Department of Information and Communication Engineering

Pabna University of Science and Technology

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

Session: 2018-2019

Course Code: ICE-2106 Title: Discrete Mathematics and Numerical Methods Sessional

Lab report

Submitted by **Md. Sohag Hossain**

Roll:190631
Registration no: 1065356
Dept. of Information and Communication Engineering
Pabna University of Science and Technology
Pabna-6600, Bangladesh
Email: mdsohaghosain.190631@s.pust.ac.bd
Mobile: 01648616567

Submitted to **Md. Anwar Hossain**

Associate Professor
Dept. of Information and Communication Engineering
Pabna University of Science and Technology
Pabna-6600, Bangladesh
Email: manwar.ice@gmail.com

Table of Contents

Problem number: 01

Problem title: Let A be the set $A = \{1, 2, 3, 4\}$. Write a program to find the ordered pairs are in the relation

I) $R1 = \{(a, b) \mid a \; divides \; b\}$ II) $R2 = \{(a, b) \mid a \leq b\}$.

Illustration of the problem:

A relation or a function is a set of ordered pairs. The set of all first coordinates of the ordered pairs is the domain of the relation or function and the second is range of the relation.

Algorithm:

1. Import product from itertools library.
2. Open the input file and read the elements as a list.
3. For relation 1: Divide the elements with each other and see if any pairs satisfies the condition i%j==0 and j%i==0 , then save them in res1.
4. For relation 2: check if any pair satisfies the condition i<=j, then save them in res2.
5. Print both results as output.

Python source code:

```
from itertools import product

with open("input.txt", "r", encoding="utf-8") as g:

        S = list(map(int, g.readlines()))

print("S= "+str(S))

res1=[(i,j) for i,j in product(S,repeat=2) if i%j==0 or j%i==0]

res2=[(i,j) for i,j in product(S,repeat=2) if i<=j]

print ("The pair list is for a/b : " + str(res1))

print ("The pair list is for a<=b : " + str(res2))
```

Sample input:

//input.txt

1

2

3

4

Sample output:

S= [1, 2, 3, 4]

The 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 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)]

Problem number: 02

Problem title: Suppose that A = {1, 2, 3} and B = {1, 2}. Let R be the relation from A to B containing (a, b) if $a \in A$, $b \in B$, 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 of the problem

A relation is a correspondence between two sets (called the domain and the range) such that to each element of the domain, there is assigned one or more elements of the range.

Algorithm

1. Import numpy into the code
2. Open the first and second sets and save them in list1 and list2 as list items.
3. To find the relation between them we check the items in both list as pairs and if any pair satisfies the condition a>b they are saved in the output list.
4. To represent the relation in matrix form we apply this condition: 1 if a>b else 0 for a in list1 for b in list2.
5. We reshape the matrix into 3x2.
6. We print both output and output2 as result.

Python source code

```
import numpy as np

with open("A.txt", "r", encoding="utf-8") as g:

    list1 = list(map(int, g.readlines()))

with open("B.txt", "r", encoding="utf-8") as g:

    list2 = list(map(int, g.readlines()))

output = [(a, b) for a in list1 for b in list2 if a > b]

output2 = [1 if a>b else 0 for a in list1 for b in list2]

data = np.array(output2).reshape(3,2)


print(output)
```

```
print(data)
```

sample input
//A.txt

1

2

3

//B.txt

1

2

Sample output
[(2, 1), (3, 1), (3, 2)]

[[0 0]

 [1 0]

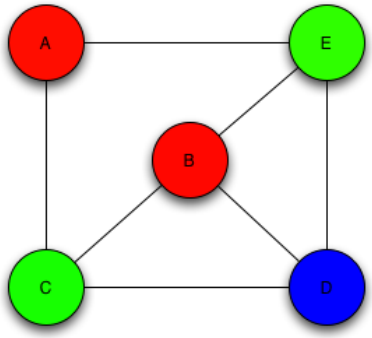
 [1 1]]

Problem number: 03

Problem title: Write a program for the solution of graph coloring problem by Welch-Powell's algorithm.

Illustration of the problem

In graph theory, vertex coloring is a way of labelling 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.



Algorithm:

1. Find the degree of each vertex.
2. List the vertices in order of descending valence.
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.

Python source code

```
def color_nodes(graph):

    nodes = sorted(graph.keys(),key=lambda x: len(graph[x]),reverse=True)

    color_map = {}

    for node in nodes:

        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


if __name__ == '__main__':
  graph = {
    'A': list('CE'),
    'B': list('ECD'),
    'C': list('ABD'),
    'D': list('BCE'),
    'E': list('ABD')
  }
print(color_nodes(graph))
```

Sample output
{'B': 0, 'C': 1, 'D': 2, 'E': 1, 'A': 0}

Problem number: 04

Problem title: Write a program to find shortest path by Warshall's algorithm.
Illustration of the problem
The Floyd Warshall Algorithm is for solving the All-Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

Algorithm

1. Initialize the shortest paths between any 2 vertices with Infinity.
2. Find all pair shortest paths that use 0 intermediate vertices, then find the shortest paths that use 1 intermediate vertex and so on, until using all N vertices as intermediate nodes.
3. Minimize the shortest paths between any pairs in the previous operation.
4. 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]).

Python source code

```
INF = 1000000000

def floyd_warshall(vertex, adjacency_matrix):
```

```
  for k in range(0, vertex):
    for i in range(0, vertex):
      for j in range(0, vertex):
        adjacency_matrix[i][j] = min(adjacency_matrix[i][j],
adjacency_matrix[i][k] + adjacency_matrix[k][j])
  print("o/d", end='')
  for i in range(0, vertex):
    print("\t{:d}".format(i+1), end='')
  print()

  for i in range(0, vertex):
    print("{:d}".format(i+1), end='')
    for j in range(0,vertex):
      print("\t{:d}".format(adjacency_matrix[i][j]), end='')
    print()

adjacency_matrix = [
          [  0,   5, INF, 10],
          [  INF,   0, 3, INF],
          [  INF,   INF, 0,   1],
          [INF, INF, INF,   0]
          ]
floyd_warshall(4, adjacency_matrix)
```

Sample output:

```
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

Problem number: 05

Problem title: Suppose that the relations R1 and R2 on a set A are represented by the matrices

$M_{R1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ and $M_{R2} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$. Write a program to find the $M_{R1 \cup R2}$ and $M_{R1 \oplus R2}$.

Illustration of the problem

Union: A list that has the common distinct element from both arrays and if there are repetitions of the element then only one occurrence is considered, known as the union of both arrays.

Intersection: A list that has common distinct elements from both arrays, is the intersection of both arrays.

Algorithm

1. Pseudocode for matrix intersection:

```
      mat_inter = []

    for i in range(len(mat1)):
       mat_inter.append([mat1[i][j] and mat2[i][j] for j in range(len(mat1[0]))])
```

Pseudocode for matrix union:

```
      mat_union = []

    for i in range(len(mat1)):
        mat_union.append([mat1[i][j] or mat2[i][j] for j in range(len(mat1[0]))])
```

Python source code

```
def matrix_intersection(mat1, mat2):

    rows = len(mat1)

    cols = len(mat1[0])

    print('Rows=', rows, 'Cols=', cols)

    mat_inter = []

    for i in range(len(mat1)):

        mat_inter.append([mat1[i][j] and mat2[i][j] for j in
range(len(mat1[0]))])
```

```
    return mat_inter

def matrix_union(mat1, mat2):
    mat_union = []
    for i in range(len(mat1)):
        mat_union.append([mat1[i][j] or mat2[i][j] for j in range(len(mat1[0]))])

    return mat_union


matrix1 = [[1, 0, 1],
           [1, 0, 0],
           [0, 1, 1]]
matrix2 = [[1, 0, 1],
           [0, 1, 1],
           [1, 0, 1]]

print('First Matrix=', matrix1)
print('Second Matrix=', matrix2)

mi = matrix_intersection(matrix1, matrix2)
print('Matrix Intersection', mi)

mu = matrix_union(matrix1, matrix2)
print('Matrix Union', mu)
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)
```

Sample output:
First Matrix= [[1, 0, 1], [1, 0, 0], [0, 1, 1]]

Second Matrix= [[1, 0, 1], [0, 1, 1], [1, 0, 1]]

Rows= 3 Cols= 3

Matrix Intersection [[1, 0, 1], [0, 0, 0], [0, 0, 1]]

Matrix Union [[1, 0, 1], [1, 1, 1], [1, 1, 1]]

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

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

Problem number: 06

Problem title: 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 of the problem

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 y1 – y0, y2 – y1, y3 – y2, ……, yn – yn–1 when denoted by dy0, dy1, dy2, ……, dyn–1 are respectively, called the first forward differences.

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.

Algorithm

1. Use x as an array for the years. And y is a matrix of the values of the years.
2. Calculate the forward difference.
3. The required year is saved as value.
4. Apply the required calculation for the population of the year.
5. Print the value and sum rounded.

Python source code

```
def u_cal(u, n):

    temp = u

    for i in range(1, n):

        temp = temp * (u - i)

    return temp


def fact(n):

    f = 1

    for i in range(2, n + 1):

        f *= i

    return f

n = 6

x = [1911, 1921, 1931, 1941, 1951, 1961]


y = [[0 for i in range(n)]

     for j in range(n)]

y[0][0] = 12

y[1][0] = 15

y[2][0] = 20
```

```
y[3][0] = 27
y[4][0] = 39
y[5][0] = 52

for i in range(1, n):
    for j in range(n - i):
        y[j][i] = y[j + 1][i - 1] - y[j][i - 1]

for i in range(n):
    print(x[i], end="\t")
    for j in range(n - i):
        print(y[i][j], end="\t")
    print("")

value = 1946
sum = y[0][0]
u = (value - x[0]) / (x[1] - x[0]);
for i in range(1, n):
    sum = sum + (u_cal(u, i) * y[0][i]) / fact(i);

print("\nValue at", value,"is", round(sum, 6))
```

Sample output:

```
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 is 32.34375

Problem number: 07

Problem title: 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 of the problem

Backward Differences: The differences y1 – y0, y2 – y1, ……, yn – yn–1 when denoted by dy1, dy2, ……, dyn, respectively, are called first backward difference. This formula is useful when the value of f(x) is required near the end of the table

Algorithm

1. Read the data from a file from user and save it as x,y by splitting.
2. Take the value for interpolation from the user.
3. Format a table.
4. Print all data into the table.

Python source code

```
import math

file_name = input("Enter file name with extension: ")

f = open(file_name, "r")

data = f.read()

print(data)

data = data.split()

x, y = [], []

for i, j in zip(data[0::2], data[1::2]):

    x.append(float(i))

    y.append(float(j))

inp = float(input("Enter value of x for interpolation: "))


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

formated_table = [["x", "f(x)", "∇f(x)"]]
for i in range(2, len(table)):
    formated_table[0].append("∇^" + str(i) + "f(x)")
for i in range(len(x)):
    row = []
    for j in range(len(table) - i):
        row.append(str(round(table[j][i], 5)))
    row.insert(0, str(x[i]))
    formated_table.append(row)
for row in formated_table:
    print(" \t".join(row))

r = (inp - x[-1]) / (x[1] - x[0])
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)/math.factorial(i)

final_result = table[0][-1] + partial_result
print("f(" + str(inp) + ") = ", final_result)
```

Sample output

Enter file name with extension: data3.txt

1 1

2 8
3 27
4 64
5 125
6 216
7 343
8 512
Enter value of x for interpolation: 7.5

x	f(x)	∇f(x)	∇^2f(x)	∇^3f(x)	∇^4f(x)	∇^5f(x)	∇^6f(x)	∇^7f(x)
1.0	1.0	7.0	12.0	6.0	0.0	0.0	0.0	0.0
2.0	8.0	19.0	18.0	6.0	0.0	0.0	0.0	
3.0	27.0	37.0	24.0	6.0	0.0	0.0		
4.0	64.0	61.0	30.0	6.0	0.0			
5.0	125.0	91.0	36.0	6.0				
6.0	216.0	127.0	42.0					
7.0	343.0	169.0						
8.0	512.0							

f(7.5) = 421.875

Problem number: 08

Problem title: Write a Python program to find the value of $f(15)$ from the following table using Newton's divided difference formula.

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

Illustration of the problem

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

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

Algorithm

1. Start the program.
2. Define n=6 and y as a matrix where f(x) is assigned.
3. Get values of y from a function divideddifftable:

```
def dividedDiffTable(x, y, n):
    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]))
    return y
```

4. Print value at assigned value from the function applyformula by rounding the result:

```
def applyFormula(value, x, y, n):
    sum = y[0][0]

    for i in range(1, n):
        sum = sum + (proterm(i, value, x) * y[0][i])

    return sum
```

5. Stop the program.

Python source code

```
def proterm(i, value, x):

    pro = 1

    for j in range(i):

        pro = pro * (value - x[j])

    return pro


def dividedDiffTable(x, y, n):

    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]))

    return y


def applyFormula(value, x, y, n):

    sum = y[0][0]


    for i in range(1, n):

        sum = sum + (proterm(i, value, x) * y[0][i])


    return sum
```

```
def printDiffTable(y, n):
    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("")


n = 6
y = [[0 for i in range(n)] for j in range(n)]
x = [4, 5, 7, 10, 11, 13]
print(x)
y[0][0] = 48
y[1][0] = 100
y[2][0] = 294
y[3][0] = 900
y[4][0] = 1210
y[5][0] = 2028
print(y)
y = dividedDiffTable(x, y, n)


printDiffTable(y, n)


value = 15
print("\nValue at", value, "is", round(applyFormula(value, x, y, n), 2))


value = 8
print("\nValue at", value, "is", round(applyFormula(value, x, y, n), 2))
```

Sample output

[4, 5, 7, 10, 11, 13]

[[48, 0, 0, 0, 0, 0], [100, 0, 0, 0, 0, 0], [294, 0, 0, 0, 0, 0], [900, 0, 0, 0, 0, 0], [1210, 0, 0, 0, 0, 0], [2028, 0, 0, 0, 0, 0]]

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 is 3150.0

Value at 8 is 448.0

Problem number: 09

Problem title: Write a Python program to find the value of y when x = 10 from the following table using Lagrange's interpolation formula.

x:	5	6	9	11
y:	12	13	14	16

Illustration of the problem

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.

Algorithm

1. Start the program
2. Define a class named data:

```
class Data:
  def __init__(self, x, y):
    self.x = x
```

```
    self.y = y
```

3. Assign a list ‘f’ where the values of x and y are processed through class ‘data’.
4. Print the value of f(10) is by interpolate the list f given ‘x’ and the lenth of f:

```
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
```

5. Stop the program.

Python source code

```
class Data:
  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)]
  print("Value of f(10) is :", interpolate(f, 10, len(f)))
```

Sample output:

Value of f(10) is : 14.666666666666666

Problem number: 10

Problem title: Write a Python program to find a real root of the equation $x^3 - 2x - 5 = 0$ using bisection method.

Illustration of the problem

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.

Algorithm

1. Start the program.
2. Define the equation in the function 'func':

```
def func(x):
    return x * x * x - 2 * x - 5
```

3. Assign the value of a and b as -1 and 3 .
4. Find the middle point c=(a+b)/2.
5. If func(c)=0 then c is the root of the solution.
6. Else if func(c) !=0
   6.1. If value func(a)*func(c) < 0 then root lies between a and c. So we recur for a and c
   6.2. Else If func (b)* func (c) < 0 then root lies between b and c. So we recur b and c.
   6.3. Else given function doesn't follow one of assumptions.
7. Stop the program.

Python source code

```
def func(x):
    return x * x * x - 2 * x - 5


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:

        c = (a + b) / 2

        if (func(c) == 0.0):
            break

        if (func(c) * func(a) < 0):
            b = c
        else:
            a = c

    print("The value of root is : ", "%.4f" % c)

a = -1
b = 3
bisection(a, b)
```

Sample output:

The value of root is : 2.0945

Problem number: 11

Problem title: Write a Python program to find a real root of the equation $x^3 - 2x - 5 = 0$ using false position method.

Illustration of the problem

False position method or 'regula falsi' method is a root-finding algorithm that combines features from the bisection method and the Secant method. As in the secant method, we use the root of a secant line (the value of x such that y=0) to compute the next root approximation for function f.

Algorithm

1. Start the program
2. Define the equation as a function:

```
def func( x ):
  return (x * x * x - 2 * x -5)
```

3.
4. Now we have to find the point which touches x axis. For that we put y = 0.
5. so that x = (a* func (b) - b* func (a)) / (func (b)- func (a)) . This will be c that is c = x.
6. If func (c) == 0, then c is the root of the solution.
7. Else func (c) != 0
    1. If value func (a)* func (c) < 0 then root lies between a and c. So we recur for a and c
    2. Else If func (b)* func (c) < 0 then root lies between b and c. So we recur b and c.
    3. Else given function doesn't follow one of assumptions.

7.continue the program until func (c)=0. Then print c as the result.

8. stop the program.

Python source code

```
MAX_ITER = 1000000

def func( x ):
  return (x * x * x - 2 * x -5)

def regulaFalsi( a , b):
  if func(a) * func(b) >= 0:
    print("You have not assumed right a and b")
    return -1
  c = a
  for i in range(MAX_ITER):

    c = (a * func(b) - b * func(a))/ (func(b) - func(a))
    if func(c) == 0:
      break
    elif func(c) * func(a) < 0:
      b = c
    else:
      a = c
  print("The value of root is : " , '%.4f' %c)
a =-200
b = 300
regulaFalsi(a, b)
```

Sample output

The value of root is: 2.0946