 0	14	14	

```
#include <stdio.h>

int main() {

 int n, q;

 scanf("%d %d ", &n, &q);

 int a[n];

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

  scanf(" %d ", &a[i]);

 }

 while (q--) {

  int l, r;

  scanf(" %d %d ", &l, &r);

  if (l == r) {

   //Do Nothing

  } else {

   int x = a[l], y;

   if ((r - l) % 2 == 1) {

    y = a[r];

   } else {

    y = a[r - 1];

   }

   int i;
```

```
      for (i = l; i + 2 <= r; i = i+2) {
        a[i] = a[i + 2];
      }
      a[i] = x;

      int st = r;
      if ((r - l) % 2 == 0)
        st--;
      for (i = st; i > l + 1; i = i-2) {
        a[i] = a[i - 2];
      }
      a[i] = y;
    }
    for (int i = 0; i < n; i = i+1) {
      printf("%d", a[i]);
      if(i != n-1)
        printf(" ");
      else if(q)
        printf("\n");
    }
  }
}
```

Lab 6

Birthday Present

Alice and Bob are invited to their friend Charlie's birthday party. Charlie is a mathematics enthusiast and really likes numbers, but in an orderly fashion. Alice has an array of `N` numbers in an increasing order. Bob has an array of `M` numbers in an increasing order. Alice and Bob wish to combine their presents and give Charlie an array of `N+M`numbers in an increasing order. You need to output the final present given to Charlie.

Input Format:
The first line contains an integer N.
The second line contains Alice’s present (N space separated integers).
The third line contains an integer M.
The fourth line contains Bob’s present (M space separated integers).

Output Format:
Output will contain `N+M` spaced separated numbers i.e. Charlie's birthday present. All the starting `N+M-1` integers will be followed by a space (' ') and the last integer will be followed by next line ('\n').

Public Test Cases

Test Case	**Output**

5
44 67 78 89 100
4
41 55 63 89
	41 44 55 63 67 78 89 89 100

1	5 44 67 78 89 100 4 41 55 63 89	41 44 55 63 67 78 89 89 100	41 44 55 63 67 78 89 89 100	
2	5 1 10 11 100 101 3 4 6 8	1 4 6 8 10 11 100 101	1 4 6 8 10 11 100 101	
3	8 0 0 5 9 45 101 600 899 8 23 67 88 103 470 909 1000 3405	0 0 5 9 23 45 67 88 101 103 470 600 899 909 1000 3405	0 0 5 9 23 45 67 88 101 103 470 600 899 909 1000 3405	

```
#include <stdio.h>

int main()

{



   int n;

   scanf("%d", &n);

   int A[n];

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

      scanf("%d", &A[i]);
}


int m;

scanf("%d", &m);

int B[m];

for(int i = 0; i < m; i++){

   scanf("%d", &B[i]);

}


int C[n+m]; // result array

int x = 0; // ptr for array A

int y = 0; // ptr for array B

int ind = 0; // ptr for array C


//Merging A and B

while(x < n && y < m){

   if(A[x] <= B[y]){

      C[ind] = A[x];

      ind++;

      x++;

   }

   else{

      C[ind] = B[y];

      ind++;

      y++;

   }

}
```

```
   // Merging left over elemnets
   while(x < n){
      C[ind] = A[x];
      ind++;
      x++;
   }
   while(y < m){
      C[ind] = B[y];
      ind++;
      y++;
   }
   for(int i = 0; i < n+m; i++){
      if(i == n+m-1) printf("%d\n", C[i]);
      else printf("%d ", C[i]);
   }
   return 0;
}
```

Complete the Code

The code below is supposed to take an arbitrary number of integers from the input and calculate the mean of the numbers. You are supposed to complete the functions `add_array()` and `get_mean()` without changing the code given in the `main` function.

- `double get_mean(int *ptr,int n)` - The function takes a pointer to the array, calculates the mean of the array, and returns the mean.
- `int *add_array(int *ptr, int n, int x)` - The function creates a new array of size `n+1`, copies the contents of the original array denoted by `ptr` to the new array, appends the value `x` at the end of the new array (i.e., after copying), and returns the pointer to the new array. (Hint: Use `malloc` and `free`.)

You **should not** use statically allocated arrays for this question. You should allocate and free arrays using dynamic memory allocation and use pointers to reference the arrays.

You **should not** change the given function prototypes.

The `main()` function is as follows.

```
int main(){
    int *ptr = (int *)malloc(4);
    int d;
    scanf("%d", &d);
    ptr[0]=d;
    int n=1;
    while(scanf("%d",&d)) {
        if (d==-1) {
            break;
        }
         ptr = add_array(ptr,n,d);
         n++;
     }
     printf("%.2lf", get_mean(ptr,n));
     return 0;
}
```

Input Format: The input is an arbitrary number of integers separated by a single space. The sequence ends with -1. The input code is given in the main function snippet.

Output Format: The output is the mean. The output is rounded to two decimal places. The output code is given in the `main` function snippet.

Example

Input: `1 2 3 4 5 -1`

Output: `3.00`

1	1 2 3 4 5 -1	3.00	3.00	
2	0 0 0 0 -1	0.00	0.00	
3	0 -2 -4 -6 -8 -1	-4.00	-4.00	

#include<stdio.h>

#include<stdlib.h>

double get_mean(int *ptr,int n){

```
    double s=0;

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

        s+=ptr[i];

    }

    return s/n;

}


int *add_array(int *ptr,int n, int x){

    int *ptr2=(int *) malloc((n+1)*sizeof(int));


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

        ptr2[i]=ptr[i];

    }

    ptr2[n]=x;

    return ptr2;

}

int main(){

            int *ptr= (int *)malloc(4);

            int d;

            scanf("%d", &d);

            ptr[0]=d;

            int n=1;

            while(scanf("%d",&d)){

                        if (d==-1) break;

                        ptr=add_array(ptr,n,d);

                        n++;

             }
             printf("%.2lf", get_mean(ptr,n));

             return 0;

}
```

In-Place String Transformation

Input a string `str` and store it in the heap memory. The maximum length of the string is 2000 characters. You are required to execute `q` queries on the string, and do in-place transformations (i.e., modify `str` directly).

Each query represents a transformation and consists of three integers: `t`, `s`, and `e`. The choice `t` specifies the type of transformation (`1 <= t <= 3`). The numbers `s` and `e` denote the start index and the end index respectively, with `0 <= s <= e < length(str)`. The particular transformation is to be applied only to the substring that lies between indices specified by `s` and `e` (both are inclusive).

A transformation could be one of the following three types. You are required to write a function to implement each transformation.

1. `void reverse(char* substr, int len)` - Reverse `substr` of length `len`. Eg: `"abc"` ? `"cba"`
2. `void cyclicShift(char* substr, int len)` - Shift `substring` cyclically towards the right by 2 places. Eg: `"abcde"` ? `"deabc"`.
3. `void swapCase(char* substr, int len)` - Swap the cases of all alphabets in `substr`. Eg: `"Ab2"` ? `"aB2"`.

You should not change the given function prototypes. You may write other helper functions if necessary.

Execute all the `q` transformations on the respective segments of `str` by calling the concerned functions with the correct parameters. Print the final string obtained after all the queries have been executed.

You **should not** use statically allocated arrays for this question. You should allocate and free space for the string using dynamic memory allocation and use pointers to reference it. Do not use array subscripting in your program. Using string library functions is also prohibited.

Input Format:

The first line is the input string `str` (newline is not part of the string). The second line specifies the number of queries `q`. The next `q` lines are triplets (`type, start, end`), representing the transformations you have to carry out in order.

Output Format:

The final string after applying all the queries.

Example

Input:

```
AbcDEFghiJ

3
```

```
1 1 4
3 0 9
2 3 7
```

Output:

`aedGHCBfIj`

Explanation:
Query 1: `AbcDEFghiJ` ? `AEDcbFghiJ`
Query 2: `AEDcbFghiJ` ? `aedCBfGHIj`
Query 3: `aedCBfGHIj` ? `aedGHCBfIj`

```
#include<stdio.h>

#include<stdlib.h>



void swap(char* a, char* b) {

    char t ;

    t = *b;

    *b = *a;

    *a = t;

}



void reverse(char* s, int n) {

    for(char* e = s + n - 1; s < e; s++, e--) {

        swap(s, e);

    }

}



void cyclicShift(char* s, int n) {

    if(n == 1) return;

    for(int j = 0; j < 2; j++) {

        for(int i = n - 2; i >= 0; i--) {

            swap(s + i, s + i + 1);
        }

    }

}


void swapCase(char* s, int n) {

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

        char c = *(s + i);

        if(c >= 65 && c <= 90) c += 32;

        else if(c >= 97 && c <= 122) c -= 32;

        *(s + i) = c;

    }

}


int main() {

    char* str = (char*)malloc(2001 * sizeof(char));

    int n = 0, q = 0;

    for(;;) {

        char c;

        scanf("%c", &c);

        if(c == '\n') break;

        *(str + (n++)) = c;

    }

    *(str + (n++)) = '\0';

    scanf("%d", &q);

    for(int i = 0; i < q; i++) {

        int t, s, e;

        scanf("%d %d %d", &t, &s, &e);
        switch(t) {
        case 1:
          reverse(str + s, e - s + 1);
          break;
        case 2:
          cyclicShift(str + s, e - s + 1);
          break;
        case 3:
          swapCase(str + s, e - s + 1);
          break;
        }
      }
      printf("%s", str);
      return 0;
    }
```

Lab 7

Balanced String

[100 Points] --- **Grading Scheme**: Visible Testcase: Testcase 1st and 2nd are of 5 marks each, and Testcase 3rd of 10 marks Hidden Testcase: 20 marks each **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); Using future concepts will deduct 20% marks Using other header files except stdio.h will lead to penalize 20% marks **Write code recursively, otherwise, 0 marks will be given.**

lexicographic order

A string S is lexicographic smaller than another string T, such that for smallest i where `S[i] != T[i]` condition `S[i] < T[i]` must hold. where `S[i]` denotes the i-th character of S, and `T[i]` denotes the i-th character of T. Here, lengths of S and T are equal.

Note : `'(' < ')' < '{' < '}'` for lexicographic order.

Problem Statement

You have to generate all balanced string of parenthesis & curly braces of length `2n + 2m` in lexicographic order.

Definition of balanced string

String with length 0 is always balanced.

A balanced string S of length `2n + 2m` is consisting of `n` pairs of parentheses and `m` pairs of curly braces, such that the number of opening parentheses `(` is equal to the number of closing parentheses `)` and number of opening curly bracket `{` is equal to number of closing curly bracket `}`. And, for every opening bracket there is a matching closing bracket such that subtring(may be with length 0) of S between these brackets is balanced.

for example, `"({})" ,"{({}())}",, "{}(()())" , "{}{}", "(){()}` are some example of balanced parentheses string. while `"){}(", "}((()", "(({)}))", "}()(){", "{({}{(})}` are not balanced parentheses string.

Input Format

The first line contains two whole numbers `n` and `m`.

Output Format

Print all the balanced strings of length `2n + 2m` in lexicographic order. Note: Don't forget to print newline character after last string.

Example Input1

`3 0`

Example Output1

```
((()))
(()())
(())()
()(())
()()()
```

Example Input2

`1 1`

Example Output2

```
1 1
(){}
({})
{()}
{}()
```

```
#include <stdio.h>

void balancedParenthesis(int n, int m, int lp, int rp, int lc, int rc, int len, char s[]) {

 if (len == 2 * n + 2 * m) {

  printf("%s\n", s);

  return;

 }

 int p = 0, c = 0;

 int idx = len - 1;

 /*

 find first open bracket (either curly or parenthesis)

 */

 while (idx >= 0) {

  if (s[idx] == ')')

   p -= 1;

  else if (s[idx] == '}')

   c -= 1;

  else if (s[idx] == '(')

   p += 1;

  else if (s[idx] == '{')

   c += 1;

  if (p > 0 || c > 0) {
      break;
    }
    idx--;
  }

  if (lp < n) {
    s[len] = '(';
    balancedParenthesis(n, m, lp + 1, rp + 1, lc, rc, len + 1, s);
  }
  if (rp > 0 && p > 0) {
    s[len] = ')';
    balancedParenthesis(n, m, lp, rp - 1, lc, rc, len + 1, s);
  }
  if (lc < m) {
    s[len] = '{';
    balancedParenthesis(n, m, lp, rp, lc + 1, rc + 1, len + 1, s);
  }
  if (rc > 0 && c > 0) {
    s[len] = '}';
    balancedParenthesis(n, m, lp, rp, lc, rc - 1, len + 1, s);
  }
}

int main() {
  int n, m;
  scanf("%d %d", &n, &m);
  char s[2 * n + 2 * m + 1];
  s[2 * n + 2 * m] = '\0';

  balancedParenthesis(n, m, 0, 0, 0, 0, 0, s);

  return 0;

}
```

Demanding List

[100 Points] -- **Grading Scheme**: Visible Testcase : 5 marks each Hidden Testcases : 45 marks each **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); Using future concept will deduct 20% marks Using other header files except stdio.h will lead to penalize 20% marks **Write code recursively, otherwise, 0 marks will be given.**

Subsequence of an array

A subsequence of an array is an ordered subset of the array's elements having the same sequential ordering as the original array. It can be obtained by deleting one or more elements form the array, while preserving the order of the remaining elements.

Example : For the array `{1,2,3,4,5,6,7}` we have valid subsequences as `{1}, {2}, {1,2}, {1,4}, {2,3,6}, {1,2,5,7}`. Some invalid subsequences would be `{2,1}, {4,1,6}, {2,1,3,6,7}`.

Problem Statement

We define a special sum of a sequence `[a[0], a[1], .... a[n-1]]` as `1*a[0] + 2*a[1] + .... + n*a[n-1]`.
Given an integer `n`, an array `a` of n elements, and an integer `sum`, print `"YES"` (without quotes) if the array has a subsequence with special sum equal to `sum`. Else print `"NO"` (without quotes).

You will be given `t` testcases per input. Print the answer for each testcase in a separate line. Make sure there are no unnecessary whitespaces.

Input Format

The first line contains an integer `t`, denoting the number of testcases. `3t` lines follow, `3` lines for each input.
The first line of each input contains an integer `n`, denoting the number of elements in the array.
The second line contains `n` space-separated integers, denoting the elements of array `a`.
The third line contains a single integer `sum`, denoting the target sum.

Output Format

For each testcase, print either `"YES"` or `"NO"` (without quotes).

Example Input

```
2
5
2 3 5 8 11
19
6
1 4 5 12 4 6
7
```

Example Output

```
YES
NO
```

Explanation

```
For the first testcase, we see that {3,8} is a valid subsequence since 1*3 + 2*8 = 19

For the second testcase, there does not exist any subsequence whose special sum is 7
```

```
#include <stdio.h>

int solve(int a[], int n, int sm, int id, int c) {
  if (sm == 0)
    return 1;
  if (id == n)
    return 0;


  if (solve(a, n, sm, id + 1, c) == 1)
    return 1;


  sm -= (c * a[id]);
  return solve(a, n, sm, id + 1, c + 1);
}

int main() {
  int t;
  scanf("%d", &t);
  while (t--) {
    int n;
    scanf("%d", &n);
    int a[n];
    for (int i = 0; i < n; i++) {
      scanf("%d", &a[i]);
    }
    int sm;
    scanf("%d", &sm);
    int ans = solve(a, n, sm, 0, 1);
    if (ans == 1)
      printf("YES");
    else
      printf("NO");
    if (t > 0)
      printf("\n");
  }
  return 0;
}
```

Division of Stones

[100 Points] -- **Grading Scheme**: For visible Testcases: 5 marks each For hidden Testcases: testcase2 of 15 marks and remaining all Testcases of 10 marks **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); Using future concepts will deduct 20% marks Using other header files except stdio.h will lead to penalize 20% marks **Write code recursively, otherwise, 0 marks will be given.**

Problem

John loves playing games and solving tough mathematical problems. Knowing this, his brother Rick decided to give him the following riddle to solve.

You are given `N` stones and a number `K`. Your task involves making a division of the `N` stones into heaps of stones such that is satisfies the following properties:

- The sum of the number of stones in all heaps must equal `N`, i.e., no stone can be left unused.
- Each heap must have at least `K` stones in it.

You have to then find the number of such divisions possible.

For example: For the case `N=10` and `K=3` we can divide `N` stones as follows:

```
10 = 3 + 3 + 4

   = 3 + 7

   = 4 + 6

   = 5 + 5

   = 10
```

So the answer in this case would be 5.

Can you help John solve this problem?

Constraints

It is guaranteed that at least 1 division will be possible for each input. Also, `N` >= `K` >=`1`

Input Format

The 1st line contains two space separated integers `N` and `K`.

Output Format:

On a single line, print the number of possible partitions.

1	10 3	5	5	
2	7 4	1	1	

3	15 2	41	41	
4	1 1	1	1	
5	100 100	1	1	

```
#include <stdio.h>

void findDivisions(int i, int n, int *ctr)

{

   if (n == 0){

      (*ctr)++;

   }



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

      findDivisions(j, n - j, ctr);

   }

}

int main(){

   int n, k;

            scanf("%d %d", &n, &k);

   int ctr = 0;

   findDivisions(k, n, &ctr);

   printf("%d", ctr);



   return 0;

}
```

Lab 8

Maximum Fencing Sum

A farmer has a plot in the form of an `n*m` matrix (`n` rows, `m` columns). He has to create a square fenced area within the plot. Every cell of the matrix has been assigned an integer value representing the ease with which one can put a fence-post in that cell. This integer can be either positive or negative. You are tasked with finding the largest square within the matrix such that the sum of the integers on the cells along its perimeter is maximized.

NOTE: A single cell is not considered a square, i.e., the side length of the mentioned squares must be at least 2.

Constraints

- `3 <= n,m <= 50`
- Value in each cell will be in range `[-100,100]`

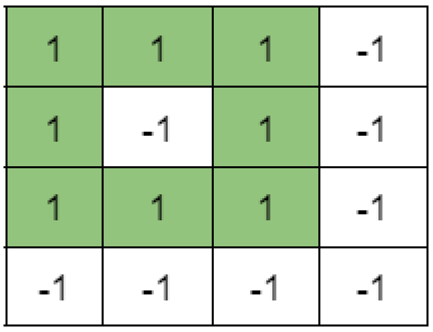
Input Format

- The first line contains 2 integers `n` and `m`.
- The next `n` lines each contain `m` space-separated integers. Each line gives a row of the matrix.

Output Format

Output two space-separated integers `s l`, where `s` is the maximum sum of values along the perimeter of the square within the matrix, and `l` is the side length of the square. In case the same sum occurs for different length squares, then print the `l` for the largest such square.

For example, consider the following matrix. The green-highlighted square of side length 3 has the highest sum on its perimeter (8). So the output for this case will be `8 3`.



1	1	1	-1
1	-1	1	-1
1	1	1	-1
-1	-1	-1	-1

Sample Testcase

Input:

```
4 4

1 1 1 -1

1 -1 1 -1

1 1 1 -1
```

```
-1 -1 -1 -1
```

Output:

```
8 3
```

2	4 3 3 -1 3 -1 3 -1 3 -1 3 -1 3 -1	8 3	8 3	
3	5 5 -5 -5 -5 -5 -5 -5 2 2 2 -5 -5 2 10 2 -5 -5 2 2 2 -5 -5 -5 -5 -5 -5	16 3	16 3	

```
#include <stdio.h>

int main(){

            int n,m;



   scanf("%d %d", &n, &m);

   int a[n][m];



   int mx;

   if(n>m) mx = n;

   else mx = m;



   int maxsum = -401;

   int maxlen = 0;



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

      for(int j=0; j<m; j++){

         scanf("%d", &a[i][j]);

      }

   }



   int sum;
```

```
for(int len=2; len<=mx; len++){
  for(int x = 0; x <= n-len; x++){
    for(int y=0; y <= m-len; y++){
      sum = 0;
      for (int i=x; i<x+len; i++){
        sum += a[i][y];
        sum += a[i][y+len-1];
      }

      for(int j=y; j<y+len; j++){
        sum += a[x][j];
        sum += a[x+len-1][j];
      }

      // Subtracting corner elements (counted twice)
      sum -= a[x][y];
      sum -= a[x][y+len-1];
      sum -= a[x+len-1][y];
      sum -= a[x+len-1][y+len-1];

      if(maxsum <= sum){
        maxsum = sum;
        maxlen = len;
      }
    }
  }
```

```
    }

    printf("%d %d", maxsum, maxlen);

    return 0;
}
```

Pacman

The objective of Pacman is to start from a starting point and reach a destination while collecting the maximum rewards along the way. The possible paths can be seen as a 2D grid, which will be given as a matrix. The entries of the matrix correspond to possible cells where Pacman can reach.
These entries of the matrix contain some numbers. -1 correspond to the cells Pacman can not reach. These are blocked cells and the walk can not go through them. Other entries which are the possible cells, contain some non-negative numbers corresponding to the value of reward given to each cell. Pacman gets this reward when it lands on the corresponding cell. Your task is to output the maximum possible value of reward that can be obtained in reaching from the point corresponding to `(0, 0)` to `(n-1, n-1)`.
The input is given as an `n×n` matrix containing integers. Pacman can not land on cells containing `-1`, while it gets reward `v` if it lands on a cell containing `v`. The rewards along the path are added, including the start and the destination. The top left entry of the matrix is the starting position `(0, 0)`, while the bottom left entry is the destination `(n-1, n-1)`. At one step, Pacman can go either down or right, i.e. from the entry `(i, j)` it can either go to `(i+1, j)` or `(i, j+1)`. However, it can not go beyond the boundaries of the grid. You have to output the maximum possible reward.

Constraints:

- The size of the matrix is `n` where `0 < n < 10`.
- Value of rewards is `v`, where `0 <= v <= 100`.
- Forbidden cells are given as `-1`.

Input:

- First line contains `n`, the size of the matrix.
- Next `n` lines contain `n` space separated numbers each, where each entry corresponds to a value of a reward `v`, or forbidden cell`-1`.

Output:

- Print a single number which corresponds to the maximum value over any walks.

Sample TestCase:

Input

```
3
```

```
0 0 5

3 -1 -1

1 0 0
```

Output:

```
4
```

Explanation:
Pacman starts from the left top cell `(0, 0)`. It can either go right to `(1, 0)` or down to `(0, 1)`, obtaining reward `0` in each case. The path going right ends at `(2, 0)` which has value `5`, but Pacman can not go down (because of `-1` at `(2, 1)`) nor right (because the grid ends). Thus it can only go down to `(1, 0)` obtaining reward `3`, and then again down to `(2, 0)` since right was not a possible move, obtaining a total reward of `3+1=4`. Then it can only go right twice to reach `(2, 2)`. The output is thus the total value of reward, which is `4`.

```
#include <stdio.h>

int max(int a, int b){

   if(a>b){

      return a;

   }

   else return b;

}

int traversal(int A[], int x, int y, int n, int val){

   if(x == (n-1) && y == (n-1)){

      return val + A[x+n*y];

   }

   else if(A[x+n*y] == -1){

      return -1;

   }

   else if(x == n-1){

      return traversal(A, x, y+1, n, val+A[x+n*y]);

   }

   else if(y == n-1){

      return traversal(A, x+1, y, n, val+A[x+n*y]);
```

```
    }
    else{
        if(traversal(A, x+1, y, n, val+A[x+n*y])!= -1 &&  traversal(A, x, y+1, n, val+A[x+n*y])!= -1){
            return max(traversal(A, x+1, y, n, val+A[x+n*y]), traversal(A, x, y+1, n, val+A[x+n*y]));
        }
        else if(traversal(A, x+1, y, n, val+A[x+n*y])!= -1){
            return traversal(A, x+1, y, n, val+A[x+n*y]);
        }
        else if(traversal(A, x, y+1, n, val+A[x+n*y])!= -1){
            return traversal(A, x, y+1, n, val+A[x+n*y]);
        }
        else return -1;
    }
    return -1;
}
int main() {
           int n;
           scanf("%d ", &n);
           int A[n+n*n];
           for(int i  = 0; i < n; i++){
               for(int j = 0; j < n; j++){
                   scanf("%d ", &A[i+n*j]);
               }
           }
    printf("%d", traversal(A, 0, 0, n, 0));
           return 0;
}
```

Neighboring-elements

[70 Points] --- **Automated Grading Scheme**:

- Visible: **5** each for all visible test cases
- Hidden: **10** each for all hidden test cases

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -20% each for using any library other than stdio.h, stdlib.h, limits.h -20% for using any built-in function -50% for using single pointer or 1 dimensional array -50% for static allocation of matrix, example int mat[100][100] or similar declaration ---

Problem Description

Given a square matrix `M` of size `N`. Calculate the sum of all neighboring elements of each diagonal element (up, down, left, right, and 4 diagonal elements -- a total of 8 elements). The neighboring elements for diagonal elements `M[0][0]` and `M[N-1][N-1]` is 3. Print these values corresponding to each diagonal element. Also print the index of that diagonal element whose corresponding sum is highest.(Row and Column index are same for a diagonal element in square matrix)

Constraint:

- **It is mandatory to use array of pointers or double pointer to store the matrix**
- **Header file: stdlib.h is allowed for dynamic memory allocation**
- **The elements of matrix can be negative integer as well**
- **Indexing starts from zero**
- **N is always greater than equal to 2**
- *Use of single pointer or one dimensional array to store matrix element will result in penalty of 50% marks.*
- *Penalty of 50% marks if using static allocation of matrix e.g. int mat[50][50] or int mat[100][100]*

Input:

- First line in each test case will contain the size of square matrix `N`.
- The remaining `N` lines contains the elements of each row in the matrix.

Output:

- The output will contains the `N+1` lines.
- First `N` line will be the sum of all neighboring elements for each diagonal element in the specified format, i.e., diagonal element followed by sum of neighboring elements.
- The last line will contain index of the diagonal element with the highest sum of the neighboring elements.
- In case two diagonal elements having same sum of neighboring elements, print the one with smaller index.

Sample TestCase

Input

```
3

8 2 3

2 3 4

4 5 6
```

Output:

```
8: 7

3: 34

6: 12

index: 1
```

Explanation:
The first `N` lines print sum of all neighboring element for each diagonal element. The sum of neighboring elements is maximum for `M[1][1]` i.e., diagonal element 3 hence index will be 1.

2	2 63 29 42 68	63: 139 68: 134 index: 0	63: 139 68: 134 index: 0	
3	4 -1 -5000 -5000 -5000 -10000 -400 -3000 -4000 -100 -1000 -2000 -3000 -400 -4000 -5000 -6000	-1: -15400 -400: -26101 -2000: -26400 -6000: -10000 index: 3	-1: -15400 -400: -26101 -2000: -26400 -6000: -10000 index: 3	

```
#include <limits.h>

#include <stdio.h>

#include <stdlib.h>



int main(int argc, char *argv[]) {

  int N, index, highest_sum = INT_MIN;

  scanf("%d", &N);

  int *sumArr = (int *)malloc(N * sizeof(int));

  int **mat = (int **)malloc(N * sizeof(int *));

  for (int i = 0; i < N; i++) {

    mat[i] = (int *)malloc(N * sizeof(int));

  }
  for (int i = 0; i < N; i++) {

    for (int j = 0; j < N; j++) {

      scanf("%d", &mat[i][j]);

    }

  }

  // single pointer

  // sumArr[i] =

  // mat[(i-1)*N+(i-1)]+mat[(i-1)*N+i]+mat[(i-1)*N+(i+1)]+mat[i*N+(i-
1)]+mat[i*N+(i+1)]+mat[(i+1)*N+(i-1)]+mat[(i+1)*N+i]+mat[(i+1)*N+(i+1)];

  if (N > 1) {

    sumArr[0] = mat[0][1] + mat[1][0] + mat[1][1];

    index = 0;

    highest_sum = sumArr[0];

    printf("%d: %d\n", mat[0][0], sumArr[0]);

    for (int i = 1; i < N - 1; i++) {

      sumArr[i] = mat[i - 1][i - 1] + mat[i - 1][i] + mat[i - 1][i + 1] + mat[i][i - 1] +

                  mat[i][i + 1] + mat[i + 1][i - 1] + mat[i + 1][i] + mat[i + 1][i + 1];

      printf("%d: %d\n", mat[i][i], sumArr[i]);

      if (sumArr[i] > highest_sum) {

        index = i;

        highest_sum = sumArr[i];

      }

    }

    sumArr[N - 1] = mat[N - 2][N - 2] + mat[N - 2][N - 1] + mat[N - 1][N - 2];

    if (highest_sum < sumArr[N - 1]) {

      index = N - 1;

      highest_sum = sumArr[N - 1];

    }
  } else {

    sumArr[0] = mat[0][0];

    index = 0;

    highest_sum = mat[0][0];

  }

  printf("%d: %d\n", mat[N - 1][N - 1], sumArr[N - 1]);

  printf("index: %d\n", index);

  free(sumArr);

  free(mat);

  return 0;

}
```

Lab 9

Find-Ranks

[15 Points] --- **Automated Grading Scheme**:

Public Test Cases (0.5 point each. 0.5*2 = 1 points) Hidden Test Cases (2 point each. 2*7 = 14 points)

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** Award 0 points if struct is not used. **Header files Allowed:** stdio.h --

You are given `N` points in 2-d plane. It is also given that x and y co-ordinate of each point is positive i.e, all the points lie in the first quadrant. You need to find rank of each point. Rank of a point is defined as the number of points that are dominated by this point. Point `A` dominates another point `B` if point `B` lies on or inside the rectangle drawn such that the upper-right corner is the point `A` and the bottom-left corner is the origin.

NOTE:

1. It is **MANDATORY** to use struct for storing the points otherwise you will be awarded **ZERO** points.
2. Output the ranks in the order as given in the input, in single line separated by space.

Input Format:

The first line contains an integer `N` denoting the total number of points. This is followed by `N` lines each containing `2` integers denoting `x` and `y` coordinates of point respectively.

Ouput Format:

Print `N` space separated integers denoting the rank of each point (**Do not** print trailing spaces).

Sample Input:

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

Sample Output

```
0 1 2
```

Explanation

According to the definition point (1, 1) does not dominate any point so rank = 0.
Point (2, 2) dominates point (1, 1) thus rank = 1.
Point (3, 3) dominates point (1, 1) and (2, 2) thus rank = 2.

1	4 1 1 2 2 3 3 4 4	0 1 2 3	0 1 2 3	
2	4 1 1 2 2 1 2 2 1	0 3 1 1	0 3 1 1	
3	5 7 63 53 55 10 4 16 46 65 58	0 2 0 1 3	0 2 0 1 3	

```
#include<stdio.h>
struct Point{
        int x;
        int y;
};

int calc_rank(struct Point p[], int index, int n)
{
        int rank = 0;
        for(int i = 0; i < n; i = i+1)
        {
                if(i != index && p[i].x <= p[index].x && p[i].y <= p[index].y)
                rank = rank+1;

        }

        return rank;

}


int main(){

        int n;

        scanf("%d", &n);

        struct Point p[n];

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

        {

                scanf("%d %d", &p[i].x, &p[i].y);

        }

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

        {

                printf("%d", calc_rank(p, i, n));

                if(i != n-1)

                        printf(" ");

        }

        return 0;

}
```

Fruit Mart

[15 Points] --- **Automated Grading Scheme**: **Public Test Cases (0.5 point each. 0.5*2 = 1 points) Hidden Test Cases (2 point each. 2*7 = 14 points)**

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** Award 0 points if struct is not used for storing the information of the shopkeepers. **Header files Allowed:** stdio.h ---
There are `n` shopkeepers (indexed from `1` to `n`) who sell apples, bananas, mangoes and oranges at their shops. Each shopkeeper has a fixed quantity of all these fruits and sells them at his own price (per kg). Alice wants to buy some fruits for herself and obviously, she will choose the

shopkeeper who can fulfil her demands at the least cost possible. Find the shopkeeper from which she will buy and the amount she will pay for her purchase.

Input

The first line of the input contains a positive integer `n`

Each of the next `n` lines contain `8` non-negative integers.
Let us say, the numbers on `i`th such line are `a A b B m M o O`.
Here,
`a` = quantity of apples that the `i`th shopkeeper has.
`A` = price charged by the `i`th shopkeeper for 1 kg of apples.
`b` = quantity of bananas that the `i`th shopkeeper has.
`B` = price charged by the `i`th shopkeeper for 1 kg of bananas.
`m` = quantity of mangoes that the `i`th shopkeeper has.
`M` = price charged by the `i`th shopkeeper for 1 kg of mangoes.
`o` = quantity of oranges that the `i`th shopkeeper has.
`O` = price charged by the `i`th shopkeeper for 1 kg of oranges.

The next line contains four non-negative integers, `x y z w`.
Here,
`x` = quantity of apples that Alice wants to buy.
`y` = quantity of bananas that Alice wants to buy.
`z` = quantity of mangoes that Alice wants to buy.
`w` = quantity of oranges that Alice wants to buy.

Output

Print on a single line (space separated) the index number of the shopkeeper from whom Alice buys the fruits (if there are multiple such shopkeepers, print the least index) and the amount she pays for her purchase. Print `-1` if no shopkeeper is able to fulfil her demands.

Note:
Solutions that do not use structs will be awarded **ZERO** points.
It is **MANDATORY** to use struct for storing the information of the shopkeepers.

Examples:

Input 1
```
1
10 10 20 20 30 30 40 40
5 10 15 20
```

Output 1
```
1 1500
```

Explanation 1
There is only `1` shopkeeper (index `1`) and he can fulfil Alice's demands. Alice will have to pay him a total of `5*10 + 10*20 + 15*30 + 20*40 = 1500`

Input 2
```
2
1 2 1 4 1 8 1 16
4 10 3 10 2 10 1 10
1 2 3 4
```

Output 2
```
-1
```

Explanation 2
It is clear that none of the two shopkeepers can fulfil Alice's demands, hence the output is `-1`.

Hidden Test Cases:
In hidden.txt

Grading Scheme: `2*0.5 + 7*2 = 15`

2	2 1 2 1 4 1 8 1 16 4 10 3 10 2 10 1 10 1 2 3 4	-1	-1	
3	6 49 73 58 30 72 44 78 23 9 40 65 92 42 87 3 27 29 40 12 3 69 9 57 60 33 99 78 16 35 97 26 12 67 10 33 79 49 79 21 67 72 93 36 85 45 28 91 94 29 1 53 8	3 2120	3 2120	

```
#include<stdio.h>

struct shop{

        int a;

        int A;

        int b;

        int B;

        int m;

        int M;

        int o;

        int O;

};

int main(){

        int n;

        scanf("%d", &n);

        struct shop arr[n];

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

                scanf("%d %d %d %d %d %d %d %d", &arr[i].a, &arr[i].A, &arr[i].b, &arr[i].B,
&arr[i].m, &arr[i].M, &arr[i].o, &arr[i].O);

        }

        int x, y, z, w;

        int cost = -1;
        int index;

        scanf("%d %d %d %d", &x, &y, &z, &w);

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

                if(arr[i].a>=x && arr[i].b>=y && arr[i].m>=z && arr[i].o>=w){

                        if(cost==-1){

                                cost = arr[i].A*x + arr[i].B*y + arr[i].M*z + arr[i].O*w;

                                index = i+1;

                                continue;

                        }

                        if(cost > arr[i].A*x + arr[i].B*y + arr[i].M*z + arr[i].O*w){

                                cost = arr[i].A*x + arr[i].B*y + arr[i].M*z + arr[i].O*w;

                                index = i+1;

                        }

                }

        }

        if(cost == -1){

                printf("%d", cost);

        }

        else{

                printf("%d %d", index, cost);

        }

        return 0;


}
```

Recover the Rectangle

[25 Points] -- Automated Grading Scheme: **Public Test Cases (2 point each. 2*2 = 4 points) Hidden Test Cases (21 points)** Test Case numbers 1 to 2 are of 2 marks each Test Case numbers 3 to 5 are of 5 marks each Test case number 6 is of 6 marks

Manual Grading Scheme Note: Any form of hard-coding will lead to zero marks. Eg: printf("0"); **Penalty:** 20% penalty if library functions of "stdlib.h" are used **Header Files allowed:** stdio.h --
Mr C had drawn a nice axis-aligned rectangle (i.e. whose sides are parallel either to the `x` or the `y` axis) on a piece of paper and decorated his drawing with a few dots. However, one of his mischievous clones came and erased the lines forming the edges of the rectangle leaving only the dots for the corners behind. Help Mr C recover his nice rectangle.

The first line of the input will give you `n`, a strictly positive number, giving you the number of points on the plane. In the next `n` lines, we will give you the `x` and `y` coordinates of n points on the 2D plane, separated by a space. The coordinates will all be integers. In your output, you have to print the area of the largest axis-aligned rectangle that can be formed out of the n points we have given you. If no axis-aligned rectangle can be formed out of the points we have given you, simply print `-1` in the output.

Caution

Rest assured that we will give you at least `4` points i.e. `n` will be greater than or equal to `4`.
The rectangle we are looking for has non-zero area. Please do not report a single point as a rectangle of area zero.
If there is no axis-aligned rectangle of non-zero area, you should print `-1` as your output.
The rectangle we are looking for must be axis aligned. Do not report a rectangle whose sides are not parallel to the x and y axes.
Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: An axis-aligned rectangle, as we discussed in class, is always uniquely identified using its lower left corner and its upper right corner. You may also want to use a structure to store the points and use an array of these structure variables to process the points given to you.
```
struct Point{
int x,y;
};
struct Point points[n];
```

Example Input

```
9
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Example Output

```
4
```

Explaination the points `(1,1) (1,3) (3,1) (3,3)` form a rectangle of area `4`.

3	5 3 4 3 5 2 6 8 4 8 5	5	5	
4	5 3 4 3 5 2 6 8 4 5 5	-1	-1	

```
#include <stdio.h>

int N;

struct point {

   int x, y;

};

int min(int a, int b){

   return a < b ? a : b;

}

int max(int a, int b){

   return a > b ? a : b;

}


int check(struct point a, struct point P[])

{

   for(int i=0; i<N; i++){

      if(a.x == P[i].x && a.y == P[i].y){

         return 1;

      }

   }

   return 0;

}


int main()

{

   scanf("%d", &N);


   struct point P[N];
```

```
for(int i=0; i<N; i++){
    scanf("%d %d", &(P[i].x), &(P[i].y));
}

int MAX_VAL = -1;
int area = MAX_VAL;

for(int i=0; i<N; i++){
    for(int j=i+1; j<N; j++){

        struct point a, b, c, d;

        a.x = min(P[i].x, P[j].x);
        a.y = min(P[i].y, P[j].y);

        b.x = min(P[i].x, P[j].x);
        b.y = max(P[i].y, P[j].y);

        c.x = max(P[i].x, P[j].x);
        c.y = min(P[i].y, P[j].y);

        d.x = max(P[i].x, P[j].x);
        d.y = max(P[i].y, P[j].y);

        if(check(a, P) && check(b, P) && check(c, P) && check(d, P)){
            int X = max(P[i].x, P[j].x) - min(P[i].x, P[j].x);
            int Y = max(P[i].y, P[j].y) - min(P[i].y, P[j].y);
```

```
                if(X*Y > 0){

                    area = max(area, X*Y);

                }

            }

        }

    }



    /*if(area == MAX_VAL){

        area = -1;

    }*/



    printf("%d\n", area);

}
```

Quiz 1 sec c

Divisor-Game

Automated Grading Scheme:

- Visible: **1** marks for each visible test case.
- Hidden: **6** marks for 1st, 2nd and 3rd test case.
- Hidden: **6.5** marks for 4th test case.
- Hidden: **7.5** marks for 5th, 6th and 7th test case.

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -20% for using any future concepts -20% each for using any library function (string library) other than printf and scanf. So getline(), gets(), fgets(), etc. are not allowed at all. -20% each for using any library other than stdio.h --

Divisor-Game

You are given a character array of N characters where each character belongs to either [A-Z] or [a-z].

You need to perform the following operation for all possible non-empty subarrays of array.

1. Find the number of subarrays in which the characters are arranged in increasing order of ASCII values.
2. Print the largest subarray in which the characters are arranged in increasing order of ASCII values.

Note: If two subarrays have the same length then the subarray which starts first will be selected. For example, in array "ABCABD", though "ABC" and "ABD" are of same length, "ABC" will be printed as it starts first.

Input Format:

The first line contains an integer `N` denoting the size of the array. The next line contains N characters without space denoting the elements of the array.

Ouput Format:

Print the number of sub arrays according to property 1 in first line and largest subarray in second line.

Sample Input:

3
abc

Sample Output

6
abc

Explanation

Our sub-arrays are: [a], [b], [c], [ab], [bc], [abc]
All subarrays are in increasing order of ASCII value. The largest subarray is abc.

1	3 abc	6 abc	6 abc	
2	3 CBA	3 C	3 C	
3	6 apqefg	12 apq	12 apq	
4	4 dbcd	7 bcd	7 bcd	

```
#include <stdio.h>

int main(){

    int n;

    scanf("%d", &n);
```

```
char a[n+1];

scanf("%s", a);

int ans = 0;

int val = 0;

int len=0,x=-1,y=-1;


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

    val= 1;

    for(int k=i+1;k<n;k++){

        if (a[k]<=a[k-1]) break;

        val++;

    }

    ans+=val;

    if(val>len){

        len=val;

        x=i;

        y=i+val-1;

    }

}


char b[len+1];

for (int i=x;i<=y;i++){

    b[i-x]=a[i];

}

b[len]='\0';

printf("%d\n%s", ans,b);

return 0;
```

}

Uppercase String

Automated Grading Scheme:

- Visible: **2.5** marks for each visible test case.
- Hidden: **2.5** marks for 1st, 2nd, 3rd and 4th test case.
- Hidden: **5** marks for 5th and 6th test case.

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -20% for using any future concepts -20% each for using any library function (string library) other than printf and scanf. So getline(), gets(), fgets(), etc. are not allowed at all. -20% each for using any library other than stdio.h **Important Note** If '\0' (null character) is not appended to end of string, the output might contain garbage value across multiple run. Try executing solution multiple times to see if no garbage value is printed as output.

Given a character array `S`. The maximum length of array is 100. The words in the string are separated by a single whitespace. The end of string is marked by `newline character(\n)`. Print the string after converting each alphabet in alternate word of string to `UPPERCASE`.

Constraint:

- **The digit and special characters in the character array remain unchanged.**
- **Make sure, the output string does not contain garbage values.**

Input:

- Each test case comprise of single line of input.
- The only line in each test case will contain the character string.
- The string will consist of uppercase, lowercase alphabets, digits and special character(@ , \ ? !) .

Output:

- The output will contains the modified string followed by a newline.

Sample TestCase :

Input

```
Hi, how are you?
```

Output:

```
HI, how ARE you?
```

Explanation:
The first and third word of the string are converted to uppercase. The special characters, second and fourth word of the string remains unmodified.

Sample TestCase :

Input

```
Hi, I AM Here.
```

Output:

```
HI, I AM Here.
```

Explanation:
The first word of the string is converted to uppercase. The third word was already in uppercase. The special characters, second and fourth word of the string remains unmodified. The uppercase alphabet in second and third word remains unchanged.

```
#include <stdio.h>

int main() {

  char str[100];

  char curr;

  int str_length = 0, index = 0;

  int word_count = 0;

  scanf("%c", &curr);

  while (curr != '\n') {

    str[str_length] = curr;

    str_length += 1;

    scanf("%c", &curr);

  }

  str[str_length] = '\0';



  // modify the given string

  for (int i = 0; i < str_length; i++) {

    if (str[i] == ' ') {

      word_count++;

      continue;

    } else if (str[i] >= 'a' && str[i] <= 'z') {

      if (word_count % 2 == 0) { // only alternate word character needs to be converted to uppercase

        str[i] = str[i] - 32;
    }
   }
  }

  printf("%s", str);
  printf("\n");
  return 0;
}
```

Quiz 1 sec d

The Hidden Key

This semester, Rohan took a course on cryptography. Being lazy, he was not able to complete his assignment on time so now he needs your help in completing his assignment on time. The assignment is as follows.
You will be given an input consisting of three lines.
In the first line, you will be given a positive integer `n` which will denote the size of message. In the next line, you will be given `n` space separated integers denoting the original message. The third line will also contain `n` space separated integers denoting the encrypted message.
The encrypted message is obtained by adding a secret key to the original message, which itself is a message of length `k` (you do not know `k`). Since the message length may be longer than the length of the key, the key is repeated as many times as required.
For example, if the plain message is [1 2 3 4 5] and the key is [1 2], then we first repeat the key till we obtain 5 elements as [1 2 1 2 1]. Note that we omitted 2 since we require only 5 integers. We now add the corresponding elements of these two messages (original message and repeated key message) to get the encrypted message. So the encrypted message in this case will be [2 4 4 6 6].

Given the original and encrypted message, you need to find the secret key used to encrypt the message. If there is no such key, then print the message `No Such Key!`. If there exists a key, then you need to output 2 lines. The first line should contain `k` which is the size of the secret key and the second line should contain `k` space separated integers denoting the secret key (output should not contain trailing spaces). If there are more than one possible secret keys, then output the secret key of the smallest length.

Note:

1. Secret keys may be of any non-negative length.
2. As the example below indicates, the integers in the message as well as in the keys, may be negative or even zero.
3. Be careful about extra/missing lines and spaces in your output.
4. It is guaranteed that the message length will always be < 100.

Input Format:

The first line contains an integer `n` denoting size of the message.
The second line contains `n` space separated integers denoting the original message.
The third line contains `n` space separated integers denoting the encrypted message.

Ouput Format:

Print two lines. First line should contain `k` and the second line should contain `k` space separated integers.

Sample Input:

```
2
31 43
31 43
```

Sample Output

```
1
0
```

Explanation

The original and encrypted messages are same. The smallest possible key is of length 1. So the secret key is [0]. Note that [0 0] is also a secret key here but we need to output the key of smallest length.

Sample Input:

```
4
1 1 5 6
2 3 6 8
```

Sample Output

```
2
1 2
```

Explanation

Note that there also exists a key of length 4 here but we need to output only smallest length secret key.

```
#include <stdio.h>

int main(){

   int n;

   scanf("%d", &n);

   int orig_msg[100], enc_msg[100];
```

```
for(int i = 0; i < n; i = i+1)
    scanf("%d", &orig_msg[i]);
for(int i = 0; i < n; i = i+1)
    scanf("%d", &enc_msg[i]);

// Generate the repeated key
int repeated_key[100];
for(int i = 0; i < n; i = i+1)
    repeated_key[i] = enc_msg[i]-orig_msg[i];

//Check for all possible key lengths
// Observe that there always exists a key of length n
int key_len = 0;
for(key_len = 1; key_len <= n; key_len = key_len+1){
    int possible_flag = 1;
    for(int i = 0; i < n; i=i+1){
        if(repeated_key[i] != repeated_key[i%key_len]){
            possible_flag = 0;
            break;
        }
    }
    if(possible_flag == 1)
        break;
}

printf("%d\n", key_len);
for(int i = 0; i < key_len; i = i+1){
        printf("%d", repeated_key[i]);

        if(i != key_len-1)

            printf(" ");

    }

    return 0;

}
```

Binary and Primes

Automated Grading Scheme:

- Visible: **5** marks for each Visible test case.
- Hidden: **5** marks for 1st, 2nd and 3rd Hidden test case.
- Hidden: **10** marks for 4th Hidden test case.

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -50% for changing template -20% for using any future concepts -20% for using built in function ---

You are given an integer `n`. Consider the binary expansion of `n` denoted by the number `s`, i.e., `s` represents a number in base 2. Let `N` denote the decimal equivalent of the same string in the decimal system. Find the smallest prime number which is just greater than or equal to `N`.
For example, if `n = 7`, the binary expansion of `n` is 111_2 where `s = 111`. Now, the decimal number represented by `s` is `N = 111`, which is "one hundred and eleven". The prime just larger than or equal to `111` will be `113`, which is the correct output.

Note:

1. Stick to the template provided to define the important functions.
2. You are not allowed to use library functions.
3. You can create other user-defined functions.
4. You are required to define the three functions given in the template. Do not change the names of the functions of the template.

Input Format:

The first line contains an integer `n`.

Constraints:

1. `2 <= n <= 50`

Output Format:

The output should contain an integer `n` which will give the correct answer.

Sample Input:

5

Sample Output:

101

1	2	11	11	
2	11	1013	1013	
3	17	10007	10007	

```
#include <stdio.h>

int pow(int a, int b){

    int p = 1;

    for(int i = 0; i < b; i++){

        p *= a;

    }

    return p;

}

int decimal_to_binary(int n){

    int N = 0, idx = 0;

    while(n > 0){

        N += (n%2)*(pow(10, idx));

        idx++;

        n = n/2;

    }

    return N;

}
```

```
int is_prime(int p){
   for(int j = 2; j < p; j++){
      if(p%j == 0){
         return 0;
      }
   }
   return 1;
}
int closest_prime(int N){
   while(!is_prime(N)){
      N++;
   }
   return N;
}
int main() {
           int n;
           scanf("%d", &n);
           printf("%d", closest_prime(decimal_to_binary(n)));
           return 0;
}
```

Lab 10

Delete the number

Write a program that takes a linked list and a number `n` as input and deletes the n th last node (nth node from the end) of the list. For this problem, `n=1` means "delete the last node," `n=2` means "delete the second last node," and so on. If `n` is greater than the length of the list (denoted by `len`), then modulus operation is performed on the absolute value of `n` by `len`, i.e., (n-1)/%len + 1 th is used.

Each node of a linked list must have the following structure.

```
struct node {
    int data;

    struct node* next;

};
```

NOTE

- Use of ARRAYS IS NOT PERMITTED. You will get marks if you use linked lists to solve the problem.
- The template contains initial code for list manipulation. You can modify it or use it as it is. DO NOT change the `print` function.
- No marks if you *do not really delete* the node. The print function provided must work as-it-is to print the modified list.

INPUT

The input will consist of two lines containing the numbers. The first line will contain a stream of numbers. Keep accepting the input until you get a -1. The second line will contain a number nn, `n` will fit in an `int`.

OUTPUT

Display the final linked list, with nthnth last node deleted from the linked list. You must use the provided `print` routine. It prints an `X` at the end to mark the end of the list.

Examples

INPUT

1 2 3 4 5 -1
1

OUTPUT

1 2 3 4 X

```
#include <stdio.h>

#include <stdlib.h>



struct node {

   int data;

   struct node* next;

};
```

```
struct node* insert(struct node* head, int data)
{
   struct node* n=(struct node*)malloc(sizeof(struct node));
   n->next=NULL;
   n->data=data;
   if(head==NULL)
      return n;
   struct node* tmp=head;
   while(tmp->next!=NULL)
      tmp=tmp->next;
   tmp->next=n;
   return head;
}
void print(struct node* head)
{
   while(head!=NULL)
   {
      printf("%d ",head->data);
      head=head->next;
   }
   printf("X\n");
   return;
}

struct node* deleteNthLast(struct node* head, int n)
{
```

```
if(head==NULL)
   return head;
struct node* offset=head;
int k=1;
while(offset!=NULL)
{
   if(k==n)
      break;
   offset=offset->next;
   k++;
}
if(offset==NULL)
   return head;
struct node* tmp=head;
struct node* parent=NULL;
while(offset->next!=NULL)
{
   parent=tmp;
   tmp=tmp->next;
   offset=offset->next;
}
if(parent==NULL)
{
   head=tmp->next;
   return head;
}
else
```

```
    {
        parent->next=tmp->next;
        return head;
    }
}

int main()
{
    struct node* head=NULL;
    int len = 0;
    while(1)
    {
        int x;
        scanf("%d", &x);
        if(x==-1)
            break;
        len++;
        head=insert(head,x);
    }
    int n;
    scanf("%d", &n);
    head = deleteNthLast(head,(n-1)%len+1);
    print(head);
    return 0;
}
```

Find Your Identity

In the first line of your input, you will be given two strictly positive integers `n` and `m`. In the next `n` lines, you will be given the `n` rows of an `n x m` matrix `A`, with each row on a separate line and two elements in a row separated by a single space. The matrix `A` will contain entries that are either `0` or `1`. In the first line of your output, print the size of the largest identity submatrix. In the next line, print the row index and column index of the top-left element of the largest identity submatrix in the format `(rowIdx,colIdx)`. Note that there are no spaces in the output. In case of multiple identity submatrices of the same largest size, print the `(rowIdx,colIdx)` of the one with the largest row number. If two matrices of the largest size have the same largest row number print the one with larger column number.

If there is no identity submatrix, print `0` in the first line of your output and print `(-1,-1)` in the second line of your output. Use zero indexing for row and column indexing.

Identity Matrix : A square matrix in which all the elements of the principal diagonal are ones and all other elements are zeros. Submatrix : A smaller matrix inside a given matrix made by fixing the left top corner and the bottom right corner.

INPUT:

The input consists of `n + 1` lines.The first line contains two integers `n` and `m`.The following `n` lines each contain a row of matrix `A`.

OUTPUT:

The output should contains two lines. The first line contains the size of largest submatrix and the next line contains `(rowIdx,colIdx)`.

CONSTRAINTS:

`1 <= n*m <= 100`

Example Input

3 4 1 1 1 1 1 1 0 1 1 0 1 0

Example Output

2 (1,1)

2	5 8 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1	2 (3,1)	2 (3,1)	
3	4 7 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 1 1 0 0 0 1 1 0 1	1 (3,6)	1 (3,6)	

```
#include <stdio.h>

#include <stdlib.h>



int min(int a, int b){

    return a < b ? a : b;
```

```
}


int isIdentity(int **arr, int ox, int oy, int size){
    int i, j, isIden = 1;
    for(i = 0; i < size; i++){
        for(j = 0; j < size; j++){
            if(i == j){
                if(arr[ox + i][oy + j] != 1)
                    isIden = 0;
            }else{
                if(arr[ox + i][oy + j] != 0)
                    isIden = 0;
            }
        }
    }
    return isIden;
}


int main() {
    int n, m, i, j, k, kMax;
    int size = 0, idx1 = -1, idx2 = -1;
    scanf("%d %d", &n, &m);
    int **arr = (int**)malloc(n * sizeof(int*));


    for(i = 0; i < n; i++){
        arr[i] = (int*)malloc(m * sizeof(int));
        for(j = 0; j < m; j++)
```

```
            scanf("%d", &arr[i][j]);
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < m; j++){
            // How far are the boundaries of the matrix
            kMax = min(n - i, m - j);
            for(k = 0; k < kMax; k++){
                // Is the k x k matrix starting at (i,j) an identity?
                if(isIdentity(arr, i, j, k+1)){
                    if(k + 1 >= size){
                        size = k + 1;
                        idx1 = i;
                        idx2 = j;
                    }
                }
            }
        }
    }

    if(size)
        printf("%d\n(%d,%d)", size, idx1, idx2);
    else
        printf("0\n(-1,-1)");

    return 0;
}
```

Reverse Alternate Segments in a Doubly-Linked List

Given a doubly-linked list (say of `n` nodes where `n > 0`) and a positive integer `k`, rearrange the links between the nodes of the linked list in the following way. The nodes of the linked list are divided into segments of `k` nodes. If `k` does not divide `n`, then the final segment is of `n % k` nodes. The order of nodes in alternate segments (1st, 3rd, 5th, ...) are to be reversed. An example is described below.

Initial linked list:
`1` ↔ `2` ↔ `3` ↔ `4` ↔ `5` ↔ `6` ↔ `7` ↔ `8` ↔ `9` ↔ `0`

Final linked list after rearrangement with `k = 3`:
`3` ↔ `2` ↔ `1` ↔ `4` ↔ `5` ↔ `6` ↔ `9` ↔ `8` ↔ `7` ↔ `0`

Note:

- The nodes in the linked list are of the following structure.

```
struct node {
  int data;
  struct node* next;
  struct node* prev;
};
```

- No marks will be given if doubly-linked list is not used.
- Arrays are prohibited. No marks will be given if arrays are used.
- Make sure that you rearrange the links in the list. A penalty will be imposed if the `data` field of any node is changed at any point after taking input.
- Do not modify the `print()` function given in the template. Use this function to print the output by providing as an argument, the pointer to the head node of the final doubly-linked list. Do not use any other `printf()` statement in the program.
- Make sure that the `prev` field of the head and the `next` field of the tail take `NULL` values.
- It is guaranteed that the linked list given is not empty. Therefore, the head passed to `print()` should not be `NULL`.

Input Format:

The input consists of two lines. The first line is a stream of integers that ends with -1 (not to be included), representing the linked list. It is guaranteed that the linked list contains at least one node. The second line is a single positive integer representing `k`.

Output Format:

The output consists of two lines. The first line is a stream of integers separated by spaces representing the final linked list after transformation. The second line is also a stream of integers that represents the same final linked list but in reverse order.
You will only need to call the `print()` function defined in the template with the pointer to the head of the final doubly-linked list as an argument. This function prints both lines in the required format.

Example

Input:

```
1 2 3 4 5 6 7 8 9 0 -1

3
```

Output:

```
3 2 1 4 5 6 9 8 7 0

0 7 8 9 6 5 4 1 2 3
```

Explanation:
Segment 1: `1 2 3` → `3 2 1`
Segment 2: `4 5 6` → `4 5 6`
Segment 3: `7 8 9` → `9 8 7`
Segment 4: `0` → `0`

```
#include <stdio.h>

#include <stdlib.h>



struct node {

    int data;

    struct node* next;

    struct node* prev;

};



void print(struct node* head) {

    if(head == NULL) {

        printf("ERROR - Head is never NULL");

        return;

    }

    while(head->next != NULL) {

        printf("%d ", head->data);

        head = head->next;

    }

    printf("%d\n", head->data);
```

```
  while(head->prev != NULL) {

    printf("%d ", head->data);

    head = head->prev;

  }

  printf("%d", head->data);

  return;

}


void insert(struct node* head, int data) {

  struct node* n = (struct node*)malloc(sizeof(struct node));

  n->next = NULL;

  n->prev = NULL;

  n->data = data;

  struct node* tmp = head;

  while(tmp->next != NULL) tmp = tmp->next;

  tmp->next = n;

  n->prev = tmp;

}


struct node* reverse(struct node* head, int k) {

  struct node* curr = head;

  struct node* prev = NULL;

  struct node* next = NULL;

  struct node* p = (head)? head->prev: NULL;

  int count = 0;

  while(count < k && curr != NULL) {

    next = curr->next;
```

```
        curr->next = prev;

        if(prev != NULL) prev->prev = curr;

        if(next != NULL) next->prev = NULL;

        prev = curr;

        curr = next;

        count++;

    }

    if(head != NULL) {

        head->next = curr;

        if(curr != NULL) curr->prev = head;

        prev->prev = p;

    }

    count = 0;

    while(count < k - 1 && curr != NULL) {

        curr = curr->next;

        count++;

    }

    if(curr != NULL) curr->next = reverse(curr->next, k);

    return prev;

}


int main() {

    int k, x;

    struct node* head = (struct node*)malloc(sizeof(struct node));

    scanf("%d", &x);

    head->next = NULL;

    head->prev = NULL;
```

```
  head->data = x;

  while(1) {

    scanf("%d", &x);

    if(x == -1) break;

    insert(head, x);

  }

  scanf("%d", &k);

  head = reverse(head, k);

  print(head);

  return 0;

}
```

Quiz 2 sec c

Unique Elements

[35 Points] -- **Automated Grading Scheme**:

- Visible: **5** each for all visible test cases
- Hidden: **5** each for all hidden test cases

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -20% each for using any library other than stdio.h, stdlib.h, -20% for using any built-in function other than malloc -50% for static allocation of matrix or array, example int mat[100][100] or similar declaration ---
Read a square matrix of size 'N'. Do the following task: Print all unique elements in the matrix. If there are no unique elements, print `No unique elements.`

Input:

- First line contain M size of matrix
- The input will span across M line.
- Each line will contain space separated M integers.

Output:

- A single line with unique (non-repetitive) elements in the matrix with space (Take care of space at the last)

Sample TestCase:

- Input

```
4
5 1 2 1
4 10 3 5
2 1 4 4
1 2 0 7
```

- Output:

```
10 3 0 7
```

Explanation:
Only elements 10,3,0 and 7 does not repeat in the matrix.

```
#include <stdio.h>
#include <stdlib.h>


void unique_elements(int **mat, int mat_size) {
  // complete your code.
  int i = 0, j = 0, k = 0;
  int element_count = 0;


  int **unique_arr = (int **)malloc(2 * sizeof(int *));
  for (i = 0; i < 2; i++) {
    unique_arr[i] = (int *)malloc((mat_size * mat_size) * sizeof(int));
  }
  for (i = 0; i < mat_size; i++) {
    for (j = 0; j < mat_size; j++) {
      unique_arr[0][i * mat_size + j] = mat[i][j];
      unique_arr[1][i * mat_size + j] = 0;
    }
```

```
 }

 for (i = 0; i < mat_size; i++) {
  for (j = 0; j < mat_size; j++) {
   for (k = 0; k < mat_size * mat_size; k++) {
    if (mat[i][j] == unique_arr[0][k]) {
     unique_arr[1][k] += 1;
     break;
    }
   }
  }
 }
 int flag=0;
 for (k = 0; (k < mat_size * mat_size && (element_count < mat_size * mat_size)); k++) {
  element_count += unique_arr[1][k];
  if (unique_arr[1][k] == 1) {
   printf("%d ", unique_arr[0][k]);
   flag=1;
  }
 }
 if(flag==0)
  printf("No unique elements.");
}

int main() {
 int N;
 int **arr;
 int i = 0, j = 0;

 scanf("%d", &N);

 arr = (int **)malloc(N * sizeof(int *));


 for (; i < N; i++) {

   arr[i] = (int *)malloc(N * sizeof(int));

 }


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

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

     scanf("%d", &arr[i][j]);

   }

 }

 // Find unique values:

 unique_elements(arr, N);


 return 0;

}
```

Is there a way

[40 Points] -- **Automated Grading Scheme**:

- Visible: **2** each for all visible test cases
- Hidden: **6** each for all hidden test cases

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -20% each for using any library other than stdio.h -25% for using global variables **Note: Use of recursion is must. Otherwise, 0 marks are rewarded.** --
You are given an `N x N` grid. You need to find if there exists a path from the cell `(x_1, y_1)` to the cell `(x_2, y_2)`.
If you are at `(x, y)` and the value at this cell of the grid is `a` then you can either move to `(x+a, y)` or to `(x, y+a)`, provided that you stay inside the grid.

Input
First line of the input contains the number `N`.

On each of the next `N`, lines there will be `N` entries where the `j`th entry on the `i`th line cooresponds to the value at the cell `(i, j)` of the grid. All these entries will be positive.
The next line contains 4 integers, `x_1, y_1, x_2` and `y_2`. All these entries are greater than or equal to `1` and less than or equal to `N`.

Output
If there exists a path from `(x_1, y_1)` to `(x_2, y_2)` print `YES`, otherwise print `NO`.

Note:
The numbering of the cells starts from `(1,1)` and goes on to `(N, N)`.

Examples:

Input 1

```
2
1 2
1 100
1 1 2 2
```

Output 1

```
YES
```

Explanation 1
From `(1, 1)` go to `(1 + 1, 1)` (as `a = 1` for `(1, 1)`) and from `(2, 1)` go to `(2, 1 + 1)` (as `a = 1` for `(2, 1)`).

Input 2

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

Output 2

```
NO
```

Explanation 2
We can move from `(1, 4)` to `(1 + 4, 4)` or `(1, 4 + 4)` and both these points take us out of the grid, so it is not possible to reach `(4, 4)` from `(1, 4)`.

```
#include<stdio.h>


int helper(int n, int grid[][n], int x1, int y1, int x2, int y2){

        int next_x = x1 + grid[x1-1][y1-1];

        int next_y = y1;

        if(next_x == x2 && next_y == y2){

                return 1;

        }

        int ans = 0;

        if(next_x <= n && next_y <= n){
            ans = helper(n, grid, next_x, next_y, x2, y2);
        }
        next_x = x1;
        next_y = y1+ grid[x1-1][y1-1];
        if(next_x == x2 && next_y == y2){
            return 1;
        }
        if(next_x <= n && next_y <= n){
            ans = (ans || helper(n, grid, next_x, next_y, x2, y2));
        }
        return ans;
}
int main(){
        int n;
        scanf("%d", &n);
        int grid[n][n];
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                scanf("%d", &grid[i][j]);
            }
        }
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        if(helper(n, grid, x1, y1, x2, y2)){
            printf("YES");
        }
        else{
                printf("NO");
        }
        return 0;
}
```

Sec d quiz 2

Fishes

[35 Points] --- **Automated Grading Scheme**:

- Visible: **1 marks** for 1st testcase and **2 marks** each for 2nd and 3rd test cases
- Hidden: **5** each for all hidden test cases

Manually Grading Scheme: **Note:** Give 0 marks for test-case component if there is any form of hard-coding. Eg: printf("0"); **Penalty :** -20% each for using any library other than stdio.h, -25% for using global varibles ---
One day Bob went for fishing in a pond of size `N x N` with a net of size `k x k`. The pond is divided into cells of size `1 x 1`. He knows the number of fishes in each cell. To collect fishes, Bob can throw his net only once. Bob wants to collect as many fishes as possible given the constraint. Where should Bob throw his net so that he collects maximum number of fishes? You need to output two integers denoting the row number and column number of the cell which should be the top left cell of the net where Bob will throw his net to collect maximum number of fishes.

NOTE:

1. Assume 1-based indexing i.e, rows and columns start from 1.
2. If there are more than one possible configurations of the net to get maximum fishes then output the configuration having the lowest row number.
3. If there are more than one possible configurations of the net to get maximum fishes having same row number then output the configuration having the lowest column number.
4. Assume that Bob cannot cut the net in parts and also the net should not go out of the boundary of the pond.

Input Format:

The first line contains two integers `N` and `k` separated by space. This is followed by `N` lines each containing `N` integers denoting number of fishes in the correspoding cell.

Constraints:

1. Assume `k <= N`.
2. `N <= 10`.
3. Maximum number of fishes in one cell is `1000`.

Ouput Format:

Print two integers separated by space denoting the row and column number of the cell.

Sample Input:

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

Sample Output

```
2 1
```

Explanation

Bob can collect maximum 10 fishes using his `2x2` net. There can be 2 possible top-left positions to get 10 fishes -> (2, 1) and (2, 2). But as mentioned in the `Note`, you have to output `2 1` as the answer.

1	2 1 7 7 1 3	1 1	1 1	
2	2 1 7 2 7 3	1 1	1 1	
3	3 2 7 8 9 2 5 3 3 9 4	1 2	1 2	

```
#include <stdio.h>

int main(){

   int n, k;

   scanf("%d %d", &n, &k);

   int fishes[10][10];

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

      for(int j = 0; j < n; j=j+1){

         scanf("%d", &fishes[i][j]);

      }

   }

   int max_sum = -1;

   int x = -1, y = -1;
```

```
    for(int row_num = 0; row_num < n-k+1; row_num=row_num+1){

        for(int col_num = 0; col_num < n-k+1; col_num=col_num+1){

            int curr_sum = 0;

            for(int i = 0; i < k; i=i+1){

                for(int j = 0; j < k; j=j+1){

                    curr_sum = curr_sum+fishes[row_num+i][col_num+j];

                }

            }

            if(curr_sum > max_sum){

                max_sum = curr_sum;

                x = row_num;

                y = col_num;

            }

        }

    }

    printf("%d %d", x+1, y+1);

    return 0;

}
```

Welcome Modiji

As you know PM Modi visited IIT Kanpur for convocation and your recently graduated seniors were getting ready to witness the event. They have been told though that because of the rowdy crowd and zigzagging queues to get into the Auditorium, it will be extremely hard to walk straight and reach the venue in time. Thus, you may have to take a few turns along the way to optimize the time in which you reach the place.

The part of the campus from your hostel to the Auditorium is represented by a 2D square matrix, where you are initially at (0, 0) and the venue is at (N-1, N-1). Each of the cells (i, j) in between has some integer points P(i, j) attached to it. P(i, j) indicates how thin the crowd is at that location, and thus the ease of travel through that cell. You have to reach (N-1, N-1) from (0, 0), while collecting the maximum number of points. You are only allowed to move down or right, that is, from (i, j), you can move to (i+1, j) or (i, j+1) only.

Find out the maximum number of points that you collect along the way.

Note: Please don't alter the template given in the question. Using dynamic memory allocation is a must.

Hint: Think recursively: max points from a cell (i, j) = points at cell (i, j) + max(max from (i+1, j), max from (i, j+1))

Constraints: N will be >=1 and P[i][j] will be >=0 for each i,j

INPUT:

The first line contains an integer N, the dimensions of the 2D array. The next N lines contain N space separated integers, denoting the array P.

OUTPUT:

One line containing a single integer, the maximum number of points that can be collected along the way.

EXAMPLE:

Input:

```
3
1 2 3
0 7 8
1 2 3
```

Output:

```
21
```

Explanation: (0, 0) -> (0, 1) -> (1, 1) -> (1, 2) -> (2, 2) gets you 1 + 2 + 7 + 8 + 3 = 21 points.

1	`3 1 2 3 0 7 8 1 2 3`	`21`	`21`	
2	`2 5 5 5 5`	`15`	`15`	
3	`2 5 5 6 5`	`16`	`16`	

4	3 1 2 3 2 2 2 3 2 1	9	9	

```
#include <stdio.h>

#include <stdlib.h>


int get_max_points(int **P, int nrow, int ncol, int N) {

 // function to compute the maximum number of points thus obtainable

 if (nrow == N-1 && ncol == N-1) {

  return P[nrow][ncol];

 }


 if (nrow == N-1) {

  return P[nrow][ncol] + get_max_points(P, nrow, ncol+1, N);

 }


 if (ncol == N-1) {

  return P[nrow][ncol] + get_max_points(P, nrow+1, ncol, N);

 }


 int right_points = get_max_points(P, nrow, ncol+1, N);

 int down_points = get_max_points(P, nrow+1, ncol, N);

 int max_points = right_points > down_points ? right_points : down_points;

 return P[nrow][ncol] + max_points;

}


int main() {

 int N, i, j;

 scanf("%d", &N);
 // create two dimensional points array

 int **P = (int**) malloc(N * sizeof(int*));

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

  P[i] = (int*) malloc(N * sizeof(int));

 }


 // take input for P

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

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

   scanf("%d", &P[i][j]);

  }

 }


 int ans = get_max_points(P, 0, 0, N);

 printf("%d\n", ans);

 return 0;

}
```