Discrete Math Practical

Name = Ashutosh Jha
University Roll No. = 23020570006
College Roll No. =20231473
Course = B.Sc. (H) Computer Science
Submitted to = Dr. Aakash

S.No	Topic
1.	Create a class SET. Create member functions to perform the following set operation:
2.	Create a class RELATION , use matrix notation to represent a relation. Include member functions to check if the relation is Reflexive ,Symmetric , Antisymmetric , Transitive . Also check whether it is equivalence or partial relation or None.
3.	Write a program that generates all the permutation of a given set of digits, with or without repetition.
4.	For any number n , write a program to list all the solutions of the equation x1+x2+ x3 + ----+xn = C ,C is a constant.
5.	Write a Program to evaluate a polynomial function. (For example store f(x) = 4n2 + 2n + 9 in an array and for a given value of n, say n = 5, compute the value of f(n)).
6.	Write a Program to check if a given graph is a complete graph. Represent the graph using the Adjacency Matrix representation.
7.	Write a Program to check if a given graph is a complete graph. Represent the graph using the Adjacency Matrix representation.
8.	Write a Program to accept a directed graph G and compute the in-degree and outdegree of each vertex.

Practical 1:

```
"""1. Create A Class SET. Create Member Functions To Perform The Following SET Operations:
    1) Is Member: Check Whether An Element Belongs To The Set Or Not And Return
    Value As True/False.
    2) Powerset: List All The Elements Of The Power Set Of A Set .
    3) Subset: Check Whether One Set Is A Subset Of The Other Or Not.
    4) Union And Intersection Of Two Sets.
    5) Complement: Assume Universal Set As Per The Input Elements From The User.
    6) Set Difference And Symmetric Difference Between Two Sets.
    7) Cartesian Product Of Sets.
Write A _Menu Driven Program To Perform The Above Functions On An Instance Of The SET Class."""


class SET:
    def __init__(self, u_set):
        self.u_set = u_set

    def is_member(self, element):
        if element in self.u_set:
            return "Element Found"
        else:
            return "Element Not Found"

    def powerset(self):
        lst=[]
        length = len(self.u_set)
        for i in range(1 << length):
            lst.append({self.u_set[j] for j in range(length) if (i & (1 << j))})
        print("Your Required Powerset Are:: ", lst)

    def subset(self, subset_set):
        if subset_set.u_set.issubset(self.u_set):
            return "This Is A Subset"
        else:
            return "This Is Not A Subset"

    def union_intersection(self, set2):
        print("Intersection Of  Your Sets Are:: \n", self.u_set.intersection(set2.u_set))
        print("Union Of Your Sets Are:: \n", self.u_set.union(set2.u_set))
```

```
    if choice == '1':
        set1 = SET(set_create())
        element = int(input("Enter Your Element:"))
        print(set1.is_member(element))
    elif choice == '2':
        set1 = SET(list(set_create()))
        set1.powerset()
    elif choice == '3':
        universal_set = SET(set_create(uni="Universal Set"))
        subset_set = SET(set_create(uni="Subset"))
        print(universal_set.subset(subset_set))
    elif choice == '4':
        set1 = SET(set_create(uni="First Set"))
        set2 = SET(set_create(uni="Another Set"))
        set1.union_intersection(set2)
    elif choice == '5':
        universal_set = SET(set_create(uni=" Universal Set "))
        complement_set = SET(set_create(uni="Set"))
        universal_set.complement(complement_set)
    elif choice == '6':
        universal_set = SET(set_create("Universal Set Or Main"))
        another_set = SET(set_create("Another Set"))
        universal_set.difference_and_symmetric_difference(another_set)
    elif choice == '7':
        set1 = SET(set_create("First set"))
        set2 = SET(set_create("Another set"))
        set1.cartesian_product(set2)
    else:
        print("Invalid Input!!\nPlease Try Again")
        main()

if __name__ == "__main__":
    for i in range(8):
        main()
```

Output:

```
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice:: 1
Enter Your Element Of set With A Space:: 1 3 5 7
Your Given set Are::  {1, 3, 5, 7}
Enter Your Element:8
Element Not Found
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice:: 2
Enter Your Element Of set With A Space:: 4 5 7 5
Your Given set Are::  {4, 5, 7}
Your Required Powerset Are::  [set(), {4}, {5}, {4, 5}, {7}, {4, 7}, {5, 7}, {4, 5, 7}]
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice::
```

```
Enter Your Element Of Universal Set With A Space:: 5 6 3 8 9
Your Given Universal Set Are::  {3, 5, 6, 8, 9}
Enter Your Element Of Subset With A Space:: 4 6 2
Your Given Subset Are::  {2, 4, 6}
This Is Not A Subset
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice:: 4
Enter Your Element Of First Set With A Space:: 1 88 4 6 2 86
Your Given First Set Are::  {1, 2, 4, 6, 86, 88}
Enter Your Element Of Another Set With A Space:: 7 8 5 6 42 5
Your Given Another Set Are::  {5, 6, 7, 8, 42}
Intersection Of  Your Sets Are::
 {6}
Union Of Your Sets Are::
 {1, 2, 4, 5, 6, 7, 8, 42, 86, 88}
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice:: 5
Enter Your Element Of  Universal Set  With A Space:: 8 54 9 5
Your Given  Universal Set  Are::  {8, 9, 5, 54}
Enter Your Element Of Set With A Space:: 8 5 4 7
Your Given Set Are::  {8, 4, 5, 7}
Your Complement Of Set Is::
```

```
Your Complement Of Set Is::
 {9, 54}
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice:: 6
Enter Your Element Of Universal Set Or Main With A Space:: 5 4 59 8 9 2
Your Given Universal Set Or Main Are::  {2, 4, 5, 8, 9, 59}
Enter Your Element Of Another Set With A Space:: 8 5 6 7 4
Your Given Another Set Are::  {4, 5, 6, 7, 8}
Difference Of Your Sets Are::
 {9, 2, 59}
Symmetric Difference of your sets are::
 {2, 6, 7, 9, 59}
Main Menu!!
    1.Check Whether An Element Belongs To The Set Or Not.
    2.List All The Elements Of The Power Set Of A Set.
    3.Check Whether One Set Is A Subset Of The Other Or Not.
    4.Find Union And Intersection Of Two Sets.
    5.Find Complement Of Set.
    6.Find Difference And Symmetric Difference Between Two Sets.
    7.Find Cartesian Product Of Sets.
    Enter Your Choice:: 7
Enter Your Element Of First set With A Space:: 85 6 8 9 4 5
Your Given First set Are::  {4, 5, 6, 8, 9, 85}
Enter Your Element Of Another set With A Space:: 8 5 6 7 4
Your Given Another set Are::  {4, 5, 6, 7, 8}
Your Cartesian Product Are::  {(5, 4), (4, 6), (5, 7), (9, 5), (85, 6), (9, 8), (8, 6), (6, 5)
, (6, 8), (4, 5), (5, 6), (4, 8), (9, 7), (8, 5), (85, 8), (9, 4), (85, 5), (8, 8), (6, 4), (6
, 7), (4, 7), (4, 4), (5, 5), (8, 4), (85, 4), (5, 8), (8, 7), (9, 6), (85, 7), (6, 6)}
Main Menu!!
```

Practical 2:

```python

"""2. Create a class RELATION, use Matrix notation to represent a relation. Include···

from numpy import array
class RELATION:
    def __init__(self, matrix):
        self.matrix = matrix
        self.length = len(matrix)

    def reflexive(self):
        for i in range(self.length):
            if not self.matrix[i][i]:
                return False
        return True

    def symmetric(self):
        for i in range(self.length):
            for j in range(self.length):
                if self.matrix[i][j] != self.matrix[j][i]:
                    return False
        return True

    def transitive(self):
        for i in range(self.length):
            for j in range(self.length):
                for k in range(self.length):
                    if self.matrix[i][j] and self.matrix[j][k] and not self.matrix[i][k]:
                        return False
        return True

    def anti_symmetric(self):
        for i in range(self.length):
            for j in range(self.length):
                if i != j and self.matrix[i][j] and self.matrix[j][i]:
                    return False
        return True
```

ctical2.py > ...

```python
class RELATION:
    def transitive(self):
        return True

    def anti_symmetric(self):
        for i in range(self.length):
            for j in range(self.length):
                if i != j and self.matrix[i][j] and self.matrix[j][i]:
                    return False
        return True

def enter_matrix():
    lst = list(map(int, input("Enter All Relation In Form Of Matrix Value With A Space:: ").split()))
    row = int(input("Enter How Many Row or Columns In Your Square Matrix:: "))
    matrix = array(lst).reshape(row, row)
    print("Your Required Matrix Are:: \n", matrix)
    return matrix

def main():
    rel = RELATION(enter_matrix())
    if rel.reflexive() and rel.symmetric() and rel.transitive():
        return "Your Relation is Equivalence Relation."
    elif rel.reflexive() and rel.anti_symmetric() and rel.transitive():
        return "Your Relation is Partial Order Relation."
    else:
        return "None"
if __name__ == "__main__":
    print(main())
```

Output:

```
Enter All Relation In Form Of Matrix Value With A Space:: 1 0 1 0 1 0 1 0 1
Enter How Many Row or Columns In Your Square Matrix:: 3
Your Required Matrix Are::
 [[1 0 1]
 [0 1 0]
 [1 0 1]]
Your Relation is Equivalence Relation.
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\prectical2.py"
Enter All Relation In Form Of Matrix Value With A Space:: 1 0 0 0 1 1 0 0 1 1 1 0 1 1 1 1
Enter How Many Row or Columns In Your Square Matrix:: 4
Your Required Matrix Are::
 [[1 0 0 0]
 [1 1 0 0]
 [1 1 1 0]
 [1 1 1 1]]
Your Relation is Partial Order Relation.
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\prectical2.py"
Enter All Relation In Form Of Matrix Value With A Space:: 1 0 1 1 0 1 0 0 1
Enter How Many Row or Columns In Your Square Matrix:: 3
Your Required Matrix Are::
 [[1 0 1]
 [1 0 1]
 [0 0 1]]
None
PS C:\Maths ptectical sem 2\Discreate>
```

Practical 3:

```
"""3. Write a Program that generates all the permutations of a given set of Set, with or
without repetition."""
from itertools import permutations,product
def generate_permutations(Set, repetition):
    if repetition:
        return list(permutations(Set))
    else:
        return list(product(Set, repeat=len(Set)))
if __name__ == "__main__":
    Set = set(map(int, input("Enter all element of set with space:").split()))
    with_repetition = generate_permutations(Set, repetition=True)
    without_repetition = generate_permutations(Set, repetition=False)
    print("Permutations with repetition:")
    for perm in with_repetition:
        print(perm)
    print("\nPermutations without repetition:")
    for perm in without_repetition:
        print(len(with_repetition),len(without_repetition))
```

Output:

```
    if repetition:
        return list(permutations(Set))
    else:
        return list(product(Set, repeat=len(Set)))
if __name__ == "__main__":
    Set = set(map(int, input("Enter all element of set with space:").split()))
    with_repetition = generate_permutations(Set, repetition=True)
    without_repetition = generate_permutations(Set, repetition=False)
    print("Permutations with repetition:")
    for perm in with_repetition:
        print(perm)
    print("\nPermutations without repetition:")
    for perm in without_repetition:
        print(len(with_repetition),len(without_repetition))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\myenv\Scripts\practical3.py"
Enter all element of set with space: 1 2 3
Permutations with repetition:
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)

Permutations without repetition:
6 27
6 27
6 27
6 27
6 27
```

Practical 4:

```python
"""4. For any number n, write a program to list all the solutions of the equation x1 + x2 +
x3 + ...+ xn = C, where C is a constant (C<=10) and x1, x2,x3,...,xn are non-negative
integers, using brute force strategy."""
def find_solutions(C, n):
    def generate_solutions(current_sum, current_solution, remaining_terms):
        if current_sum == C:
            solutions.append(current_solution[:])
            return
        if not remaining_terms:
            return
        for i in range(remaining_terms[0], C - current_sum + 1):
            current_solution.append(i)
            generate_solutions(current_sum + i, current_solution, remaining_terms[1:])
            current_solution.pop()

    solutions = []
    generate_solutions(0, [], list(range(C + 1)))
    return solutions

if __name__ == "__main__":
    n = int(input("Enter number of terms::"))
    C = int(input("Enter value of constant::"))
    all_solutions = find_solutions(C, n)
    print(f"All solutions for {n} terms equation which sum is {C}")
    for solution in all_solutions:
        print(solution)
```

Output:

```
Enter number of terms::5
Enter value of constant::6
All solutions for 5 terms equation which sum is 6
[0, 1, 2, 3]
[0, 1, 5]
[0, 2, 4]
[0, 3, 3]
[0, 4, 2]
[0, 6]
[1, 1, 4]
[1, 2, 3]
[1, 3, 2]
[1, 5]
[2, 1, 3]
[2, 2, 2]
[2, 4]
[3, 1, 2]
[3, 3]
[4, 2]
[5, 1]
[6]
PS C:\Maths ptectical sem 2\Discreate>
```

Practical 5:

```python
"""5. Write a Program to evaluate a polynomial function. (For example store f(x) = 4n2 +
2n + 9 in an array and for a given value of n, say n = 5, compute the value of f(n))."""

def solve_polynomial():
    func = list(map(int, input("Enter Your polynomial coefficient Seperated With Space::").split()))
    num = int(input("Enter Value Of Your Variable::"))
    value = 0
    for i in range(-1, -len(func)-1, -1):
        value += func[i]*num**(-i-1)
    return value
print(solve_polynomial())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\myenv\Scripts\practical5.py"
Enter Your polynomial coefficient Seperated With Space::3 6 9 4
Enter Value Of Your Variable::3
166
PS C:\Maths ptectical sem 2\Discreate>
```

Practical 6:

```python
"""6. Write a Program to check if a given graph is a complete graph. Represent the
graph using the Adjacency Matrix representation."""
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adj_matrix = [[0] * vertices for _ in range(vertices)]

    def add_edge(self, u, v):
        if graph_type== 1:
            self.adj_matrix[u][v] = 1
            self.adj_matrix[v][u] = 1
        else:
            self.adj_matrix[u][v] = 1

    def is_complete(self):
        for i in range(self.vertices):
            for j in range(self.vertices):
                if i != j and self.adj_matrix[i][j] == 0:
                    return False
        return True
    def get_matrix(self):
        return self.adj_matrix

if __name__ == "__main__":
    graph_type =int(input("Enter Your Graph Type(1.Undirected 2.Directed)::"))
    num_vertices = int(input("Enter number of vertices::"))
    g = Graph(num_vertices)
    num=int(input("Enter number of edges::"))
    for i in range(num):
        a=int(input(f"Enter first vertice of {i+1} edge:: "))- 1
        b=int(input(f"Enter second vertice of same edge:: "))- 1
        g.add_edge(a,b)
    print("Your Adjacency Matrix is::\n",g.get_matrix())
    if g.is_complete():
        print("The graph is a complete graph.")
    else:
        print("The graph is not a complete graph.")
```

Output:

```
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\myenv\Scripts\practical6.py"
Enter Your Graph Type(1.Undirected 2.Directed)::1
Enter number of vertices::2
Enter number of edges::1
Enter first vertice of 1 edge:: 1
Enter second vertice of same edge:: 1
Your Adjacency Matrix is::
 [[1, 0], [0, 0]]
The graph is not a complete graph.
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\myenv\Scripts\practical6.py"
Enter Your Graph Type(1.Undirected 2.Directed)::1
Enter number of vertices::2
Enter number of edges::1
Enter first vertice of 1 edge:: 1
Enter second vertice of same edge:: 2
Your Adjacency Matrix is::
 [[0, 1], [1, 0]]
em 2\Discreate\myenv\Scripts\practical6.py"
Enter Your Graph Type(1.Undirected 2.Directed)::2
Enter number of vertices::2
Enter number of edges::2
Enter first vertice of 1 edge:: 1
Enter second vertice of same edge:: 1
Enter first vertice of 2 edge:: 1
Enter second vertice of same edge:: 1
Your Adjacency Matrix is::
 [[1, 0], [0, 0]]
The graph is not a complete graph.
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\myenv\Scripts\practical6.py"
Enter Your Graph Type(1.Undirected 2.Directed)::2
Enter number of vertices::2
Enter number of edges::2
Enter first vertice of 1 edge:: 1
Enter second vertice of same edge:: 2
Enter first vertice of 2 edge:: 2
Enter second vertice of same edge:: 1
Your Adjacency Matrix is::
 [[0, 1], [1, 0]]
The graph is a complete graph.
PS C:\Maths ptectical sem 2\Discreate>
```

Practical 7:

```python
"""7. Write a Program to check if a given graph is a complete graph. Represent the
graph using the Adjacency List representation."""
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adj_list = [[] for _ in range(vertices)]

    def add_edge(self, u, v):
        if graph_type== 1:
            self.adj_list[u].append(v)
            self.adj_list[v].append(u)
        else:
            self.adj_list[v].append(u)

    def is_complete(self):
        for i in range(self.vertices):
            for j in range(self.vertices):
                if i != j and j not in self.adj_list[i]:
                    return False
        return True
    def get_list(self):
        return self.adj_list

if __name__ == "__main__":
    graph_type =int(input("Enter Your Graph Type(1.Undirected 2.Directed)::"))
    num_vertices = int(input("Enter number of vertices::"))
    g = Graph(num_vertices)
    num=int(input("Enter number of edges::"))
    for i in range(num):
        a=int(input(f"Enter first vertice of {i+1} edge:: "))
        b=int(input(f"Enter second vertice of same edge:: "))
        g.add_edge(a,b)
    print("Your Adjacency Matrix is::\n",g.get_list())
    if g.is_complete():
        print("The graph is a complete graph.")
    else:
        print("The graph is not a complete graph.")
```

Output:

```
Enter number of vertices::3
Enter number of edges::3
Enter first vertice of 1 edge:: 0
Enter second vertice of same edge:: 1
Enter first vertice of 2 edge:: 1
Enter second vertice of same edge:: 2
Enter first vertice of 3 edge:: 0
Enter second vertice of same edge:: 2
Your Adjacency Matrix is::
 [[1, 2], [0, 2], [1, 0]]
The graph is a complete graph.
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\practical7.py"
Enter Your Graph Type(1.Undirected 2.Directed)::1
Enter number of vertices::3
Enter number of edges::2
Enter first vertice of 1 edge:: 1
Enter second vertice of same edge:: 1
Enter first vertice of 2 edge:: 1
Enter second vertice of same edge:: 2
Your Adjacency Matrix is::
 [[], [1, 1, 2], [1]]
The graph is not a complete graph.
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\practical7.py"
Enter Your Graph Type(1.Undirected 2.Directed)::2
Enter number of vertices::2
Enter number of edges::2
Enter first vertice of 1 edge:: 0
Enter second vertice of same edge:: 1
Enter first vertice of 2 edge:: 1
Enter second vertice of same edge:: 0
Your Adjacency Matrix is::
 [[1], [0]]
The graph is a complete graph.
PS C:\Maths ptectical sem 2\Discreate>
```

Practical 8:

```python
"""8. Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex."""
class DirectedGraph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adj_list = [[] for _ in range(vertices)]

    def add_edge(self, u, v):
        self.adj_list[u].append(v)

    def compute_degrees(self):
        in_degrees = [0] * self.vertices
        out_degrees = [0] * self.vertices

        for u in range(self.vertices):
            for v in self.adj_list[u]:
                out_degrees[u] += 1
                in_degrees[v] += 1

        return in_degrees, out_degrees

if __name__ == "__main__":
    num_vertices = int(input("Enter number of vertices::"))
    g = DirectedGraph(num_vertices)
    num=int(input("Enter number of edges::"))
    for i in range(num):
        a=int(input(f"Enter first vertice of {i+1} edge:: "))- 1
        b=int(input(f"Enter second vertice of same edge:: "))- 1
        g.add_edge(a,b)

    print("Vertex\tIn-Degree\tOut-Degree")
    in_degrees,out_degrees=g.compute_degrees()
    for v in range(num_vertices):
        print(f"{v}\t{in_degrees[v]}\t\t{out_degrees[v]}")
```

Output:

```
        self.adj_list[u].append(v)

    def compute_degrees(self):
        in_degrees = [0] * self.vertices
        out_degrees = [0] * self.vertices

        for u in range(self.vertices):
            for v in self.adj_list[u]:
                out_degrees[u] += 1
                in_degrees[v] += 1
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Maths ptectical sem 2\Discreate> python -u "c:\Maths ptectical sem 2\Discreate\practical8.py"
Enter number of vertices::3
Enter number of edges::2
Enter first vertice of 1 edge:: 1
Enter second vertice of same edge:: 2
Enter first vertice of 2 edge:: 2
Enter second vertice of same edge:: 3
Vertex  In-Degree       Out-Degree
0       0               1
1       1               1
2       1               0
PS C:\Maths ptectical sem 2\Discreate>
```