A STUDY ON BFS AND DFS ADAPTIVE ALGORITHMS WITH APPLICATION IN COMPUTER SCIENCE

Abstract

Many real life problems exhibit a connectivity structure in nature. Before, data solving techniques for including, bioinformatics, communication network, image data, wireless networks etc., are more complicated because of high computational complexity. Hence, nowadays, there are lots of graph models and these can be solved using graph theory algorithms such as BFS, DFS, Dijkstra's algorithm and so on. These algorithms are applied in data structures. This paper explains BFS and DFS algorithms with application.

Keywords: Graph, Directed Graph, Connectivity, Trees, Subgraph, Spanning subgraph, Spanning trees.

Author

Girija B
Research Scholar
Oscar College
Vellore, India.
girijaranjith2017@gmail.com

I. INTRODUCTION

Mathematics plays an immense role in many fields; especially Graph Theory occupies an important role in the field of computer science. Graph theory is a mathematical model of pair wise relations between objects. Graphs are the convenient to represent mathematical objects. There is a wide range of application of graph theory in computer science.

Here, we will see, the algorithms such as BFS ANS DFS, with applications.

II. THE BREADTH FIRST TRAVERSAL ALGORITHM

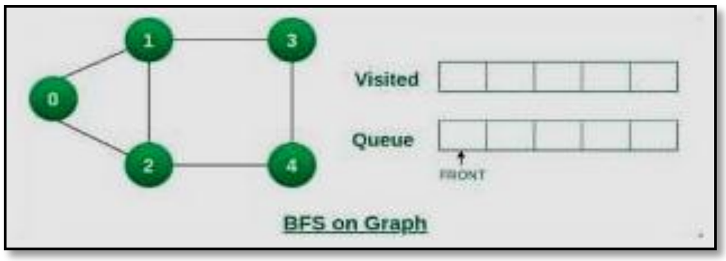
This algorithm is used to search a graph data structure with vertices. It starts at the root of the graph and travels all the vertices at the current depth level. The BFS for a graph is similar to BFS of a tree. The only difference is graphs contain cycles but trees are not. To avoid repeated travel of same vertices, we divide into two categories:

- Visited
- Not visited.

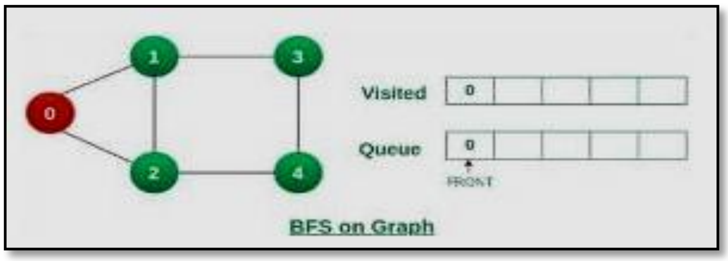
First we assume that all the vertices are reachable with the starting vertex. BFS uses a queue data structure for traversal. Starting from the first vertex, all the vertices in a particular level are visited first and the vertices in the next level are visited. All the adjacent unvisited vertices are pushed into the queue, and the vertices of current level are marked visited, and popped from the queue.

Let us understand the working algorithm of BFS with the following simple example:

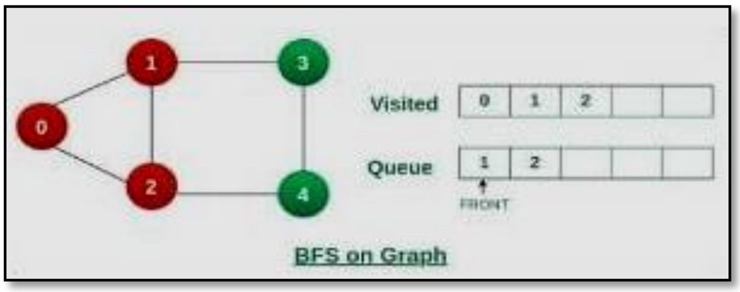
Step-1: Initially the queue and visited arrays are empty.



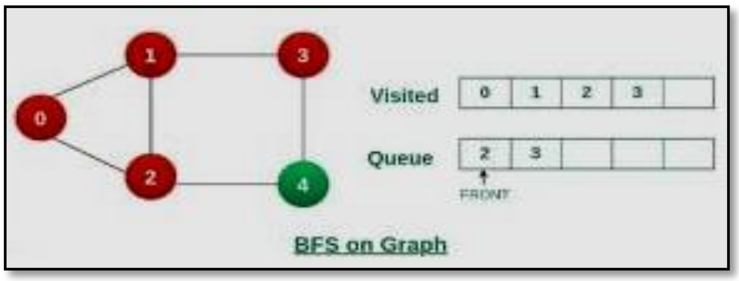
Step-2: Push node 0 into queue and mark visited.



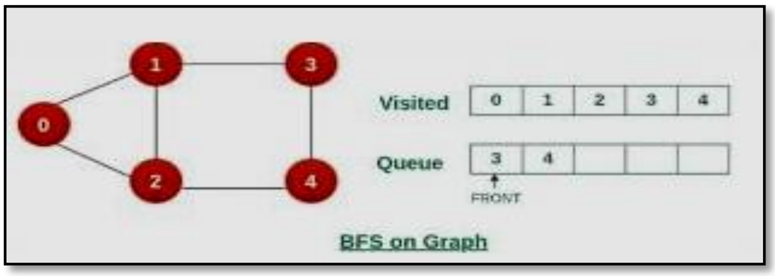
Step-3: Remove node 0 from the front of the queue and visit the unvisited neighbors and push them into queue.



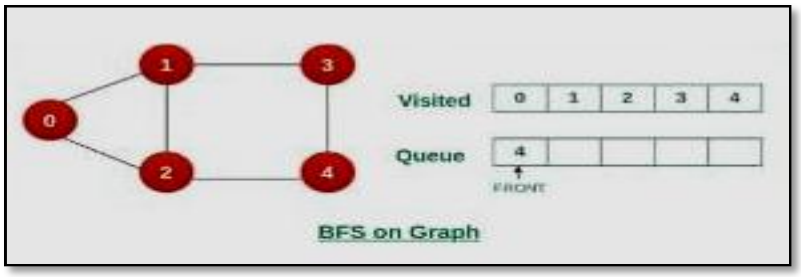
Step-4: Remove node 1 from the front of the queueand visit the unvisited neighbors and push them into queue.



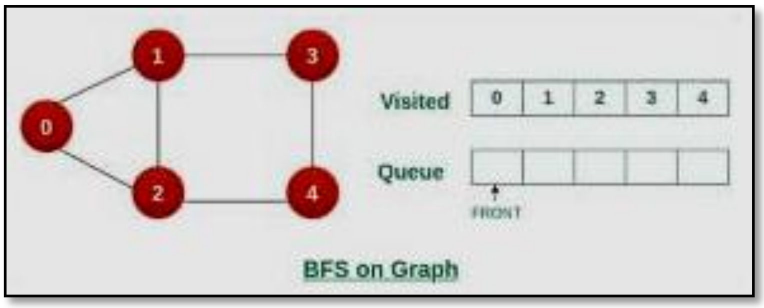
Step-5: Remove node 2 from the queue and visit the unvisited neighbors and push them into queue.



Step-6: Remove node 3 form the queue and visit the unvisited neighbors. As we can see that every neighbor of node 3 visited, so move to the next node that is in front of the queue.



Step-7: Remove node 4 from the front of the queue and visit the unvisited neighbors. As we can see that neighbors of node 4 visited, so move to the next node that is in front of the queue.



Now, queue become empty, so terminate the process of iteration.

III. ILLUSTRATION C PROGRAM FOR BFS ALGORITHM:

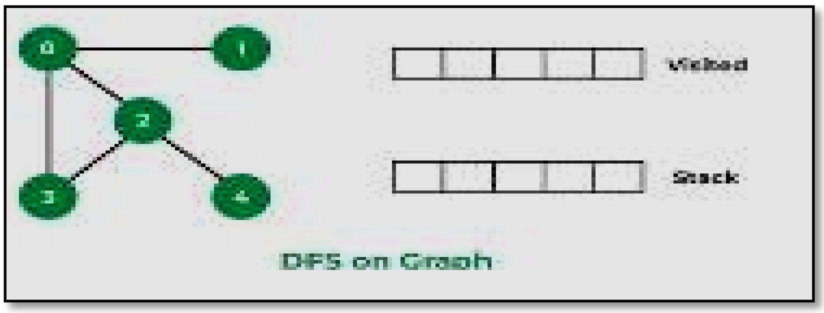
```
        #define MAX_VERTICES 50
        typedef struct Graph_t {

        // No. of vertices
        int V;
        bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
        Graph* Graph_create(int V)
{
        Graph* g = malloc(sizeof(Graph));
        g->V = V;

        for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                        g->adj[i][j] = false;
        }
        }

        return g;
}
void Graph_destroy(Graph* g) { free(g); }
void Graph_addEdge(Graph* g, int v, int w)
{
        g->adj[v][w] = true;
}
void Graph_BFS(Graph* g, int s)
{
        bool visited[MAX_VERTICES];
        for (int i = 0; i < g->V; i++) {
                visited[i] = false;
        }
        int queue[MAX_VERTICES];
        int front = 0, rear = 0;
        visited[s] = true;
        queue[rear++] = s;
        while (front != rear) {
                s = queue[front++];
                printf("%d ", s);
                for (int adjacent = 0; adjacent < g->V;
                        adjacent++) {
                        if (g->adj[s][adjacent] && !visited[adjacent]) {
                                visited[adjacent] = true;
                                queue[rear++] = adjacent;
                        }
                }
        }
}
int main()
{
        // Create a graph
        Graph* g = Graph_create(4);
        Graph_addEdge(g, 0, 1);
        Graph_addEdge(g, 0, 2);
        Graph_addEdge(g, 1, 2);
        Graph_addEdge(g, 2, 0);
        Graph_addEdge(g, 2, 3);
        Graph_addEdge(g, 3, 3);
        printf("Following is Breadth First Traversal "
        Graph_BFS(g, 2);
                "(starting from vertex 2) \n");
        Graph_destroy(g);
        return 0;
}
```

IV. THE DEPTH FIRST SEARCH ALGORITHM

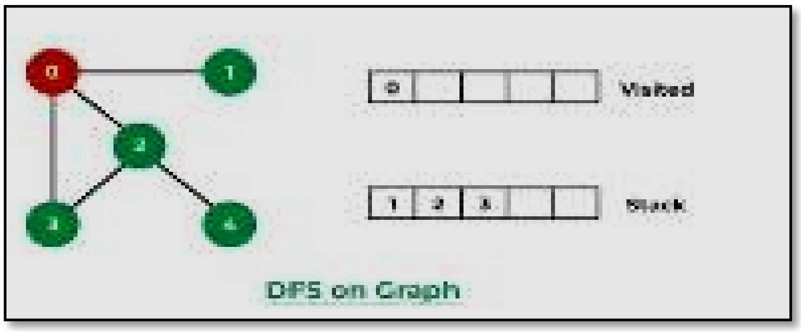
DFS of a graph is similar to DFS traversal of a tree. A graph can have more than one DFS traversal. This algorithm for traversing or searching tree or graph data structures. The algorithm starts with a root node ans explores as far as possible along each branch before backtracking.

Let us understand the working of DFS with the following illustration.

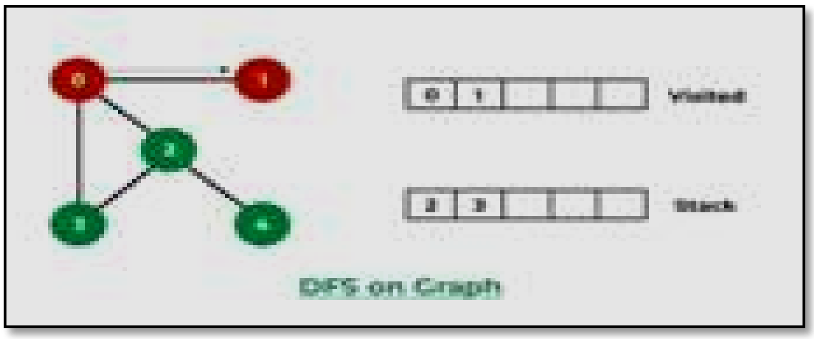
Step 1: Initially stack and visited arrays are empty



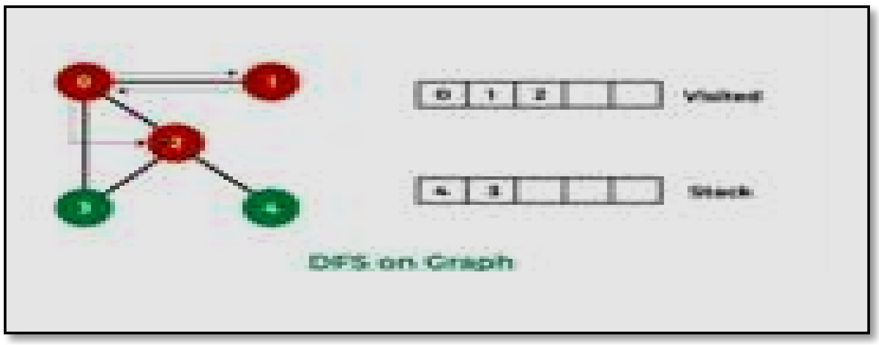
Step2 : Visit 0 and put its adjacent nodes which are not visited yet into the stack



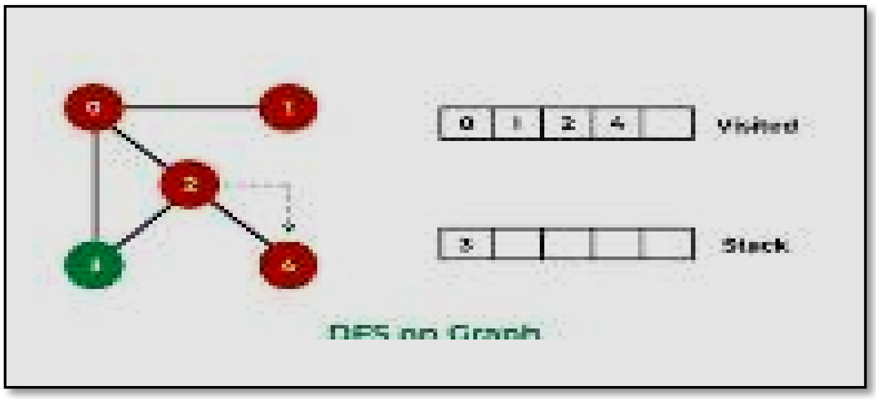
Step 3 : Now,Nodes 1 at the top of the stack ,So bisit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack



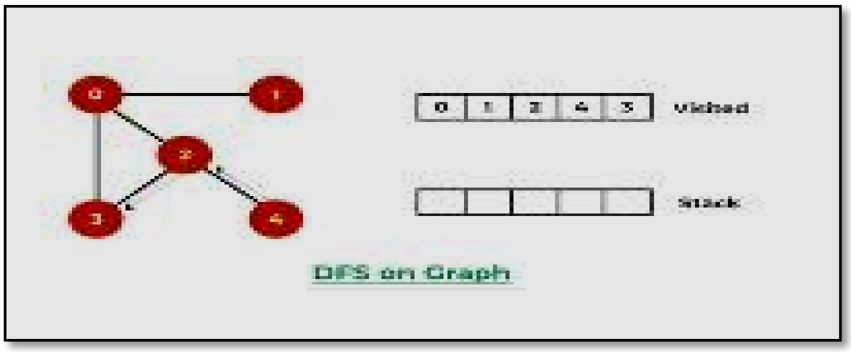
Step 4 : Now,Node 2 at the top of the stack,So visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (eg : 3,4) in the stack



Step 5: Now, Node 4 at the top of the stack,So visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



Step 6 : Now, Node 3 at the top of the stack,So visit node 3 pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



Now stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

Let us implement above process by C Program

```
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;
  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);
  while (temp != NULL) {
    int connectedVertex = temp->vertex;
    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}
  struct node* newNode = malloc(sizeof(struct node));
struct node* createNode(int v) {
// Create a node
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));
  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
}
  addEdge(graph, 2, 3);
  printGraph(graph);
  DFS(graph, 2);
  return 0;
  addEdge(graph, 1, 2);
```

V. THE COMPARISON BETWEEN BFS AND DFS ALGORITHMS

BFS is a vertex based technique and it uses queue data structure whereas DFS is an edge based technique and uses stack data structure.

The comparison between output of BFS and DFS Algorithms: In both BFS and DFS algorithms we get same output. But BFS builds the tree level by level whereas DFS by sub tree by sub tree level. BFS is used to search nearby vertices for given source whereas DFS is used to when the vertices are away from the given source. BFS requires more memory but DFS doesn't. The time complexity O(V+E) is same for both algorithms. But BFS nees more space than DFS.

VI. CONCLUSION

In this paper, we discuss about the algorithms of BFS and DFS with example and with implementation. Both the algorithms are very useful and easy to understand. We can use either BFS or DFS which suits for our program

REFERENCES

[1] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
[2] https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
[3] https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/
Elumalai, "Graph theory applications in computer science and engineerin", Malaya Journal of Matematik, Vol. S, No : 4025-4027, 2000.
[4] Serafino Cicerone, Gabriele Di Stafano, "Graph algorithms and Applications" ISBN 978-3-0365-1542-7 (Hbk), ISBN 978-3-0365-1541-0 (PDF)