

28/02/24

Week 9

a) Write a program to traverse a graph using BFS

```
#include <stdio.h>
#define MAX_VERTICES 100
int n, adj[100][100], visited[100], queue[100], front = 0, rear = 0;
void bfs (int v) {
    visited[v] = 1;
    queue[rear++] = v;
    while (front < rear) {
        int current = queue[front++];
        for (int i = 0; i < n; i++) {
            if (adj[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
}
```

in main () {

int v;

printf("Enter the number of vertices");

scanf("%d", &n);

for (i = 0; i < n; i++)

visited[i] = 0;

}

graph data in vertex form

scanf ("Enter the starting vertex: ");

for (i = 0; i < n; i++)

for (j = 0; j < n; j++)

scanf ("%d", &adj[i][j]);

```

        cout << "Enter the starting vertex: ";
        cin >> s;
        cout << endl;
        cout << "Scanning..." << endl;
        dfs(s);
        for (i = 0; i < n; i++) {
            if (!visited[i]) {
                cout << "DFS is not possible. Not all
                nodes are reachable." << endl;
                return 0;
            }
        }
        cout << endl;
        cout << "DFS Traversal: ";
        for (i = 0; i < n; i++) {
            if (visited[i])
                cout << i << " ";
        }
        cout << endl;
    }

    void dfs(int v) {
        visited[v] = true;
        cout << v << " ";
        for (int i = 0; i < n; i++) {
            if (adj[v][i] && !visited[i])
                dfs(i);
        }
    }
}

```

**Output:** Enter the number of vertices: 7

Enter the adjacency matrix:

```

0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 0 0 6 1 0 0

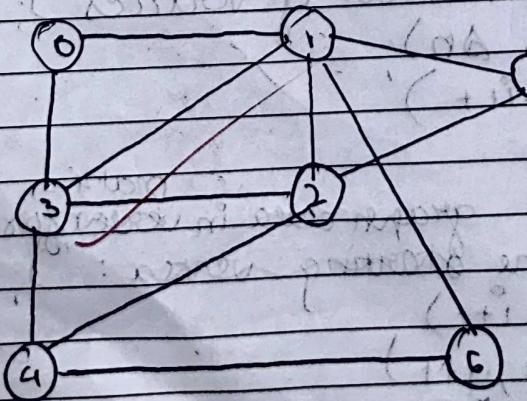
```

Enter the starting vertex: 2

```

2 1 3 4 5 0 6

```



(a) Write a program  
program as conn

```

#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 7

```

```

void dfs(int v) {
    visited[v] = true;
    cout << v << " ";
    for (int i = 0; i < MAX_VERTICES; i++) {
        if (adj[v][i] && !visited[i])
            dfs(i);
    }
}

```

```

int main() {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int v, e, i, j;
    bool visited[MAX_VERTICES];
    for (i = 0; i < MAX_VERTICES; i++)
        visited[i] = false;
    cout << "Enter the number of vertices: ";
    cin >> v;
    cout << endl;
    cout << "Enter the adjacency matrix: ";
    for (i = 0; i < v; i++) {
        for (j = 0; j < v; j++) {
            cin >> adj[i][j];
        }
    }
    cout << endl;
    cout << "Enter the starting vertex: ";
    cin >> e;
    cout << endl;
    cout << "DFS Traversal: ";
    dfs(e);
    cout << endl;
    return 0;
}

```

```

bool isConected(int v, int e) {
    int adj[7][7];
    int v, e, i, j;
    bool visited[7];
    for (i = 0; i < 7; i++)
        visited[i] = false;
    cout << "Enter the number of vertices: ";
    cin >> v;
    cout << endl;
    cout << "Enter the adjacency matrix: ";
    for (i = 0; i < v; i++) {
        for (j = 0; j < v; j++) {
            cin >> adj[i][j];
        }
    }
    cout << endl;
    cout << "Enter the starting vertex: ";
    cin >> e;
    cout << endl;
    cout << "DFS Traversal: ";
    dfs(e);
    cout << endl;
    return true;
}

```

```

bool isConected(int v, int e) {
    int adj[7][7];
    int v, e, i, j;
    bool visited[7];
    for (i = 0; i < 7; i++)
        visited[i] = false;
    cout << "Enter the number of vertices: ";
    cin >> v;
    cout << endl;
    cout << "Enter the adjacency matrix: ";
    for (i = 0; i < v; i++) {
        for (j = 0; j < v; j++) {
            cin >> adj[i][j];
        }
    }
    cout << endl;
    cout << "Enter the starting vertex: ";
    cin >> e;
    cout << endl;
    cout << "DFS Traversal: ";
    dfs(e);
    cout << endl;
    return true;
}

```

```

bool isConected(int v, int e) {
    int adj[7][7];
    int v, e, i, j;
    bool visited[7];
    for (i = 0; i < 7; i++)
        visited[i] = false;
    cout << "Enter the number of vertices: ";
    cin >> v;
    cout << endl;
    cout << "Enter the adjacency matrix: ";
    for (i = 0; i < v; i++) {
        for (j = 0; j < v; j++) {
            cin >> adj[i][j];
        }
    }
    cout << endl;
    cout << "Enter the starting vertex: ";
    cin >> e;
    cout << endl;
    cout << "DFS Traversal: ";
    dfs(e);
    cout << endl;
    return true;
}

```

return 1;

(f) write a program to check whether the given graph is connected or not using DFS method

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 10

void dfs (int graph [MAX\_VERTICES][MAX\_VERTICES],  
int num\_vertices, bool visited [MAX\_VERTICES],  
int vertex) {  
 visited[vertex] = true;

int i;

for (i=0; i<num\_vertices; ++i) {  
 if (graph[vertex][i] == 1 && !visited[i]) {  
 dfs(graph, num\_vertices, visited, i);  
 }  
 }

}

bool isconnected (int graph[MAX\_VERTICES][MAX\_VERTICES])

[MAX\_VERTICES], int num\_vertices) {

bool visited [MAX\_VERTICES] = {false};

dfs (graph, num\_vertices, visited, 0);

int i;

for (i=0; i<num\_vertices; ++i) {

if (!visited[i]) {  
 return false;  
 }  
 }

}

~~return true;~~

}

longest path

```

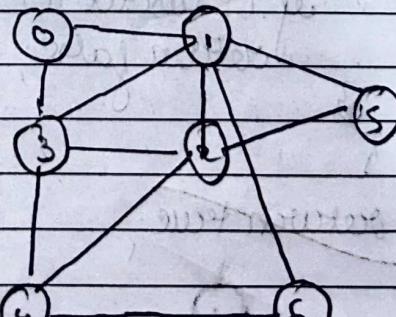
    cout << "Enter num vertices"; cin >> numVertices;
    cout << "Enter the number of vertices : ";
    cin >> numVertices;
    cout << "Enter the adjacency matrix : [n][n] ";
    int i, j;
    adjGraph = new int*[numVertices];
    for (int i = 0; i < numVertices; ++i) {
        adjGraph[i] = new int[numVertices];
        for (int j = 0; j < numVertices; ++j) {
            adjGraph[i][j] = 0;
        }
    }
    cout << "Enter the adjacency matrix : [n][n] ";
    for (int i = 0; i < numVertices; ++i) {
        for (int j = 0; j < numVertices; ++j) {
            cout << "Enter edge " << i << " " << j << ": ";
            cin >> adjGraph[i][j];
        }
    }
    cout << "The graph is ";
    if (isConnected(adjGraph, numVertices)) {
        cout << "connected." << endl;
    } else {
        cout << "not connected." << endl;
    }
}

```

```

returns 0.0 / XAMT returns true because it is a
y      3' (another min tree, Feasible XAMT)
- floydF = PARMAX XAMT? initially
outPut: Min. cost of min. weight is
Enter the number of vertices: 7
Enter the adjacency matrix
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
2 1 2 3 4 5 6

```



The graph is connected