**Answer to the Question No.1**

**There are two widely used method to represent a graph in memory.**

1. Adjacency Matrix
2. Adjacency List

**Difference between Adjacency Matrix and Adjacency List**

|  |  |
| --- | --- |
| **Adjacency Matrix** | **Adjacency List** |
| To represent in adjacency matrix we need a 2d array. | To represent in adjacency List we need a arrays of linked list or vector . |
| Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j | An Adjacency list is an array consisting of the address of all the linked lists. |
| When the graph is dense,it is better to use graph representation | When the graph is spouse, it is better to use list representation. |
| Adjacency matrix for undirected graph is always symmetric. |  |

**Advantages of Adjacency Matrix Representation**

1. We can determine if two vertices are adjacent to each other in constant time.
2. We can add an edge in the graph in constant time.
3. We can delete an edge form the graph in constant time.

**Disadvantages of Adjacency Matrix Representation**

1. The size of adjacency matrix is V2. Suppose there is a graph with 1000 vertices and 1 edge. We are using an array of size 10002 for storing one edge which is a waste of memory.
2. Traversing the graph using algorithms like DPS/BFS requires O(V2) time in case of adjacency matrix whereas we can traverse the graph in O(V+E) time using adjacency list.
3. the main disadvantage is its large memory complexity.

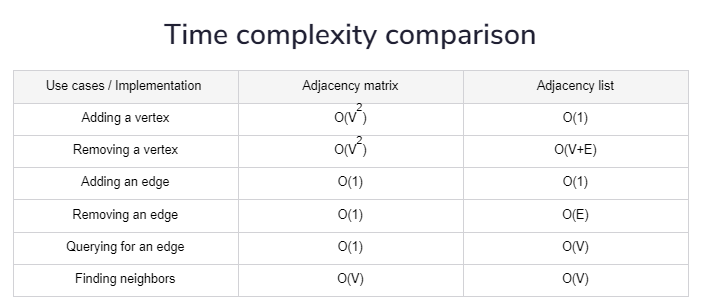
**Advantages of Adjacency Matrix Representation**

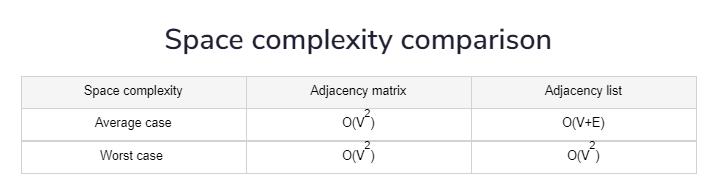
1. The advantage of the adjacency list implementation is that it allows us to compactly represent a sparse graph.
2. The adjacency list also allows us to easily find all the links that are directly connected to a particular vertex.

**Disadvantages of Adjacency List Representation**

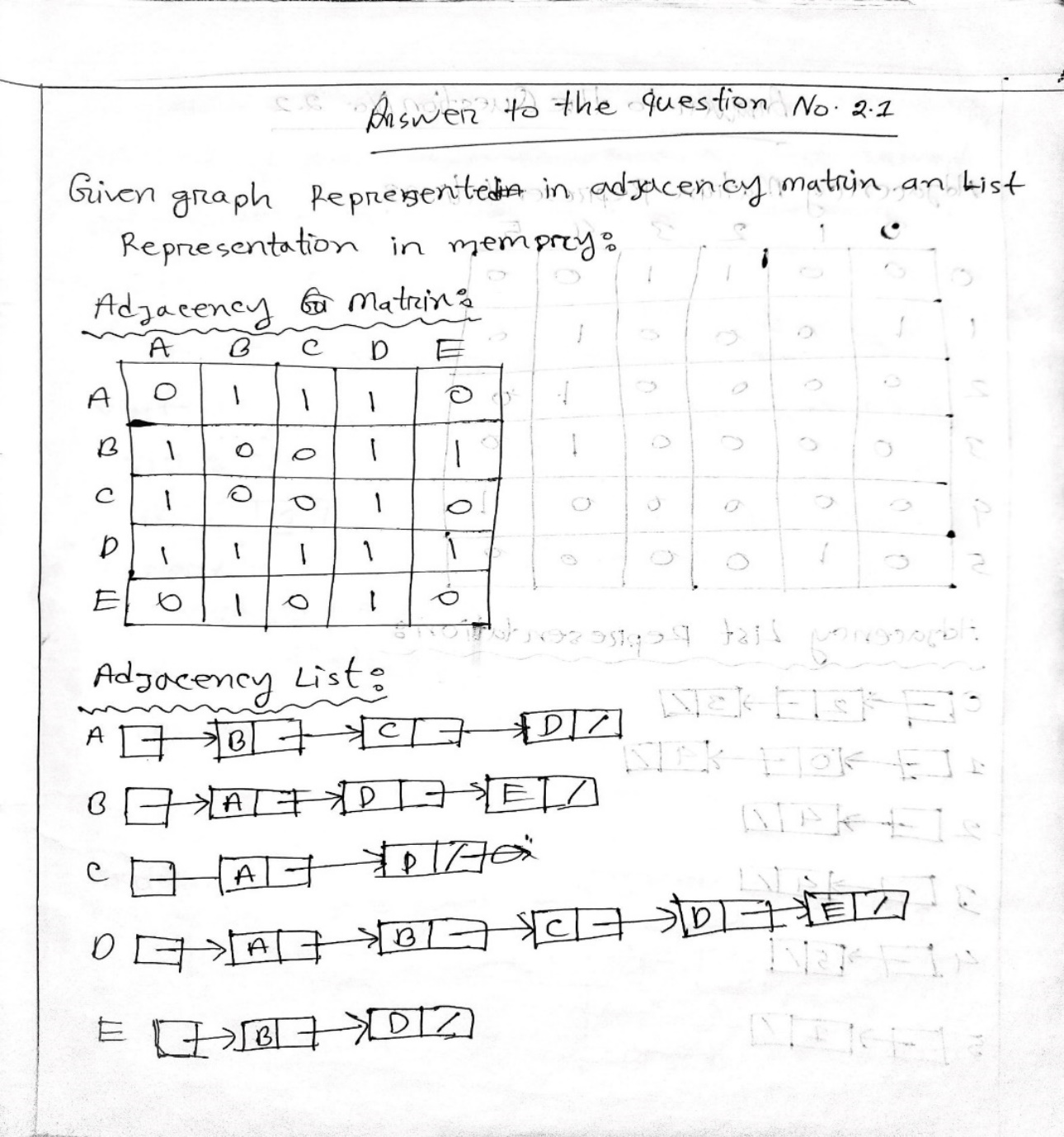
1. A potential disadvantage of the adjacency-list representation is that it provides no quicker way to determine whether a given edge (u, v) is present in the graph than to search for v in the adjacency list Adj[u]

**Complexity**

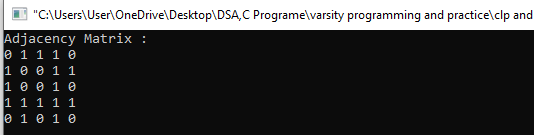


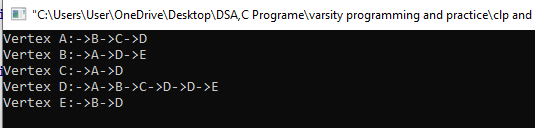


**Answer to the Question No: 2.1**

****

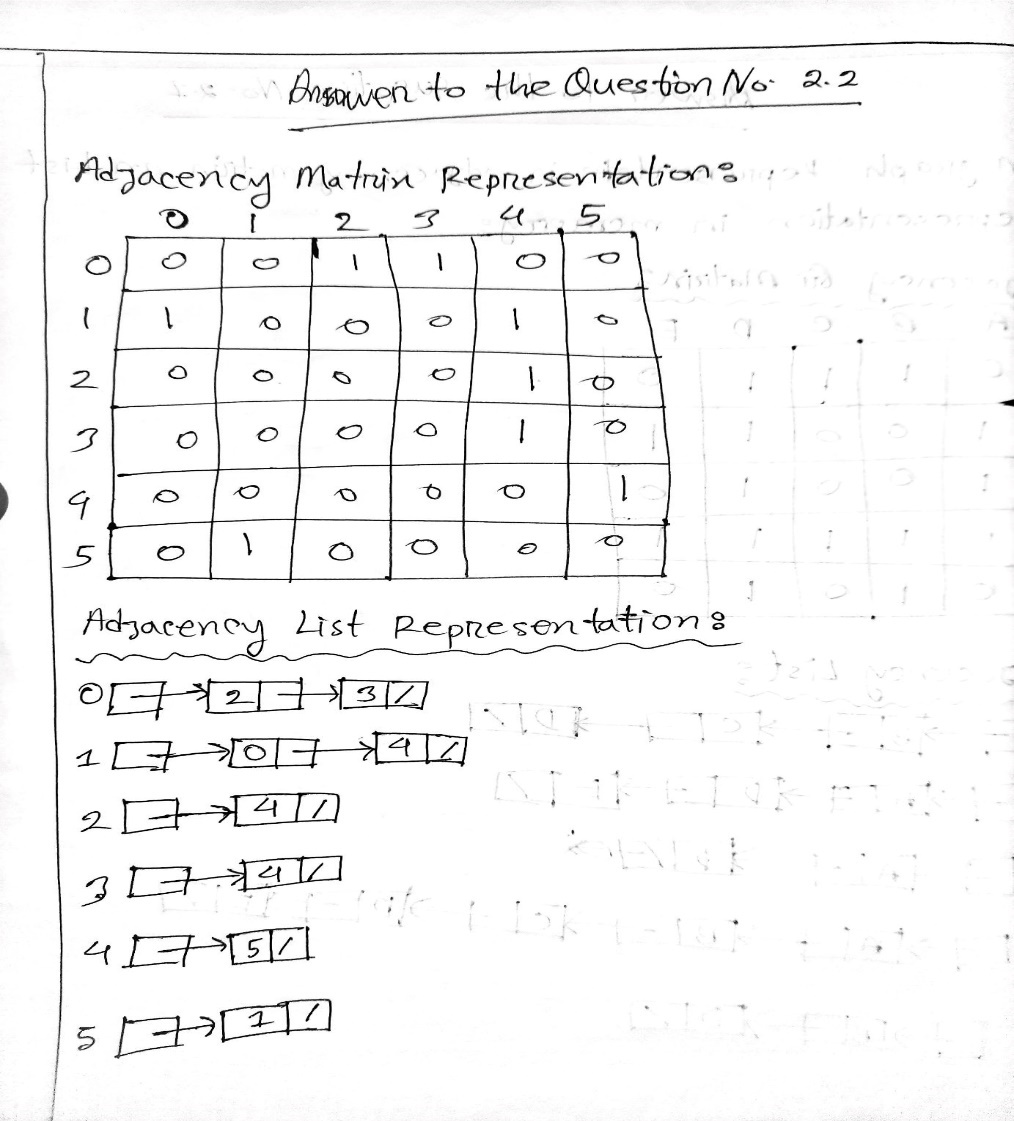
**Output Screenshot 2.1 Matrix and List**

****

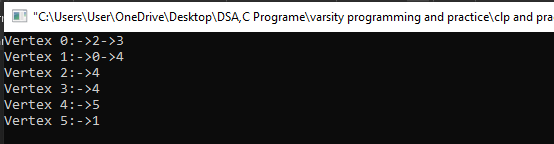
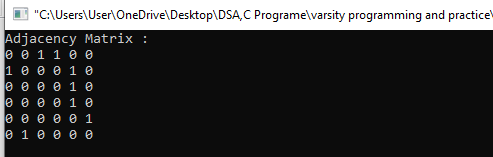
****

**Graph 2.1 And 2.2 cpp file link:**

<https://drive.google.com/drive/folders/1KOrV-xJmQ0a23SOh0ba2x5Im7CBCS6pE?usp=sharing>

**Answer to the question No. 2.2**

**Adjacency List Output Screenshot: graph 2.2**

****

**Answer to the Question No.3**

**Difference Between DFS And BFS**

|  |  |  |
| --- | --- | --- |
| **Term** | **BFS** | **DFS** |
| **1.Stands for** | BFS stands for Breadth First Search | DFS stands for Depth First Search. |
| **2.Data Structure** | BFS(Breadth First Search) uses Queue data structure for finding the shortest path. | DFS(Depth First Search) uses Stack data structure. |
| **3.Definition** | BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level. | DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes. |
| **4.Conceptual Difference** | BFS builds the tree level by level. | DFS builds the tree sub-tree by sub-tree. |
| |  |  | | --- | --- | | **5.** | **Approach used** | | It works on the concept of FIFO (First In First Out). | It works on the concept of LIFO (Last In First Out). |
| |  |  | | --- | --- | | **6.** | **Suitable for** | | BFS is more suitable for searching vertices closer to the given source. | DFS is more suitable when there are solutions away from source. |
| |  |  | | --- | --- | | **7.** | **Backtracking** | | In BFS there is no concept of backtracking. | DFS algorithm is a recursive algorithm that uses the idea of backtracking |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **8.** | **Memory** | | BFS requires more memory. | DFS requires less memory. |
| |  |  | | --- | --- | | **9.** | **Optimality** | | BFS is optimal for finding the shortest path. | DFS is not optimal for finding the shortest path. |
| |  |  | | --- | --- | | **10.** | **Speed** | | BFS is slow as compared to DFS. | DFS is fast as compared to BFS. |

**Application of DFS Algorithm**

1. For finding the path
2. To test if the graph is bipartite
3. For finding the strongly connected components of a graph
4. For detecting cycles in a graph

**BFS Algorithm Applications**

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. In Ford-Fulkerson algorithm to find maximum flow in a network
5. Cycle detection in an undirected graph
6. In [minimum spanning tree](https://www.programiz.com/dsa/spanning-tree-and-minimum-spanning-tree)

**Complexity**

**Time Complexity BFS:**

The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.

**Time Complexity DFS:**

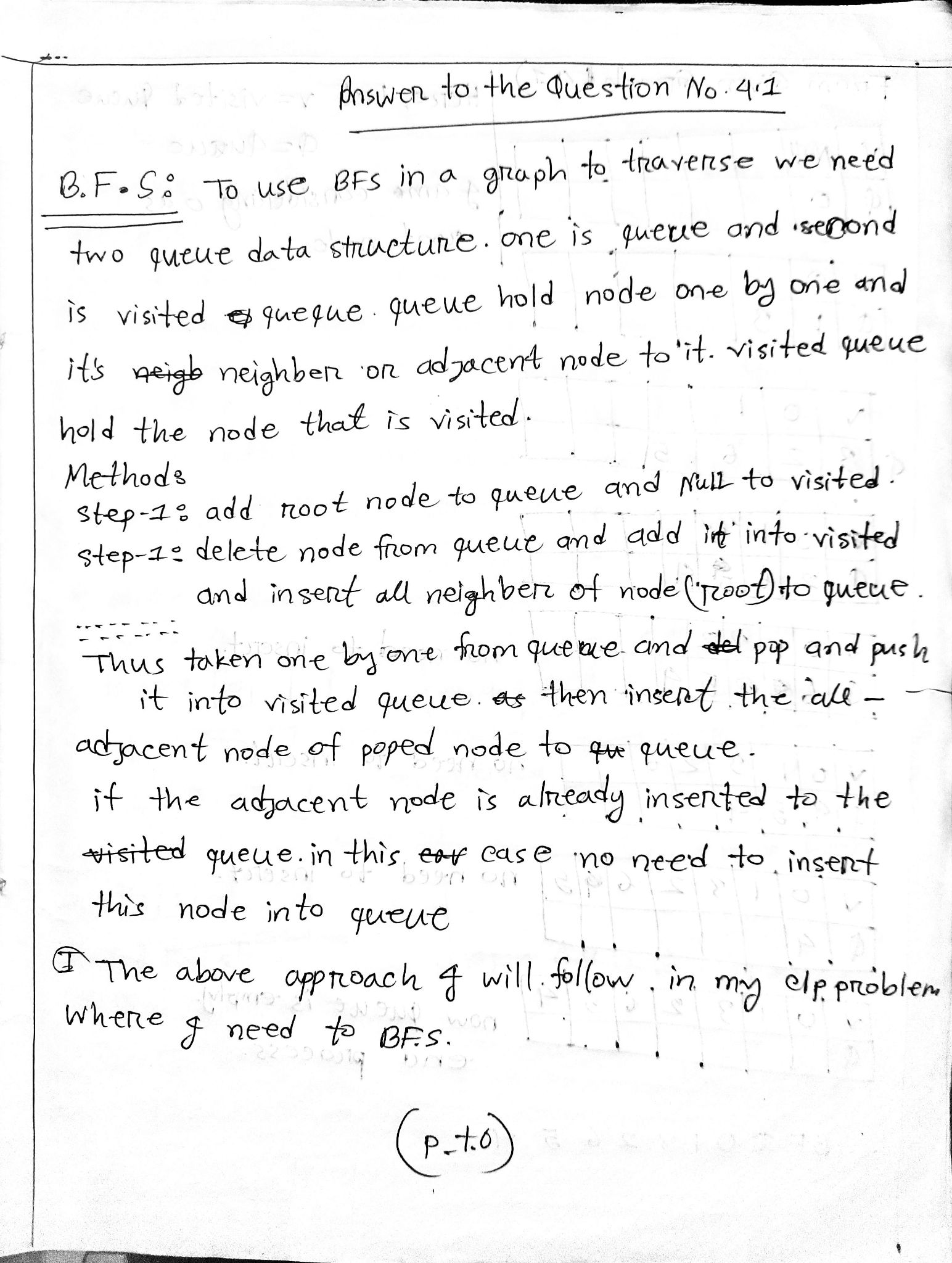
The Time complexity of DFS is also O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.

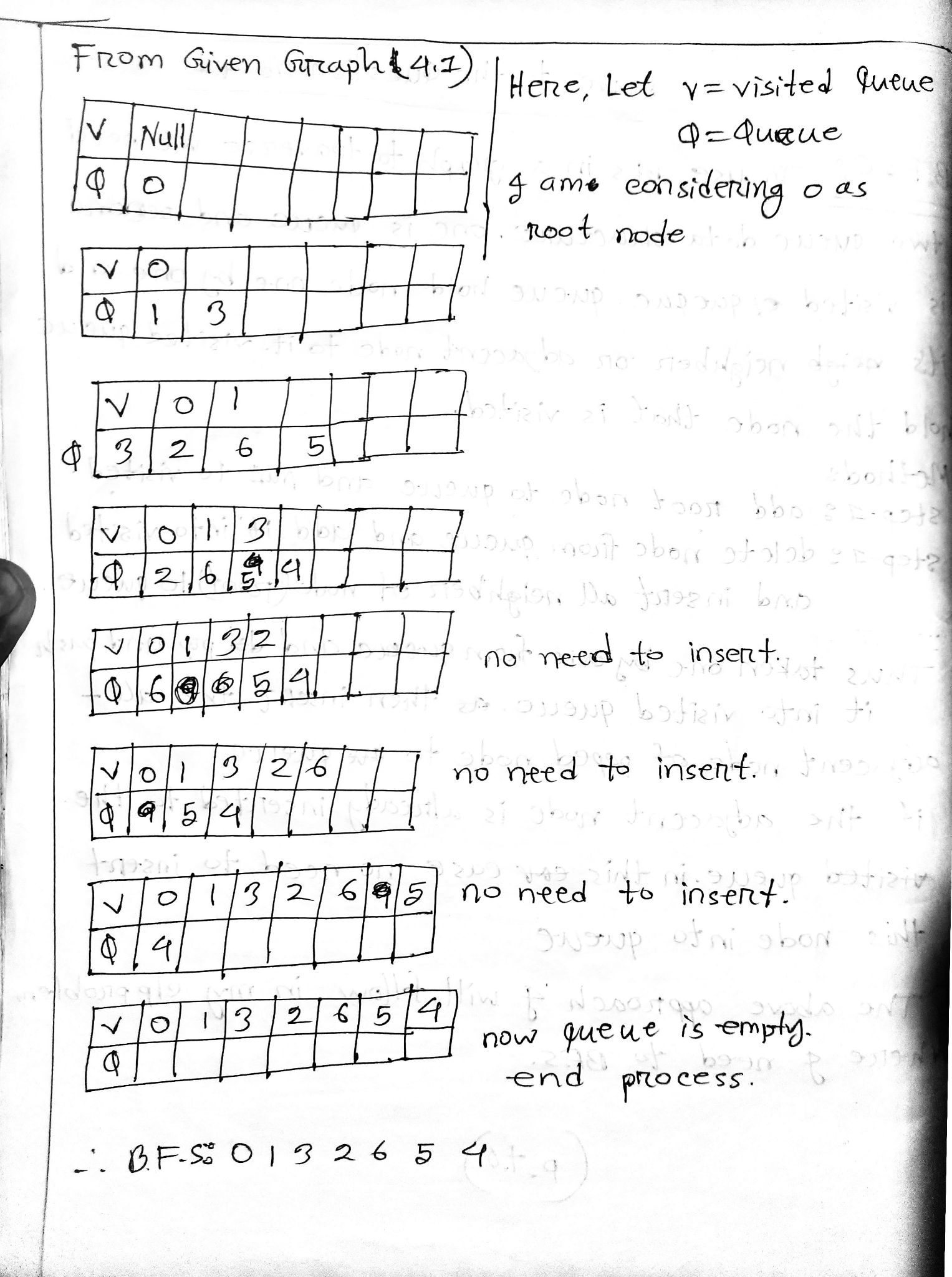
**Space complexity BFS:**

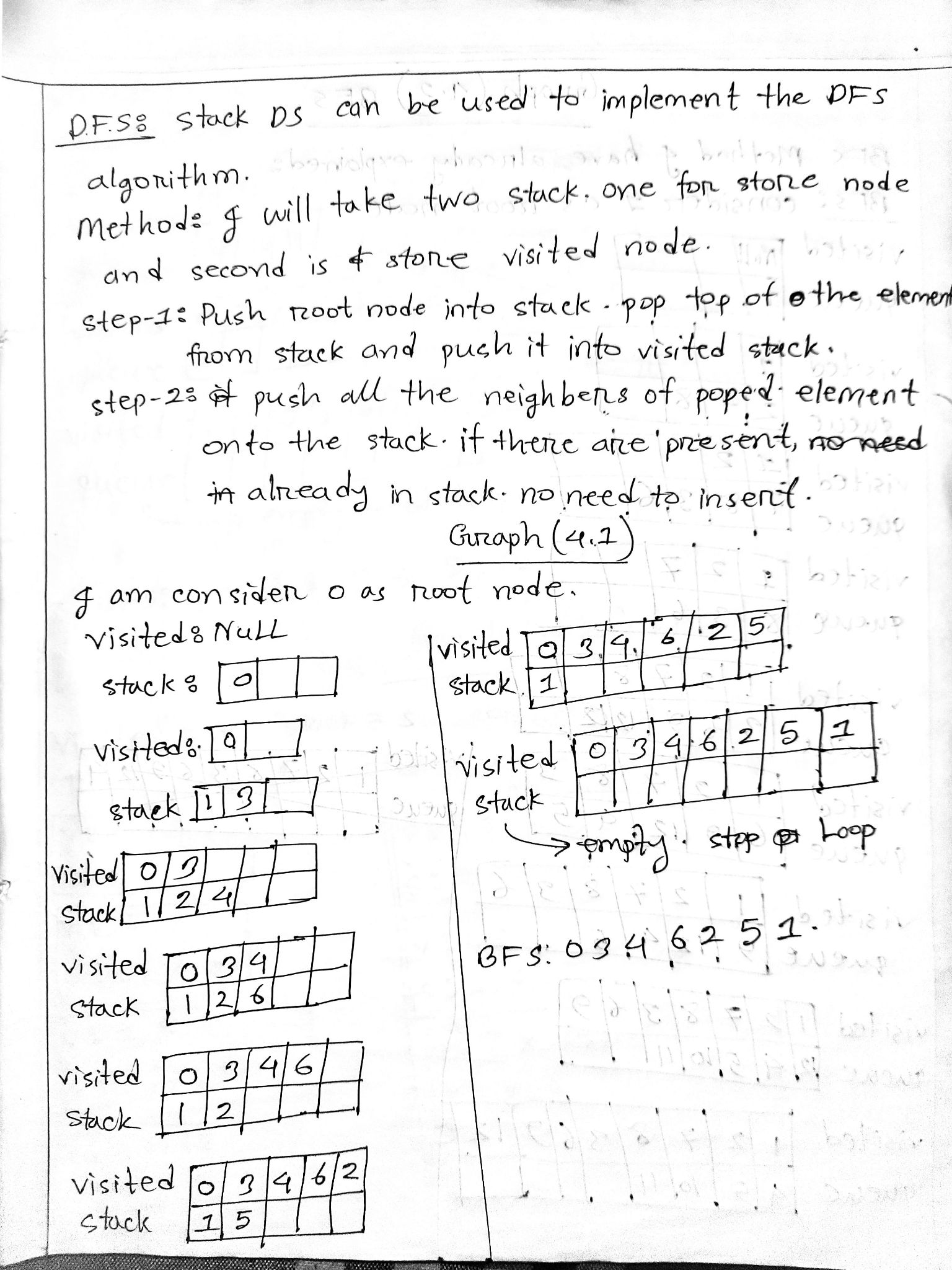
In BFS, the space complexity is more critical as compared to time complexity.

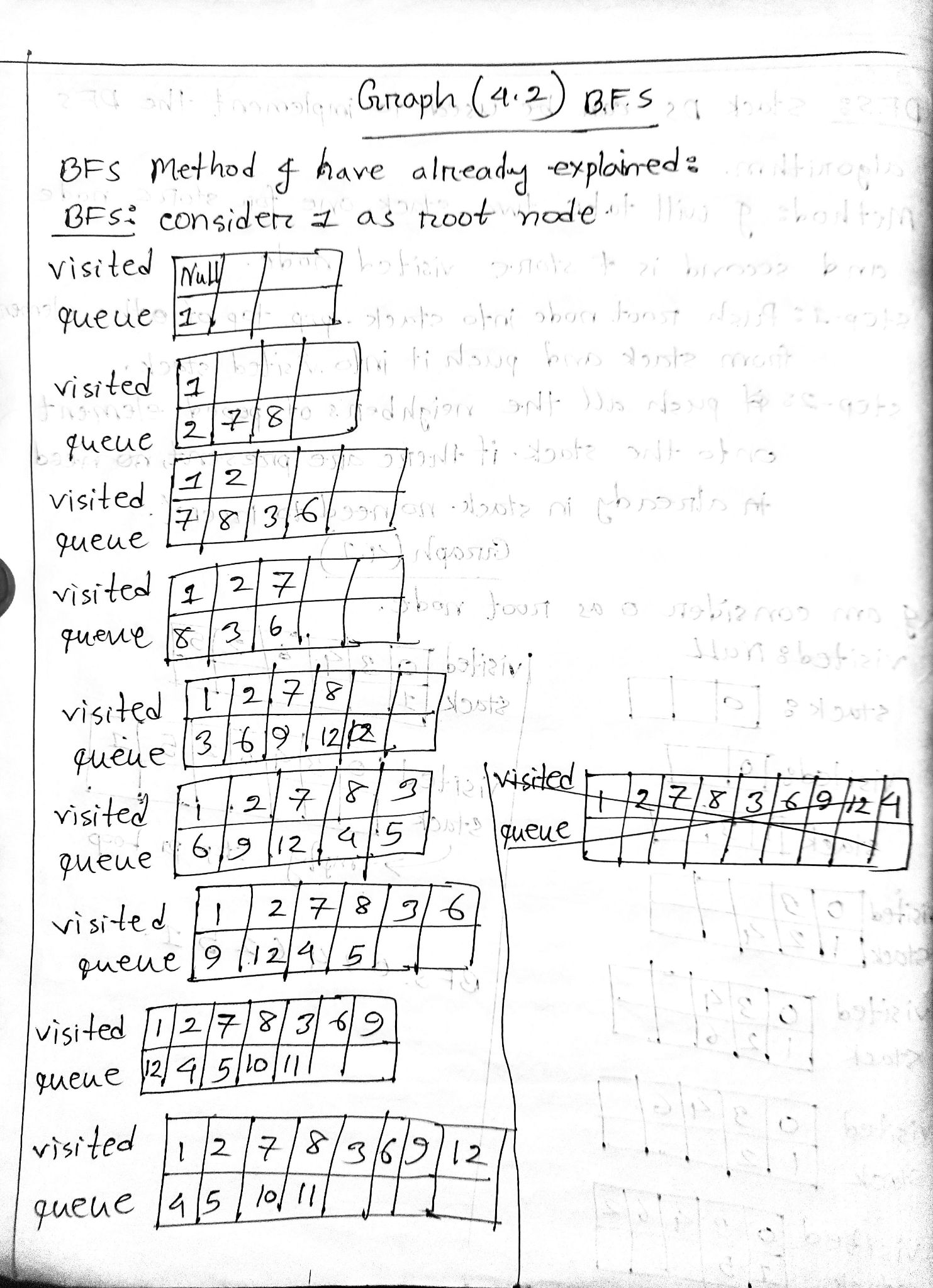
**Space Complexity DFS :**

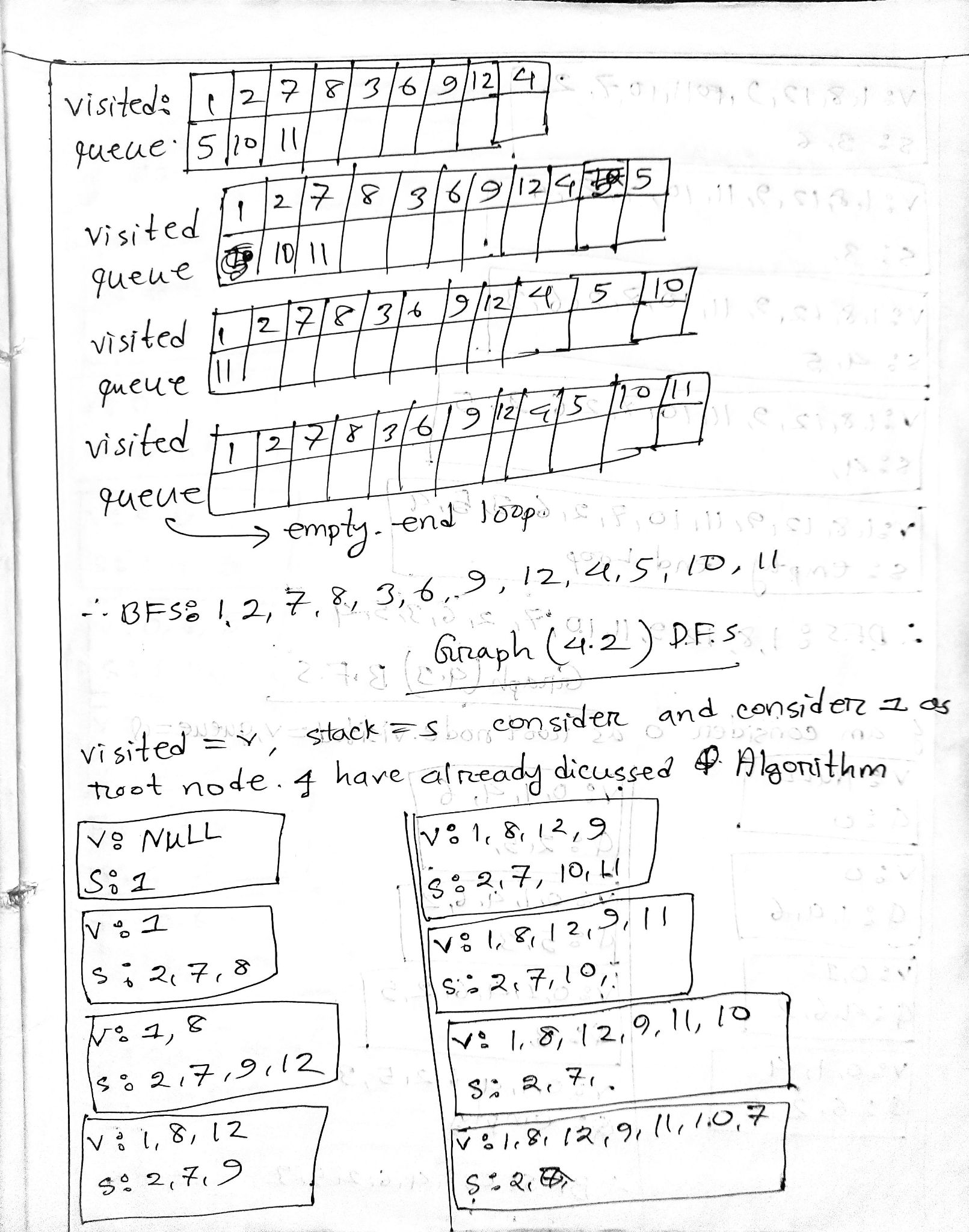
DFS has lesser space complexity because at a time it needs to store only a single path from the root to the leaf node.

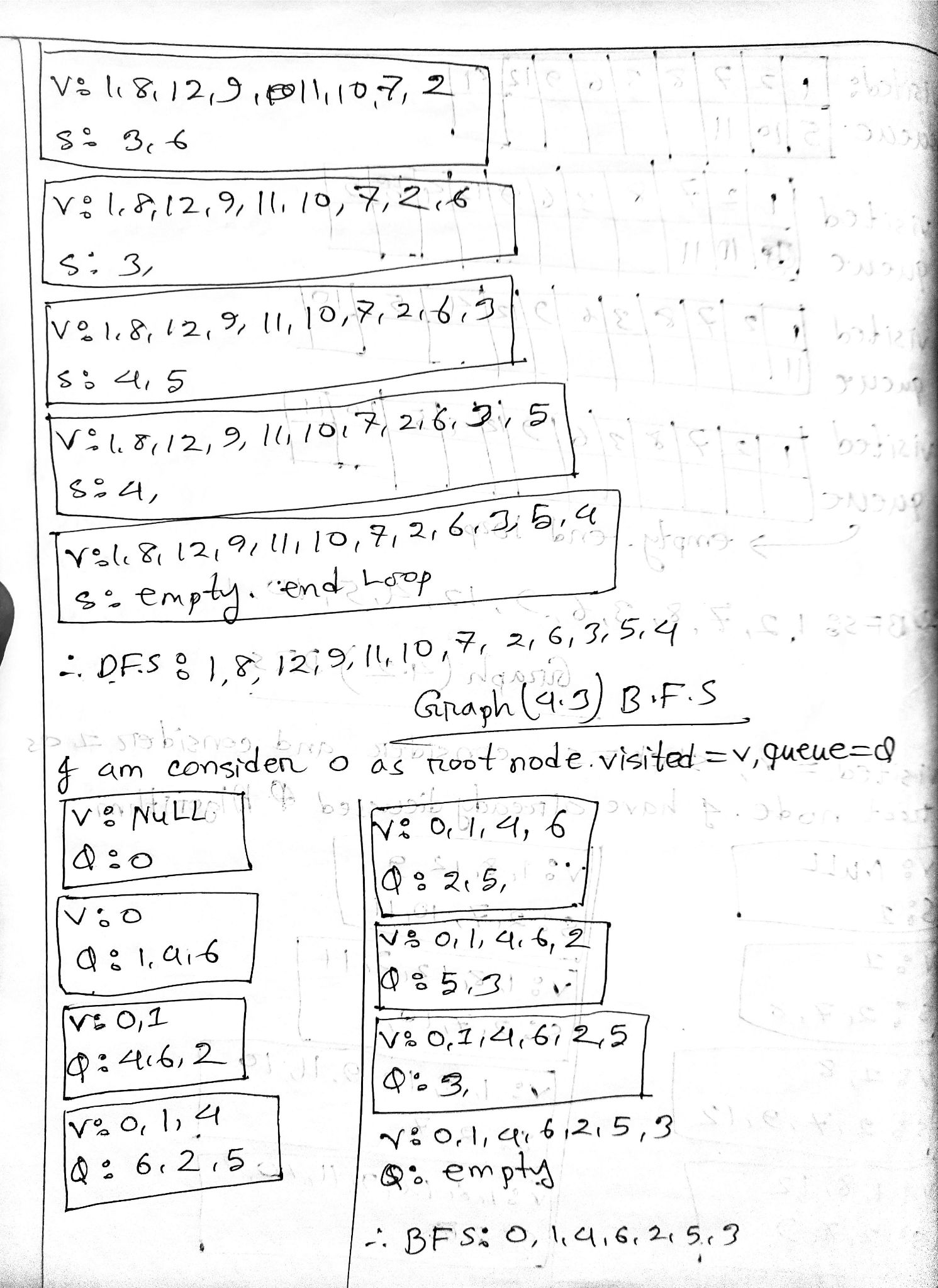
**Answer to the Question No: 4.1 to 4.4**

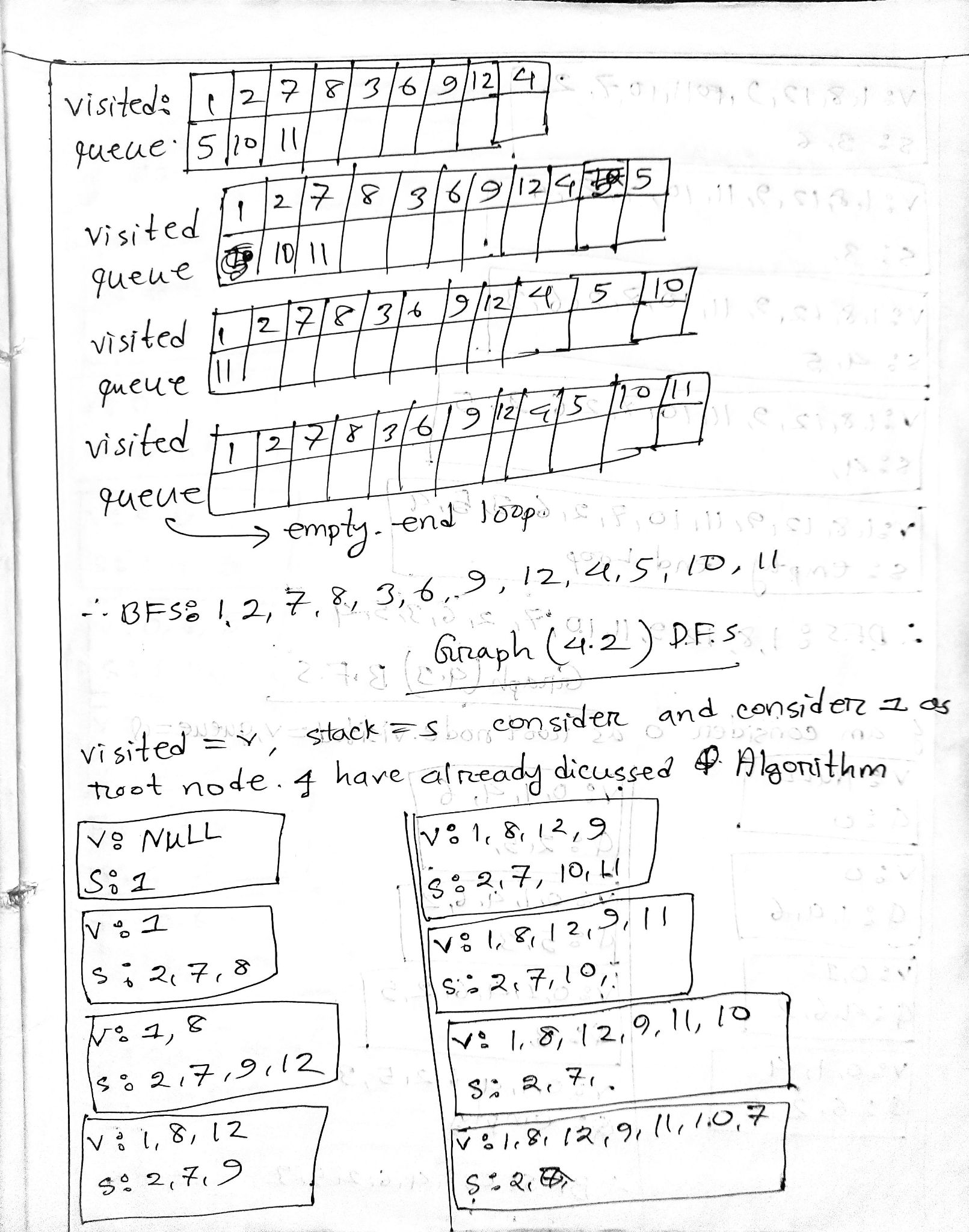
****

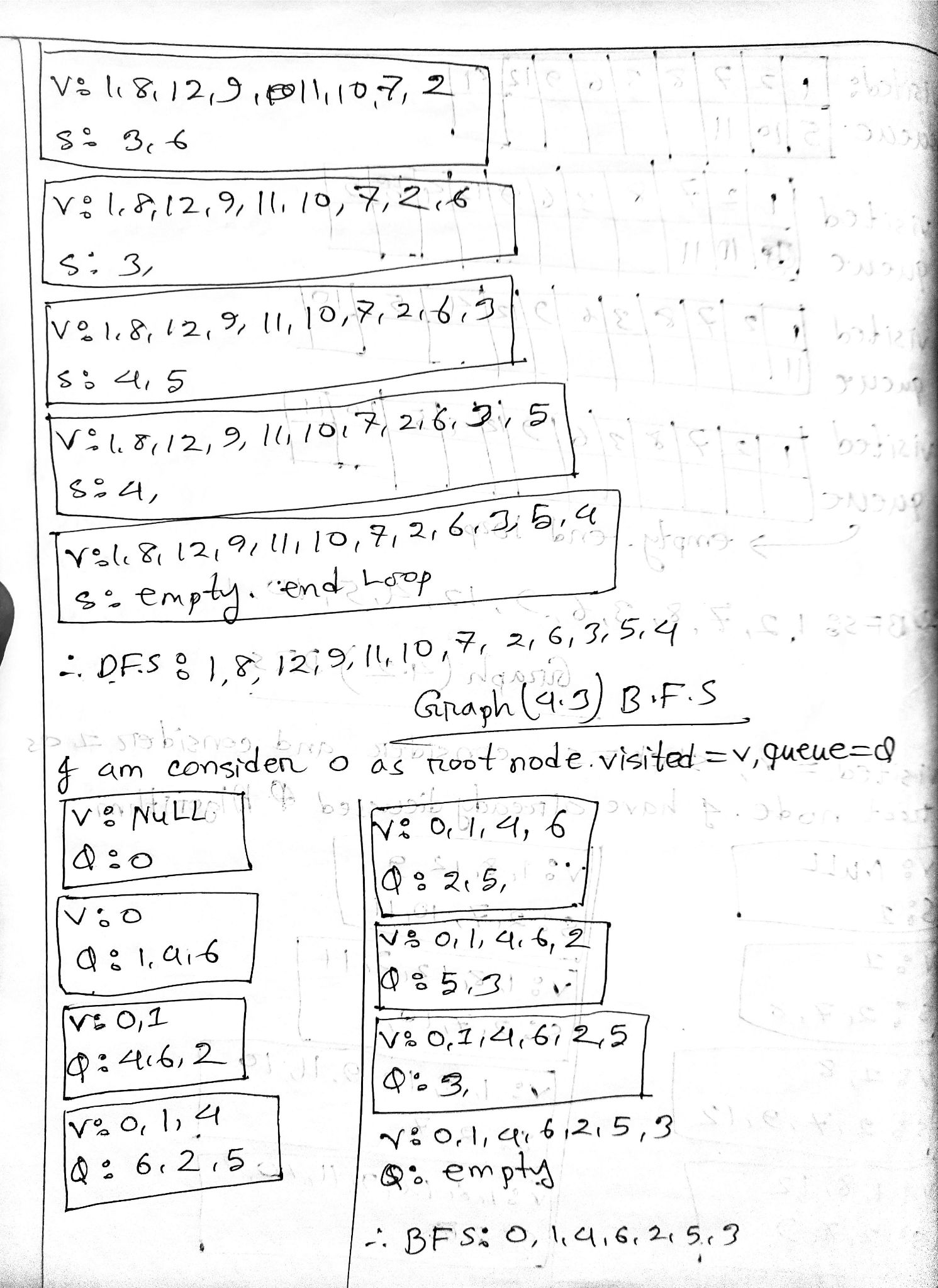
****

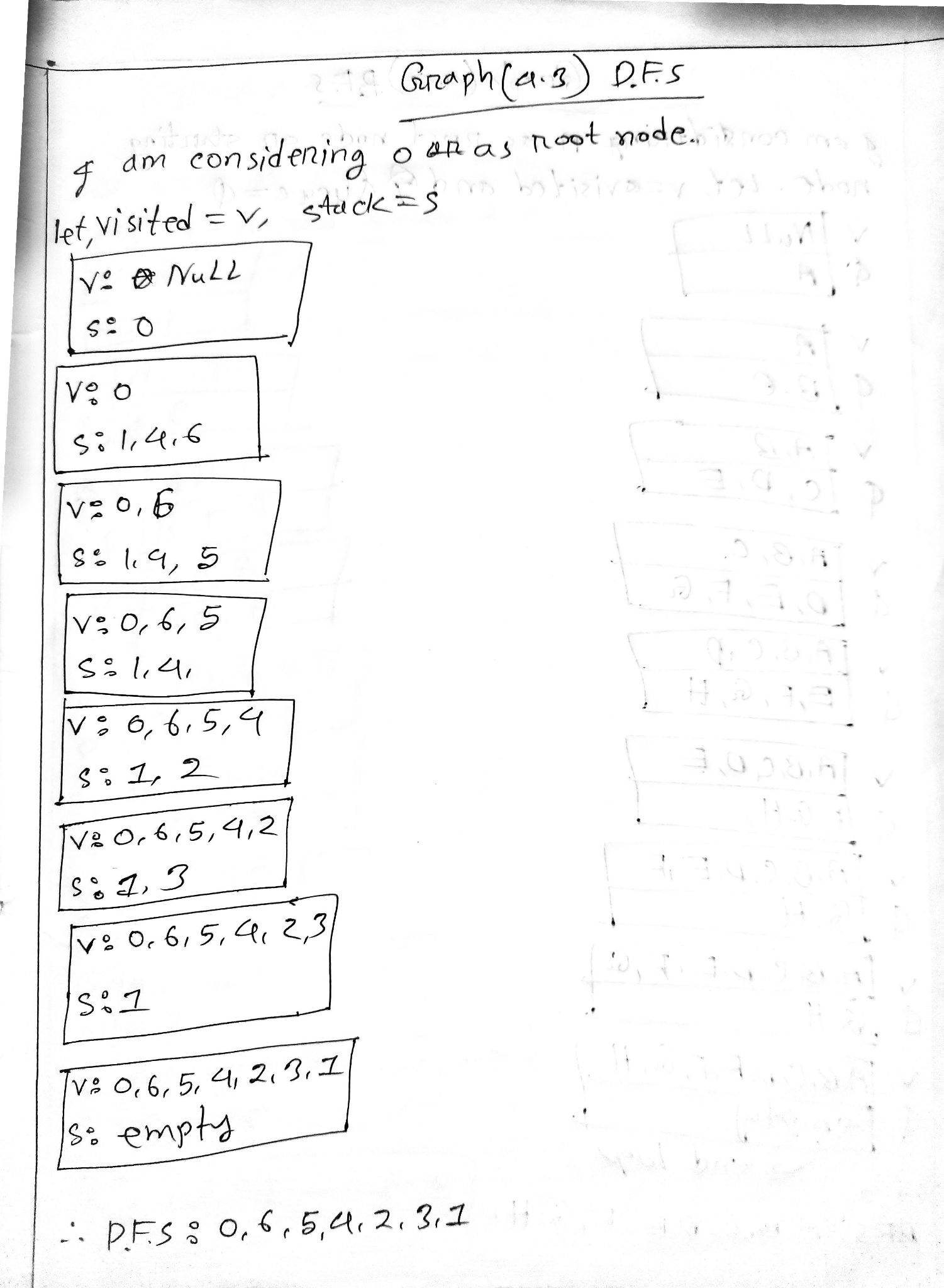
****

****

****

****

****

****

**All bfs and dfs cpp file given in above drive link.**