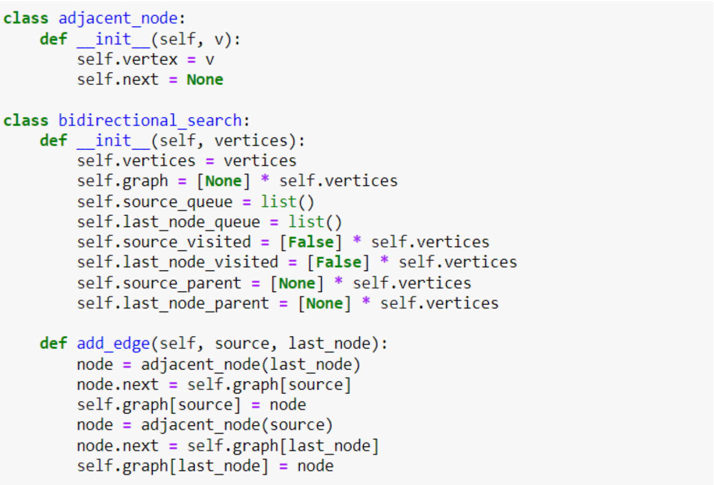
NAME : ANKIT KUMAR SAHU

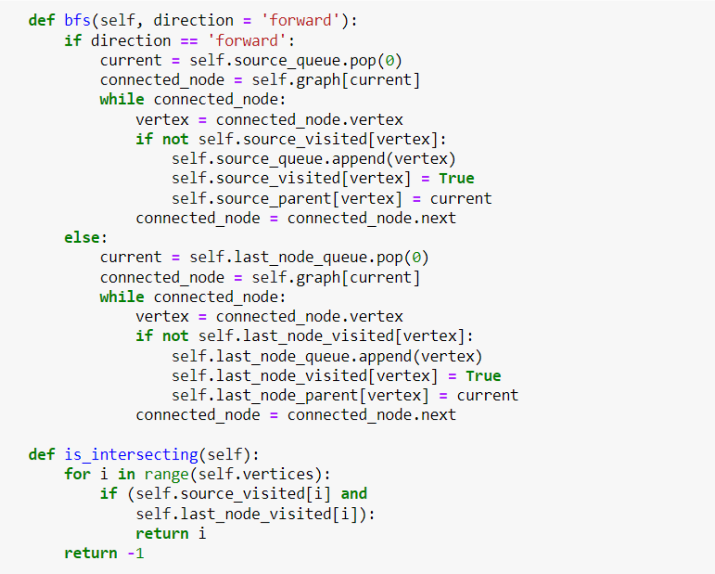
REG NO. : 20BAI1005

LAB 6

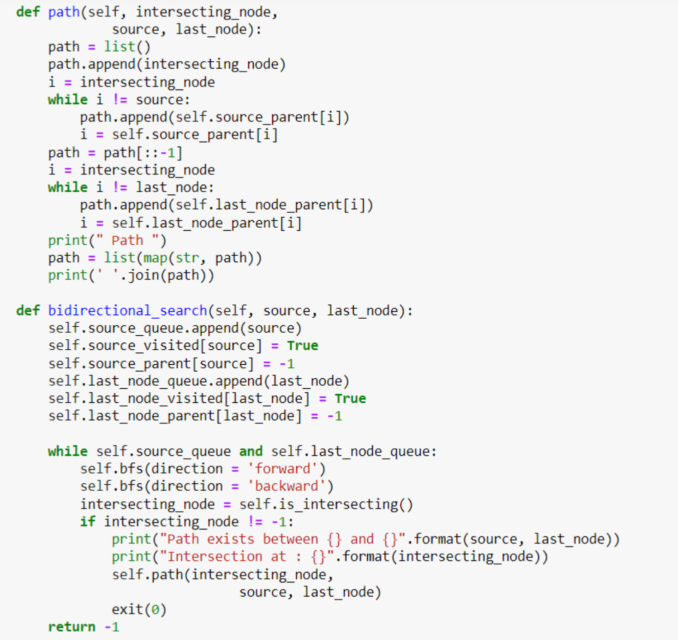
BIDIRECTIONAL A* SEARCH



```
class adjacent_node:
    def __init__(self, v):
        self.vertex = v
        self.next = None

class bidirectional_search:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = [None] * self.vertices
        self.source_queue = list()
        self.last_node_queue = list()
        self.source_visited = [False] * self.vertices
        self.last_node_visited = [False] * self.vertices
        self.source_parent = [None] * self.vertices
        self.last_node_parent = [None] * self.vertices

    def add_edge(self, source, last_node):
        node = adjacent_node(last_node)
        node.next = self.graph[source]
        self.graph[source] = node
        node = adjacent_node(source)
        node.next = self.graph[last_node]
        self.graph[last_node] = node
```



```
    def bfs(self, direction = 'forward'):
        if direction == 'forward':
            current = self.source_queue.pop(0)
            connected_node = self.graph[current]
            while connected_node:
                vertex = connected_node.vertex
                if not self.source_visited[vertex]:
                    self.source_queue.append(vertex)
                    self.source_visited[vertex] = True
                    self.source_parent[vertex] = current
                connected_node = connected_node.next
        else:
            current = self.last_node_queue.pop(0)
            connected_node = self.graph[current]
            while connected_node:
                vertex = connected_node.vertex
                if not self.last_node_visited[vertex]:
                    self.last_node_queue.append(vertex)
                    self.last_node_visited[vertex] = True
                    self.last_node_parent[vertex] = current
                connected_node = connected_node.next

    def is_intersecting(self):
        for i in range(self.vertices):
            if (self.source_visited[i] and
                self.last_node_visited[i]):
                return i
        return -1
```



```
    def path(self, intersecting_node,
             source, last_node):
        path = list()
        path.append(intersecting_node)
        i = intersecting_node
        while i != source:
            path.append(self.source_parent[i])
            i = self.source_parent[i]
        path = path[::-1]
        i = intersecting_node
        while i != last_node:
            path.append(self.last_node_parent[i])
            i = self.last_node_parent[i]
        print(" Path ")
        path = list(map(str, path))
        print(' '.join(path))

    def bidirectional_search(self, source, last_node):
        self.source_queue.append(source)
        self.source_visited[source] = True
        self.source_parent[source] = -1
        self.last_node_queue.append(last_node)
        self.last_node_visited[last_node] = True
        self.last_node_parent[last_node] = -1

        while self.source_queue and self.last_node_queue:
            self.bfs(direction = 'forward')
            self.bfs(direction = 'backward')
            intersecting_node = self.is_intersecting()
            if intersecting_node != -1:
                print("Path exists between {} and {}".format(source, last_node))
                print("Intersection at : {}".format(intersecting_node))
                self.path(intersecting_node,
                          source, last_node)
                exit(0)
        return -1
```

```python
n = 20
source = 1
last_node = 15
graph = bidirectional_search(n)
graph.add_edge(1, 4)
graph.add_edge(2, 4)
graph.add_edge(3, 6)
graph.add_edge(5, 6)
graph.add_edge(4, 8)
graph.add_edge(6, 8)
graph.add_edge(8, 9)
graph.add_edge(9, 10)
graph.add_edge(10, 11)
graph.add_edge(11, 13)
graph.add_edge(11, 14)
graph.add_edge(10, 12)
graph.add_edge(12, 15)
graph.add_edge(13, 15)
out = graph.bidirectional_search(source, last_node)
if out == -1:
    print("No path between {} and {}".format(source, last_node))
```

```
Path exists between 1 and 15
Intersection at : 9
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 9
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 8
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 4
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 3
 Path
1 4 8 6 3 6 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 1
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 1
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 1
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 1
 Path
1 4 8 9 10 12 15
Path exists between 1 and 15
Intersection at : 1
 Path
1 4 8 9 10 12 15
No path between 1 and 15
```