qpset one—1 to10    I sessional qb

qpsettwo -- 4,5,6,8,9 from II sessional qb.

4. Create an adjacency list for graph and implement BFS.

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_NODES 20

// Structure to represent a node in the adjacency list

struct Node {

int data;

struct Node\* next;

};

// Structure to represent the graph

struct Graph {

int numNodes;

struct Node\* adjList[MAX\_NODES];

};

// Function to add an edge to the graph

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

// Create a new node for the destination vertex

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

newNode->data = dest;

newNode->next = graph->adjList[src];

graph->adjList[src] = newNode;

// Since the graph is undirected, add an edge from dest to src as well

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

newNode->data = src;

newNode->next = graph->adjList[dest];

graph->adjList[dest] = newNode;

}

// Function to print the adjacency list of the graph

void printAdjList(struct Graph\* graph) {

for (int i = 0; i < graph->numNodes; ++i) {

printf("Adjacency list for node %d: ", i);

struct Node\* current = graph->adjList[i];

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

}

// Function to perform Breadth-First Search on the graph

void BFS(struct Graph\* graph, int startNode) {

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

int queue[MAX\_NODES];

int front = 0, rear = 0;

visited[startNode] = true;

queue[rear++] = startNode;

while (front < rear) {

int currentNode = queue[front++];

printf("%d ", currentNode);

struct Node\* temp = graph->adjList[currentNode];

while (temp != NULL) {

int adjacentNode = temp->data;

if (!visited[adjacentNode]) {

visited[adjacentNode] = true;

queue[rear++] = adjacentNode;

}

temp = temp->next;

}

}

}

int main() {

struct Graph graph;

graph.numNodes = 4;

for (int i = 0; i < MAX\_NODES; ++i) {

graph.adjList[i] = NULL;

}

addEdge(&graph, 0, 1);

addEdge(&graph, 0, 2);

addEdge(&graph, 0, 3);

addEdge(&graph, 1, 2);

addEdge(&graph, 2, 3);

addEdge(&graph, 1, 3);

printf("");

printAdjList(&graph);

int startNode = 0;

printf("BFS starting from node %d: ", startNode);

BFS(&graph, startNode);

return 0;

}

5. Create an adjacency matrix for graph and implement DFS.

#include <stdio.h>

#include <stdlib.h>

#define n 3

void DFS(int);

int G[10][10], visited[10];

void main()

{

int i,j;

/\* printf("vertices : ");

scanf("%d ",&n);\*/

//print the matrix

printf("Adjacency matrix for grpah: ");

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

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

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

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

visited[i]=0;

DFS(0);

}

void DFS(int i){

int j;

printf("\n%d ", i);

visited[i]=1;

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

if(!visited[j]&&G[i][j]==1)

DFS(j); } }

6.Create an adjacency list for graph and find its connected components using DFS.

Write addedge function and print adjacency list function

8. Let the parent fork and let the child execute ls command with exec. Observe the result with and without having wait() system call in the parent.

9. Create two threads in a main program, let the first thread execute a function to display a message namely ”this is thread one” , similarly let the second thread displays ”this is thread two”.