

SPPU-BE-COMP-CONTENT - KSKA Git

Q1> What is BFS and DFS?

ANS.

(1) BFS - Breadth First Search :-

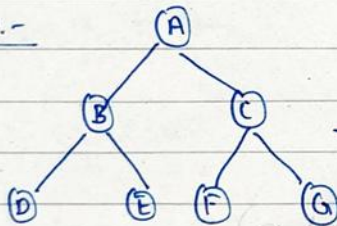
- BFS is a graph traversal algorithm that explores node level by level.
- It starts from a source node and visits all its neighbors first, then neighbors of neighbors.
- Uses a Queue Datastructure.
- Follows First in First Out (FIFO)

Key Points:-

- Finds shortest path (Unweighted graph)
- Visits nodes in layers.
- Uses Queue.

Example Flow:- Start \rightarrow Visit Neighbors \rightarrow The Next Level \rightarrow Continue.

For Eg:-



BFS Traversal :-

ABCEDEFG

(2) DFS - Depth First Search

- DFS Explores as far as possible along one branch before backtracking.
- Uses stack datastructures.
- Follows Last in First Out (LIFO)

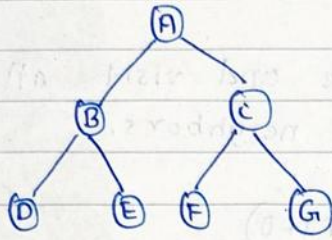
Key points:-

- Goes Depth First, then backtracks.
- Uses stack or recursion.
- Good For Cycle detection, path finding.

SPPU-BE-COMP-CONTENT - KSKA Git

For Example:-

Start → Go Deep → Reach End → Backtrack → Continue.



DFS Traversal:-

ABDECFG

Q2.)

ANS.

Write a parallel BFS and DFS Algorithm using OpenMP

(1) Parallel BFS

```
#include <stdio.h>
```

```
#include <omp.h>
```

```
#include <stdbool.h>
```

```
#define MAX 100
```

```
int graph[MAX][MAX];
```

```
bool visited[MAX];
```

```
void parallelBFS (int start, int n) {
```

```
    int queue[MAX], front = 0, rear = 0;
```

```
    visited[start] = true;
```

```
    queue[rear++] = start;
```

```
    while (front < rear) {
```

```
        int node = queue[front++];
```

```
        printf("visited %d\n", node);
```

```
    }  
    #pragma omp parallel for
```


SPPU-BE-COMP-CONTENT - KSKA Git

```
for (int i=0; i<n; i++) {  
    if (graph[node][i] && !visited[i]) {  
        #pragma omp critical  
        {  
            if (!visited[i]) {  
                visited[i] = true;  
                queue[rear++] = i;  
            }  
        }  
    }  
}
```

(2) Parallel DFS

```
#include <stdio.h>  
#include <omp.h>  
#include <stdbool.h>
```

```
#define MAX 100
```

```
int graph[MAX][MAX];  
bool visited[MAX];
```

```
void parallelDFS(int node, int n) {  
    printf("Visited %d\n", node);  
    visited[node] = true;
```

```
#pragma omp parallel for
```

```
for (int i=0; i<n; i++) {
```

```
    if (graph[node][i] && !visited[i]) {
```

```
        #pragma omp task
```

```
        parallelDFS(i, n);
```

```
    } } }
```

SPPU-BE-COMP-CONTENT - KSKA Git

- DFS Parallelization is harder due to recursion and dependencies.
- Proper Synchronization (like critical, task, atomic) is required.

Q3. > What is the Advantage of using parallel programming in DFS and BFS?

ANS. The Advantages of using parallel programming in DFS and BFS are:-

1. Faster Execution

- Multiple nodes are processed simultaneously.
- Reduces traversal time for large graphs.

2. Efficient For Large Graphs

Useful in:-

- Social Networks
- Web Crawling
- AI Search Problem

3. Better CPU Utilization.

- Uses Multi-core processors effectively.

4. Scalable

- Performance improves with more cores.

	Feature	Parallel BFS	Parallel DFS.
1.	Easier to parallelize	Yes	Difficult.
2.	Structure	Level-based	Recursive
3.	Performance	High	Moderate
4.	Synchronization	Easier	Complex.