Artificial Intelligence
(CS13217)

Lab Report 3

Name:	Rabia anwar
Registration #:	csu-s15-114
Lab Report #:	03
Dated:	16-04-2018
Submitted To:	Mr. Usman Ahmed

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

Experiment 3
Implementing breath first searchi

Objective
Breadth-first search **Software Tool**
1. opertaing system window 10
2.subline version 3.0
3. python

1 Theory

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'[1]) and explores the neighbor nodes first, before moving to the next level neighbours.

BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse and Michael Burke, in their (rejected) Ph.D.
1. Root case : The traversal queue is initially empty so the root node must be added before the general case..
2. General case: Process any items in the queue, while also expanding their children, stop if the queue was empty. The general case will halt after processing the bottom level as leaf nodes have no children....
3.Input: A search problem. A search-problem abstracts out the problem specific requirements from the actual search algorithm.

Output: An ordered list of actions to be followed to reach from start state to the goal state. ...

Figure 1: Time Independent Feature Set

2 Task

2.1 Procedure: Task 3

2.2 Procedure: Task 3

```
graph = { # sample graph implemented as a dictionary
graph = { # sample graph implemented as a dictionary
    'A' : ['B','C'],
    'B' : ['D','E'],
    'C' : ['A','E'],
    'D' : ['B','E'],
    'E' : ['C','F','D','B'],
    'F' : ['D','E'],}
# visits all the nodes of a graph (connected component) using BFS
def bfs_connected_component(graph, start):
    explored = [] # keep track of all visited nodes
    queue = [start] # keep track of nodes to be checked
    levels = {}          # this dict keeps track of levels
    levels[start]= 0     # depth of start node is 0
    visited= [start]      # to avoid inserting the same node twice into the
    # keep looping until there are nodes still to be checked
    while queue:
        node = queue.pop(0) # pop shallowest node (first node) from queue
        explored.append(node)
        neighbours = graph[node]
        for neighbour in neighbours: # add neighbours of node to queue
            if neighbour not in visited:
                queue.append(neighbour)
```

```
                visited.append(neighbour)
                levels[neighbour]= levels[node]+1 # print(neighbour, ">>",

    print(levels)
    return explored
ans = bfs_connected_component(graph,'A') # returns ['A', 'B', 'C', 'E', 'D
```

3 Conclusion