

Name: Richard Nadar
Reg no: RA1911030010109
Date: 15-2-22

EXP 5: Implementation of Best First Search

CODE:

```
from queue import PriorityQueue

v = 14

graph = [[] for i in range(v)]

def best_first_search(source, target, n):
    visited = [0] * n
    visited[0] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()
```

```
# Function for adding edges to graph
```

```
def addedge(x, y, cost):
```

```
    graph[x].append((y, cost))
```

```
    graph[y].append((x, cost))
```

```
# The nodes shown in above example(by alphabets) are
```

```
# implemented using integers addedge(x,y,cost);
```

```
adddedge(0, 1, 3)
```

```
adddedge(0, 2, 6)
```

```
adddedge(0, 3, 5)
```

```
adddedge(1, 4, 9)
```

```
adddedge(1, 5, 8)
```

```
adddedge(2, 6, 12)
```

```
adddedge(2, 7, 14)
```

```
adddedge(3, 8, 7)
```

```
adddedge(8, 9, 5)
```

```
adddedge(8, 10, 6)
```

```
adddedge(9, 11, 1)
```

```
adddedge(9, 12, 10)
```

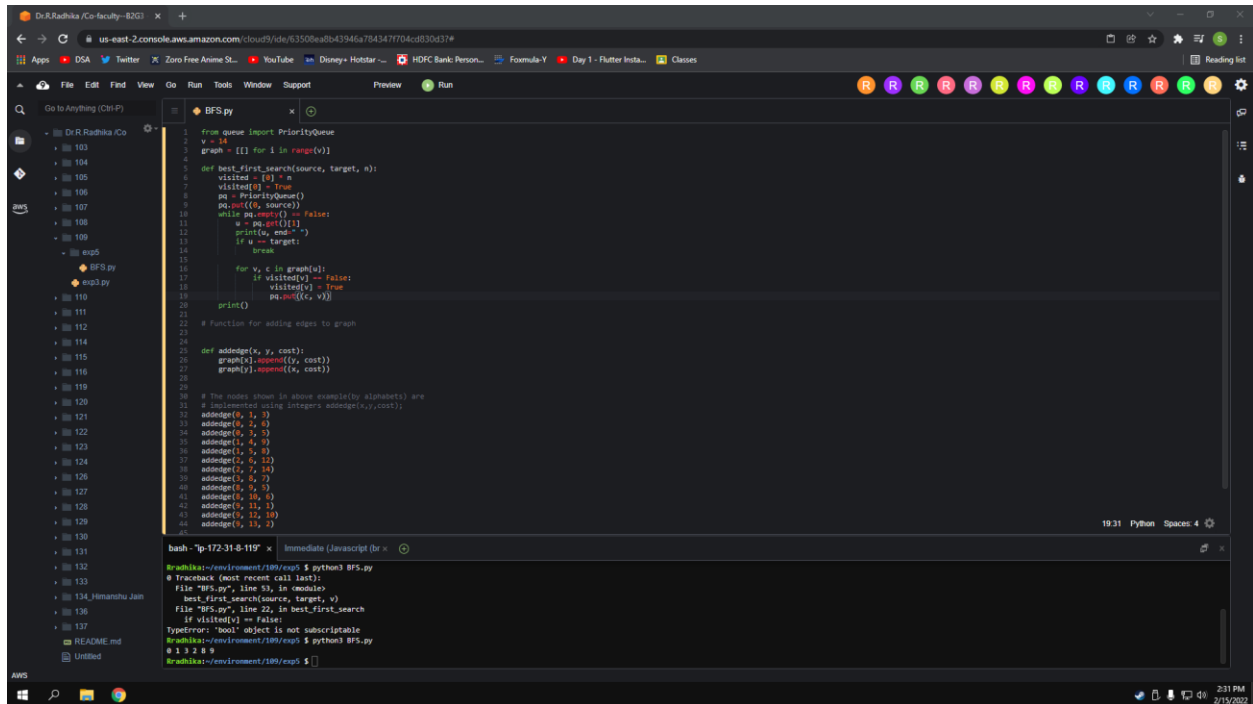
```
adddedge(9, 13, 2)
```

```
source = 0
```

```
target = 9
```

```
best_first_search(source, target, v)
```

OUTPUT:



The screenshot shows a VS Code editor window with a file named `BFS.py` open. The file contains a Python implementation of a Breadth-First Search (BFS) algorithm. The code defines a `best_first_search` function that takes `source` and `target` as arguments. It uses a `PriorityQueue` to explore nodes, starting from the `source` and visiting nodes in order of their distance from the source. The code also includes a `addedge` function to add edges to the graph. The graph is represented as a dictionary where keys are nodes and values are lists of adjacent nodes. The code includes comments explaining the steps and the data structures used.

```
1 from queue import PriorityQueue
2 n = 14
3 graph = [[] for i in range(n)]
4
5 def best_first_search(source, target, n):
6     visited = [0] * n
7     visited[source] = True
8     pq = PriorityQueue()
9     pq.put((0, source))
10    while pq.empty() == False:
11        u = pq.get()[1]
12        print(u, end=" ")
13        if u == target:
14            break
15        for v, c in graph[u]:
16            if visited[v] == False:
17                visited[v] = True
18                pq.put((c, v))
19    print()
20
21 # Function for adding edges to graph
22
23
24
25 def addedge(x, y, cost):
26     graph[x].append((y, cost))
27     graph[y].append((x, cost))
28
29
30 # The nodes shown in above example (u, alphabet) are
31 # implemented using integers addedge(x,y,cost);
32
33 addedge(0, 1, 10)
34 addedge(0, 2, 4)
35 addedge(0, 3, 5)
36 addedge(1, 4, 3)
37 addedge(1, 5, 8)
38 addedge(2, 6, 12)
39 addedge(2, 7, 14)
40 addedge(3, 8, 7)
41 addedge(4, 9, 5)
42 addedge(5, 10, 6)
43 addedge(6, 11, 1)
44 addedge(6, 12, 10)
45 addedge(7, 13, 2)
```

The terminal output shows the execution of the `BFS.py` file. It displays a traceback for a `TypeError: 'bool' object is not subscriptable` error. The error occurs in the `best_first_search` function, specifically in the line `if visited[u] == False:`. The error message indicates that the variable `u` is a boolean value, which is not subscriptable. The terminal output also shows the command `python3 BFS.py` being executed.

```
bash - "ip-172-31-0-119" x Immediate (JavaScript (br x)
kradhika:~/environment/189/exp5 $ python3 BFS.py
# Traceback (most recent call last):
# File "BFS.py", line 53, in <module>
#   best_first_search(source, target, n)
# File "BFS.py", line 22, in best_first_search
#   if visited[u] == False:
# TypeError: 'bool' object is not subscriptable
kradhika:~/environment/189/exp5 $ python3 BFS.py
# 1 2 3 4 5
kradhika:~/environment/189/exp5 $
```