Assignment No:02

CSE-0408 Summer 2021

Md. Khaled Saifullah Sadi
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
mdsadi4@gmail.com

***Abstract*—Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.**

n

***Index Terms*—heuristic, 8 puzzle**

I. Introduction

Breadth First Search (BFS) is an algorithm for traversing or searching layerwise in tree or graph data structures.If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step

II. Literature Review

Breadth First Search (BFS) is a prominent workbench model for measuring the performance of heuristic search algorithms [Nilsson, 1969; Gaschnig, 1970; Pearl, 1995; Fardeen, 1992], learning methods [Laird et a/., 1977] and the use of macro operators [Korf, 1995a].

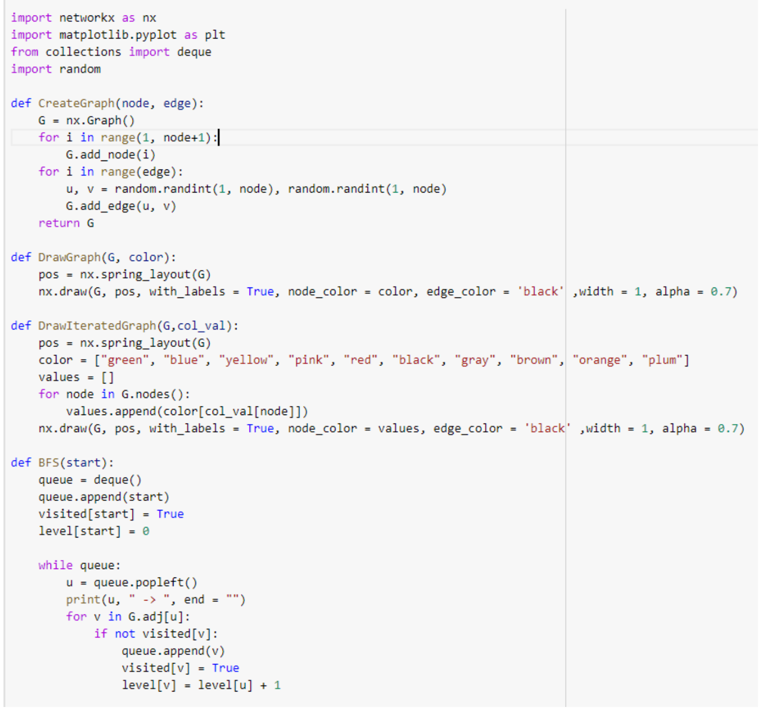
III. Proposed Methodology

.Begin the search algorithm, by knowing the key which is to be searched. Once the key/element to be searched is decided the searching begins with the root (source) first

1) procedure BFS(G, root) is
2) let Q be a queue
3) label root as explored
4) Q.enqueue(root)
5) while Q is not empty do
6) v := Q.dequeue()
7) if v is the goal then
8) return v
9) for all edges from v to w in
10) for all edges from v to w in
11) G.adjacentEdges(v) do
12) if w is not labeled as explored then
13) label w as explored
14) Q.enqueue(w)

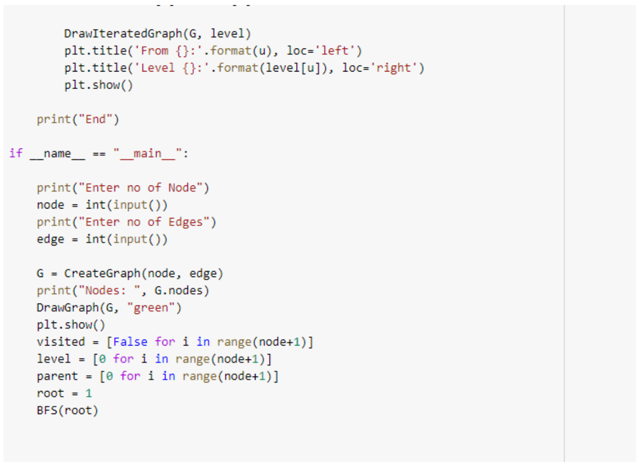
We are using NetworkX for creating Graph. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks(Graph). Also using matplotlib.pyplot, we can graphically represent our graph.

IV. Some Screansoot from Code



```python
import networkx as nx
import matplotlib.pyplot as plt
from collections import deque
import random

def CreateGraph(node, edge):
    G = nx.Graph()
    for i in range(1, node+1):
        G.add_node(i)
    for i in range(edge):
        u, v = random.randint(1, node), random.randint(1, node)
        G.add_edge(u, v)
    return G

def DrawGraph(G, color):
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels = True, node_color = color, edge_color = 'black' ,width = 1, alpha = 0.7)

def DrawIteratedGraph(G,col_val):
    pos = nx.spring_layout(G)
    color = ["green", "blue", "yellow", "pink", "red", "black", "gray", "brown", "orange", "plum"]
    values = []
    for node in G.nodes():
        values.append(color[col_val[node]])
    nx.draw(G, pos, with_labels = True, node_color = values, edge_color = 'black' ,width = 1, alpha = 0.7)

def BFS(start):
    queue = deque()
    queue.append(start)
    visited[start] = True
    level[start] = 0

    while queue:
        u = queue.popleft()
        print(u, " -> ", end = "")
        for v in G.adj[u]:
            if not visited[v]:
                queue.append(v)
                visited[v] = True
                level[v] = level[u] + 1
```

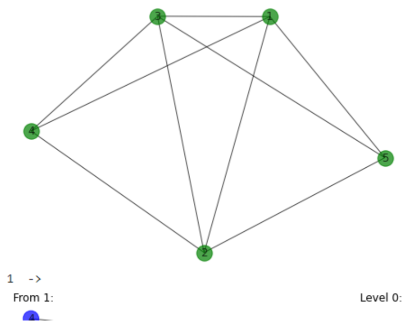
Fig. 1. Code



```
        DrawIteratedGraph(G, level)
        plt.title('From {}:'.format(u), loc='left')
        plt.title('Level {}:'.format(level[u]), loc='right')
        plt.show()

    print("End")

if __name__ == "__main__":

    print("Enter no of Node")
    node = int(input())
    print("Enter no of Edges")
    edge = int(input())

    G = CreateGraph(node, edge)
    print("Nodes: ", G.nodes)
    DrawGraph(G, "green")
    plt.show()
    visited = [False for i in range(node+1)]
    level = [0 for i in range(node+1)]
    parent = [0 for i in range(node+1)]
    root = 1
    BFS(root)
```

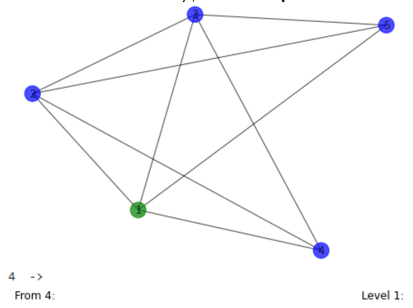
Fig. 2. Code

V. Some Screansoot from OutPut



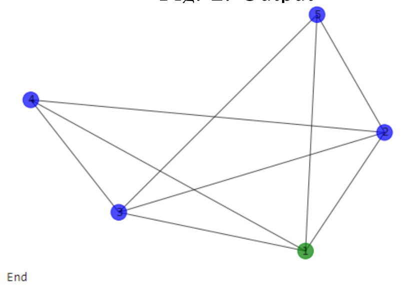
```
1  ->
```

Fig. 1. Output



```
4  ->
```

Fig. 2. Output



```
End
```

Fig. 2. Output

VI. Conclusion

The BFS algorithm is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Piltaver, R., Lustrek, M., and Gams, M. (2015). Breadth First Search (BFS). Journal of Experimental and Theoretical Artificial Intelligence, 24(1), 65-94