WEEK-13

Date:22-10-2025

List of programs:

1. Write a C progam to implement Breadth First search Traversal.
2. Write a C progam to implement Depth First search Traversal.

1. Aim: To Write a C progam to implement Breadth First search Traversal.

Program:

```
#include <stdio.h>
int bfs[20],rear = -1,front = -1,vt[20];
void store(int n)
{
   bfs[++rear] = n;
}
int delet()
{
   return bfs[++front];
}
void bfsearch(int a[][10], int n, int id)
{
   int i;
   for (i=0;i<n;i++)
      vt[i] = 0;
   id=id-1;
   store(id);
   vt[id] = 1;
   printf("\nBFS visited vertices: ");
   while (front!=rear)
   {
```

```
        id = delet();
        printf("%d ", id + 1);
        for (i = 0; i < n; i++)
        {
            if (a[id][i] == 1 && vt[i] == 0)
            {
                store(i);
                vt[i] = 1;
            }
        }
    }
}
int main()
{
    int a[10][10], n, i, j, id;
    printf("Enter number of vertices on graph:");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);
    printf("Enter starting vertex:");
    scanf("%d", &id);
    bfsearch(a, n, id);
    return 0;
}
```

Output:

```
Enter number of vertices on graph:5
Enter adjacency matrix:
0 1 1 0 0
1 0 0 1 1
1 0 0 1 0
0 1 1 0 1
0 1 0 1 0
Enter starting vertex:1

BFS visited vertices: 1 2 3 4 5

=== Code Execution Successful ===
```

2. Aim: To Write a C progam to implement Depth First Search Graph Traversal.

Program:

```
#include <stdio.h>
int vt[10], dfs[10], top = -1;
void push(int n)
 {
   dfs[++top] = n;
}
void pop()
{
   top--;
}
void dfs_Traversal(int a[][10], int n, int start)
{
   int i, found;
   for (i = 0; i < n; i++)
      vt[i] = 0;
   start = start - 1;
   push(start);
   vt[start] = 1;
   printf("DFS visited nodes are: %d", start + 1);
   while (top != -1)
{
      start = dfs[top];
      found = 0;
      for (i = 0; i < n; i++) {
         if (a[start][i] == 1 && vt[i] == 0) {
            push(i);
```

```
            printf(" -> %d", i + 1);
            vt[i] = 1;
            found = 1;
            break;
          }
        }
        if (found == 0)
          pop();
      }
    }
    int main() {
      int a[10][10], n, i, j, id;
      printf("Enter number of vertices in graph: ");
      scanf("%d", &n);

      printf("Enter adjacency matrix:\n");
      for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
          scanf("%d", &a[i][j]);
      printf("Enter starting vertex: ");
      scanf("%d", &id);
      dfs_Traversal(a, n, id);
      return 0;
    }
```

Output:

```
Enter number of vertices in graph: 4
Enter adjacency matrix:
1 0 0 1
0 1 1 0
0 1 1 0
1 0 0 1
Enter starting vertex: 1
DFS visited nodes are: 1 -> 4

=== Code Execution Successful ===
```

Inferences:

- **BFS** explores nodes level by level, visiting all neighbors before moving deeper.
- It guarantees the shortest path in unweighted graphs and helps identify levels or layers.
- **DFS** explores as far as possible along each branch before backtracking.
- It's useful for pathfinding, cycle detection, and topological sorting.
- BFS uses more memory, while DFS is more memory-efficient but may not find the shortest path.