Khushi panjwani_D10B42

Experiment 11

THEORY-**Breadth-First Search (BFS)**

- **Method**: Explores nodes layer by layer using a queue.
- **Complexity**:
  - Time: O(V+E)O(V + E)O(V+E) (vertices + edges)
  - Space: O(V)O(V)O(V) (for the queue and visited array)
- **Uses**: Finding the shortest path in unweighted graphs, checking connectivity, and level order traversal.

Depth-First Search (DFS)

- **Method**: Explores as deep as possible along a branch using a stack (or recursion).
- **Complexity**:
  - Time: O(V+E)O(V + E)O(V+E)
  - Space: O(V)O(V)O(V) (for the recursion stack)
- **Uses**: Pathfinding, topological sorting, cycle detection, and backtracking.

Key Differences

- **Traversal**: BFS is layer-wise; DFS dives deep into branches.
- **Shortest Path**: BFS guarantees the shortest path in unweighted graphs; DFS does not.
- **Data Structure**: BFS uses a queue; DFS uses a stack.

INPUT-

BFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
void bfs(int graph[MAX_VERTICES][MAX_VERTICES], int start, int n) {
   int visited[MAX_VERTICES] = {0};
   int queue[MAX_VERTICES], front = 0, rear = -1;
   visited[start] = 1;
   queue[++rear] = start;
   printf("BFS Traversal: ");
   while (front <= rear) {
      int vertex = queue[front++];
      printf("%d ", vertex);
      for (int i = 0; i < n; i++) {
         if (graph[vertex][i] == 1 && !visited[i]) {
            visited[i] = 1;
            queue[++rear] = i;
         }}}
   printf("\n");
```

```
}
int main() {
    int graph[MAX_VERTICES][MAX_VERTICES] = {
        {0, 1, 0, 0},
        {0, 0, 1, 0},
        {0, 0, 0, 1},
        {0, 0, 0, 0}
    };
    int n = 4;  // Number of vertices
    int start_vertex = 0;
    bfs(graph, start_vertex, n);
    return 0;
}
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int vertex, int visited[], int n) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < n; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(graph, i, visited, n);
        }}}
```

DFS IMPLENTATION

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int vertex, int visited[], int n) {
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < n; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(graph, i, visited, n);
        }
    }}
int main() {
    int graph[MAX_VERTICES][MAX_VERTICES] = {
        {0, 1, 0, 0},
        {0, 0, 1, 0},
        {0, 0, 0, 1},
        {0, 0, 0, 0}
    };
```

Khushi panjwani_D10B42

```
    int n = 4;  // Number of vertices
    int visited[MAX_VERTICES] = {0};
    int start_vertex = 0;
    printf("DFS Traversal: ");
    dfs(graph, start_vertex, visited, n);
    printf("\n");
    return 0;
}
```

OUTPUT

```
/tmp/l0j1ZadF8I.o
BFS Traversal: 0 1 2 3
```

```
/tmp/oODgE900gd.o
DFS Traversal: 0 1 2 3
```

CONCLUSION- BFS and DFS are key graph traversal algorithms.

- BFS: Explores nodes layer by layer, ideal for finding the shortest path in unweighted graphs.
- DFS: Explores as deeply as possible before backtracking, suited for deep exploration and problems like topological sorting