BFS,DFS

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

#define SIZE 40

struct queue {

int items[SIZE];

int front;

int rear;

};

struct queue\* createQueue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void printQueue(struct queue\* q);

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph {

int numVertices;

struct node\*\* adjLists;

int\* visited;

};

// BFS algorithm

void bfs(struct Graph\* graph, int startVertex) {

struct queue\* q = createQueue();

graph-&gt;visited[startVertex] = 1;

enqueue(q, startVertex);

while (!isEmpty(q)) {

printQueue(q);

int currentVertex = dequeue(q);

printf(&quot;Visited %d\n&quot;, currentVertex);

struct node\* temp = graph-&gt;adjLists[currentVertex];

while (temp) {

int adjVertex = temp-&gt;vertex;

if (graph-&gt;visited[adjVertex] == 0) {

graph-&gt;visited[adjVertex] = 1;

enqueue(q, adjVertex);

}

temp = temp-&gt;next;

}

}

}

// Creating a node

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode-&gt;vertex = v;

newNode-&gt;next = NULL;

return newNode;

}

// Creating a graph

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph-&gt;numVertices = vertices;

graph-&gt;adjLists = malloc(vertices \* sizeof(struct node\*));

graph-&gt;visited = malloc(vertices \* sizeof(int));

int i;

for (i = 0; i &lt; vertices; i++) {

graph-&gt;adjLists[i] = NULL;

graph-&gt;visited[i] = 0;

}

return graph;

}

// Add edge

void addEdge(struct Graph\* graph, int src, int dest) {

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode-&gt;next = graph-&gt;adjLists[src];

graph-&gt;adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode-&gt;next = graph-&gt;adjLists[dest];

graph-&gt;adjLists[dest] = newNode;

}

// Create a queue

struct queue\* createQueue() {

struct queue\* q = malloc(sizeof(struct queue));

q-&gt;front = -1;

q-&gt;rear = -1;

return q;

}

// Check if the queue is empty

int isEmpty(struct queue\* q) {

if (q-&gt;rear == -1)

return 1;

else

return 0;

}

// Adding elements into queue

void enqueue(struct queue\* q, int value) {

if (q-&gt;rear == SIZE - 1)

printf(&quot;\nQueue is Full!!&quot;);

else {

if (q-&gt;front == -1)

q-&gt;front = 0;

q-&gt;rear++;

q-&gt;items[q-&gt;rear] = value;

}

}

// Removing elements from queue

int dequeue(struct queue\* q) {

int item;

if (isEmpty(q)) {

printf(&quot;Queue is empty&quot;);

item = -1;

} else {

item = q-&gt;items[q-&gt;front];

q-&gt;front++;

if (q-&gt;front &gt; q-&gt;rear) {

printf(&quot;Resetting queue &quot;);

q-&gt;front = q-&gt;rear = -1;

}

}

return item;

}

// Print the queue

void printQueue(struct queue\* q) {

int i = q-&gt;front;

if (isEmpty(q)) {

printf(&quot;Queue is empty&quot;);

} else {

printf(&quot;\nQueue contains \n&quot;);

for (i = q-&gt;front; i &lt; q-&gt;rear + 1; i++) {

printf(&quot;%d &quot;, q-&gt;items[i]);

}

}

}

int main() {

struct Graph\* graph = createGraph(6);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2); addEdge(graph, 1, 4);

addEdge(graph, 1, 3);

addEdge(graph, 2, 4);

addEdge(graph, 3, 4);

bfs(graph, 0);

return 0;

}

Depth First Search

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph {

int numVertices;

int\* visited;

// We need int\*\* to store a two dimensional array.

// Similary, we need struct node\*\* to store an array of Linked lists

struct node\*\* adjLists;

};

// DFS algo

void DFS(struct Graph\* graph, int vertex) {

struct node\* adjList = graph-&gt;adjLists[vertex];

struct node\* temp = adjList;

graph-&gt;visited[vertex] = 1;

printf(&quot;Visited %d \n&quot;, vertex);

while (temp != NULL) {

int connectedVertex = temp-&gt;vertex;

if (graph-&gt;visited[connectedVertex] == 0) {

DFS(graph, connectedVertex);

}

temp = temp-&gt;next;

}

}

// Create a node

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode-&gt;vertex = v;

newNode-&gt;next = NULL;

return newNode;

}

// Create graph

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph-&gt;numVertices = vertices;

graph-&gt;adjLists = malloc(vertices \* sizeof(struct node\*));

graph-&gt;visited = malloc(vertices \* sizeof(int));

int i;

for (i = 0; i &lt; vertices; i++) {

graph-&gt;adjLists[i] = NULL;

graph-&gt;visited[i] = 0;

}

return graph;

}

// Add edge

void addEdge(struct Graph\* graph, int src, int dest) {

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode-&gt;next = graph-&gt;adjLists[src];

graph-&gt;adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode-&gt;next = graph-&gt;adjLists[dest];

graph-&gt;adjLists[dest] = newNode;

}

// Print the graph

void printGraph(struct Graph\* graph) {

int v;

for (v = 0; v &lt; graph-&gt;numVertices; v++) {

struct node\* temp = graph-&gt;adjLists[v];

printf(&quot;\n Adjacency list of vertex %d\n &quot;, v);

while (temp) {

printf(&quot;%d -&gt; &quot;, temp-&gt;vertex);

temp = temp-&gt;next;

}

printf(&quot;\n&quot;);

}

}

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 3);

printGraph(graph);

DFS(graph, 2);

return 0;

}