Start Date   :...................   Date of Completion:......................

Group A

Assignment No: 1(B)

Title of the Assignment: Design and implement Parallel Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for DFS

Objective of the Assignment: Students should be able to perform Parallel Depth First Search based on existing algorithms using OpenMP

Prerequisite:

1.   Basic of programming language

2.   Concept of DFS

3.   Concept of Parallelism

Contents for Theory:

1.   **What is DFS?**

2.   **Example of DFS**

3.   **Concept of OpenMP**

4.   **How Parallel DFS Work**

What is DFS?

DFS stands for Depth-First Search. It is a popular graph traversal algorithm that explores as far as possible along each branch before backtracking. This algorithm can be used to find the shortest path between two vertices or to traverse a graph in a systematic way. The algorithm starts at the root node and explores as far as possible along each branch before backtracking. The backtracking is done to explore the next branch that has not been explored yet.

DFS can be implemented using either a recursive or an iterative approach. The recursive approach is simpler to implement but can lead to a stack overflow error for very large graphs. The iterative approach uses a stack to keep track of nodes to be explored and is preferred for larger graphs.

DFS can also be used to detect cycles in a graph. If a cycle exists in a graph, the DFS algorithm will eventually reach a node that has already been visited, indicating that a cycle exists.

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
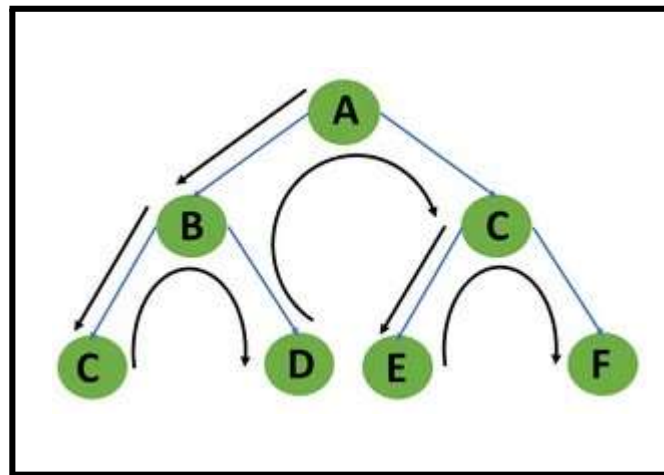
Example of DFS:

To implement DFS traversal, you need to take the following stages.

Step 1: Create a stack with the total number of vertices in the graph as the size.

Step 2: Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to

the stack.

Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.

Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.

Step 5 - If there are no new vertices to visit, go back and pop one from the stack using backtracking.

Step 6 - Continue using steps 3, 4, and 5 until the stack is empty.
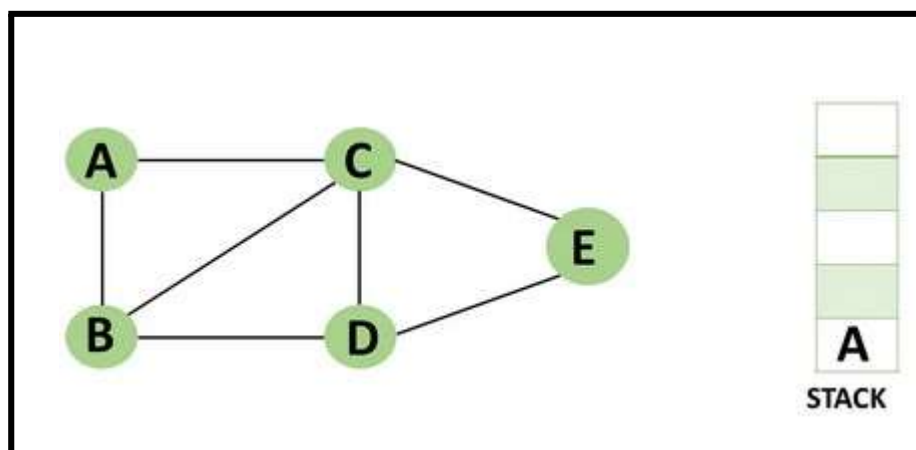
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

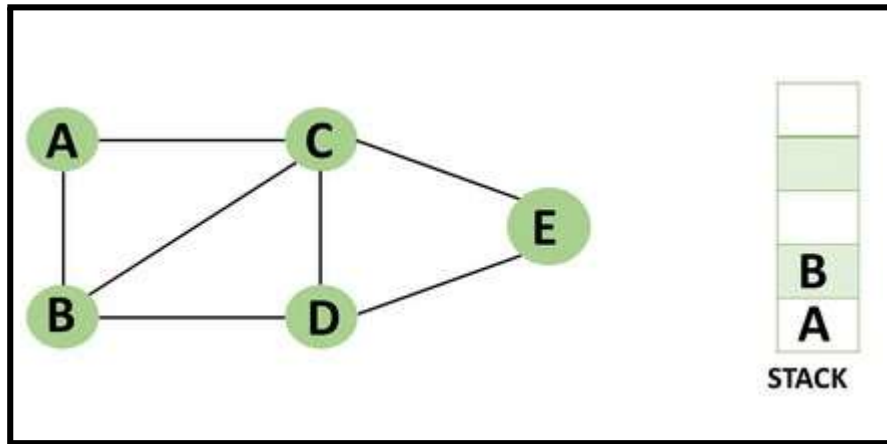Consider the following graph as an example of how to use the dfs algorithm.
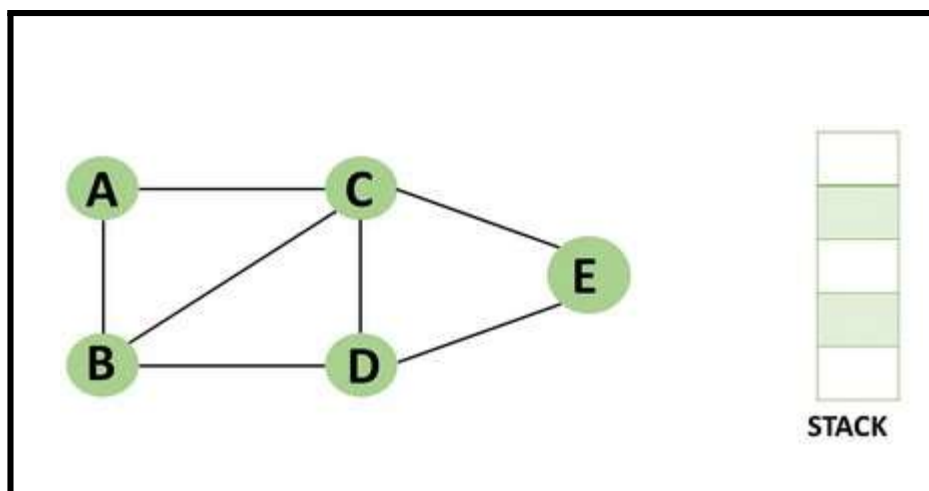
Step 1: Mark vertex A as a visited source node by selecting it as a source node.

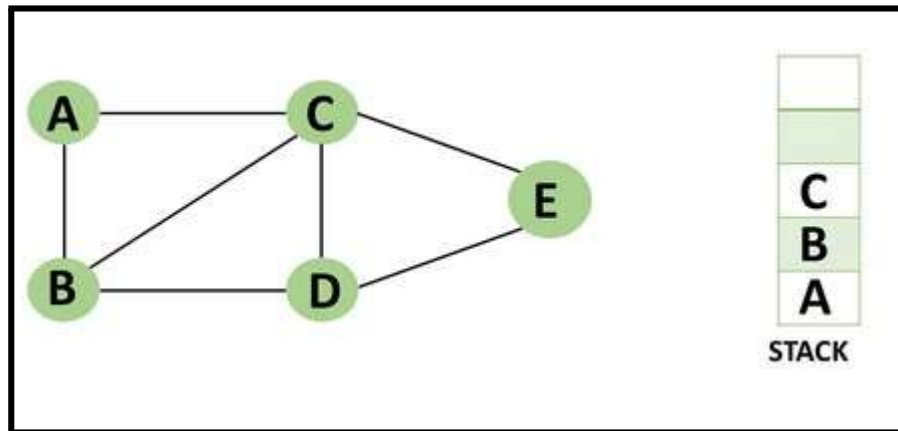● You should push vertex A to the top of the stack.

Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

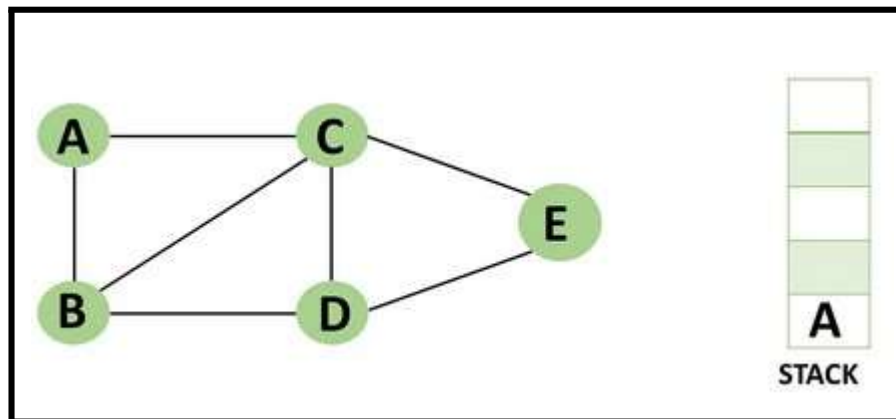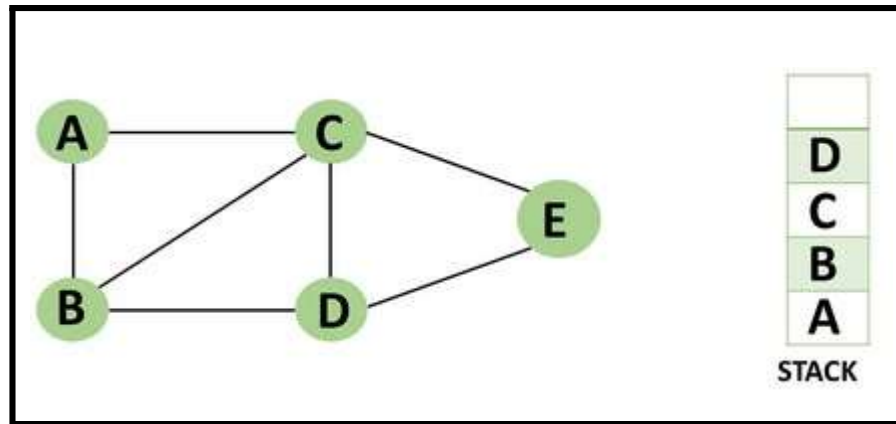You should push vertex B to the top of the stack

Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.
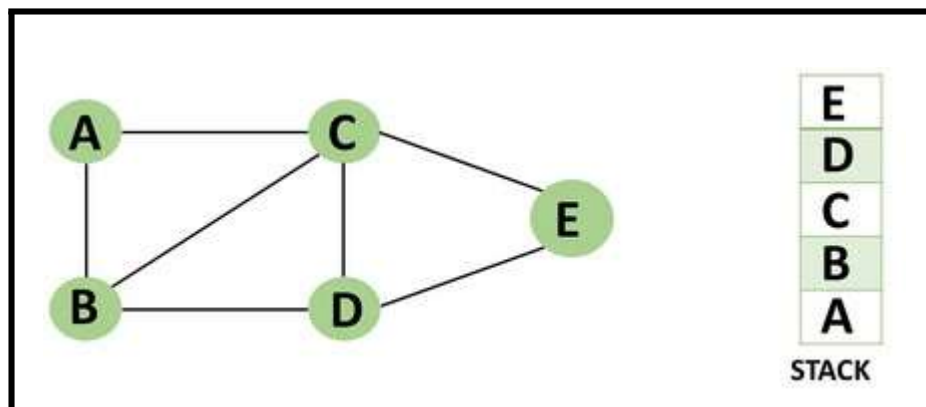
- Vertex C is pushed to the top of the stack.

Step 4: You can visit any nearby unvisited vertices of vertex C, you need to select vertex D and designate it as a visited vertex.

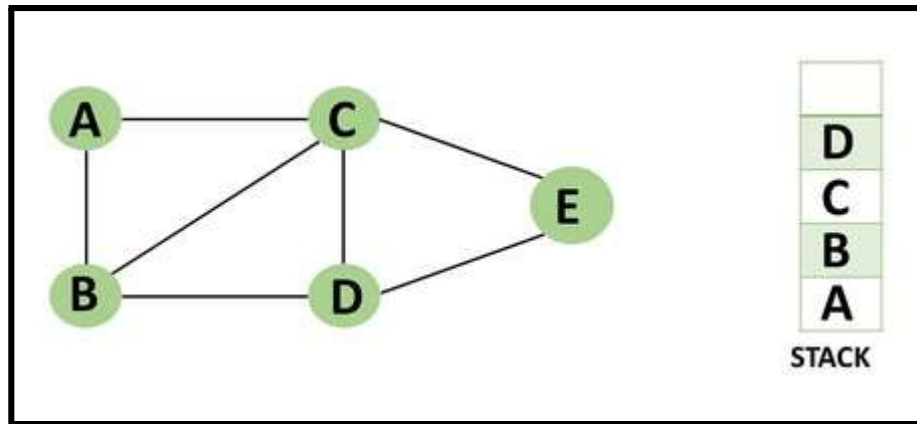- Vertex D is pushed to the top of the stack.

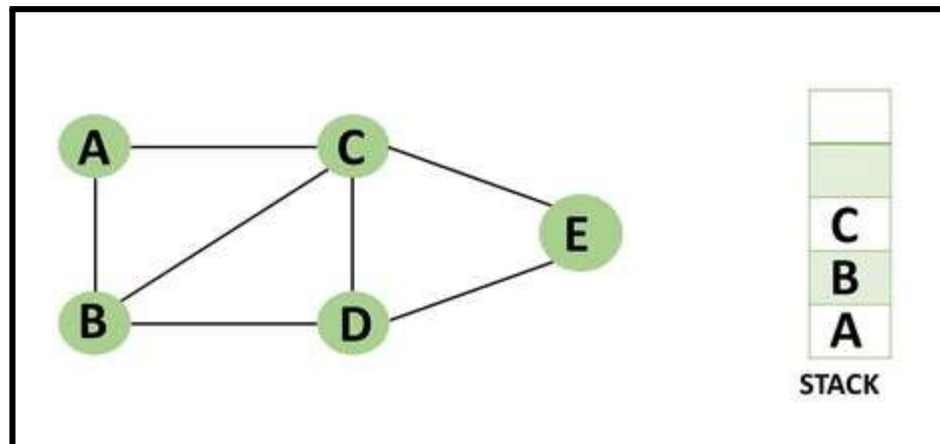Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

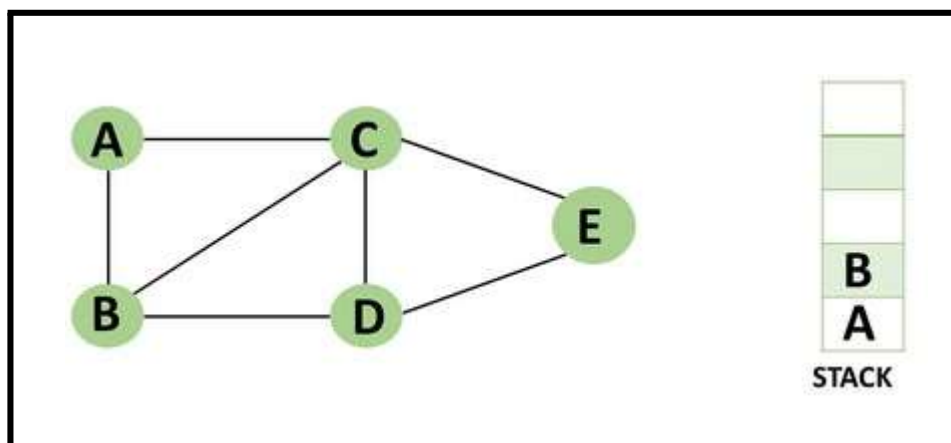● Vertex E should be pushed to the top of the stack.

Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.
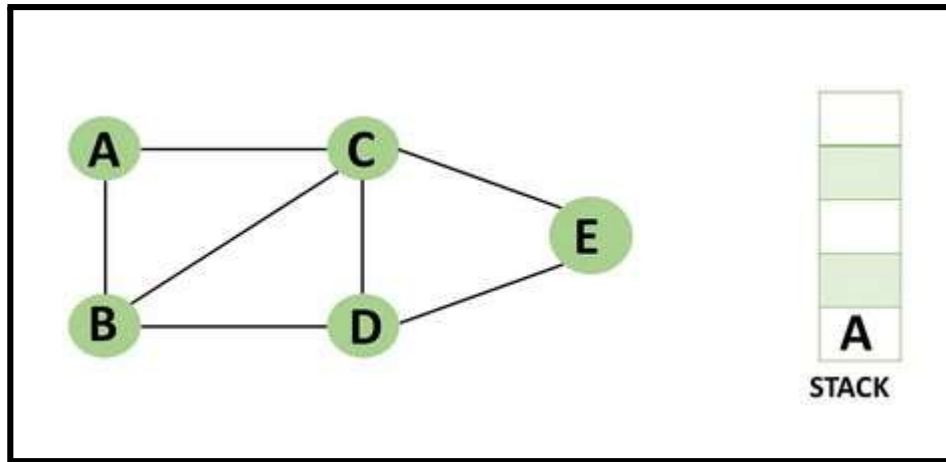
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.
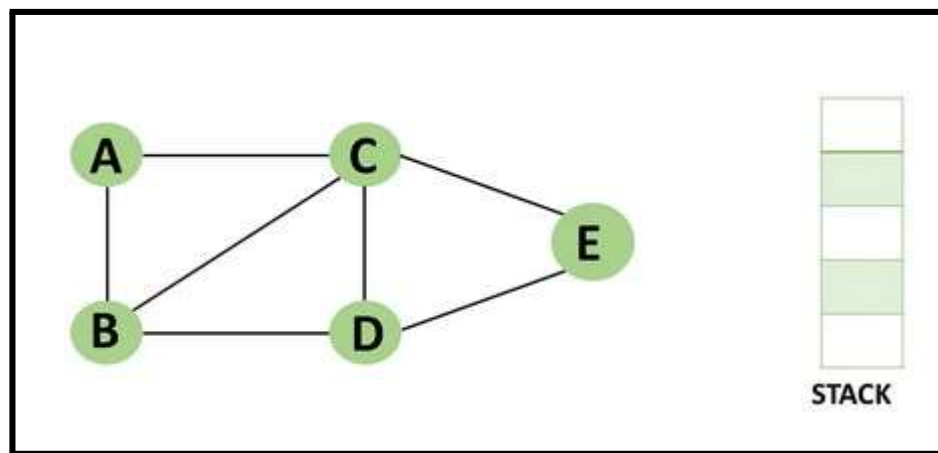

Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.

Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.

Step 10: All of the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.

Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.

- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.

- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to

execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

How Parallel DFS Work

- Parallel Depth-First Search (DFS) is an algorithm that explores the depth of a graph structure to search for nodes. In contrast to a serial DFS algorithm that explores nodes in a sequential manner, parallel DFS algorithms explore nodes in a parallel manner, providing a significant speedup in large graphs.

- Parallel DFS works by dividing the graph into smaller subgraphs that are explored simultaneously. Each processor or thread is assigned a subgraph to explore, and they work independently to explore the subgraph using the standard DFS algorithm. During the exploration process, the nodes are marked as visited to avoid revisiting them.

- To explore the subgraph, the processors maintain a stack data structure that stores the nodes in the order of exploration. The top node is picked and explored, and its adjacent nodes are pushed onto the stack for further exploration. The stack is updated concurrently by the processors as they explore their subgraphs.

- Parallel DFS can be implemented using several parallel programming models such as OpenMP, MPI, and CUDA. In OpenMP, the #pragma omp parallel for directive is used to distribute the work among multiple threads. By using this directive, each thread operates on a different part of the graph, which increases the performance of the DFS algorithm.

Conclusion-   In this way we can achieve parallelism while implementing DFS

Assignment Question

1. **What if DFS?**
2. **Write a parallel Depth First Search (DFS) algorithm using OpenMP**
3. **What is the advantage of using parallel programming in DFS?**
4. **How can you parallelize a DFS algorithm using OpenMP?**
5. **What is a race condition in parallel programming, and how can it be avoided in OpenMP?**

Reference link
- https://www.programiz.com/dsa/graph-dfs
- https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm