



THE UNIVERSITY  
OF LAHORE  
**ISLAMABAD  
CAMPUS**

## **Artificial Intelligence (CS13217)**

### **Lab Report**

Name: Sameea Naeem  
Registration #: CSU-XS16-139  
Lab Report #: 03  
Submitted To: Mr. Usman Ahmed

The University of Lahore, Islamabad Campus  
Department of Computer Science & Information Technology

# Experiment # 3

## Implementing Breadth first search algorithm

### Objective

To understand and implement the BFS algorithm.

### Software Tool

1. Sub lime text
2. python
3. miktex

## 1 Theory

Breadth First Search (BFS) searches breadth-wise in the problem space. Breadth-First search is like traversing a tree where each node is a state which may be a potential candidate for solution. It expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found. It is very easily implemented by maintaining a queue of nodes. Initially the queue contains just the root. In each iteration, node at the head of the queue is removed and then expanded. The generated child nodes are then added to the tail of the queue.

## 2 Task

Implement BFS on graph shown in figure 2

### 2.1 Code

```
graph = { # sample graph implemented as a dictionary
    '0' : ['1', '2', '3', '4'],
    '1' : ['0', '5'],
```

```

File Edit Selection Find View Goto Tools Project Preferences Help
graph.py
1 graph = { # sample graph implemented as a dictionary
2   '0': ['1','2','3','4'],
3   '1': ['0','5'],
4   '2': ['0','5'],
5   '3': ['0','5'],
6   '4': ['0','5'],
7   '5': ['1','2','3','4'],
8   '6': ['7','4','7'],
9   '7': ['6','5'],
10 }
11 # visits all the nodes of a graph (connected component) using BFS
12 def bfs_connected_component(graph, start):
13     explored = [] # keep track of all visited nodes
14     queue = [start] # keep track of nodes to be checked
15     levels = {} # this dict keeps track of levels
16     levels[start] = 0 # depth of start node is 0
17     visited[start] = True # to avoid inserting the same node twice into the queue
18     # keep looping until there are nodes still to be checked
19     while queue:
20         node = queue.pop(0) # pop shallowest node (first node) from queue
21         explored.append(node)
22         neighbours = graph[node]
23         for neighbour in neighbours: # add neighbours of node to queue
24             if neighbour not in visited:
25                 queue.append(neighbour)
26                 visited.append(neighbour)
27                 levels[neighbour] = levels[node] + 1 # print(neighbour, ">", levels[neighbour])
28     print(levels)
29     return explored
30 ans = bfs_connected_component(graph, '0') # returns ['A', 'B', 'C', 'D', 'E', 'F', 'G']
31 print(ans)
32
33 { '1': 1, '0': 0, '3': 1, '2': 1, '5': 2, '4': 1, '7': 3, '6': 2 }
34 ['0', '1', '2', '3', '4', '5', '6', '7']
35 [finished in 0.3s]

```

Figure 1: Implementation of BFS using python

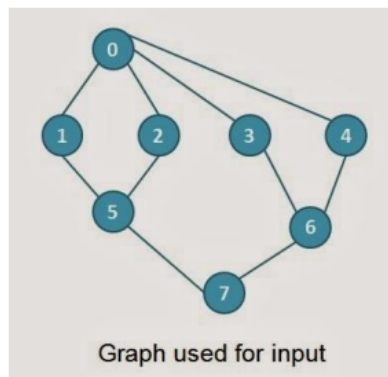


Figure 2: Graph

```

    '2' : [ '0', '5' ],
    '3' : [ '0', '6' ],
    '4' : [ '0', '6' ],
    '5' : [ '1', '2', '7' ],
    '6' : [ '3', '4', '7' ],
    '7' : [ '6', '5' ],
}
# visits all the nodes of a graph (connected component) using BFS
def bfs_connected_component(graph, start):
    explored = []
    queue = [start]
    levels = {}
    levels[start] = 0
    visited = [start]

    while queue:
        node = queue.pop(0)
        explored.append(node)
        neighbours = graph[node]
        for neighbour in neighbours:
            if neighbour not in visited:
                queue.append(neighbour)
                visited.append(neighbour)
        levels[neighbour] = levels[node] + 1

    print(levels)
    return explored
ans = bfs_connected_component(graph, '0')
# returns [ '1', '2', '3', '4', '5', '6', '7' ]
print(ans)

```

### 3 Conclusion

With nodes starting from root node, 0 at the first level, 1,2,3 and 4 at the second level and 5, 6 and 7 at the third level. If we want to search for node 7 then BFS will search level by level. First it will check if 7 exists at the root. Then it will check nodes at the second level. Finally it will find 7 at the third level.