DFS AND BFS APPLICATIONS

Team:

Aditya Arandhara(RA1911003010376)

Arunabh Kalita(RA1911003010375)

Parth Moghekar(RA1911003010368)

Saachi Almeida(RA1911003010370)

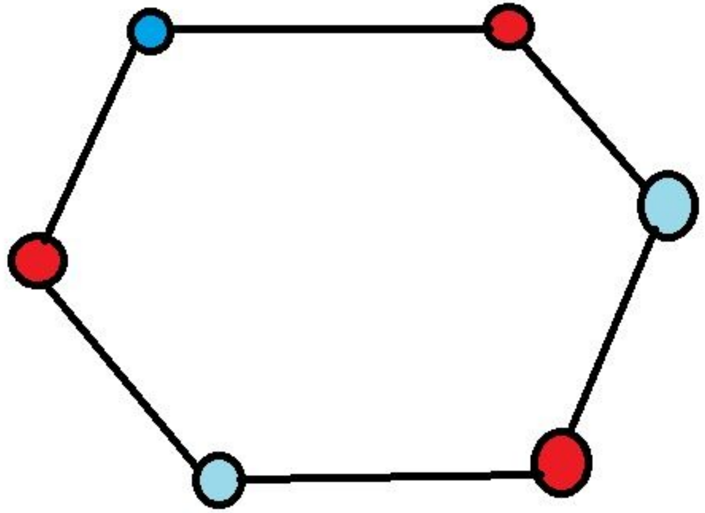
Shubham Goel(RA1911003010379)

Check if graph is Bipartite using DFS

One of the applications of Dfs is using to check whether a given graph is bipartite or not.

A bipartite graph, also called a bigraph, is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent.

In order to check this, we use graph coloring using only two colours such that vertices in a set are colored with the same color.



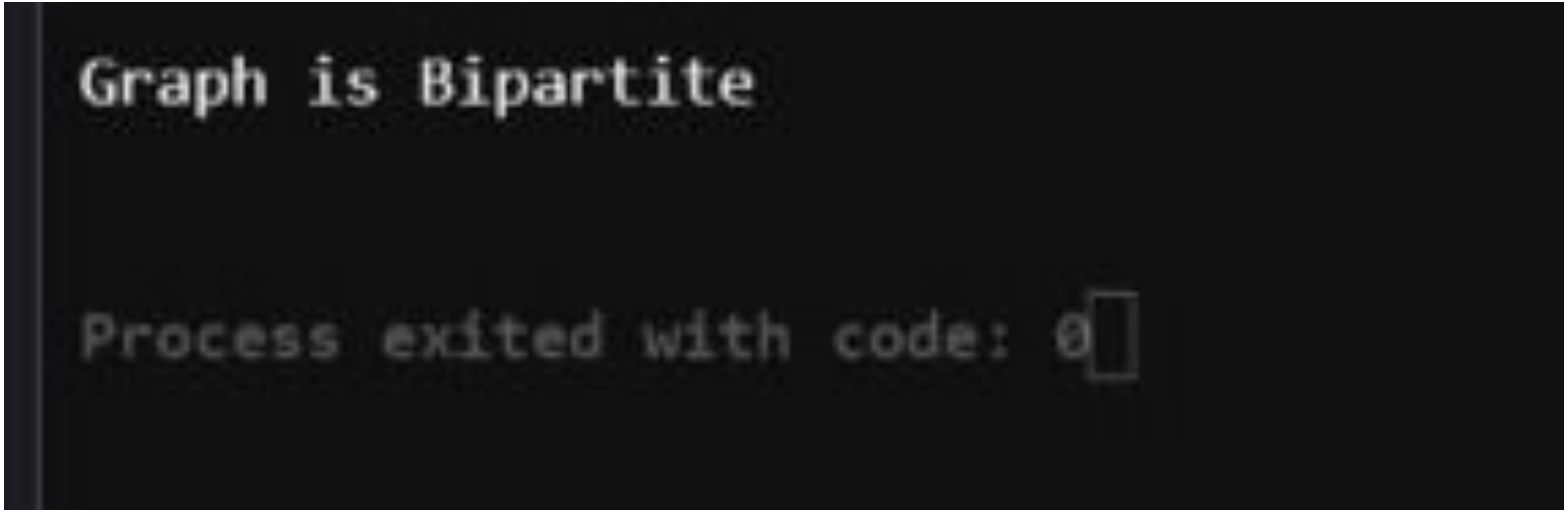
ALGORITHM

- Use a *color[]* array which stores 0 or 1 for every node which denotes opposite colors.
- Call the function DFS from any node.
- If the node u has not been visited previously, then assign !color[v] to color[u] and call DFS again to visit nodes connected to u.
- If at any point, color[u] is equal to color[v], then the node is not bipartite.
- Modify the DFS function such that it returns a boolean value at the end.

CODE

```
def addEdge(adj, u, v):

    adj[u].append(v)
    adj[v].append(u)

def isBipartite(adj, v, visited, color):

    for u in adj[v]:

        if (visited[u] == False):
            visited[u] = True
            color[u] = not color[v]

            if (not isBipartite(adj, u, visited, color)):
                return false

        elif (color[u] == color[v]):
            return Talse
    return True

if _name=='main_':

    N = 6
    adj = [[] for i in range(N + 1)]
    visited = [0 for i in range(N + 1)]
    color = [0 for i in range(N + 1)]

    addEdge(adj, 1, 2)
    addEdge(adj, 2, 3)
    addEdge(adj, 3, 4)
    addEdge(adj, 4, 5)
    addEdge(adj, 5, 6)
    addEdge(adj, 6, 1)

    visited[1] = True

    color[1] = 0

    if (isBipartite(adj, 1, visited, color)):
        print("Graph is Bipartite")
    else:
        print("Graph is not Bipartite
```

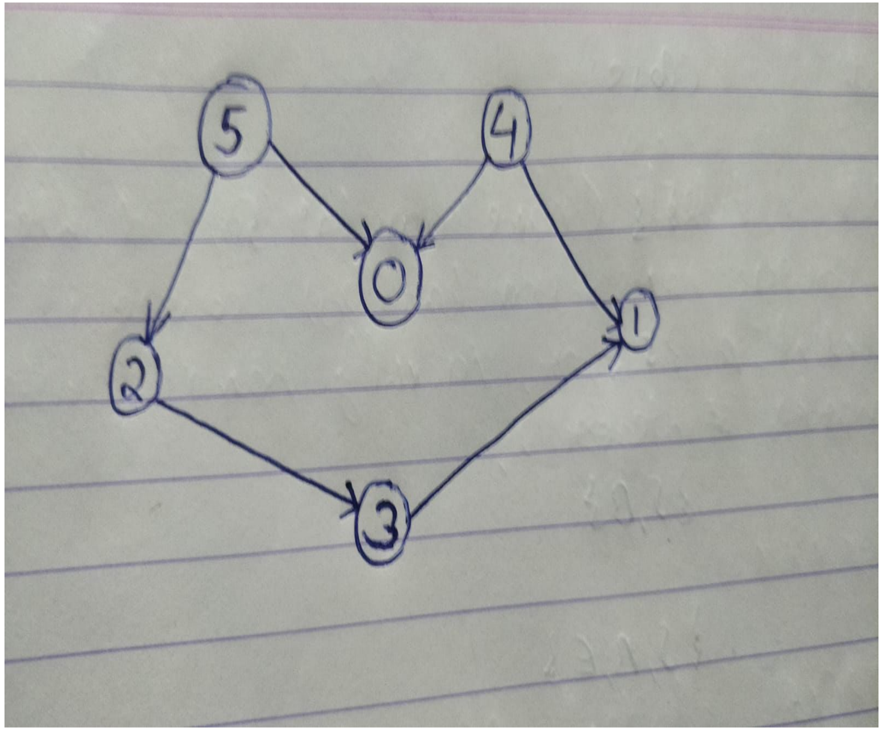
OUTPUT



```
Graph is Bipartite


Process exited with code: 0
```

Topological Sort using BFS

Topological sort is one of the applications of BFS.Topological sort must require a Directed Acyclic Graph and is a linear ordering of its vertices in which u occurs before v in the ordering for every directed edge uv from vertex u to vertex v.



ALGORITHM

Step-1: Compute in-degree (number of incoming edges) for each of the vertex present in the DAG and initialize the count of visited nodes as 0.

Step-2: Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)

Step-3: Remove a vertex from the queue (Dequeue operation) and then.

1. **Increment count of visited nodes by 1.**
2. **Decrease in-degree by 1 for all its neighbouring nodes.**
3. **If in-degree of a neighbouring nodes is reduced to zero, then add it to the queue.**

Step 4: Repeat Step 3 until the queue is empty.

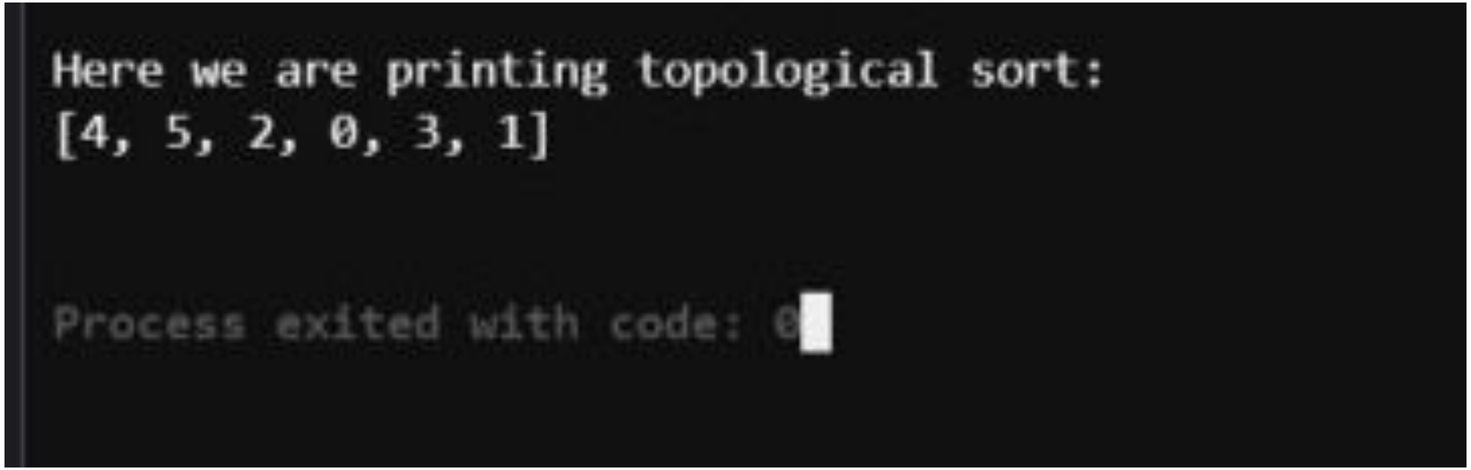
Step 5: If count of visited nodes is not equal to the number of nodes in the graph then the topological sort is not possible for the given graph.

CODE

```
from collections import defaultdict

class Graph:
	def _init_(self, vertices):
		self.graph = defaultdict(list)
		self.V = vertices

	def addEdge(self, u, v):
		self.graph[u].append(v)

	def topologicalSort(self):

		in_degree = [0]*(self.V)

		for i in self.graph:
			for j in self.graph[i]:
				in_degree[j] += 1

		queue = []
		for i in range(self.V):
			if in_degree[i] == 0:
				queue.append(i)
		cnt = 0
		top_order = []
```

```
        while queue:
            u = queue.pop(0)
            top_order.append(u)

            for i in self.graph[u]:
                in_degree[i] -= 1
                if in_degree[i] == 0:
                    queue.append(i)

            cnt += 1

        if cnt != self.V:
            print ("There exists a cycle in the graph")
        else :
            print (top_order)


g = Graph(6)
g.addEdge(5, 2);
g.addEdge(5, 0);
g.addEdge(4, 0);
g.addEdge(4, 1);
g.addEdge(2, 3);
g.addEdge(3, 1);

print ("Here we are printing topological sort: ")
g.topologicalSort()
```

OUTPUT



```
Here we are printing topological sort:
[4, 5, 2, 0, 3, 1]


Process exited with code: 0
```

THANK YOU!