University of Baltistan Skardu

Submitted To:

Ma’am Noreen Maryam

Submitted By:

Basit Ali

Registration No:

S23BSCS012

Department:

Computer Science

Date:

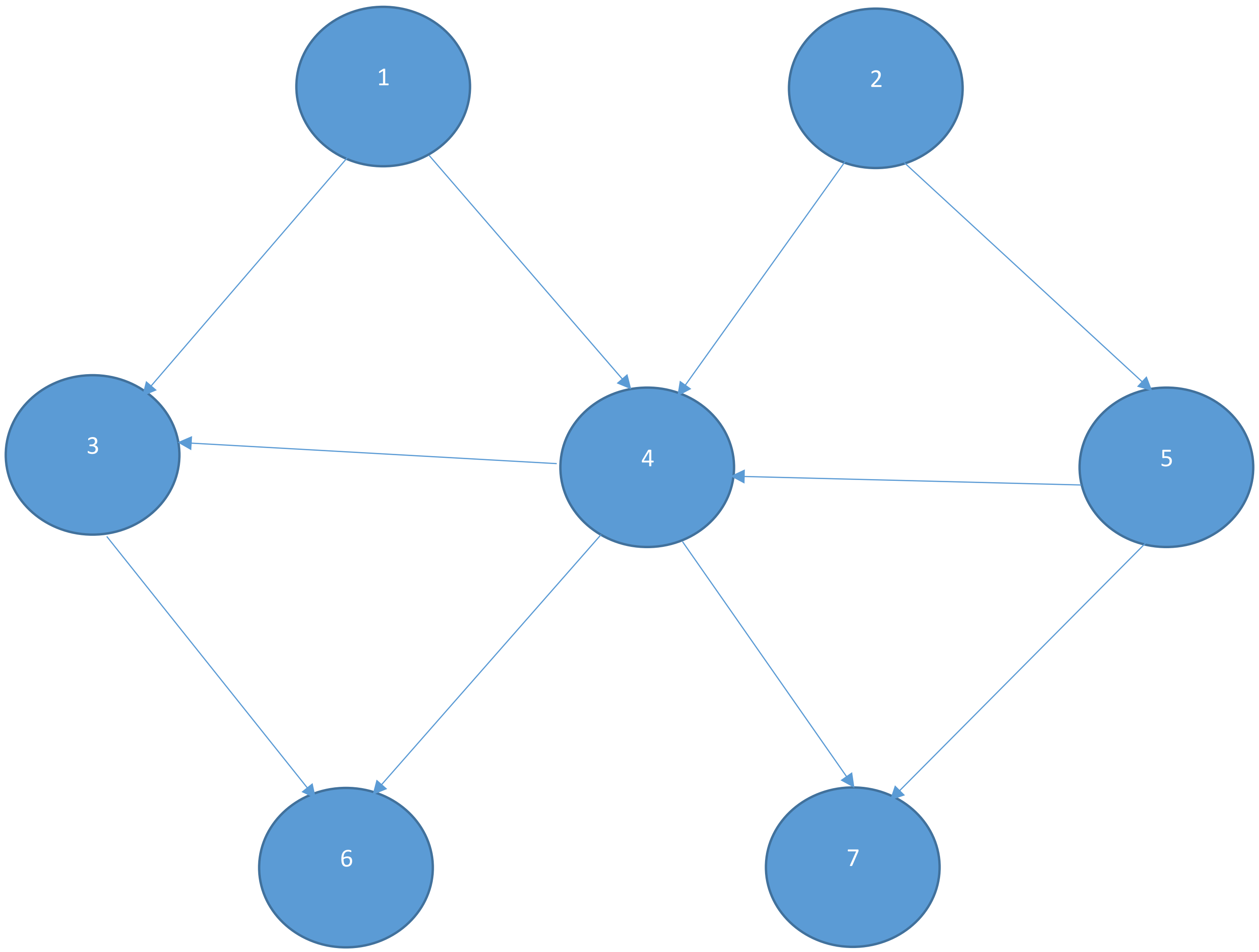
May 13,2024

Section:

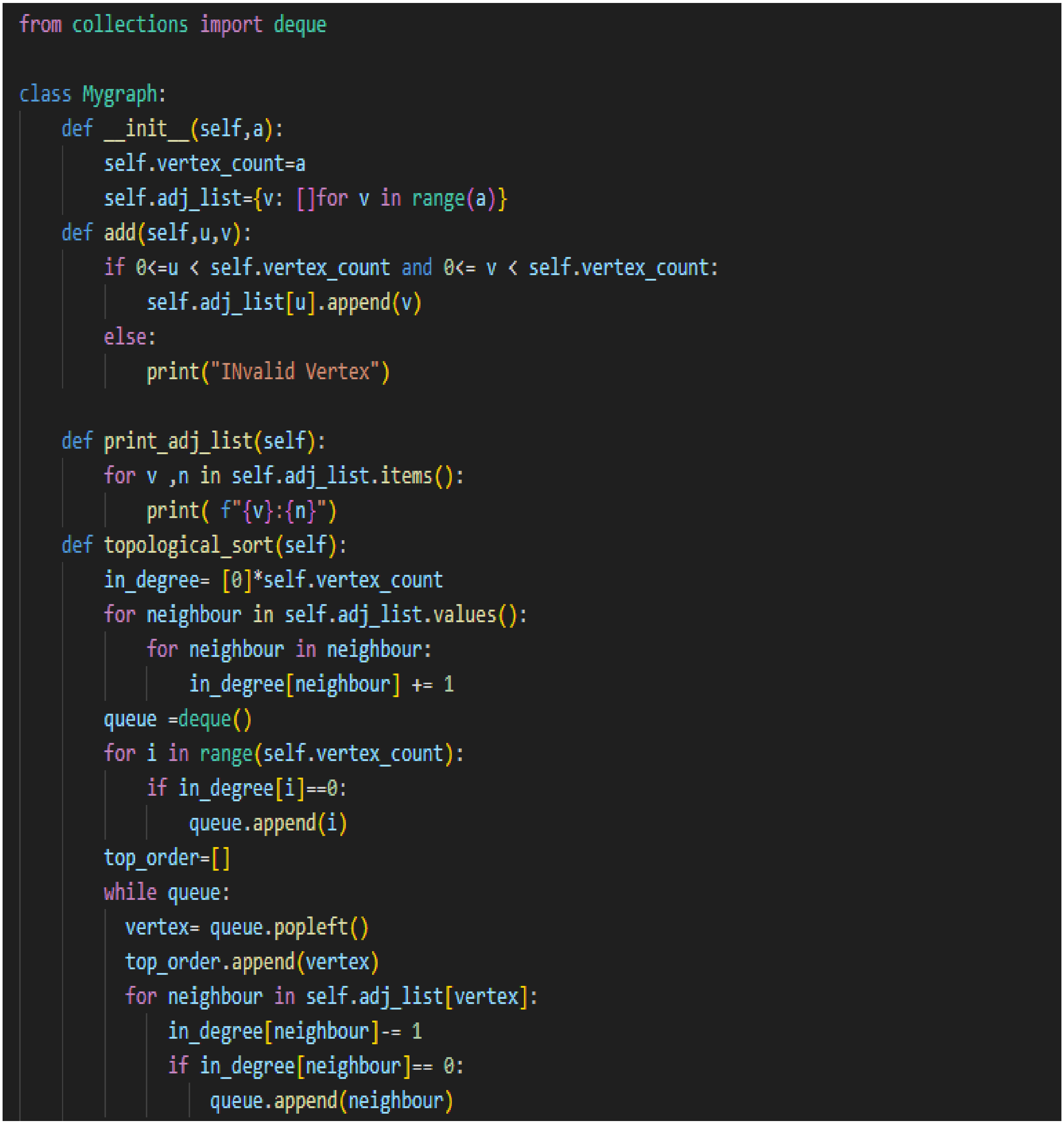
“A”

LAB#17:

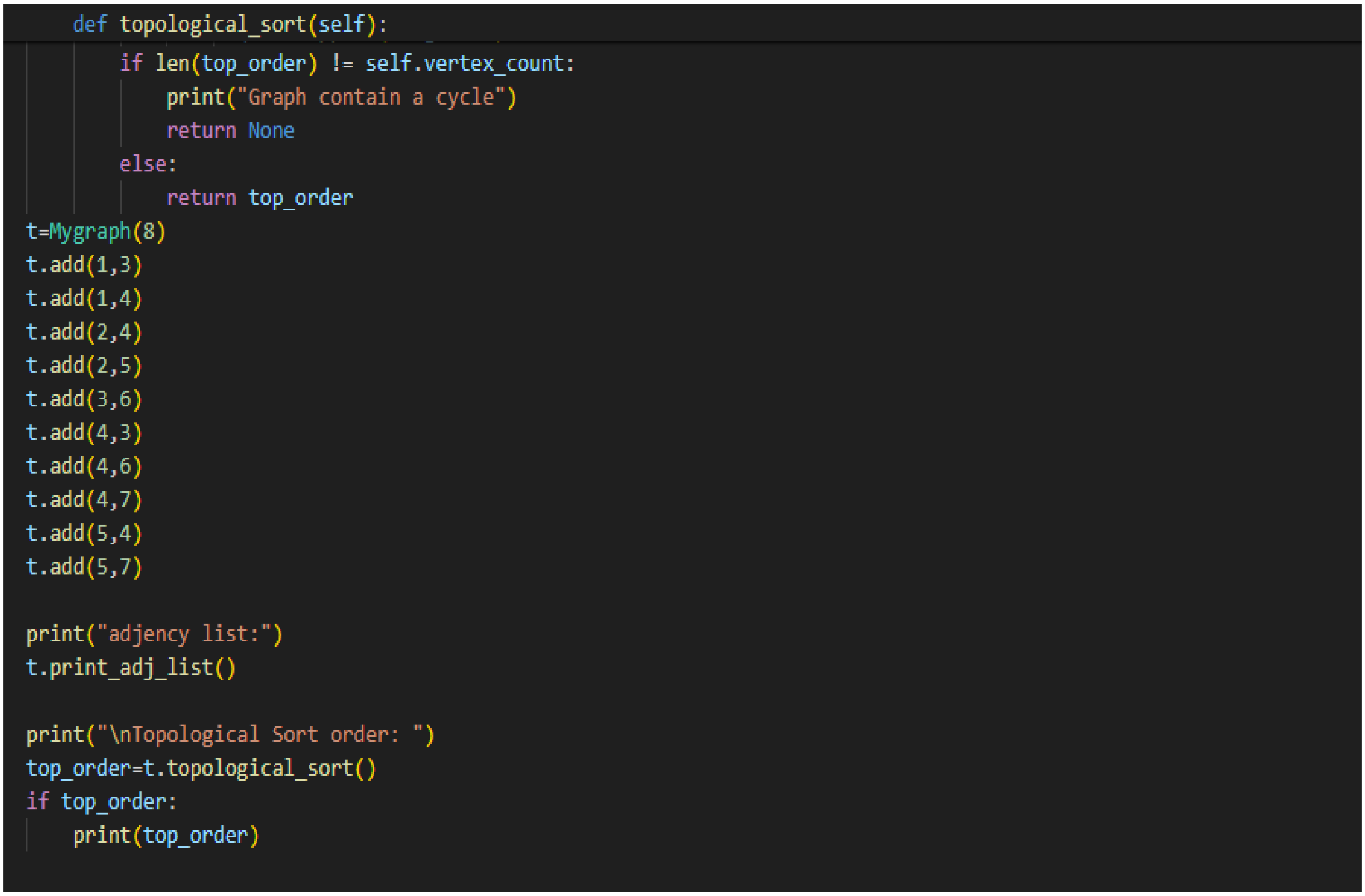
Q1: Write a program of the given graph and print the output of this graph in topological order and also explain the code.



Program:

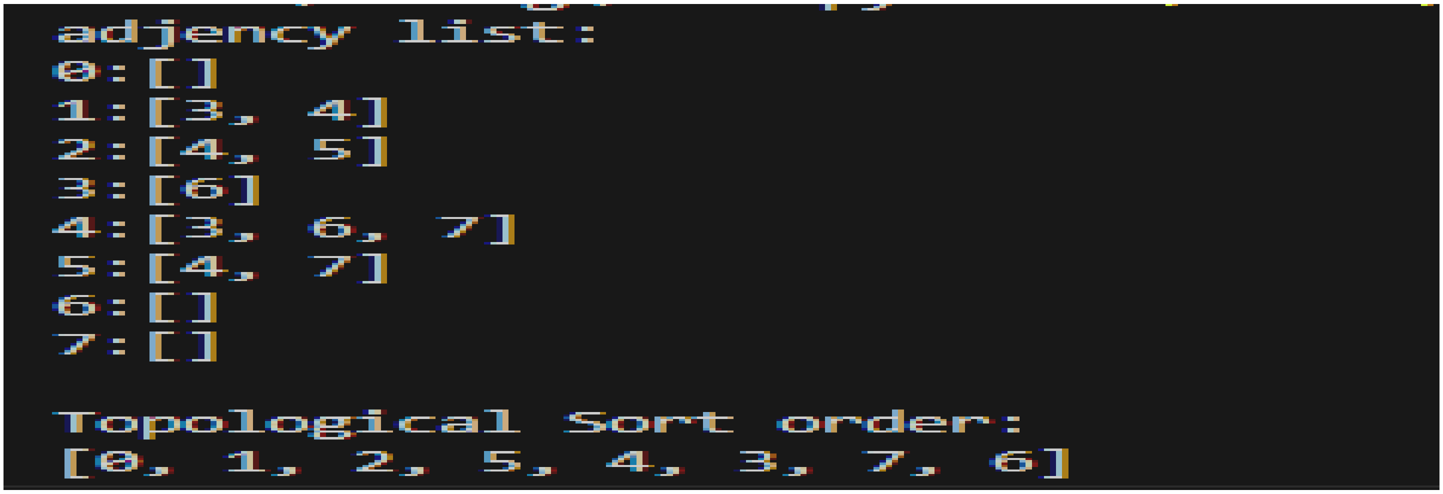


```
from collections import deque

class Mygraph:
    def __init__(self,a):
        self.vertex_count=a
        self.adj_list={v: []for v in range(a)}
    def add(self,u,v):
        if 0<=u < self.vertex_count and 0<= v < self.vertex_count:
            self.adj_list[u].append(v)
        else:
            print("INvalid Vertex")

    def print_adj_list(self):
        for v ,n in self.adj_list.items():
            print( f"{v}:{n}")
    def topological_sort(self):
        in_degree= [0]*self.vertex_count
        for neighbour in self.adj_list.values():
            for neighbour in neighbour:
                in_degree[neighbour] += 1
        queue =deque()
        for i in range(self.vertex_count):
            if in_degree[i]==0:
                queue.append(i)
        top_order=[]
        while queue:
          vertex= queue.popleft()
          top_order.append(vertex)
          for neighbour in self.adj_list[vertex]:
              in_degree[neighbour]-= 1
              if in_degree[neighbour]== 0:
                  queue.append(neighbour)
```



```
    def topological_sort(self):
        if len(top_order) != self.vertex_count:
            print("Graph contain a cycle")
            return None
        else:
            return top_order
t=Mygraph(8)
t.add(1,3)
t.add(1,4)
t.add(2,4)
t.add(2,5)
t.add(3,6)
t.add(4,3)
t.add(4,6)
t.add(4,7)
t.add(5,4)
t.add(5,7)

print("adjency list:")
t.print_adj_list()

print("\nTopological Sort order: ")
top_order=t.topological_sort()
if top_order:
    print(top_order)
```

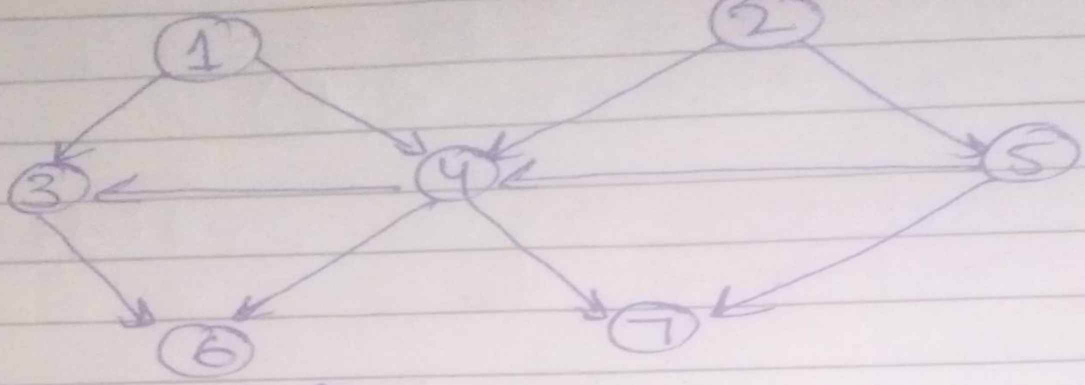
OUTPUT:



```
adjency list:
0:[]
1:[3, 4]
2:[4, 5]
3:[6]
4:[3, 6, 7]
5:[4, 7]
6:[]
7:[]

Topological Sort order:
[0, 1, 2, 5, 4, 3, 7, 6]
```

Explanation of code:

Explanation of code



Output:

Topological order
0 1 2 5 4 3 7 6

Explain

(i) i = 0

queue = (0)
top-order = (0)

- There is no neighbour of 0 therefor 0 is first append in the top-order.

queue = (1,2,5,4,3,
top- order = (1)

vertex = queue.popleft
vertex = 1

As vertex = 1 then we delete the arrow that are outgoing from vertex 1.

iii) i = 2
queue = (2,5,4,3,7,6)
top-order = (1,2)
vertex = 1,2

As vertex = 2 then we delete the arrow that are outgoing from verte 2.

Then we take the vertex where minimum arr are coming.

iv) i = 5
queue = (5,4,3,7,6)
top-order = (1,

vertex = 1, 2, 5

v) i = 4
queue = (4, 3, 7, 6)
top-order = (1, 2, 5, 4)
vertex = (1, 2, 5, 4)

vi) i = 3
queue = (~~4~~ 3, 7, 6)
top-order = (1, 2, 5, 4, 3)
vertex = (1, 2, 5, 4, 3)

vii) i = 7
queue = (7, 6)
top-order = (1, 2, 5, 4, 3, 7)
vertex = (1, 2, 5, 4, 3, 7)

viii) i = 6
queue = (6)
top-order = (1, 2, 5, 4, 3, 7, 6)
vertex = (1, 2, 5, 4, 3, 7, 6)

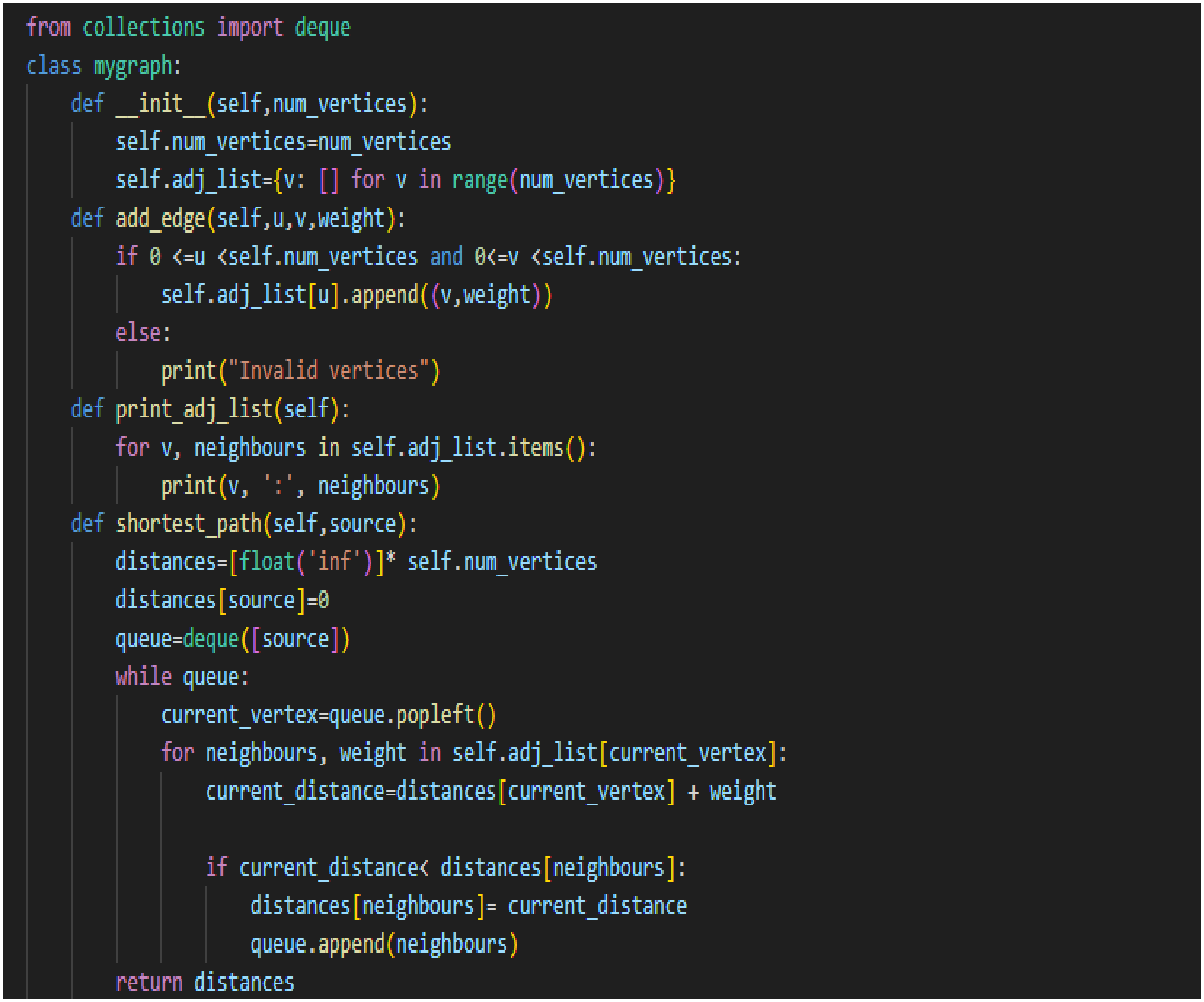
Then we pop 6 from queue therefor the queue is empty and output is

1, 2, 5, 4, 3, 7, 6

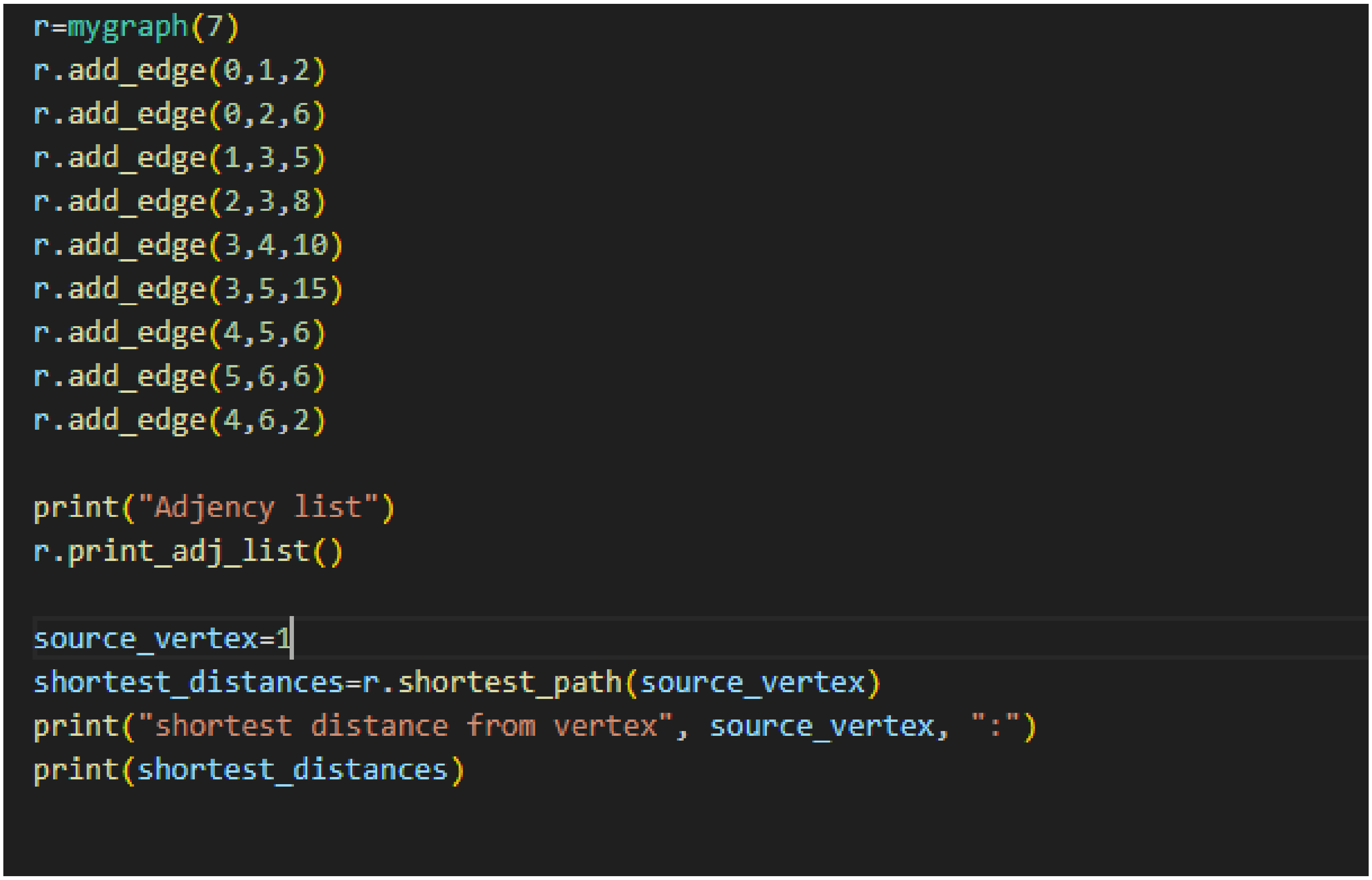
Lab#18:

Q1: Write a program that find a shortest path of above program of lab#18. We take the source vertex as 1.

Program:

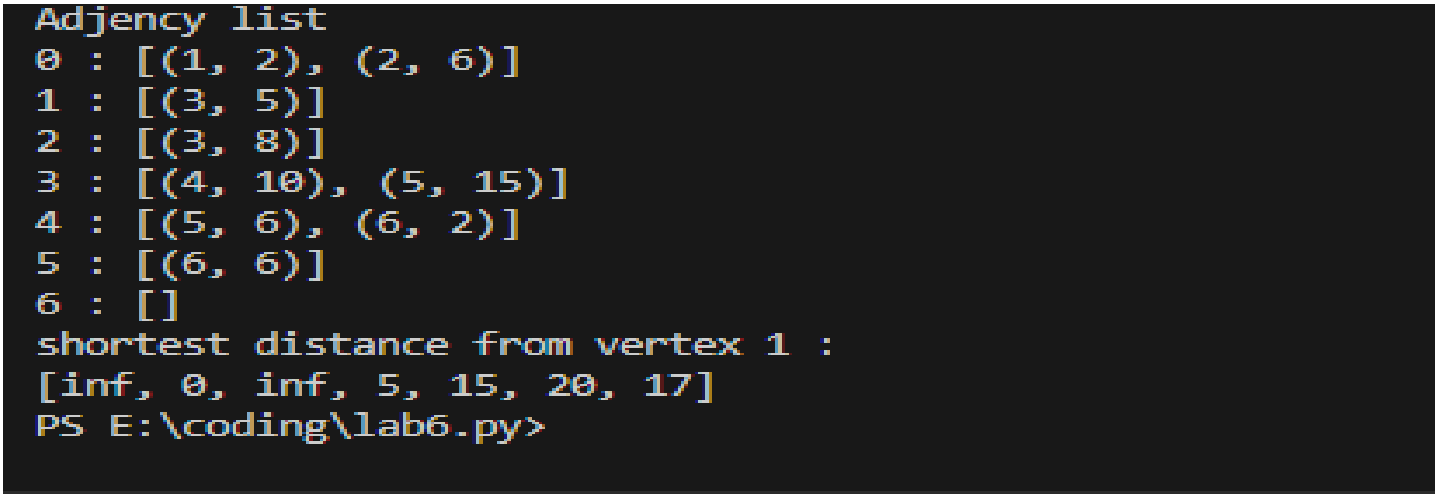


```python
from collections import deque
class mygraph:
    def __init__(self,num_vertices):
        self.num_vertices=num_vertices
        self.adj_list={v: [] for v in range(num_vertices)}
    def add_edge(self,u,v,weight):
        if 0 <=u <self.num_vertices and 0<=v <self.num_vertices:
            self.adj_list[u].append((v,weight))
        else:
            print("Invalid vertices")
    def print_adj_list(self):
        for v, neighbours in self.adj_list.items():
            print(v, ':', neighbours)
    def shortest_path(self,source):
        distances=[float('inf')]* self.num_vertices
        distances[source]=0
        queue=deque([source])
        while queue:
            current_vertex=queue.popleft()
            for neighbours, weight in self.adj_list[current_vertex]:
                current_distance=distances[current_vertex] + weight

                if current_distance< distances[neighbours]:
                    distances[neighbours]= current_distance
                    queue.append(neighbours)
        return distances
```



```
r=mygraph(7)
r.add_edge(0,1,2)
r.add_edge(0,2,6)
r.add_edge(1,3,5)
r.add_edge(2,3,8)
r.add_edge(3,4,10)
r.add_edge(3,5,15)
r.add_edge(4,5,6)
r.add_edge(5,6,6)
r.add_edge(4,6,2)

print("Adjency list")
r.print_adj_list()

source_vertex=1
shortest_distances=r.shortest_path(source_vertex)
print("shortest distance from vertex", source_vertex, ":")
print(shortest_distances)
```

__Output__:



```
Adjency list
0 : [(1, 2), (2, 6)]
1 : [(3, 5)]
2 : [(3, 8)]
3 : [(4, 10), (5, 15)]
4 : [(5, 6), (6, 2)]
5 : [(6, 6)]
6 : []
shortest distance from vertex 1 :
[inf, 0, inf, 5, 15, 20, 17]
PS E:\coding\lab6.py>
```

__Explanation:__

Date:

Explanation of Code

Source-vertex = 1

(i) For node 1 to 0.

current distance = $1 + \infty$
$= \infty$

If $\infty < \infty$
return ∞

(ii) For node 1 to 1

current distance = $1 + 0$
$= 0$

If $0 < \infty$
$d_v = 0$

) For node 1 to 2
current distance = ∞

If $\infty < \infty$
return ∞

(iv) For node 1 to 3

current distance = 5

If $5 < \infty$

$d_v = 5$

(v) For node 1 to 4

current distance = 15

If $15 < \infty$

$d_v = 15$

(vi) For node 1 to 5

~~If~~

current distance = 20

If $20 < \infty$

$d_v = 20$

(vii) For node 1 to 6

current distance = 17

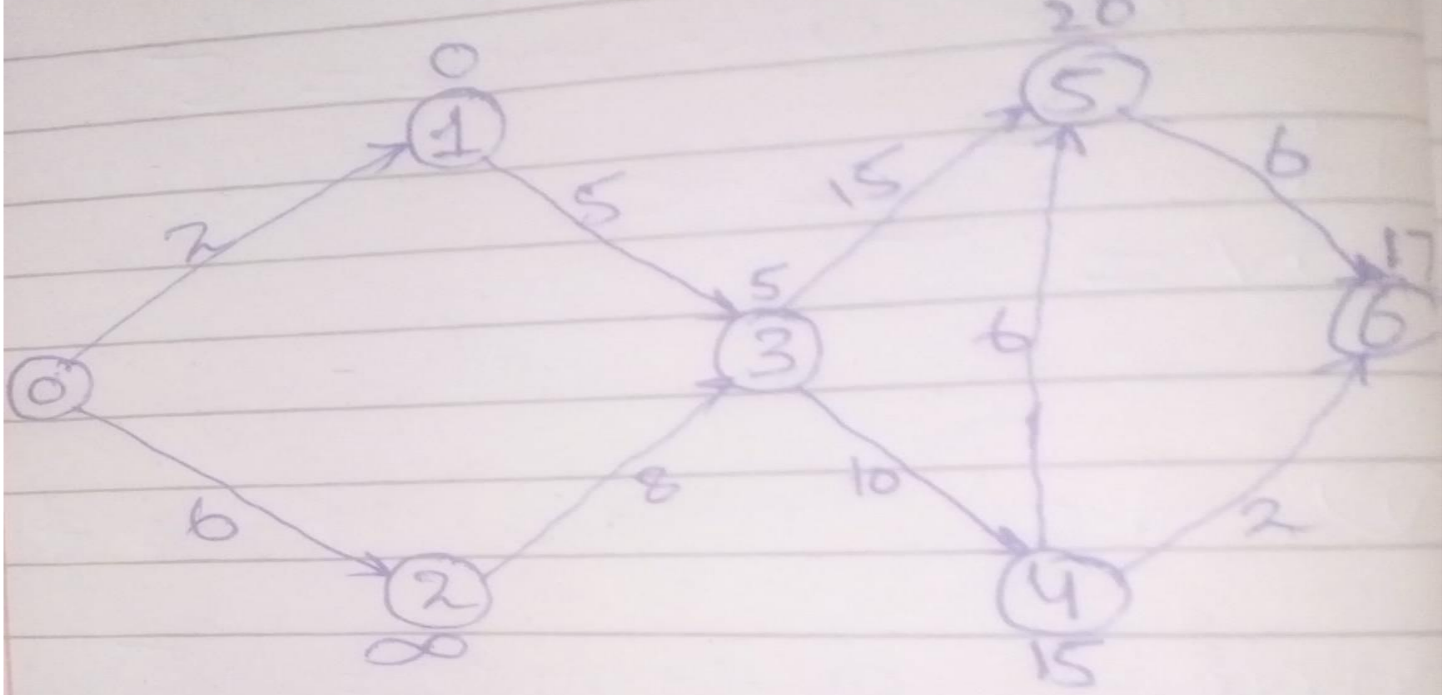
If $17 < \infty$

$d_v = 17$

Then the last result is

$\{\infty, 0, \infty, 5, 15, 20, 17]$

Diagramatical show



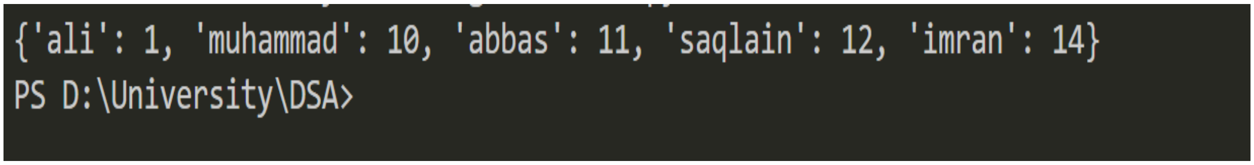
There is no path to go from node 1 to node 0 and 2 therefor the shortest path of 0 and 2 from node 1 is equal to ∞.

Lab#19:

Q1: Write a program to apply the merge sort algorithm on an unsorted dictionary to sort its values.

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    merged = []
    left_index, right_index = 0, 0
    while left_index < len(left) and right_index < len(right):
        if left[left_index] < right[right_index]:
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged

# Given dictionary
d = {"ali": 14, "muhammad": 1, "abbas": 12, "saqlain": 10, "imran":
11}

# Extract values from the dictionary
values = list(d.values())

# Perform merge sort on the extracted values
sorted_values = merge_sort(values)

# Create a new dictionary with sorted values and original keys
sorted_d = dict(zip(d.keys(), sorted_values))

print(sorted_d)
```

Output:



```
{'ali': 1, 'muhammad': 10, 'abbas': 11, 'saqlain': 12, 'imran': 14}
PS D:\University\DSA>
```