VIT-AP
UNIVERSITY

# LAB REPORT

Assignment 8

NAME : P.SATHWIKA

REGISTRATION NUMBER : 21BCE8118

SUBJECT : Design and Analysis of Algorithms.

LAB SLOT : L53+L54

## IMPLEMENT STRONGLY CONNECTED COMPONENTS

A directed graph is termed strongly connected if there is a path in both directions between every pair of vertices in the graph. Even if the entire graph is not strongly connected, two vertices u and v are said to be strongly connected if there exists a path in both directions between them.

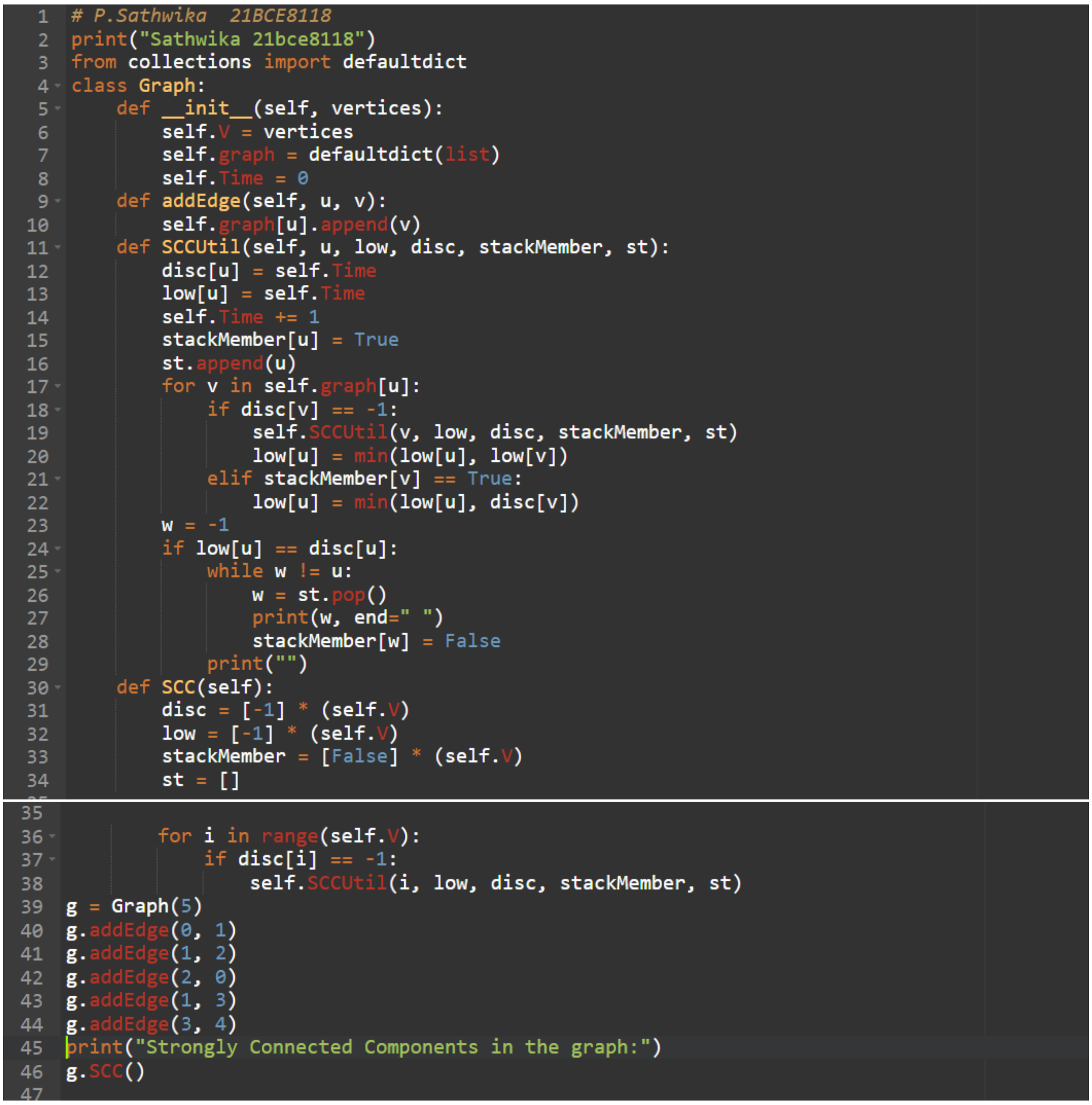
**ALGORITHM:**

1. Perform a DFS traversal on the entire graph and mark the visited vertices as 'done'. If a vertex leads to an already visited vertex, push it into the stack. This step helps us determine the order of vertices in their respective strongly connected components.
2. Reverse the edges in the graph to create its transpose.
3. Perform another DFS traversal on the reversed graph starting from each vertex in the order they were pushed into the stack during step 1. Each group of vertices reached during this traversal forms an SCC.

**CODE:**

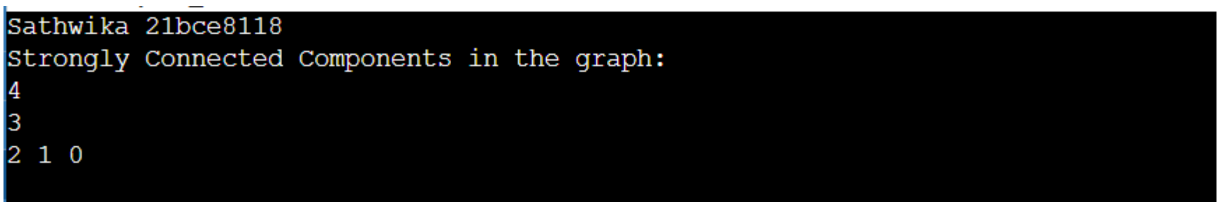
```
# P.Sathwika  21BCE8118

print("Sathwika 21bce8118")

from collections import defaultdict

class Graph:

    def __init__(self, vertices):

        self.V = vertices

        self.graph = defaultdict(list)

        self.Time = 0

    def addEdge(self, u, v):

        self.graph[u].append(v)

    def SCCUtil(self, u, low, disc, stackMember, st):

        disc[u] = self.Time

        low[u] = self.Time

        self.Time += 1

        stackMember[u] = True
        st.append(u)

        for v in self.graph[u]:
            if disc[v] == -1:
                self.SCCUtil(v, low, disc, stackMember, st)
                low[u] = min(low[u], low[v])
            elif stackMember[v] == True:
                low[u] = min(low[u], disc[v])

        w = -1
        if low[u] == disc[u]:
            while w != u:
                w = st.pop()
                print(w, end=" ")
                stackMember[w] = False
            print("")

    def SCC(self):
        disc = [-1] * (self.V)
        low = [-1] * (self.V)
        stackMember = [False] * (self.V)
        st = []

        for i in range(self.V):
            if disc[i] == -1:
                self.SCCUtil(i, low, disc, stackMember, st)

g = Graph(5)
```

```
g.addEdge(0, 1)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(1, 3)

g.addEdge(3, 4)


print("Strongly Connected Components in the graph:")

g.SCC()
```



```
# P.Sathwika  21BCE8118
print("Sathwika 21bce8118")
from collections import defaultdict
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = defaultdict(list)
        self.Time = 0
    def addEdge(self, u, v):
        self.graph[u].append(v)
    def SCCUtil(self, u, low, disc, stackMember, st):
        disc[u] = self.Time
        low[u] = self.Time
        self.Time += 1
        stackMember[u] = True
        st.append(u)
        for v in self.graph[u]:
            if disc[v] == -1:
                self.SCCUtil(v, low, disc, stackMember, st)
                low[u] = min(low[u], low[v])
            elif stackMember[v] == True:
                low[u] = min(low[u], disc[v])
        w = -1
        if low[u] == disc[u]:
            while w != u:
                w = st.pop()
                print(w, end=" ")
                stackMember[w] = False
            print("")
    def SCC(self):
        disc = [-1] * (self.V)
        low = [-1] * (self.V)
        stackMember = [False] * (self.V)
        st = []

        for i in range(self.V):
            if disc[i] == -1:
                self.SCCUtil(i, low, disc, stackMember, st)
g = Graph(5)
g.addEdge(0, 1)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(1, 3)
g.addEdge(3, 4)
print("Strongly Connected Components in the graph:")
g.SCC()
```

**OUTPUT:**



```
Sathwika 21bce8118
Strongly Connected Components in the graph:
4
3
2 1 0
```