

## exp\_9/a.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX_VERTICES 100
5
6  struct Node {
7      int vertex;
8      struct Node* next;
9  };
10
11 struct Graph {
12     int numVertices;
13     struct Node* adjLists[MAX_VERTICES];
14     int visited[MAX_VERTICES];
15 };
16
17 struct Node* createNode(int v) {
18     struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
19     newNode->vertex = v;
20     newNode->next = NULL;
21     return newNode;
22 }
23
24 struct Graph* createGraph(int vertices) {
25     struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
26     graph->numVertices = vertices;
27     for (int i = 0; i < vertices; i++) {
28         graph->adjLists[i] = NULL;
29         graph->visited[i] = 0;
30     }
31     return graph;
32 }
33
34 void addEdge(struct Graph* graph, int src, int dest) {
35     struct Node* newNode = createNode(dest);
36     newNode->next = graph->adjLists[src];
37     graph->adjLists[src] = newNode;
38
39     // For undirected graph, add edge from dest to src too
40     newNode = createNode(src);
41     newNode->next = graph->adjLists[dest];
42     graph->adjLists[dest] = newNode;
43 }
44
45 void DFS(struct Graph* graph, int vertex) {
46     struct Node* adjList = graph->adjLists[vertex];
47     struct Node* temp = adjList;
48 }
```

```
49     graph->visited[vertex] = 1;
50     printf("%d ", vertex);
51
52     while (temp != NULL) {
53         int connectedVertex = temp->vertex;
54         if (graph->visited[connectedVertex] == 0) {
55             DFS(graph, connectedVertex);
56         }
57         temp = temp->next;
58     }
59 }
60
61 void BFS(struct Graph* graph, int startVertex) {
62     int queue[MAX_VERTICES];
63     int front = 0, rear = 0;
64
65     graph->visited[startVertex] = 1;
66     queue[rear++] = startVertex;
67
68     while (front != rear) {
69         int currentVertex = queue[front++];
70         printf("%d ", currentVertex);
71         struct Node* temp = graph->adjLists[currentVertex];
72
73         while (temp) {
74             int adjVertex = temp->vertex;
75             if (graph->visited[adjVertex] == 0) {
76                 graph->visited[adjVertex] = 1;
77                 queue[rear++] = adjVertex;
78             }
79             temp = temp->next;
80         }
81     }
82 }
83
84 void resetVisited(struct Graph* graph) {
85     for (int i = 0; i < graph->numVertices; i++) {
86         graph->visited[i] = 0;
87     }
88 }
89
90 int main() {
91     int vertices, edges, u, v, choice, startVertex;
92     printf("Enter number of vertices: ");
93     scanf("%d", &vertices);
94     struct Graph* graph = createGraph(vertices);
95
96     printf("Enter number of edges: ");
97     scanf("%d", &edges);
98     printf("Enter edges (source destination):\n");
```

```
99     for (int i = 0; i < edges; i++) {
100         scanf("%d %d", &u, &v);
101         addEdge(graph, u, v);
102     }
103
104     while (1) {
105         printf("\n1: BFS\n2: DFS\n0: Exit\nEnter choice: ");
106         scanf("%d", &choice);
107         if (choice == 0) break;
108         printf("Enter starting vertex: ");
109         scanf("%d", &startVertex);
110
111         resetVisited(graph);
112
113         if (choice == 1) {
114             printf("BFS traversal: ");
115             BFS(graph, startVertex);
116             printf("\n");
117         } else if (choice == 2) {
118             printf("DFS traversal: ");
119             DFS(graph, startVertex);
120             printf("\n");
121         } else {
122             printf("Invalid choice\n");
123         }
124     }
125
126     free(graph);
127     return 0;
128 }
129
```