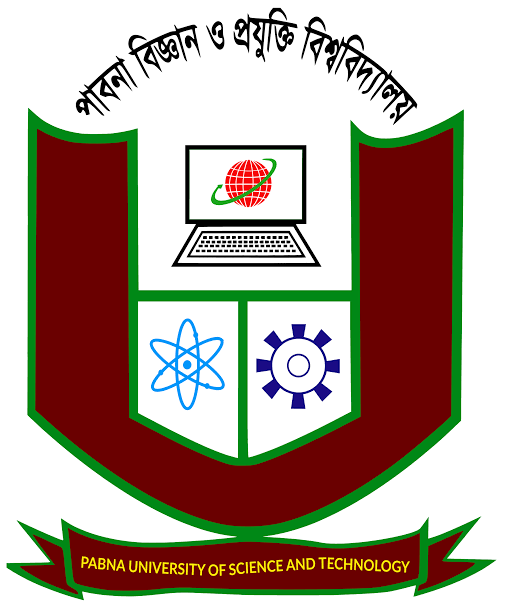
****Description: Image result for PUST SYMBOLDescription: Image result for PUST SYMBOL

**Faculty of Engineering & Technology**

***Department of Information and Communication Engineering***

***B.Sc. (Engineering) 2nd Year 1st Semester Examination-2020***

******

**Practical Lab Report**

**Session:** **2018-2019** **Course Code**: **ICE-2106**

**Title:** **Discrete Mathematics and Numerical Methods Sessional.**

**Submitted to**

**Md. Anwar Hossain**

Associate Professor

Dept. of Information and Communication Engineering

Pabna University of Science and Technology

Pabna-6600, Bangladesh

E-mail: [manwar.ice@gmail.com](mailto:manwar.ice@gmail.com)

**Submitted by**

|  |  |
| --- | --- |
|  | **Md. Rahatul Rabbi**  Roll: 190609  Registration No:1065334  Dept. of Information and Communication Engineering  Pabna University of Science and Technology  Pabna-6600, Bangladesh  E-mail: rahatul.190609@s.pust.ac.bd  Mobile: 017902-24950 |

***Table of Contents***

|  |  |
| --- | --- |
|  | Let A be the set {1, 2, 3, 4}. Write a program to find the ordered pairs are in the relation  I) = {(a, b) | a divides b} II) = {(a, b) | a ≤ b}.............................................. 3 - 5 |
|  | Suppose that A = {1, 2, 3} and B = {1, 2}. Let R be the relation from A to B containing (a, b) if a ∈ A, b ∈ B, and a > b. Write a program to find the relation R and also represent this relation in matrix form if a1 = 1, a2 = 2, and a3 = 3, and b1 = 1 and b2 = 2………………………………………………………………………………….. 5 - 7 |
|  | Write a program for graph coloring by Welch- Powell’s algorithm…...………… 8 - 12 |
|  | Write a program to find shortest path by Wars hall’s algorithm…………...…… 12 - 14 |
|  | Suppose that the relations R1 and R2 on a set A are represented by the matrices  and . Write a program to find the MR1∪R2 and MR1∩R2 …………………………………………………………………………... 15 - 18 |
|  | The following table gives the population of a town during the last six censuses. Write a Python program to find the population in the year of 1946 using Newton-Gregory forward interpolation formula…………………………………………………... 19 - 23   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Year: | 1911 | 1921 | 1931 | 1941 | 1951 | 1961 | | Population: | 12 | 15 | 20 | 27 | 39 | 52 | |
|  | Write a Python program to find ***f(7.5)*** form the following table using Newton-Gregory backward interpolation formula……………………………………………...…. 24 - 28   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | *x*: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | *f(x)*: | 1 | 8 | 27 | 64 | 125 | 216 | 343 | 512 | |
|  | Write a Python program to find the value of f(15) from the following table using Newton’s divided difference formula for unequal intervals …………………… 28 - 32   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | *x:* | 4 | 5 | 7 | 10 | 11 | 13 | | *f(x):* | 48 | 100 | 294 | 900 | 1210 | 2028 | |
|  | Write a Python program to find the value of y when x=10 from following table using Lagrange’s interpolation formula for unequal intervals……………………..….. 33 - 35   |  |  |  |  |  | | --- | --- | --- | --- | --- | | *x:* | 5 | 6 | 9 | 11 | | *y:* | 12 | 13 | 14 | 16 | |
|  | Write a Python program to find a real root of the equation that lies between -1 and 3 using bisection method…………………………………….… 35 - 38 |
|  | Write a Python program to find a real root of the function in the range 1<x<3 using false position method………………………………….…… 38 - 41 |

**Experiment No:** **01**

**Name of the Experiment:** Let A be the set {1, 2, 3, 4}. Write a program to find the ordered pairs are in the relation-

I) = {(a, b) | a divides b} II) = {(a, b) | a ≤ b}

**Illustration:**

Relationships between elements of sets are represented using a structure called a relation, which is just a subset of the Cartesian product of the sets. The most direct way to express a relationship between elements of two sets is to use ordered pairs made up of two related elements.

In this relation we use two conditions for first condition a/b we use, if i % j = = 0 and j % i = = 0 and in the second condition a <= b we use, i <= j. Finally we find the pair list of a/b and a <= b.

**Procedure:**

Firstly, I imported the product module from the itertools package that used as a method. I created a dada.txt file in same folder and here putted values of the set A = {1, 2, 3, 4} for getting input. Now, for reading the data.txt file primarily, I used input function for getting file name with extension from the user and putted the file name with a variable named file\_name. After that I used open method and assigned the file with a variable named file.

Then I used a map function for putted values of the data.txt file as a list with a variable named Set. Usually map function received two parameters such as function and iterables object and it returns a list. After that I printed the values of set A = [1, 2, 3, 4].

Afterword’s I used product method for calculating the ordered pairs and it returned a list that assigned with a variable named Data.

For calculating the ordered pair when relation following = {(a, b) | a divides b}, at first I declared an empty list as variable named result\_1. After that I used “for” loop. Inside this ‘for’ loop I created an ‘if’ statement where checked two conditions i % j == 0 or j % i == 0. If it return “True” then append the result (i, j) in the variable named result\_1.

Again, I used a comprehension list for calculating the ordered pair when relation following = {(a, b) | a ≤ b} that contained an ‘if’ statement where checked condition i<=j and assigned the result with variable named result\_2. Then I printed two results for this two relations such as I) = {(a, b) | a divides b} II) = {(a, b) | a ≤ b}. Finally I closed the dada.txt file.

**Python Source Code:**

|  |
| --- |
| **from** itertools **import** product  *# Variable initialization* result\_1 = []  *# Getting Set Input From file* file\_name = input(**"Enter file name with extension : "**) *# code and input file should be in same folder  # read input value from file* file = open(file\_name,**"r"**) Set = list(map(int, file.readlines()))  *# Printing Set A* print(**"\n\nValue of the Set, A = "**,Set,**"\n"**)  *# Calculating the ordered pairs* Data = list(product(Set,repeat=2))  *# Calculating the ordered pair when relation following R1 ={(a,b) | a divides b}* **for** i,j **in** Data:  **if** i%j==0 **or** j%i==0:  result\_1.append((i,j))  *# Calculating the ordered pair when relation following R2 ={(a,b) | a ≤ b}* result\_2 = [(i,j) **for** i,j **in** Data **if** i<=j]  *# printing results* print (**"The ordered pair list is for a / b : \n\t\t\t"**,result\_1) print (**"The ordered pair list is for a <= b : \n\t\t\t"** ,result\_2) file.close() |

**Input:**

**File:**

data.txt

1

2

3

4

Enter file name with extension: data.txt

**Output:**

Value of the Set, A = [1, 2, 3, 4]

The ordered pair list is for a / b:

[(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 4), (3, 1), (3, 3), (4, 1), (4, 2), (4, 4)]

The ordered pair list is for a <= b:

[(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)]

**Experiment No:** **02**

**Name of the Experiment:** Suppose that A = {1, 2, 3} and B = {1, 2}. Let R be the relation from A to B containing (a, b) if a ∈ A, b ∈ B, and a > b. Write a program to find the relation R and also represent this relation in matrix form if a1 = 1, a2 = 2, and a3 = 3, and b1 = 1 and b2 = 2.

**Illustration:**

In this problem, first we declare NumPy. NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Here we print Output1 which is a > b relation and output2 which is 3, 2 order matrix. So we use two condition first use for loops and arrange a relation and output2 we use a condition 1 for a > b and 0 for a!> b .So finally we find two output-

1. Relation of a>b and 2. Matrix of a > b.

**Procedure:**

This program started with “import numpy as np”. NumPy is a Python library used for working with arrays. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The means of “import numpy as np” is the NumPy package can be referred to as np instead of numpy.

Firstly, I created two files dada1.txt file and data2.txt file in the same folder and here putted values of the sets respectively A = {1, 2, 3, 4}, B = {1, 2} for getting input. Now, for reading the data1.txt file primarily, I used input function for getting first file name with extension from the user and assigned the first file name with a variable named file\_name1. After that I used open method and putted the first file with a variable named file1. Similarly, I putted the second file name with a difference variable named file\_name2 and also assigned the second file with a difference variable named file2.

After that, I used map function. It had received two parameters, such as function and iterables object, it returned a list and I assigned values of the data1.txt file as a list with a variable named list1. I used the same things again, but this time I assigned values of the data2.txt file as a list with a different variable named list2.

Afterwards, I used a comprehension list for calculating some specific mathematical tasks. First task is assigned with a variable named output1 and second task is assigned with a variable named data. Then I used an attribute from numpy named np.array and putted the result with a variable named newArray. After that, I used reshaping array that means changing the shape of an array. I used newArray.reshape(4, 4) and assigned the result with a variable named output2. Then, I printed the result. Finally I closed those files.

**Python Source Code:**

|  |
| --- |
| **import** numpy **as** np  *# Getting Set Input From file* file\_name1 = input(**"Enter First file name with extension : "**) *# code and input file should be in same folder* file\_name2 = input(**"Enter Second file name with extension : "**) *# code and input file should be in same folder  # reads input value from file* file1 = open(file\_name1,**"r"**) file2 = open(file\_name2,**"r"**) list1 = list(map(int, file1.readlines())) list2 = list(map(int, file2.readlines()))  *# using list comprehension* output1 = [(a, b) **for** a **in** list1  **for** b **in** list2 **if** a > b] data = [1 **if** a > b **else** 0 **for** a **in** list1  **for** b **in** list2]  *# Calculating matrix form relation of R* newArray = np.array(data) output2 = newArray.reshape(3, 2)  *# Printing Result* print(**"\nRelation of R : "**,output1) print(**"\nRepresent relation of R in matrix form is : "**,output2) *# Closing files* file1.close() file2.close() |

**Input:**

**Files:**

data2.txt

1

2

data1.txt

1

2

3

Enter First file name with extension: data1.txt

Enter Second file name with extension: data2.txt

**Output:**

Relation of R: [(2, 1), (3, 1), (3, 2)]

Represent relation of R in matrix form is:

**Experiment No:** **03**

**Name of the Experiment:** Write a program for graph coloring by Welch- Powell’s algorithm.

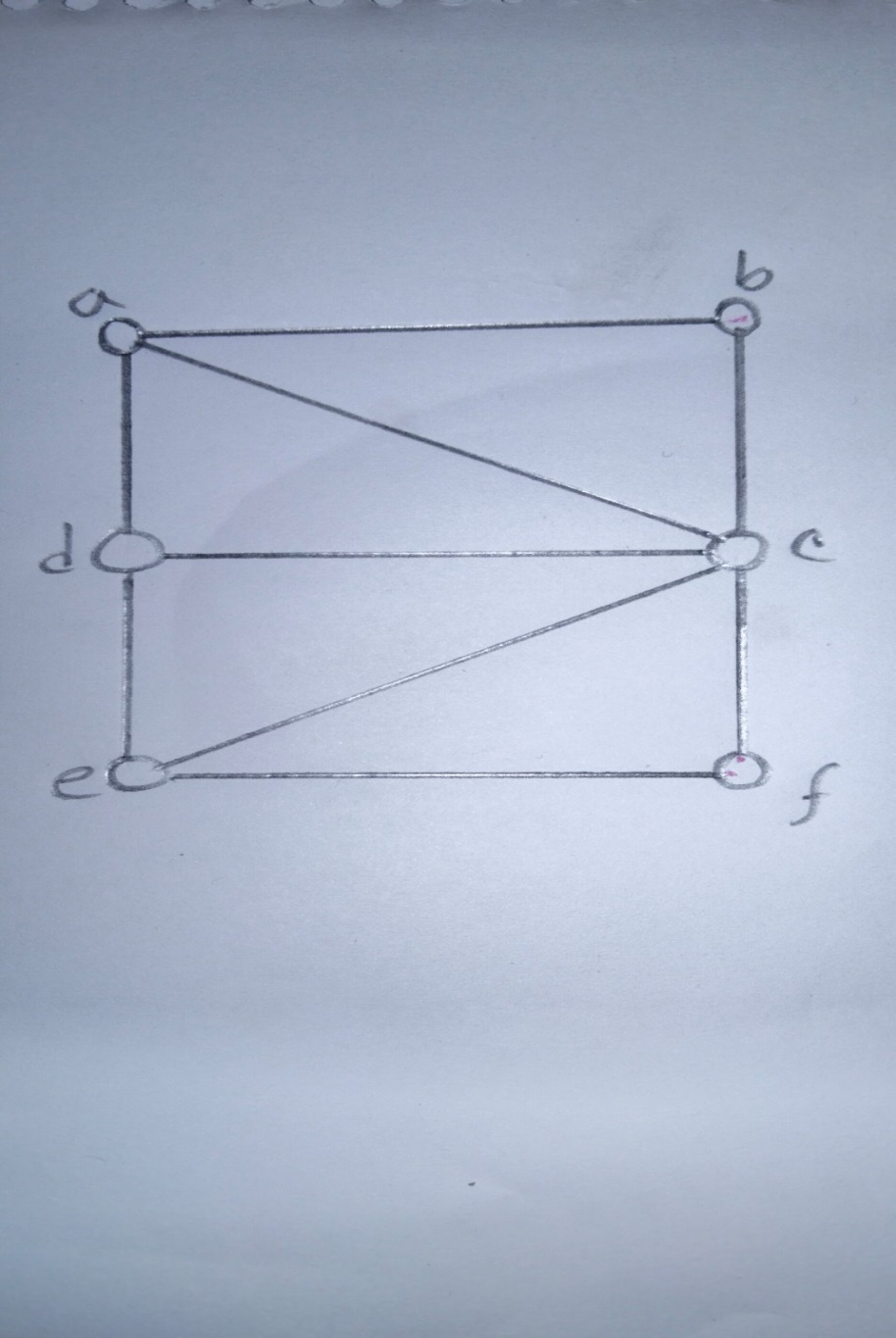
**Illustration:**

In graph theory, vertex coloring is a way of labeling each individual vertex such that no two adjacent vertexes have same color. But we need to find out the number of colors we need to satisfy the given condition. It is not desirable to have a large variety of colors or labels. So, we have an algorithm called welsh Powell algorithm that gives the minimum colors we need. This algorithm is also used to find the chromatic number of a graph. This is an iterative greedy approach.

**Welsh Powell Algorithm consists of following Steps:**

1. Find the degree of each vertex
2. List the vertices in order of descending degrees.
3. Color the first vertex with color 1.
4. Move down the list and color all the vertices not connected to the colored vertex, with the same color.
5. Repeat step 4 on all uncolored vertices with a new color, in descending order of degrees until all the vertices are colored.

By starting with the highest degree, we make sure that the vertex with the highest number of conflicts can be taken care of as early as possible.



***Fig-(i)***

|  |  |
| --- | --- |
| **Vertex** | **Degree** |
| a | 3 |
| b | 2 |
| c | 5 |
| d | 3 |
| e | 3 |
| f | 2 |

First, order the list in descending order of degrees. In case of tie, we can randomly choose any ways to break it.

So, the new order will be: c, a, d, e, b, f

Now, Following Welsh Powell Graph coloring Algorithm,

c – Color Pink

a – Don’t color pink, as it connects to c

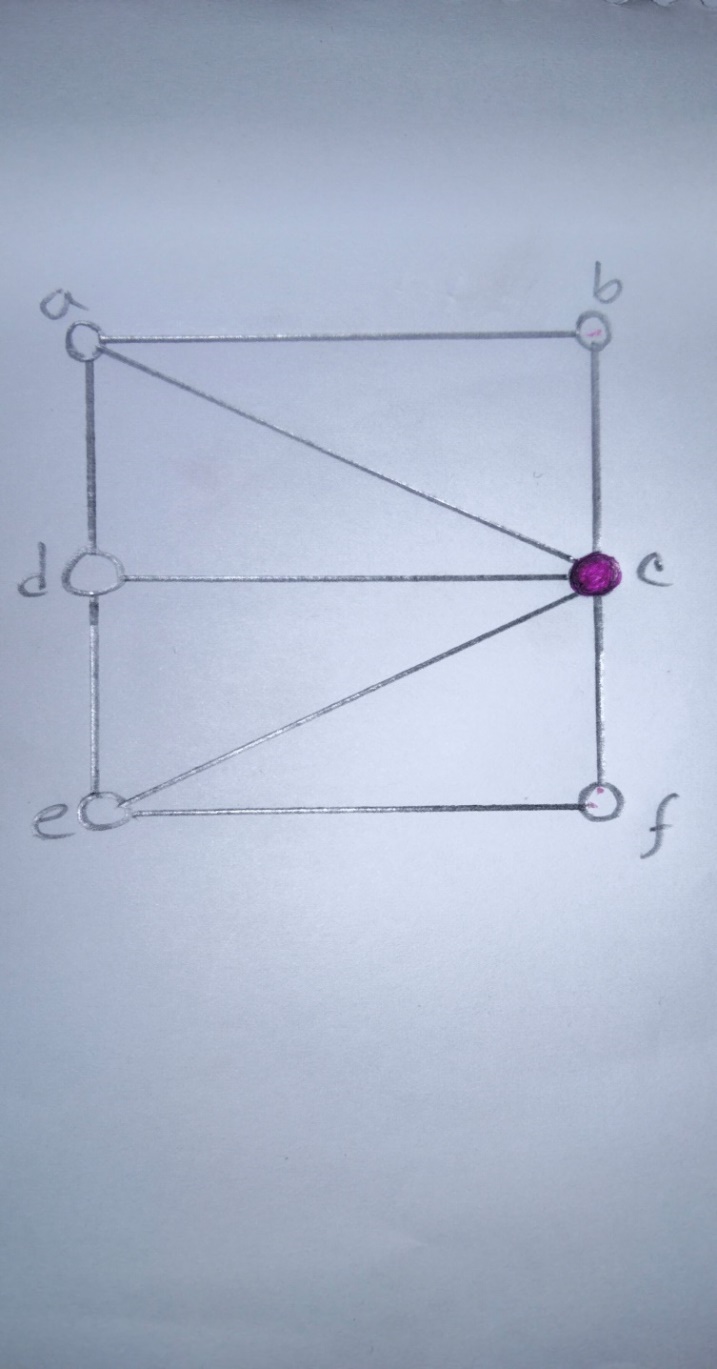
b – Don’t color pink, as it connects to c

d – Don’t color pink, as it connects to c

e – Don’t color pink, as it connects to c

f – Don’t color pink, as it connects to c

After this, the graph will look like the one below.



***Fig-(ii)***

Ignoring the vertices already colored, we are left with: a, d, e, b, f

We can repeat the process with the second color Green-

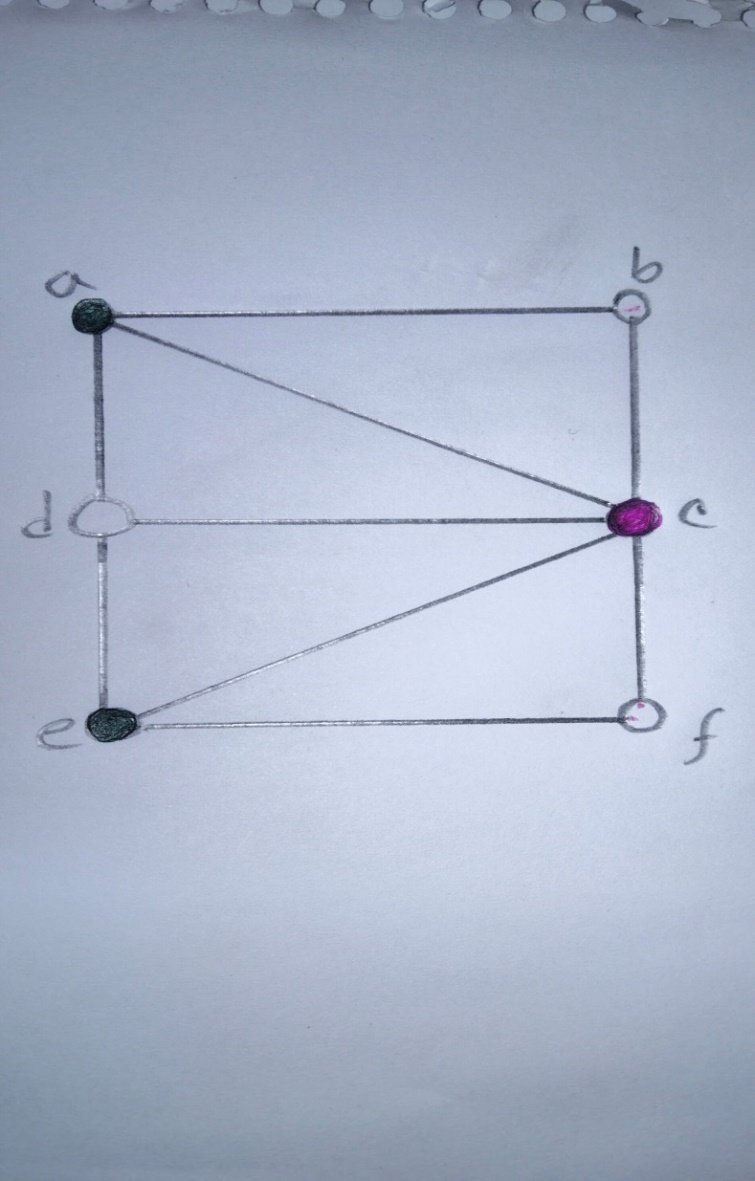
a – Color green

b – Don’t color green, as it connects with a

d – Don’t color green, as it connects with a

e – Color green

f – Don’t color green, as it connects to e



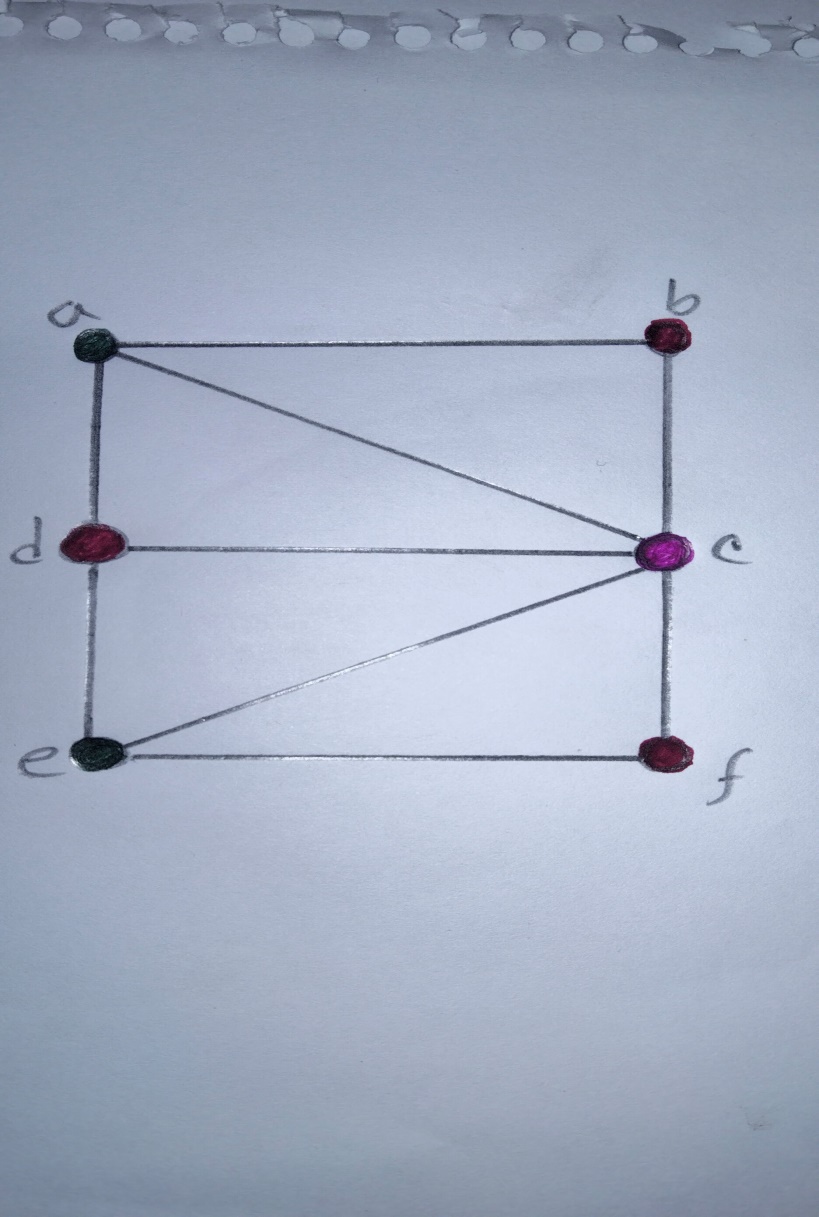
***Fig-(iii)***

Again, ignoring the colored vertices, we are left with d, e, b, f let’s color it with Red.

d – Color Red

b – Color Red

f – Color Red



***Fig-(iv)***

The final figure is shown below. Now, we can see that using Welsh Powell’s algorithm we can color the vertices with only 3 types of colors (chromatic number of this graph is 3) which is the optimal solution.

**Procedure:**

Firstly, I created a user define functions called color\_nodes for calculating graph coloring by Welch- Powell’s algorithm and should be used ‘def’ for definition keyword before function name for created this. This function contained not only one parameter but also received one argument named ‘graph’.

After that, I created an empty list and assigned with a variable named color\_map in the function. Again, I created a ‘for’ loop for consider nodes in descending degree (Most neighbors into Least neighbors) using sorted function that’s received three parameters such as iterable objects, key, reverse. Here iterable object is graph, key is ‘lambda’ function that calculated length of graph using len() function and reverse means ‘True’. Inside this ‘for’ loop I used a comprehension list for calculated neighbor colors and assigned the appropriate result with a variable named neighbor\_colors as a set. Then I also used a next function. Inside the next function I created a comprehension list which contained an ‘if’ statement that checked the condition “color **not in** neighbor\_colors” and assigned the appropriate result with a variable named color\_map. Finally, I returned the result of color\_map.

After that, I created a dictionary for stored the adjacent list and assigned this dictionary with a variable named graph. Dictionaries are used to stored data values in key: value pairs. A dictionary is a collection which is ordered\*, changeable and does not allow duplicates.

Afterwards, I called the function named color\_nodes with one argument named ‘graph’ for solution of graph coloring problem by ‘Welch-Powell’s” algorithm and printed the appropriate result of graph coloring problem using print function.

**Python Source Code:**

|  |
| --- |
| *# Function Declaration* **def** color\_nodes(graph):  color\_map = {}   *# Consider nodes in descending degree (Most neighbors .... Least neighbors)* **for** node **in** sorted(graph, key=**lambda** x: len(graph[x]), reverse=**True**):  neighbor\_colors = set(color\_map.get(neigh) **for** neigh **in** graph[node])  color\_map[node] = next(  color **for** color **in** range(len(graph)) **if** color **not in** neighbor\_colors  )  **return** color\_map  *# Adjacent list* graph = {**'a'**:list(**'bcd'**),**'b'**: list(**'ac'**),**'c'**: list(**'abdef'**),**'d'**: list(**'ace'**),**'e'**: list(**'cdf'**),**'f'**: list(**'ce'**)} *# Printing result* print(**"The solution of graph coloring problem by \'Welch-Powell’s\' algorithm is : \n"**,12 \***"\t"**,color\_nodes(graph)) |

**Output:**

**The solution of graph coloring problem by 'Welch-Powell’s' algorithm is:**

{'c': 0, 'a': 1,’d’: 2, 'e': 1, 'b': 2, 'f': 2}

**Experiment No:** **04**

**Name of the Experiment:** Write a program to find shortest path by Wars hall’s algorithm.

**Illustration:**

Floyd-Wars hall Algorithm is an algorithm based on dynamic programming technique to compute the shortest path between all pair of nodes in a graph.

**Limitations:**

* The graph should not contain negative cycles.
* The graph can have positive and negative weight edges.

**For a graph with vertices:**

* Initialize the shortest paths between any 2 vertices with Infinity (INT.maximum).
* Find all pair shortest paths that use 0 intermediate vertices, and then find the shortest paths that use 1 intermediate vertex and so on, until using all N vertices as intermediate nodes.
* Minimize the shortest paths between any pairs in the previous operation.
* For any 2 vertices i and j, one should actually minimize the distances between this pair using the first K nodes, so the shortest path will be: minimum( D[i][k] + D[k][j], D[i][j]).

**Applications:**

The applications of Floyd Wars hall Algorithm is as follows:

* Detecting the Presence of a Negative Cycle
* Transitive Closure of a Directed Graph

**Procedure:**

Firstly, I initialized variable named INF = 1000000000 and I also created a user define functions called floyd\_warshall for calculating shortest path by Wars hall’s algorithm and should be used ‘def’ for definition keyword before function name for created this. This function contained two parameters ‘vertex’ and ‘adjacency\_matrix’ and received two arguments ‘4’ and ‘adjacency\_matrix’.

After that, I created a ‘for’ loop that ran value of vertex that means four times. Inside this ‘for’ loop I created a nested ‘for’ loop that ran value of vertex and I also created ‘for’ loop that ran value of vertex that means four times under this nested ‘for’ loop for calculating all pair shortest path. Under this ‘for’ loop that’s mean in the most inner ‘for’ loop, I calculated minimum value between the “adjacency\_matrix[i][j]” and “adjacency\_matrix[i][k] + adjacency\_matrix[k][j]” using formula “min(adjacency\_matrix[i][j], adjacency\_matrix[i][k] + adjacency\_matrix[k][j])” following the steps relax the distance from i to j by allowing vertex k as intermediate vertex and consider which one is better, going through vertex k or the previous value for find shortest path by “Wars hall’s” algorithm and assigned the result with a variable named adjacency\_matrix[i][j].

Thus, I printed the pretty graph of shortest path by ‘Wars hall’s’ algorithm in the function. Inside the print function I used string formatting command {:d} that used to string number (integer) formatting. In this graph o/d means the leftmost row is the origin vertex and the topmost column as destination vertex.

Afterwards, I initialized the variable named adjacency\_matrix = [

[ 0, 5, INF, 10 ],

[ INF, 0, 3, INF],

[ INF, INF, 0, 1 ],  
 [ INF, INF, INF, 0 ]  
 ]

Finally, I called the function named Floyd\_warshall with two arguments ‘4’ and ‘adjacency\_matrix’ for appropriate result.

**Python Source Code:**

|  |
| --- |
| INF = 1000000000 **def** floyd\_warshall(vertex, adjacency\_matrix):   *# calculating all pair shortest path* **for** k **in** range(0, vertex):  **for** i **in** range(0, vertex):  **for** j **in** range(0, vertex):   *# relax the distance from i to j by allowing vertex k as intermediate vertex  # consider which one is better, going through vertex k or the previous value* adjacency\_matrix[i][j] = min(adjacency\_matrix[i][j], adjacency\_matrix[i][k] + adjacency\_matrix[k][j])   *# pretty print the graph  # o/d means the leftmost row is the origin vertex  # and the topmost column as destination vertex* print(**"Shortest path by \'Wars hall’s\' algorithm is : \n"**)  print(**"\t\t\t\t\t\t\t\t\t\to/d"**, end=**''**)  **for** i **in** range(0, vertex):  print(**"\t\t\t{:d}"**.format(i+1), end=**'\t '**)  print();  **for** i **in** range(0, vertex):  print(**"\t\t\t\t\t\t\t\t\t\t{:d}"**.format(i+1), end=**''**)  **for** j **in** range(0,vertex):  **if** i==0:  print(end=**" \t"**)  **if** j>=1 **and** i<=j:  print(end=**"\t "**)  **if** i ==1 **and** j == 2 **or** i ==1 **and** j == 3 **or** i ==2 **and** j == 3:  print(end=**" \t"**)  print(**"\t\t{:d}"**.format(adjacency\_matrix[i][j]), end=**''**)  print();  **""" input is given as adjacency matrix, input represents this undirected graph  A--1--B  | /  3 /  | 1  | /  C--2--D should set infinite value for each pair of vertex that has no edge  """** *# Initialized variable* adjacency\_matrix = [  [ 0, 5, INF, 10],  [ INF, 0, 3, INF],  [ INF, INF, 0, 1],  [ INF, INF, INF, 0]  ] *# Calling function* floyd\_warshall(4, adjacency\_matrix); |

**Output:**

**Shortest path by 'Wars hall’s' algorithm is:**

o/d 1 2 3 4

1 0 5 8 9

2 1000000000 0 3 4

3 1000000000 1000000000 0 1

4 1000000000 1000000000 1000000000 0

**Experiment No:** **05**

**Name of the Experiment:** Suppose that the relations R1 and R2 on a set A are represented by the matrices-

And. Write a program to find the MR1∪R2 and MR1∩R2

**Illustration:**

The matrix of a relation on a set, which is a square matrix, can be used to determine whether the relation has certain properties.

**Union, intersection of relations**

* Suppose R1 and R2 are relations on a set A represented by MR1 and MR2
* The matrices representing the union and intersection of these relations are

MR1⋃R2 = MR1 MR2

MR1⋂R2 = MR1 MR2

**Example**

* Suppose that the relations R1 and R2 on a set A are represented by the matrices

What are the matrices for R1⋃R2 and R1⋂R2?

MR1⋃R2 = MR1 MR1⋂R2 = MR1

**Procedure:**

Firstly, I created two user define functions called matrix\_intersection and matrix\_union for calculated matrix intersection and union and should be used ‘def’ for definition keyword before function name for created this. Both function contained two parameters mat1 and mat2 and received two arguments matrix1 and matrix2.

Primarily, I created an empty list with a variable named mat\_inter in the function named matrix\_intersection and also created ‘for’ loop that ran length of rows times. Inside this ‘for’ loop I created an empty list with a variable named List and also created nested ‘for’ loop that ran length of columns times. I used an ‘and’ keyword for calculating Intersection between two matrix. Then I append the result with a variable named List. Afterwards, I append the result of List with a variable named mat\_inter using ‘append’ library function. Finally I returned the result of mat\_inter list.

Similarly, I created an empty list with a difference variable named mat\_union in another function named matrix\_union. I used same process again, but this time I used an ‘or’ keyword for calculating Union between two matrix. Finally I returned the result of mat\_union list.

After that, I declared two matrix such as matrix1 = [[1, 0, 1],[1, 0, 0],[0, 1, 1]] and matrix2 = [[1, 0, 1],[ 0, 1, 1],[1, 0, 1]]. At first I printed first matrix, second matrix, number of rows and number of columns respectively. Then I called the first function named matrix\_intersection with two arguments such as matrix1 and matrix2 for calculating intersection and printed matrix intersection = [[1, 0, 1],[0, 0, 0],[0, 0, 1]]. Again I called the second function named matrix\_union with two arguments such as matrix1 and matrix2 for calculating union and printed matrix union = [[1, 0, 1],[1, 1, 1],[1, 1, 1]].

**Python Source Code:**

|  |
| --- |
| *# First Function* **def** matrix\_intersection(mat1, mat2):  mat\_inter = []  **for** rows **in** range(len(mat1)):  List = []  print(4 \* **"\t"**, end=**" "**)  **for** cols **in** range(len(mat1[0])):  List.append(mat1[rows][cols] **and** mat2[rows][cols])  print(mat1[rows][cols] **and** mat2[rows][cols], end=**" "**) *# Calculating intersection* mat\_inter.append(List)  print()  **return** mat\_inter  *# second function* **def** matrix\_union(mat1, mat2):  mat\_union = []  **for** rows **in** range(len(mat1)):  List = []  print(4 \* **"\t"**, end=**" "**)  **for** cols **in** range(len(mat1[0])):  List.append(mat1[rows][cols] **or** mat2[rows][cols])  print(mat1[rows][cols] **or** mat2[rows][cols], end=**" "**) *# Calculating union* mat\_union.append(List)  print()  **return** mat\_union  *# Input declared Matrix1 and Matrix2* matrix1 = [[1, 0, 1],  [1, 0, 0],  [0, 1, 1]] matrix2 = [[1, 0, 1],  [0, 1, 1],  [1, 0, 1]]  *# printing First Matrix* print(**'First Matrix M\_R1: '**) **for** rows **in** range(len(matrix1)):  print(4\***"\t"**,end=**" "**)  **for** cols **in** range(len(matrix1[0])):  print(matrix1[rows][cols],end=**" "**)  print()  *# printing Second Matrix* print(**'Second Matrix M\_R2: '**) **for** rows **in** range(len(matrix2)):  print(4\***"\t"**,end=**" "**)  **for** cols **in** range(len(matrix2[0])):  print(matrix2[rows][cols],end=**" "**)  print()  *# printing Rows and Columns of matrix* print(**"\nRows:"**,len(matrix1),**"; Cols:"**,len(matrix1[0]))  *# printing Matrix Intersection* print(**'\nMatrix Intersection:'**) mi = matrix\_intersection(matrix1, matrix2)  *# printing Matrix Union* print(**'Matrix Union: '**) mu = matrix\_union(matrix1, matrix2) v = [**'p'**, **'q'**, **'r'**] r1 = [] **for** i **in** range(len(mi)):  **for** j **in** range(len(mi[0])):  **if** mi[i][j] == 1:  r1.append((v[i], v[j])) print(r1)  r2 = [] **for** i **in** range(len(mu)):  **for** j **in** range(len(mu[0])):  **if** mu[i][j] == 1:  r2.append((v[i], v[j]))  print(r2) |

**Input:**

matrix2 = [[1, 0, 1],  
 [0, 1, 1],  
 [1, 0, 1]]

matrix1 = [[1, 0, 1],  
 [1, 0, 0],  
 [0, 1, 1]]

**Output:**

First Matrix M\_R1:

Second Matrix M\_R2:

Rows: 3 ; Cols: 3

Matrix Intersection:

Matrix Union:

[('p', 'p'), ('p', 'r'), ('r', 'r')]

[('p', 'p'), ('p', 'r'), ('q', 'p'), ('q', 'q'), ('q', 'r'), ('r', 'p'), ('r', 'q'), ('r', 'r')]

**Experiment No:** **06**

**Name of the Experiment:** The following table gives the population of a town during the last six censuses. Write a Python program to find the population in the year of 1946 using Newton-Gregory forward interpolation formula-

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Year: | 1911 | 1921 | 1931 | 1941 | 1951 | 1961 |
| Population: | 12 | 15 | 20 | 27 | 39 | 52 |

**Illustration:**

Interpolation is the technique of estimating the value of a function for any intermediate value of the independent variable, while the process of computing the value of the function outside the given range is called extrapolation.

**Forward Differences**: The differences when denoted by are respectively, called the first forward differences. Thus the first forward differences are:

**Forward difference table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x** | **y** |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Newton’s Gregory Forward Interpolation Formula:**

This formula is particularly useful for interpolating the values of f(x) near the beginning of the set of values given. h is called the interval of difference and  Here a is first term.

Afterwards, the reason for the name “forward” interpolation formula lies in the fact that this formula contains values of the tabulated function from f(a) onward to the right and none to the left if this value. This formula is used mainly for interpolating the values of y near the beginning of a set of tabulated values and for extrapolating values of y a short distance backward (i.e. to the left) from f(a).

**Procedure:**

This program started with “from math import factorial”. Math is a Python library used for working with factorial calculation here. In Python we have long code that serve the purpose of factorial calculation, but they are timed and complexity to process. The means of “from math import factorial” is the math package can be referred to as factorial instead of math.

Firstly, I created a user define function called ‘u\_cal’ for easily calculating u and should be used ‘def’ for definition keyword before function name for created this. This function contained two parameters u and n and received two arguments u and i. Primarily, I assigned value of u with a variable named temp in the function named ‘u\_cal’ and also created ‘for’ loop that ran length of table times. Inside this ‘for’ loop I created equation ‘temp \* (u - i)’ and assigned the result with a variable named temp. Finally, I returned the result of temp.

After that, I created a dada.txt file in same folder and here putted values of the table Year = [1911, 1921, 1931, 1941, 1951, 1961] and Population = [12, 15, 20, 27, 39, 52] for getting input. Now, for reading the data.txt file primarily, I used input function for getting file name with extension from the user and putted the file name with a variable named file\_name. Then I used open method and assigned the file with a variable named file and also used read function for printed the values of file. After that I printed the values of table.

Then I used a ‘split’ function for putted values of the data.txt file as a list with a variable named data. I created two empty lists with different variables named x and r. I used a zip function for append values of year with a variable named x and also append values of population with a different variable named r in the empty list. Usually zip function receives one parameter such as iterables object and it returned a list. After that I initialized the values of population in the variable named y such as y[0][0] = 12, y[1][0] = 15, y[2][0] = 20, y[3][0] = 27, y[4][0] = 39, y[5][0] = 52 and I took value of x for interpolation from user using input function and assigned with a variable named value as a float number.

After that, I calculated the forward difference table applying Newton-Gregory forward interpolation formula y[j][i] = y[j + 1][i - 1] - y[j][i - 1] and printed the result of forward difference table.

Afterwards, I assigned to value of y[0][0] with a variable named sum and I also assigned result of the formula (value - x[0]) / (x[1] - x[0]) with a different variable named u. Then I created a ‘for’ loop that ran length of table times. Inside the ‘for’ loop applying equation “sum + (u\_cal(u, i) \* y[0][i]) / factorial(i)” and assigned the result with a variable named sum. Here, the means of “u\_cal(u, i)” is called function named ‘u\_cal’ with two arguments such as u and i. Finally, I printed the result of sum for interpolation.

**Python Source Code:**

|  |
| --- |
| **from** math **import** factorial  *# calculating u* **def** u\_cal(u, n):  temp = u;  **for** i **in** range(1, n):  temp = temp \* (u - i);  **return** temp;  *# read input value from file* file\_name = input(**"Enter file name with extension: "**) *# code and input file should be in same folder* file = open(file\_name, **"r"**) data = file.read() print(**"Values of table :"**) print(data) print() data = data.split() n = int(len(data)/2) x,r = [],[] **for** i, j **in** zip(data[0::2], data[1::2]):  x.append(int(i))  r.append(int(j))  *# y[][] is used for difference table* y = [[0 **for** i **in** range(n)]  **for** j **in** range(n)];  *# with y[][0] used for input* **for** i **in** range(n):  y[i][0] = r[i]  *# Value to interpolate at* value = float(input(**"Enter value of x for interpolation: "**));  print(**"\nForward difference table: "**) print(**"Here, x = Year and f(x) = Population"**)  *# Calculating the forward difference table* **for** i **in** range(1, n):  **for** j **in** range(n - i):  y[j][i] = y[j + 1][i - 1] - y[j][i - 1]  *# Displaying the forward difference table* formated\_table = [**"x"**, **"f(x)"**, **"∇f(x)"**] **for** i **in** range(2, n+1):  formated\_table.append(**"∇^"** + str(i) + **"f(x)"**) print(**" "**, end=**""**) **for** i **in** range(n+1):  print(formated\_table[i], end=**" \t"**); print()  **for** i **in** range(n):  print(x[i], end=**" \t"**);  **for** j **in** range(n - i):  **if** j>1:  print(end=**"\t"**)  print(y[i][j], end=**"\t\t"**);  print();  *# initializing u and sum* sum = y[0][0]; u = (value - x[0]) / (x[1] - x[0]);  *# Result Calculation* **for** i **in** range(1, n):  sum = sum + (u\_cal(u, i) \* y[0][i]) / factorial(i);  *# Printing Result* print(**"\nValue at"**, value,  **"is"**, round(sum, 6)); |

**Input:**

**File:**

data.txt

1911 12  
1921 15  
1931 20  
1941 27  
1951 39  
1961 52

**Enter file name with extension:** data.txt

**Enter value of x for interpolation:** 1946

**Output:**

Values of table:

1911 12

1921 15

1931 20

1941 27

1951 39

1961 52

**Forward difference table:**

Here, x = Year and f(x) = Population

**x f(x) ∇f(x) ∇^2f(x) ∇^3f(x) ∇^4f(x) ∇^5f(x)**

1911 12 3 2 0 3 -10

1921 15 5 2 3 -7

1931 20 7 5 -4

1941 27 12 1

1951 39 13

1961 52

**Value at 1946.0 is** 32.34375

**Experiment No:** **07**

**Name of the Experiment:** Write a Python program to find ***f(7.5)*** form the following table using Newton-Gregory backward interpolation formula-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *x*: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *f(x)*: | 1 | 8 | 27 | 64 | 125 | 216 | 343 | 512 |

**Illustration:**

Interpolation is the technique of estimating the value of a function for any intermediate value of the independent variable, while the process of computing the value of the function outside the given range is called extrapolation**.**

**Backward Differences:** The differences when denoted by , respectively, are called first backward difference. Thus the first backward differences are:

**Backward difference table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x** | **y** |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Newton’s Gregory Backward Interpolation Formula:**

This formula is useful when the value of f(x) is required near the end of the table. h is called the interval of difference and Here ‘an’ is last term.

Afterwards, it is called the formula for “backward” interpolation because it contains values of the tabulated function from *f(a + nh)* backward to the left and none to the right of *f(a + nh).* This formula is used mainly for interpolating values of y near the of a set of tabulated values, and also for extrapolating values of y a short distance ahead (to the right) of *f(a + nh)*.

**Procedure:**

This program started with “from math import factorial”. Math is a Python library used for working with factorial calculation here. In Python we had long code that serve the purpose of factorial calculation, but they are timed and complexity to process. The means of “from math import factorial” is the math package can be referred to as factorial instead of math.

Firstly, I created a dada.txt file in same folder and here putted values of the table x = [1, 2, 3, 4, 5, 6, 7, 8] and f(x) = [1, 8, 27, 64, 125, 216, 343, 512] for getting input. Now, for reading the data.txt file primarily, I used input function for getting file name with extension from the user and putted the file name with a variable named file\_name. Then I used open method and assigned the file with a variable named file and also used read function for printed the values of file. After that I printed the values of table.

Then I used a ‘split’ function for putted values of the data.txt file as a list with a variable named data. I created two empty lists with different variables named x and y. I used a zip function for append values of x with a variable named x and also append values of f(x) with a different variable named y in the empty list. Usually zip function received one parameter such as iterables object and it returned a list. After that, I took value of x for interpolation from user using input function and assigned with a variable named inp as a float number.

After that, I calculated the backward difference table applying Newton-Gregory backward interpolation formula. Primarily, I assigned to value of y with a variable named table as a list and I also created a ‘for’ loop that ran ((length of y) – 1) times. Inside the ‘for’ loop I created an empty list with a variable named yn and also created a nested ‘for’ loop that contained a zip function. In this ‘for’ loop calculated the equation (i - k) and append the result in the yn empty list. Finally, I append the result of yn in the variable named table and printed the result of backward difference table.

Afterwards, I assigned to value of ‘1’ with a variable named r\_component and result of the formula “(inp - x[-1]) / (x[1] - x[0]) with a different variable named r and I also initialized ‘r\_component = 0’. Then I created a ‘for’ loop that ran length of table times. Inside the ‘for’ loop applying equation “r\_component \* (r + i - 1)” and assigned the result with a variable named r\_component and I also applying the equation “partial\_result + (table[i][-1] \* r\_component) / factorial(i)” and assigned the result with a different variable named partial\_result. Finally, I calculated the equation “table[0][-1] + partial\_result” and assigned the result with a variable named final\_result. At last, I printed the result of final\_result for interpolation.

**Python Source Code:**

|  |
| --- |
| **from** math **import** factorial *# read input value from file* file\_name = input(**"Enter file name with extension: "**) *# code and input file should be in same folder* file = open(file\_name, **"r"**) data = file.read() print(**"Values of table :"**) print(data,end=**"\n"**) data = data.split() x, y = [], [] **for** i, j **in** zip(data[0::2], data[1::2]):  x.append(int(i))  y.append(int(j))  *# Value to interpolate at* inp = float(input(**"Enter value of x for interpolation: "**)) print(**"\nBackward difference table: "**)  *# Calculating the backward difference table* table = [y] **for** l **in** range(len(y) - 1):  yn = []  **for** i, k **in** zip(y[1::1], y[0::1]):  yn.append(i - k)  table.append(yn)  y = yn  *# Displaying the backward difference table* formated\_table = [**"x"**, **"f(x)"**, **"∇f(x)"**] **for** i **in** range(2, len(table)+1):  formated\_table.append(**"∇^"** + str(i) + **"f(x)"**) **for** i **in** range(len(table)+1):  print(formated\_table[i], end=**" \t"**); print() **for** i **in** range(len(table)):  print(x[i], end=**" \t"**);  **for** j **in** range(len(table) - i):  **if** j>1:  print(end=**"\t"**)  print(table[j][i], end=**"\t\t"**);  print();  *# calculation of r* r = (inp - x[-1]) / (x[1] - x[0])  *# result calculation* r\_component = 1 partial\_result = 0 **for** i **in** range(1, len(table)):  r\_component = r\_component \* (r + i - 1)  partial\_result = partial\_result + (table[i][-1] \* r\_component) / factorial(i) final\_result = table[0][-1] + partial\_result  *# Printing Result* print(**"\nValue of"**, **"f("**,inp,**") is : "**, final\_result); |

**Input:**

**File:**

data.txt

1 1  
2 8  
3 27  
4 64  
5 125  
6 216  
7 343  
8 512

**Enter file name with extension:** data.txt

**Enter value of x for interpolation:** 7.5

**Output:**

**Values of table:**

1 1

2 8

3 27

4 64

5 125

6 216

7 343

8 512

**Backward difference table:**

**x f(x) ∇f(x) ∇^2f(x) ∇^3f(x) ∇^4f(x) ∇^5f(x) ∇^6f(x) ∇^7f(x)**

1 1 7 12 6 0 0 0 0

2 8 19 18 6 0 0 0

3 27 37 24 6 0 0

4 64 61 30 6 0

5 125 91 36 6

6 216 127 42

7 343 169

8 512

**Value of *f*(7.5) is :** 421.875

**Experiment No:** **08**

**Name of the Experiment:** Write a Python program to find the value of *f(15)* from the following table using Newton’s divided difference formula for unequal interval-

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *x:* | 4 | 5 | 7 | 10 | 11 | 13 |
| *f(x):* | 48 | 100 | 294 | 900 | 1210 | 2028 |

**Illustration:**

Interpolation is an estimation of a value within two known values in a sequence of values.

Newton’s divided difference interpolation formula is an interpolation technique used when the interval difference is not same for all sequence of values.

Suppose be the (n+1) values of the function y=f(x) corresponding to the arguments where interval differences are not same.

**Then the first divided difference is given by**

**The second divided difference is given by**

And so on…

Divided differences are symmetric with respect to the arguments i.e. **independent of the order of arguments.**

So,

By using first divided difference, second divided difference as so on .A table is formed which is called the divided difference table.

**Divided difference table:**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Newton’s divided difference interpolation formula:**

**Procedure:**

Firstly, I created a user define function called ‘u\_cal’ for easily calculating and should be used ‘def’ for definition keyword before function name for created this. This function contained three parameters i, value and x and received three arguments i, value and x. Primarily, I assigned to value of ‘1’ with a variable named pro in the function named ‘u\_cal’ and also created ‘for’ loop that ran value of ‘i’ times. Inside this ‘for’ loop I created equation ‘pro \* (value - x[j])’ and assigned the result with a variable named pro. Finally, I returned the result of pro.

After that, I created a dada.txt file in same folder and here putted values of the table x = [4, 5, 7, 10, 11, 13] and *f(x)* = [48, 100, 294, 900, 1210, 2028] for getting input. Now, for reading the data.txt file primarily, I used input function for getting file name with extension from the user and putted the file name with a variable named file\_name. Then I used open method and assigned the file with a variable named file and also used read function for printed the values of file. After that I printed the values of table.

Then I used a ‘split’ function for putted values of the data.txt file as a list with a variable named data. I created two empty lists with different variables named x and r. I used a zip function for append values of x with a variable named x and also append values of *f(x)* with a different variable named r in the empty list. Usually zip function received one parameter such as iterables object and it returned a list. After that I initialized the values of population in the variable named y such as *y[0][0] = 48, y[1][0] = 100, y[2][0] = 294, y[3][0] = 900, y[4][0] = 1210, y[5][0] = 2028* and I took value of x for interpolation from user using input function and assigned with a variable named value as a float number.

After that, I calculated the divided difference table applying Newton’s divided difference formula for unequal interval *y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j]))* and printed the result of divided difference table.

Afterwards, I assigned to value of y[0][0] with a variable named sum. Then I created a ‘for’ loop that ran length of table times. Inside the ‘for’ loop applying equation “sum + (u\_cal(i, value, x) \* y[0][i])” and assigned the result with a variable named sum. Here, the means of “u\_cal(I, value, x)” is called function named ‘u\_cal’ with three arguments such as i, value and x. Finally, I printed the result of sum for interpolation.

**Python Source Code:**

|  |
| --- |
| *# function defined* **def** u\_cal(i, value, x):  pro = 1  **for** j **in** range(i):  pro = pro \* (value - x[j])  **return** pro  *# read input value from file* file\_name = input(**"Enter file name with extension: "**) *# code and input file should be in same folder* file = open(file\_name, **"r"**) data = file.read() print(**"Values of table :"**) print(data,end=**"\n"**) data = data.split() n = int(len(data)/2) x,r = [],[] **for** i, j **in** zip(data[0::2], data[1::2]):  x.append(int(i))  r.append(int(j))  *# y[][] is used for divided difference table* y = [[0 **for** i **in** range(n)]  **for** j **in** range(n)];  *# with y[][0] used for input* **for** i **in** range(n):  y[i][0] = r[i]  *# Value to interpolate at* value = float(input(**"Enter value of x for interpolation: "**)); print(**"\nDivided difference table: "**) *# calculating divided difference table* **for** i **in** range(1, n):  **for** j **in** range(n - i):  y[j][i] = ((y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j]))  *# displaying divided difference table* formated\_table = [**"x"**, **"f(x)"**, **"∇f(x)"**] **for** i **in** range(2, n+1):  formated\_table.append(**"∇^"** + str(i) + **"f(x)"**) **for** i **in** range(n+1):  print(formated\_table[i], end=**"\t\t"**); print()  **for** i **in** range(n):  print(x[i], end=**"\t\t "**);  **for** j **in** range(n - i):  print(y[i][j], end=**" \t\t"**);  print();  *# initializing sum* sum = y[0][0]; *# applying Newton's divided difference formula # Result Calculation* **for** i **in** range(1, n):  sum = sum + (u\_cal(i,value,x) \* y[0][i]);  *# Printing Result* print(**"\nValue at"**, value,  **"is"**, round(sum, 2)); |

**Input:**

**data.txt**

4 48  
5 100  
7 294  
10 900  
11 1210  
13 2028

**Enter file name with extension:** data.txt

**Enter value of x for interpolation:** 15

**Output:**

**Values of table:**

4 48

5 100

7 294

10 900

11 1210

13 2028

**Divided difference table:**

**x f(x) ∇f(x) ∇^2f(x) ∇^3f(x) ∇^4f(x) ∇^5f(x)**

4 48 52.0 15.0 1.0 -0.0 -0.0

5 100 97.0 21.0 1.0 -0.0

7 294 202.0 27.0 1.0

10 900 310.0 33.0

11 1210 409.0

13 2028

**Value at 15.0 is** 3150.0

**Experiment No:** **9**

**Name of the Experiment:** Write a Python program to find the value of y when x=10 from following table using Lagrange’s interpolation formula for unequal intervals-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *x:* | 5 | 6 | 9 | 11 |
| *y:* | 12 | 13 | 14 | 16 |

**Illustration:**

Interpolation is a method of finding new data points within the range of a discrete set of known data points. In other words interpolation is the technique to estimate the value of a mathematical function, for any intermediate value of the independent variable.  
For example, in the given table we’re given 4 set of discrete data points, for an unknown function f(x):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 |
|  | 0 | 1 | 2 | 5 |
|  | 2 | 3 | 12 | 147 |

**How to find?**  
Here we can apply the Lagrange’s interpolation formula to get our solution.  
The Lagrange’s Interpolation formula:  
If, y = f(x) takes the values corresponding to x = then,

This is Lagrange’s interpolation formula and can be used for both equal and unequal intervals. This method is preferred over its counterparts like Newton’s method because it is applicable even for unequally spaced values of x.

**The main uses of “Lagrange’s” formula are:**

1. To find any value of a function when the given values of the independent variable are not equidistant.
2. To find the value of the independent variable corresponding to a given value of the function.

**Afterwards,** we notice that Lagrange’s formula is tedious to apply and involves a great deal of computation. It must also be used with care and caution, tor if the values of the independent variable are not taken close together the results are liable to be very inaccurate.

**Procedure:**

The program started with a class named ‘Data’. Under this class I created a parameterized constructor. This constructor had received two parameters namely, x and y. I used a comment above the constructor for easily understanding this code. Constructors are generally used for instantiating an object. The task of constructors is to initialized (assign values) to the data members of the class when an object of class is created. I assigned the receivable x and y into this class variable x and y respectively, this means self.x = x and self.y = y.

After that, I created a user defined function named ‘interpolate’. It has received three parameters, namely, f, xi and n. Here, f: list means f is a list as well as xi: int and n: int means xi and n is integer. Also I used a type converter symbol (->), that’s mean all of the parameterized variables type is changed to be float. Under this function I created a variable named result and assign the value is result = 0.0.

Under interpolate function I also used a ‘for’ loop. Under this for loop I declared a variable named ‘term’ and compute individual terms of Lagrange’s Interpolation formula and assigned the value into ‘term’ variable. This ‘for’ loop is execute n times which is the interpolate function variable n.

After that, I used another ‘for’ loop under first ‘for’ loop that’s mean I used nested for loop. This for loop is also executes n times. Under this for loop I used ‘if’ statement that checked the condition “j != i”. If the condition is true then Lagrange’s Interpolation formula will be evaluated. This process execute continues until the 'for' loop is false. Then put the term variable value into the result variable. This processed is continues n times that’s mean four times because the value of n is four. And finally returned the result variable.

Afterwards, I created a list named ‘f’ and into the list I called the Data class and also initialized the value. I called the Data class four times.

Finally, I called the interpolate function with given three arguments and same time also printed the interpolate function.

**Python Source Code:**

|  |
| --- |
| *# Created class* **class** Data:  *# Initialized Constructor* **def** \_\_init\_\_(self, x, y):  self.x = x  self.y = y  **def** interpolate(f: list, xi: int, n: int) -> float:  result = 0.0  **for** i **in** range(n):  term = f[i].y  **for** j **in** range(n):  **if** j != i:  term = term \* (xi - f[j].x) / (f[i].x - f[j].x)  result += term   **return** result  **if** \_\_name\_\_ == **"\_\_main\_\_"**:  f = [Data(5, 12), Data(6, 13), Data(9, 14), Data(11, 16)]  *# printing result* print(**"Value of f(10) is :"**, interpolate(f, 10, len(f))) |

**Output:**

Value of f(10) is : 14.666666666666666

**Experiment No:** **10**

**Name of the Experiment:** Write a Python program to find a real root of the equation that lies between -1 and 3 using bisection method.

**Illustration:**

Given a function f(x) on floating number x and two numbers ‘a’ and ‘b’ such that *f(a)\*f(b)* < *0* and *f(x)* is continuous in [a, b]. Here *f(x)* represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

**Example:**

**Input:** A function of x, for example and two values: a = -200 and b = 300 such that

*f(a)\*f(b) < 0*, i.e., f(a) and f(b) have opposite signs.

**Output:** The value of root is: -1.0025. OR any other value with allowed deviation from root.

**What are Algebraic and Transcendental functions?**

Algebraic function are the one which can be represented in the form of polynomials like *f(x)*= where ,,… are constants and x is a variable.

Transcendental functions are non-algebraic functions, for example *f(x)* = sin(x)\*x – 3 or f(x) = or f(x) = ln(x) + x ….

**What is Bisection Method?**

The method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of ‘x’ for which f(x) = 0.

The method is based on **The Intermediate Value Theorem** which states that if *f(x)* is a continuous function and there are two real numbers a and b such that *f(a)\*f(b)* < 0 and *f(b) < 0),* then it is guaranteed that it has at least one root between them.

**Assumptions:**

1. *f(x)* is a continuous function in interval [a, b]

2. *f(a) \* f(b) < 0*

**Steps:**

1. Find middle point c = .

2. If *f(c) == 0*, then c is the root of the solution.

3. Else *f(c) != 0*

1. If value *f(a)\*f(c) < 0* then root lies between a and c. So we recur for a and c

2. Else If *f(b)\*f(c) < 0* then root lies between b and c. So we recur for b and c.

3. Else given function doesn’t follow one of assumptions.

Since root may be a floating point number, we repeat above steps while difference between ‘a’ and b is less than ‘a’ value (A very small value).

**Procedure:**

Firstly, I created two user define functions called func and bisection for find a real root of the equation using bisection method and should be used ‘def’ for definition keyword before function name for created this. The func function contained one parameter such as x and received one argument but bisection function contained two parameters ‘a’ and ‘b’ and received two arguments a and b.

Primarily, I created an equation “x \* x \* x - 2 \* x – 5” in the function named func and I returned the result of equation.

After that, I used an ‘if’ statement where checked the condition “func(a) \* func(b) >= 0” in the function named bisection. If it returned ‘true’ then, it had printed “You have not assumed right a and b”. Then I used return keyword that means finished the work of function named bisection. Otherwise I assigned the value of ‘a’ with a variable named c. Hence I created a ‘while’ loop that checked the condition “(b - a) > = 0.0001” and if it returned false then, it had dismissed. Then I calculated a middle point using formula “(a + b) / 2” and assigned the result with a variable named c. Again, I used an ‘if’ statement where checked the condition “func(c) == 0.0” in this loop and if it returned ‘true’ then I had used break keyword that means this loop stopped. After that, I also used an ‘if’ statement where checked the condition “func(c) \* func(a) < 0” in this loop and if it returned ‘true’ then I had assigned value of c with a variable named b. Else I assigned the value of c with a different variable named a. Finally, I printed the result of c for a real root of the equation using bisection method.

Afterwards, I initialized a = -1 and b = 3 that had assumed. Then I called the function named bisection with two arguments ‘a’ and ‘b’ for find the real root of the equation using bisection method.

**Python Source Code:**

|  |
| --- |
| *# Python program for implementation of Bisection Method for solving equations # an example function whose solution is determined using Bisection Method. # The function is x^3 - 2x - 5 = 0* **def** func(x):  **return** x \* x \* x - 2 \* x - 5  *# Prints root of func(x) with error of EPSILON* **def** bisection(a, b):  **if** func(a) \* func(b) >= 0:  print(**"You have not assumed right a and b\n"**)  **return** c = a  **while** (b - a) >= 0.0001:   *# Find middle point* c = (a + b) / 2   *# Check if middle point is root* **if** (func(c) == 0.0):  **break** *# Decide the side to repeat the steps* **if** (func(c) \* func(a) < 0):  b = c  **else**:  a = c  *# printing result* print(**"The value of root is : "**, **"%.4f"** % c)  *# Driver code # Initial values assumed* a = -1 b = 3 *# calling function* bisection(a, b) |

**Input:**

**a = -1; b = 3;**

**Output:**

The value of root is: 2.0945

**Experiment No:** **11**

**Name of the Experiment:** Write a Python program to find a real root of the function in the range 1<x<3 using false position method.

**Illustration:**

Given a function f(x) on floating number x and two numbers ‘a’ and ‘b’ such that f(a)\*f(b) < 0 and f(x) is continuous in [a, b]. Here f(x) represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

Input: A function of x, for example and two values: a = -200 and b = 300 such that

*f(a)\*f(b)* < 0, i.e., f(a) and f(b) have opposite signs.

Output: The value of root is: -1.00. OR any other values close to root.

**Similarities with Bisection Method:**

1. Same Assumptions: This method also assumes that function is continuous in [a, b] and given two numbers 'a' and 'b' are such that *f(a) \* f(b) < 0.*

2. Always Converges: like Bisection, it always converges, usually considerably faster than Bisection--but sometimes very much more slowly than Bisection.

**Differences with Bisection Method:**   
It differs in the fact that we make a chord joining the two points [a, f(a)] and [b, f(b)]. We consider the point at which the chord touches the x axis and named it as c.

**Steps:**

**1**. Write equation of the line connecting the two points.

y – f(a) = ( (f(b)-f(a))/(b-a) )\*(x-a)

Now we have to find the point which touches x axis.

For that we put y = 0.

so x = a - (f(a)/(f(b)-f(a))) \* (b-a)

x = (a\*f(b) - b\*f(a)) / (f(b)-f(a))

This will be our c that is c = x.

**2. If** f(c) == 0, then c is the root of the solution.

**3. Else** f(c) != 0

**1. If** value f(a)\*f(c) < 0 then root lies between a and c. So we recur for a and c

**2. Else If** f(b)\*f(c) < 0 then root lies between b and c. So we recur for b and c.

**3. Else** given function doesn't follow one of assumptions.

Since root may be a floating point number and may converge very slow in worst case, we iterate for a very large number of times such that the answer becomes closer to the root.

**Procedure:**

Firstly, I initialized a variable MAX\_ITER = 1000000 and created two user define functions called func and regulaFalsi for find a real root of the equation in the range 1<x<3 using false position method and should be used ‘def’ for definition keyword before function name for created this. The func function contained one parameter such as x and received one argument but bisection function contained two parameters ‘a’ and ‘b’ and received two arguments ‘a’ and ‘b’.

Primarily, I created an equation “x \* x \* x - 2 \* x – 5” in the function named func and I returned the result of equation.

After that, I used an ‘if’ statement where checked the condition “func(a) \* func(b) >= 0” in the function named regulaFalsi. If it returned ‘true’ then, it had printed “You have not assumed right a and b”. Then I used return keyword that means finished the work of function named regulaFalsi. Otherwise I assigned the value of ‘a’ with a variable named c. Hence I created a ‘for’ loop that ran value of MAX\_ITER times. Then I calculated the result using formula “(a \* func(b) - b \* func(a))/ (func(b) - func(a))” and assigned the result with a variable named c. Again, I used an ‘if’ statement where checked the condition “func(c) == 0.0” in this loop and if it returned ‘true’ then I had used break keyword that means this loop dismissed. After that, I also used an ‘if’ statement where checked the condition “func(c) \* func(a) < 0” in this loop and if it returned ‘true’ then I had assigned value of c with a variable named b. Else I assigned the value of c with a different variable named a. Finally, I printed the result of c for a real root of the equation in the range 1<x<3 using false position method.

Afterwards, I initialized a = -200 and b = 300 that had assumed. Then I called the function named regulaFalsi with two arguments ‘a’ and ‘b’ for find the real root of the equation in the range 1<x<3 using false position method.

**Python Source Code:**

|  |
| --- |
| *# Python program for implementation of false position Method for solving equations # An example function whose solution is determined using false position Method.* MAX\_ITER = 1000000  *# The function is x^3 - 2x - 5 = 0* **def** func( x ):  **return** (x \* x \* x - 2 \* x -5)  *# Prints root of func(x) with error of EPSILON* **def** regulaFalsi( a , b):  **if** func(a) \* func(b) >= 0:  print(**"You have not assumed right a and b"**)  **return** c = a  **for** i **in** range(MAX\_ITER):  c = (a \* func(b) - b \* func(a))/ (func(b) - func(a))  **if** func(c) == 0:  **break** *# Decide the side to repeat the steps* **if** func(c) \* func(a) < 0:  b = c  **else**:  a = c  *# printing result* print(**"The value of root is : "** , **'%.4f'** %c)  *# Driver code # Initial values assumed* a =-200 b = 300 *# calling function* regulaFalsi(a, b) |

**Input:**

**a = -200; b = 300;**

**Output:**

The value of root is: 2.0946