
Godavari College Of Engineering, Jalgaon.



**ACADEMIC YEAR
2020 - 2021**

VI SEMESTER

**LAB MANUAL
COMPETITIVE PROGRAMMING-II
[BTCOC606]**

FACULTY : Mr. RAHUL GAIKWAD

**DEPARTMENT OF
COMPUTER ENGINEERING**

**Godavari Foundation's
GODAVARI COLLEGE OF ENGINEERING, JALGAON
(NAAC Accredited)**

(An affiliated to Dr. Babasaheb Ambedkar Technological University)



CERTIFICATE

This is to certify that Miss. **Shweta Ravindra Patil** , Roll No: **34** , PRN No: **1951711245011**, of **T.Y. COMPUTER** class has satisfactorily carried out the practical work in the Subject : **Competitive Programming-II [BTCOC606]** as per laid down in the syllabus, in this Laboratory and that this journal represent **her** bonafide work in the year **2020-2021**.

Date :

Signature
Mr. RAHUL GAIKWAD
Faculty in Charge

Signature
Mr. PRAMOD B. GOSAVI
H.O.D.

Dr. V. H. PATIL
PRINCIPAL
Godavari Foundation's
Godavari College of Engineering, Jalgaon

Index

Sr. No	Practical Name	Page No.	Dates
1	Write Program To Find Prime Numbers.	1 - 2	06-04-2021
2	Write Program To Find GCD Of Given Number .	3 - 4	06-04-2021
3	Write Program To Find Modulous Of Two Numbers.	5 - 6	13-04-2021
4	Write Program Of Diophantine Equations.	7 - 9	20-04-2021
5	Write Program To Solve 15-Puzzle Problem.	10 - 12	27-04-2021
6	Write Program To Solve Tug of War Problem.	13 – 15	04-05-2021
7	Write Program To Solve Tower of Cubes Problem.	16 – 20	11-05-2021
8	Write Program To Solve Tower Of Hanoi.	21 – 22	18-05-2021
9	Write Program To Solve Number of Stopping Station Problem.	23 – 24	25-05-2021
10	Write Program To Solve Is Bigger Smarter Problem.	25 – 27	01-06-2021
11	Write Program To Solve Unidirectional TSP Problem.	28 - 31	08-06-2021

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 1

Date : 06-04-2021

Title : Write Program To Find Prime Numbers.

Aim : Implement Program To Find Prime Numbers.

Theory:

Prime Number : A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. The first few prime numbers are {2, 3, 5, 7, 11,}

The idea to solve this problem is to iterate through all the numbers starting from 2 to \sqrt{N} using a for loop and for every number check if it divides N. If we find any number that divides, we return false. If we did not find any number between 2 and \sqrt{N} which divides N then it means that N is prime and we will return True.

Why did we choose \sqrt{N} ?

The reason is that the smallest and greater than one factor of a number cannot be more than the \sqrt{N} of N. And we stop as soon as we find a factor. For example, if N is 49, the smallest factor is 7. For 15, smallest factor is 3. Below is the C program to check if a number is prime:

Program:-

```
// C program to check if a
// number is prime

#include <math.h>
#include <stdio.h>
int main()
{
    int n, i, flag = 1;

    // Ask user for input
    printf("Enter a number: \n");

    // Store input number in a variable
    scanf("%d", &n);
```

```

// Iterate from 2 to sqrt(n)
for (i = 2; i <= sqrt(n); i++) {

    // If n is divisible by any number between
    // 2 and n/2, it is not prime
    if (n % i == 0) {
        flag = 0;
        break;
    }
}

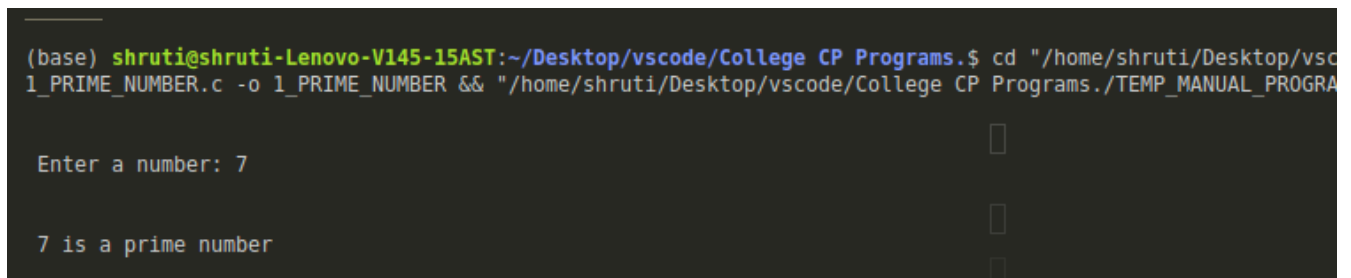
if (n <= 1)
    flag = 0;

if (flag == 1) {
    printf("%d is a prime number", n);
}
else {
    printf("%d is not a prime number", n);
}

return 0;
}

```

Output:



```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs.$ cd "/home/shruti/Desktop/vscode/College CP Programs."
1_PRIME_NUMBER.c -o 1_PRIME_NUMBER && "/home/shruti/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRA

Enter a number: 7

7 is a prime number

```

Conclusion:- In this Practical i learned to implement prime number program.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 2

Date : 06-04-2021

Title : Write Program To Find GCD Of Given Number.

Aim : To Implement Program To Find GCD Of Given Number.

Theory:

GCD Of Two Numbers : It is the highest number that completely divides two or more numbers. It is abbreviated for GCD. It is also known as the Greatest Common Factor (GCF) and the Highest Common Factor (HCF). It is used to simplify the fractions. The HCF or GCD of two integers is the largest integer that can exactly divide both numbers (without a remainder).

How to Find the Greatest Common Factor

- Write all the factors of each number.
- Select the common factors.
- Select the greatest number, as GCF.

Example: Find the GCF of 12 and 8.

Solution:

Factors of 12: 1, 2, 3, 4, 6, 12

Factors of 8: 1, 2, 4, 8

Common Factors: 1, 2, 4

Greatest Common Factor: 4

Program:-

```
#include <stdio.h>
int main()
{
    int n1, n2, i, gcd;
```

```

printf("Enter two integers: ");
scanf("%d %d", &n1, &n2);

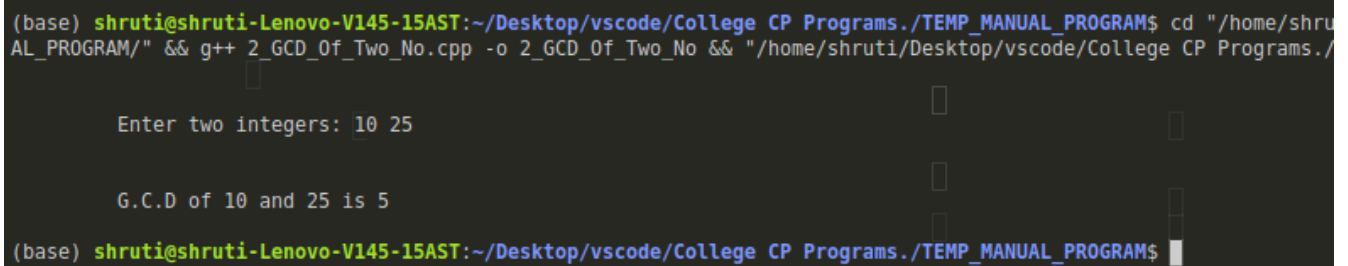
for(i=1; i <= n1 && i <= n2; ++i)
{
    // Checks if i is factor of both integers
    if(n1%i==0 && n2%i==0)
        gcd = i;
}

printf("G.C.D of %d and %d is %d", n1, n2, gcd);

return 0;
}

```

Output:



```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM$ cd "/home/shruti/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM/" && g++ 2_GCD_of_Two_No.cpp -o 2_GCD_of_Two_No && "/home/shruti/Desktop/vscode/College CP Programs./2_GCD_of_Two_No"
Enter two integers: 10 25
G.C.D of 10 and 25 is 5
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM$

```

Conclusion:- In this Practical i learned to Implement Program To Find GCD Of Given Number.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 3

Date : 13-04-2021

Title : Write Program To Find Modulous Of Two Numbers.

Aim : To Implement Program To Find Modulous Of Two Numbers.

Theory:

Modulus Operator : The **modulus operator** is a symbol used in various programming languages. It is denoted by the percentage symbol (%). It is a modulus operator that is used in the arithmetic operator. It determines the remainder. In some cases, the remainder may be 0, it means the number is completely divisible by the divisor.

Syntax : remainder = a % b;

In the above Syntax, a and b are two integers, and the % (Percent) symbol is a modulus operator that divides a by b and returns the remainder.

Program:-

```
#include <stdio.h>
int main() {
    int dividend, divisor, quotient, remainder;
    printf("Enter dividend: ");
    scanf("%d", &dividend);
    printf("Enter divisor: ");
    scanf("%d", &divisor);

    // Computes quotient
    quotient = dividend / divisor;

    // Computes remainder
    remainder = dividend % divisor;

    printf("Quotient = %d\n", quotient);
    printf("Remainder = %d", remainder);
    return 0;
}
```


Output:

```
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM$ cd "/home/shruti/AL_PROGRAM/" && g++ 3 Modulous.cpp -o 3_Modulous && "/home/shruti/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM/3_Modulous"
Enter dividend: 10
Enter divisor: 2
Quotient = 5
Remainder = 0
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM$
```

Conclusion:- In this Practical i learned to Implement Program To Find Modulous Of Two Numbers.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 4

Date : 20-04-2021

Title : Write Program Of Diophantine Equations.

Aim : To Implement Program Of Diophantine Equations.

Theory:

Diophantine equation : A Diophantine equation is a polynomial equation, usually in two or more unknowns, such that only the integral solutions are required. An Integral solution is a solution such that all the unknown variables take only integer values.

Given three integers a, b, c representing a linear equation of the form : $ax + by = c$. Determine if the equation has a solution such that x and y are both integral values.

Examples :

Input : a = 3, b = 6, c = 9

Output: Possible

Explanation : The Equation turns out to be,

$3x + 6y = 9$ one integral solution would be

$x = 1, y = 1$

Solution: For linear Diophantine equation equations, integral solutions exist if and only if, the GCD of coefficients of the two variables divides the constant term perfectly. In other words the integral solution exists if, $\text{GCD}(a, b)$ divides c.

Thus the algorithm to determine if an equation has integral solution is pretty straightforward.

1. Find GCD of a and b
2. Check if $c \% \text{GCD}(a, b) == 0$
3. If yes then print Possible
4. Else print Not Possible

Below is the implementation of above approach.

Program:-

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

using namespace std;

int gcd(int a, int b)
{
    int gcd,i;
    gcd=0;
    for(i=1; i <= a && i <= b; ++i)
    {
        if(a%i==0 && b%i==0)
            gcd = i;
    }

    return gcd;
    //return (a%b == 0)? abs(b) : gcd(b,a%b);
}

bool isPossible(int a, int b, int c)
{
    return (c%gcd(a,b) == 0);
}

int main()
{
    int a,b,c;

    printf("\n\n\t Enter The Value For A : ");
    scanf("%d",&a);
    printf("\n\n\t Enter The Value For B : ");
    scanf("%d",&b);
    printf("\n\n\t Enter The Value For C : ");
    scanf("%d",&c);

    printf("\n\n=====\\n\\n");

    if(isPossible(a,b,c))
    {
        printf("\n\n\t Diophantine equation Solution is Possible. \\n\\n");
    }
    else
    {
        printf("\n\n\t Diophantine equation Solution is Not Possible. \\n\\n");
    }
}
```

```

    }

    return 0;
}

```

Output:

```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM$ cd "/home/shruti/Desktop/vscode/College CP Programs./3_Diophantine Equations/3.1 Program/" && g++ Approach2.cpp -o Approach2 && "/home/shruti/Desktop/vscode/College CP Programs./3_Diophantine Equations/3.1 Program/Approach2"
h2
Enter The Value For A : 3
Enter The Value For B : 6
Enter The Value For C : 9

=====

Diophantine equation Solution is Possible.
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./3_Diophantine Equations/3.1 Programs$

```

Conclusion:- In this Practical i learned to Implement Program Of Diophantine Equations.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 5

Date : 27-04-2021

Title : Write Program To Solve 15-Puzzle Problem.

Aim : To Implement Program To Solve 15-Puzzle Problem.

Theory:

15-Puzzle Problem : The 15 puzzle is also called as Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and many others. In 15 Puzzels Problem we have given a 4×4 board with 15 tiles (every tile has one number from 1 to 15) and one empty space. The objective is to place the numbers on tiles in order using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space. For example,

Initial Configuration

2	1	3	4
5	6	7	8
9	10	11	12
13	14	15	X

Final Configuration

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	X

Here X marks the spot to where the

elements can be shifted and the final configuration always remains the same the puzzle is solvable.

In general, for a given grid of width N, we can find out check if a $N \times N - 1$ puzzle is solvable or not by following below simple rules :

1. If N is odd, then puzzle instance is solvable if number of inversions is even in the input state.
2. If N is even, puzzle instance is solvable if
 - the blank is on an even row counting from the bottom (second-last, fourth-last, etc.) and number of inversions is odd.
 - the blank is on an odd row counting from the bottom (last, third-last, fifth-last, etc.) and number of inversions is even.
3. For all other cases, the puzzle instance is not solvable.

Program:-

```
#include <iostream>
#define N 4
using namespace std;

int getInvCount(int arr[])
{
    int inv_count = 0;
    for (int i = 0; i < N * N - 1; i++)
    {
        for (int j = i + 1; j < N * N; j++)
        {
            // count pairs(i, j) such that i appears
            // before j, but i > j.
            if (arr[j] && arr[i] && arr[i] > arr[j])
                inv_count++;
        }
    }
    return inv_count;
}

int findXPosition(int puzzle[N][N])
{
    // start from bottom-right corner of matrix
    for (int i = N - 1; i >= 0; i--)
        for (int j = N - 1; j >= 0; j--)
            if (puzzle[i][j] == 0)
                return N - i;
}

bool isSolvable(int puzzle[N][N])
{
    // Count inversions in given puzzle
    int invCount = getInvCount((int*)puzzle);

    // If grid is odd, return true if inversion
    // count is even.
    if (N & 1)
    {
        return !(invCount & 1);
    }
    else // grid is even
    {
        int pos = findXPosition(puzzle);
```

```

        if (pos & 1)
            return !(invCount & 1);
        else
            return invCount & 1;
    }
}

int main()
{
    int puzzle[N][N] =
    {
        {12, 1, 10, 2},
        {7, 11, 4, 14},
        {5, 0, 9, 15}, // Value 0 is used for empty space
        {8, 13, 6, 3},
    };

    isSolvable(puzzle)? cout << "Solvable":cout << "Not Solvable";
    return 0;
}

```

Output:



```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS$ cd "/home/shruti/Desktop/vscode/College CP Programs/" && g++ 5_15_puzzle_problem.cpp -o 5_15_puzzle_problem && "/home/shruti/Desktop/vscode/College CP Programs/5_15_puzzle_problem"
Entered Matrix is :
12      1      10      2
7       11      4      14
5       0       9      15
8       13      6      3

Target Matrix is :
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      0

***** OUTPUT *****

Solvable

End Of Program

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS$

```

Conclusion:- In this Practical i learned to Implement Program To Solve 15-Puzzle Problem.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 6

Date : 04-05-2021

Title : Write Program To Solve Tug of War Problem.

Aim : To Implement Program To Solve Tug of War Problem.

Theory:

Tug Of War Problem : In Tug Of War Problem We have Given a set of n integers, divide the set in two subsets of $n/2$ sizes each such that the difference of the sum of two subsets is as minimum as possible. If n is even, then sizes of two subsets must be strictly $n/2$ and if n is odd, then size of one subset must be $(n-1)/2$ and size of other subset must be $(n+1)/2$.

1. You are given an array of n integers.
 2. You have to divide these n integers into 2 subsets such that the difference of sum of two subsets is as minimum as possible.
 3. If n is even, both set will contain exactly $n/2$ elements. If n is odd, one set will contain $(n-1)/2$ and other set will contain $(n+1)/2$ elements.
 4. If it is not possible to divide, then print "-1".
- Note : Check out the question video and write the recursive code as it is intended without changing signature. The judge can't force you but intends you to teach a concept.

Program :

```
#include <iostream>
#include <stdlib.h>
#include <limits.h>
using namespace std;

void TugofWarRecur(int* array, int n, bool* current_elements, int selected_elements_count, bool*
solution, int* min_diff, int sum, int current_sum, int current_position)
{
    if (current_position == n)
    {
        return;
    }
}
```



```

    if ((n/2 - selected_elements_count) > (n - current_position))
    {
        return;
    }

    TugofWarRecur(array, n, current_elements, selected_elements_count, solution, min_diff, sum,
current_sum, current_position+1);

    selected_elements_count++;
    current_sum = current_sum + array[current_position];
    current_elements[current_position] = true;
    if (selected_elements_count == n/2)
    {
        if (abs(sum/2 - current_sum) < *min_diff)
        {
            *min_diff = abs(sum/2 - current_sum);
            for (int i = 0; i<n; i++)
            {
                solution[i] = current_elements[i];
            }
        }
    }
    else
    {
        TugofWarRecur(array, n, current_elements, selected_elements_count, solution, min_diff, sum,
current_sum, current_position+1);
    }
    current_elements[current_position] = false;
}

void TugOfWar(int *array, int n)
{
    bool* current_elements = new bool[n];
    bool* solution = new bool[n];
    int min_diff = INT_MAX;
    int sum = 0;
    for (int i=0; i<n; i++)
    {
        sum = sum + array[i];
        current_elements[i] = solution[i] = false;
    }
    TugofWarRecur(array, n, current_elements, 0, solution, &min_diff, sum, 0, 0);

    for (int i=0; i<n; i++)
    {
        if(solution[i] == true)
        {
            cout << array[i] << " ";
        }
    }
}

```

```

    }

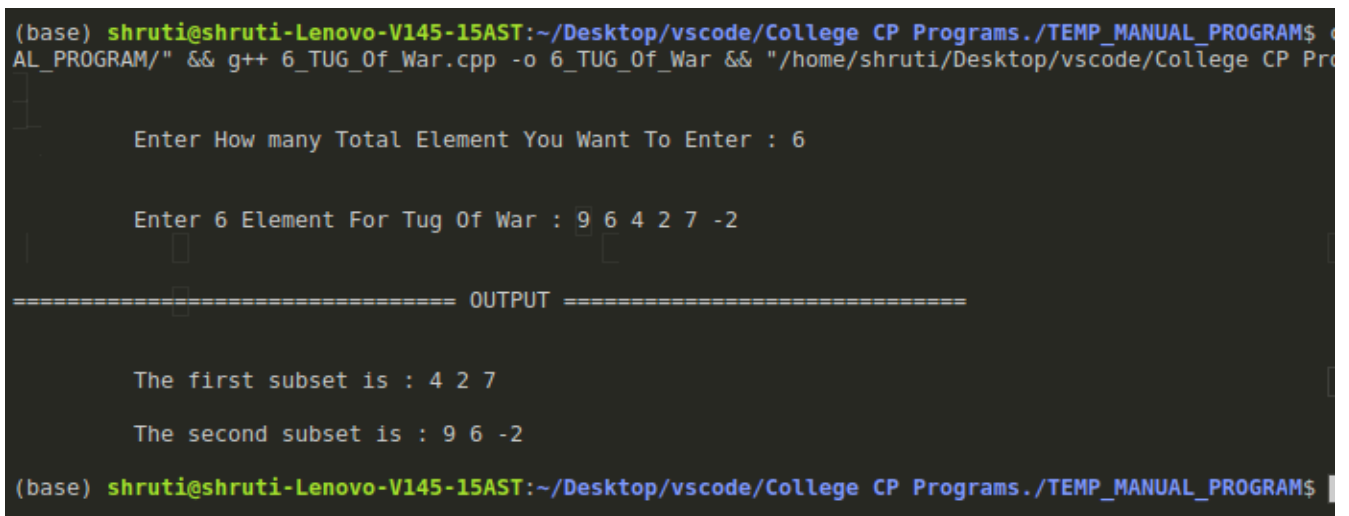
    for (int i=0; i<n; i++)
    {
        if(solution[i] == false)
        {
            cout << array[i] << " ";
        }
    }
}

int main()
{
    int n;
    cin>>n;
    int input_array[n];
    for(int i=0;i<n;i++)
    {
        cin>>input_array[i];
    }
    TugOfWar(input_array,n);

    return 0;
}

```

Output:



```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS: $ g++ 6_TUG_Of_War.cpp -o 6_TUG_Of_War && ./6_TUG_Of_War
Enter How many Total Element You Want To Enter : 6
Enter 6 Element For Tug Of War : 9 6 4 2 7 -2
===== OUTPUT =====
The first subset is : 4 2 7
The second subset is : 9 6 -2
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS: $

```

Conclusion:- In this Practical i learned to Implement Program To Solve Tug of War Problem.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 7

Date : 11-05-2021

Title : Write Program To Solve Tower of Cubes Problem.

Aim : To Implement Program To Solve Tower of Cubes Problem.

Theory:

Tower of Cubes Problem : In this problem you are given N colorful cubes each having a distinct weight. Each face of a cube is colored with one color. Your job is to build a tower using the cubes you have subject to the following restrictions:

- Never put a heavier cube on a lighter one.
- The bottom face of every cube (except the bottom cube, which is lying on the floor) must have the same color as the top face of the cube below it.
- Construct the tallest tower possible.

Input & Output Rules:

1) Input Rules :

1. The input may contain multiple test cases. The first line of each test case contains an integer N
- Here Value of N is : ($1 \leq N \leq 500$) indicating the number of cubes you are given. The ith ($1 \leq i \leq N$) of the next N lines contains the description of the ith cube.
2. A cube is described by giving the colors of its faces in the following order: front, back, left, right, top and bottom face. For your convenience colors are identified by integers in the range 1 to 100.
3. You may assume that cubes are given in the increasing order of their weights, that is cube 1 is the lightest and cube N is the heaviest.

2) Output Rules :

1. For each test case in the input first print the test case number on a separate line as shown in the sample output. On the next line print the number of cubes in the tallest tower you have built.
2. From the next line describe the cubes in your tower from top to bottom with one description per line.

3. Each description contains an integer (giving the serial number of this cube in the input) followed by a single white space character and then the identification string (front, back, left, right, top or bottom) of the top face of the cube in the tower.
- Note that there may be multiple solutions and any one of them is acceptable.

Program :

```
#include <iostream>

using namespace std;

const int Skipped = 6;
int N;
int cubes[500][6];
int numberCanStack[500][100];
int bestTopFaceToUse[500][100];

string ToFace(int facePos)
{
    switch (facePos)
    {
        case 0:
            return "front";
        case 1:
            return "back";
        case 2:
            return "left";
        case 3:
            return "right";
        case 4:
            return "top";
        default:
            return "bottom";
    }
}

void PrintOut(int cube, int color)
{
    if (cube == N)
        return;

    if (bestTopFaceToUse[cube][color] == Skipped)
    {
        PrintOut(cube + 1, color);
    }

    else
    {
        int faceUsed = bestTopFaceToUse[cube][color];
```

```

    PrintOut(cube + 1, cubes[cube][faceUsed]);

    cout << "\t" << (N - cube) << ' ' << "\t | \t\t" << ToFace(faceUsed) << "\n";
}
}

int NumCanStack(int cube, int color)
{
    if (cube == N)
        return 0;

    int &num = numberCanStack[cube][color];

    if (num == -1) bestTopFaceUse
    {
        num = 0;
        int &bestTopFace = bestTopFaceToUse[cube][color];
        bestTopFace = Skipped;

        for (int face = 0; face < 6; ++face)
        {
            if (cubes[cube][face] == color)
            {
                int topFace = face ^ 1;
                int amount = NumCanStack(cube + 1, cubes[cube][topFace]) + 1;

                if (amount > num)
                {
                    num = amount;
                    bestTopFace = topFace;
                }
            }
        }
        int amount = NumCanStack(cube + 1, color);
        if (amount > num)
        {
            num = amount;
            bestTopFace = Skipped;
        }
    } return num;
}

int main()
{
    int i = 1;
    cout << "\n\n\t Enter Number of Cubes You Want : ";
    while (cin >> N, N)
    {

```

```

if (i == 1)
{
    cout << "\n\n===== ";
    cout << "\n Enter the Values for Faces OR Side Of Cubes : \n\n";
    cout << "front \t back \t left \t right \t top \t bottom \n";
    i++;
}
for (int cube = N - 1; cube >= 0; --cube)
{
    for (int face = 0; face < 6; ++face)
        cin >> cubes[cube][face];
    cout << "\t";
}

for (int cube = 0; cube < N; ++cube)
{
    for (int color = 0; color < 105; ++color)
        numberCanStack[cube][color] = -1;
}

int bestNum = 0, bestStartCube = -1, bestColor;
for (int cube = 0; cube < N; ++cube)
{
    for (int face = 0; face < 6; ++face)
    {
        int num = NumCanStack(cube, cubes[cube][face]);
        if (num > bestNum)
        {
            bestNum = num;
            bestStartCube = cube;
            bestColor = cubes[cube][face];
        }
    }
}

cout << "\nTotal Height Of Tower : " << bestNum << "\n\n";
cout << "Index Of Cube \t | " << " Face Of Cube Attache To Upper Cube" << "\n";
cout << "-----\n\n";

PrintOut(bestStartCube, bestColor);

exit(0);
}
}

```

Output:

```
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./5_Is B
_Of_Cubes.cpp -o 1_Tower_Of_Cubes && "/home/shruti/Desktop/vscode/College CP Program

Enter Number of Cubes You Want : 3

=====
Enter the Values for Faces OR Side Of Cubes :

front    back    left    right    top    bottom
1         2         2         2         1         2
3         3         3         3         3         3
3         2         1         1         1         1

***** Output *****

Total Height Of Tower : 2

-----
Index Of Cube | Face Of Cube Attache To Upper Cube
-----
1             | front
3             | back
```

•

Conclusion:- In this Practical i learned to Implement Program To Solve Tower of Cubes Problem.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 8

Date : 18-05-2021

Title : Write Program To Solve Tower Of Hanoi.

Aim : To Implement Program To Solve Tower Of Hanoi.

Theory:

Tower Of Hanoi : Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Program:-

```
#include<stdio.h>
void towers(int, char, char, char);
int main()
{
    int num;
    printf ("Enter the number of disks : ");
    scanf ("%d", &num);
    printf ("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers (num, 'A', 'C', 'B');
    return 0;
}
void towers( int num, char from peg, char topeg, char auxpeg)
{
    if (num == 1)
    {
        printf ("\n Move disk 1 from peg %c to peg %c", from peg, topeg);
        return;
    }
    towers (num - 1, from peg, auxpeg, topeg);
    printf ("\n Move disk %d from peg %c to peg %c", num, from peg, topeg);
    towers (num - 1, auxpeg, topeg, from peg);
}
```


Output:

```
Move disk 1 from peg A to peg C(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP
AL_PROGRAM/" && g++ 8_Tower_Of_Hanoi.cpp -o 8_Tower_Of_Hanoi && "/home/shruti/Desktop/vscode/Colle

Enter the number of disks : 3

=====

The sequence of moves involved in the Tower of Hanoi are :

Move disk 1 from Rod A to Rod C
Move disk 2 from Rod A to Rod B
Move disk 1 from Rod C to Rod B
Move disk 3 from Rod A to Rod C
Move disk 1 from Rod B to Rod A
Move disk 2 from Rod B to Rod C
Move disk 1 from Rod A to Rod C

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAM$
```

Conclusion:- In this Practical i learned to Implement Program To Solve Tower Of Hanoi.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 9

Date : 25-05-2021

Title : Write Program To Solve Number of Stopping Station Problem.

Aim : To Implement Program To Solve Number of Stopping Station Problem.

Theory:

Problem statement – A program to find the number of ways in which a train will stop in **r** stations out of **n** stations so that no two stopping stations are consecutive.

Problem Explanation : This program will calculate the number of ways i.e. permutations in which the train will stop. Here, the train will travel from point X to Y. Between these points, there are **n** stations. The train will stop on **r** stations of these **n** stations given the conditions that while stopping on **r** stations the train should not stop in two consecutive stations.

This permutation can be found using the direct nPr formula.

Let's take a few examples,

Input : $n = 16$, $r = 6$

Output : 462

Explanation : The number of ways the train can stop at 6 stops out of 16 stops fulfilling the condition is found using the permutation formula given by

$$nPr \text{ or } p(n, r) = n! / (n-r)!$$

Algorithm: Input --> total numbers of stations **n** and number of stations train can stop **r**.

Step 1 : For values of **n** and **r** calculate the value of $p(n, r) = n! / (n-r)!$

Step 2 : print the value of $p(n, r)$ using std print method.

Program:-

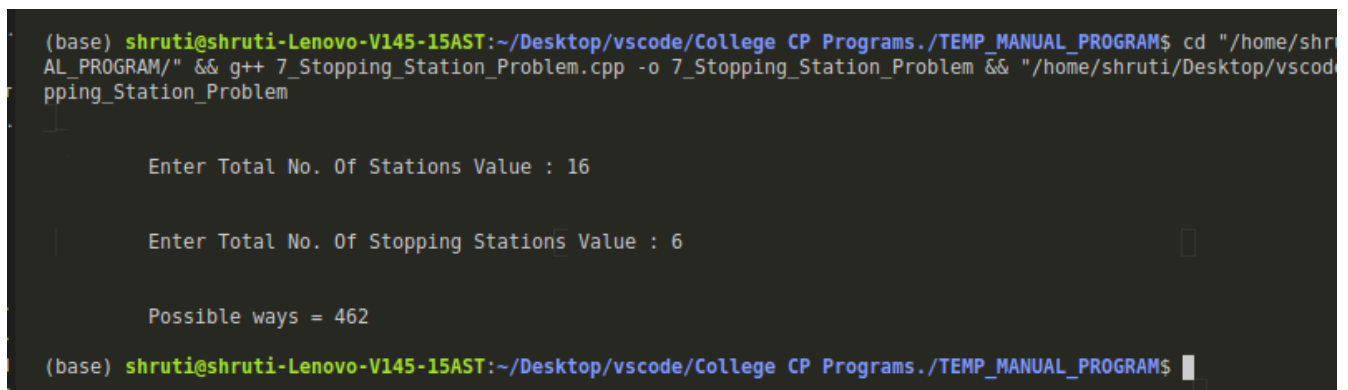
```
#include<stdio.h>
int main(){
    int n = 16, s = 6;
    printf("Total number of stations = %d\nNumber of stopping station = %d\n", s, n);
    int p = s;
    int num = 1, dem = 1;
```

```

while (p!=1) {
    dem*=p;
    p--;
}
int t = n-s+1;
while (t!=(n-2*s+1)) {
    num *= t;
    t--;
}
if ((n-s+1) >= s)
    printf("Possible ways = %d", num / dem);
else
    printf("no possible ways");
}

```

Output:



```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS$ cd "/home/shruti/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS/" && g++ 7_Stopping_Station_Problem.cpp -o 7_Stopping_Station_Problem && "/home/shruti/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS/7_Stopping_Station_Problem"

Enter Total No. Of Stations Value : 16

Enter Total No. Of Stopping Stations Value : 6

Possible ways = 462

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./TEMP_MANUAL_PROGRAMS$

```

Conclusion:- In this Practical i learned to Implement Program To Solve Number of Stopping Station Problem.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 10

Date : 01-06-2021

Title : Write Program To Solve Is Bigger Smarter Problem.

Aim : To Implement Program To Solve Is Bigger Smarter Problem.

Theory:

Is Bigger Smarter Problem : Some people think that the bigger an elephant is, the smarter it is. To disprove this, you want to take the data on a collection of elephants and put as large a subset of this data as possible into a sequence so that the weights are increasing, but the IQ's are decreasing.

Input & Output Rules:

1) Input Rules :

- The input will consist of data for a bunch of elephants, one elephant per line.)
- The data for a particular elephant will consist of a pair of integers: the first representing its size in kilograms and the second representing its IQ in hundredths of IQ points. Both integers are between 1 and 10000.
- The data will contain information for at most 1000 elephants. Two elephants may have the same weight, the same IQ, or even the same weight and IQ.

2) Output Rules :

1. Say that the numbers on the i-th data line are $W[i]$ and $S[i]$. Your program should output a sequence of lines of data; the first line should contain a number n ;
2. If these n integers are $a[1], a[2], \dots, a[n]$ then it must be the case that
 - $W[a[1]] < W[a[2]] < \dots < W[a[n]]$ &
 - $S[a[1]] > S[a[2]] > \dots > S[a[n]]$
3. In order for the answer to be correct, n should be as large as possible. All inequalities are strict: weights must be strictly increasing, and IQs must be strictly decreasing.

Program:-

```
#include<iostream>
#include<algorithm>
using namespace std;
```

```

int dp[1000];
int path[1000];
int pos;
int ans;

struct ele{
    int w;
    int iq;
    int id;
}e[1000];

bool cmp(struct ele a,struct ele b)
{
    if(a.w==b.w)
    {
        return a.iq>b.iq;
    }
    else
    {
        return a.w<b.w;
    }
}

void print(int x){
    if(ans--){
        print(path[x]);
        printf("\t%d\t\t %d \t\t %d \n",e[x].id,e[x].w,e[x].iq);
    }
}

int main()
{
    int cnt=0;
    int N;

    printf("\n\n\t Enter How many Elephant : ");
    scanf("%d",&N);

    printf("\n\n\t Enter value of Weight & IQ for each Elephant : ")
    for(int i=0;i<N;i++)
    {

        scanf("%d%d",&e[cnt].w,&e[cnt].iq);
        e[cnt].id=cnt+1;
        dp[cnt]=1;
        ++cnt;
    }
}

```

```

sort(e,e+cnt,cmp);

for(int i=0;i<cnt;i++)
{
    for(int j=0;j<i;j++)
    {
        if(e[i].w>e[j].w && e[i].iq<e[j].iq && dp[j]+1>dp[i])
        {
            dp[i]=dp[j]+1;
            path[i]=j;// Answer Elephant path sequence
        }
    }
    if(ans<=dp[i])
    {
        pos=i;
        ans=dp[i];
    }
}
printf("\n\n\t Length Of Longest common Subsequent of Elephants : %d \n\n\n",ans);
print(pos);//recursive output sequence
return 0;
}

```

Output:

```

TERMINAL OUTPUT DEBUG CONSOLE PROBLEMS 1: Code
Enter How many Elephant : 9

=====

Enter value of Weight & IQ for each Elephant :

Weight    IQ
6008      1300
6000      2100
500       2000
1000      4000
1100      3000
6000      2000
8000      1400
6000      1200
2000      1900

===== OUTPUT =====

Length Of Longest common Subsequent of Elephants : 4

Elephant ID | Elephant Weight | Elephant IQ
-----
4           | 1000            | 4000
5           | 1100            | 3000
9           | 2000            | 1900
7           | 8000            | 1400
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./5_Is Bigger Smarter$

```

Conclusion:- In this Practical i learned to mplement Program To Solve Is Bigger Smarter Problem.

Godavari College Of Engineering, Jalgaon.

Subject Name: CP-II

Class: T. Y. Btech. (Comp.)

Practical No. : 11

Date : 08-06-2021

Title : Write Program To Solve Unidirectional TSP Problem.

Aim : To Implement Program To Solve Unidirectional TSP Problem.

Theory:

Unidirectional TSP Problem : Problems that require minimum paths through some domain appear in many different areas of computer science. For example, one of the constraints in VLSI routing problems is minimizing wire length.

- The Traveling Salesperson Problem (TSP) — finding whether all the cities in a salesperson's route can be visited exactly once with a specified limit on travel time — is one of the canonical examples of an NP-complete problem; solutions appear to require an inordinate amount of time to generate, but are simple to check. This problem deals with finding a minimal path through a grid of points while traveling only from left to right.
- Given an $m \times n$ matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 1 (the first column) and consists of a sequence of steps terminating in column n (the last column). A step consists of traveling from column i to column $i + 1$ in an adjacent (horizontal or diagonal) row.
- The first and last rows (rows 1 and m) of a matrix are considered adjacent, i.e., the matrix “wraps” so that it represents a horizontal cylinder. Legal steps are illustrated on the right.
- The weight of a path is the sum of the integers in each of the n cells of the matrix that are visited. For example, two slightly different 5×6 matrices are shown below (the only difference is the numbers in the bottom row).
- The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

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

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

Input & Output Rules:

1) Input Rules :

1. The input consists of a sequence of matrix specifications. Each matrix specification consists of the row and column dimensions in that order on a line followed by $m \cdot n$ integers where m is the row dimension and n is the column dimension.
2. The integers appear in the input in row major order, i.e., the first n integers constitute the first row of the matrix, the second n integers constitute the second row and so on. The integers on a line will be separated from other integers by one or more spaces.
 - Note: integers are not restricted to being positive.
3. For each specification the number of rows will be between 1 and 10 inclusive; the number of columns will be between 1 and 100 inclusive.

2) Output Rules :

1. Two lines should be output for each matrix specification in the input file:
 - a) the first line represents a minimal-weight path, and
 - b) the second line is the cost of a minimal path.
2. The path consists of a sequence of n integers (separated by one or more spaces) representing the rows that constitute the minimal path.
3. If there is more than one path of minimal weight the path that is lexicographically smallest should be output.

Note: Lexicographically means the natural order on sequences induced by the order on their elements.

Program:

```
#include <iostream>
using namespace std;
int C, R;

bool reached[100][10];
int grid[100][10];
int bestCost[100][10];
int bestTo[100][10];

int GetBest(int c, int r)
{
    if (c == C)
        return 0;

    int &cost = bestCost[c][r];

    if (!reached[c][r])
    {
```



```

reached[c][r] = true;
int &to = bestTo[c][r];
to = (r - 1 + R) % R;
cost = GetBest(c + 1, to);

int newCost = GetBest(c + 1, r);
if (newCost < cost || (newCost == cost && r < to))
{
    cost = newCost;
    to = r;
}
int newR = (r + 1) % R;
newCost = GetBest(c + 1, newR);
if (newCost < cost || (newCost == cost && newR < to))
{
    cost = newCost;
    to = newR;
}
cost += grid[c][r];
} return cost;
}

int main()
{
    cout<<"\n\n\t Enter Row Dimension For Graph : ";
    cin>>R;

    cout<<"\n\n\t Enter Column Dimension For Graph : ";
    cin>>C;

    for (int r = 0; r < R; ++r)
    {
        for (int c = 0; c < C; ++c)
        {
            cin >> grid[c][r];
            reached[c][r] = false;;
        }
    }

    int bestCost = 1e9, bestStart;
    for (int r = 0; r < R; ++r)
    {
        int cost = GetBest(0, r);
        if (cost < bestCost)
        {
            bestStart = r;

```

```

        bestCost = cost;
    }
}

cout<<"\n\n===== OUTPUT =====";
cout << "\n\n Minimum Path is : \t";
int current = bestStart;
cout << current + 1;
current = bestTo[0][current];
int c = 1;
while (c < C)
{
    cout <<"\t"<< (current + 1);
    current = bestTo[c][current];
    ++c;
}
cout << "\n\n Minimum Path Cost is : "<< bestCost << "\n';

cout<<"\n\n===== End =====\n\n";
}

```

Output:

```

(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./6_Unidirectional_TSP$ cd "
ectional_TSP/" && g++ Shweta_Patil_Unidirectional_TSP_Problem_Solution.cpp -o Shweta_Patil_Unidirection
ollege CP Programs./6_Unidirectional_TSP/"Shweta_Patil_Unidirectional_TSP_Problem_Solution

Enter Row Dimension For Graph : 5

Enter Column Dimension For Graph : 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 1 2 3

===== OUTPUT =====
Minimum Path is :      1      2      1      5      4      5
Minimum Path Cost is : 11

===== End =====
(base) shruti@shruti-Lenovo-V145-15AST:~/Desktop/vscode/College CP Programs./6_Unidirectional_TSP$

```

Conclusion:- In this Practical i learned to Implement Program To Solve Unidirectional TSP Problem.