Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Diagram

Description automatically generated with low confidence

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Q6

Graphical user interface, text, application, email

Description automatically generated

5

1 2

1 3

4 5

A

Enter the number of vertices:

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

The graph is not connected.

#include <stdio.h>

#include <stdlib.h>

typedef struct \_listnode

{

int vertex;

    struct \_listnode \*next;

} ListNode;

typedef ListNode StackNode;

typedef struct \_graph{

int V;

int E;

int \*visited; //store the visited vertices

int \*\*matrix;

}Graph;

typedef struct \_linkedlist

{

    int size;

    ListNode \*head;

} Stack;

// You should not change the prototypes of these functions

//////STACK///////////////////////////////////////////

void push(Stack \*sPtr, int vertex);

int pop(Stack \*sPtr);

int peek(Stack s);

int isEmptyStack(Stack s);

void removeAllItemsFromStack(Stack \*sPtr);

////GRAPH////////////////////////////////////////////

void printGraphMatrix(Graph );

////////////////////////////////////////////

int Connected (Graph);

int main()

{

Graph g;

int i,j;

printf("Enter the number of vertices:\n");

scanf("%d",&g.V);

g.E = 0;

g.matrix = (int \*\*)malloc(g.V\*sizeof(int \*));

for(i=0;i<g.V;i++)

g.matrix[i] = (int \*)malloc(g.V\*sizeof(int));

for(i=0;i<g.V;i++)

for(j=0;j<g.V;j++)

g.matrix[i][j] = 0;

g.visited = (int \*) malloc(sizeof(int)\*g.V);

for(i=0;i<g.V;i++)

g.visited[i] = 0;

int V1, V2;

printf("Enter two vertices which are adjacent to each other: (press a to stop)\n");

while(scanf("%d %d",&V1,&V2)==2)

{

if(V1>0 && V1<=g.V && V2>0 && V2<=g.V)

{

g.matrix[V1-1][V2-1] = 1;

g.matrix[V2-1][V1-1] = 1;

g.E++;

}

else

break;

printf("Enter two vertices which are adjacent to each other: (press a to stop)\n");

}

scanf("%\*c");

int res = Connected(g);

if(res ==1)

printf("The graph is connected.\n");

else

printf("The graph is not connected.\n");

return 0;

}

int Connected (Graph g)

{

//write your code here

int vertice;

Stack s;

s.size = 0;

s.head = NULL;

    push(&s, 1);

    g.visited[0] = 1;

int count = 1;

int stop\_Node = 0;

    while(isEmptyStack(s) != 1)

    {

        int j = peek(s); // see top

        stop\_Node = 0;

        for(vertice=1; vertice<=g.V; vertice++) //standard

        {

            if( g.visited[vertice-1] == 0 && g.matrix[j-1][vertice-1] == 1)

            {

                push(&s, vertice);

                g.visited[vertice-1] = 1;

                count++;

                stop\_Node = 1;

if(count == g.V) // check after

                {

                    return 1;

                }

                break;

            }

        }

        if(stop\_Node == 0)

        {

            pop(&s);

        }

    }

    return 0;

}

void printGraphMatrix(Graph g)

{

int i,j;

for(i=0;i<g.V;i++){

for(j=0;j<g.V;j++)

printf("%d\t",g.matrix[i][j]);

printf("\n");

}

}

void push(Stack \*sPtr, int vertex)

{

    StackNode \*newNode;

newNode= malloc(sizeof(StackNode));

newNode->vertex = vertex;

newNode->next = sPtr->head;

sPtr->head = newNode;

sPtr->size++;

}

int pop(Stack \*sPtr)

{

if(sPtr==NULL || sPtr->head==NULL){

return 0;

}

else{

StackNode \*temp = sPtr->head;

sPtr->head = sPtr->head->next;

free(temp);

sPtr->size--;

return 1;

}

}

int isEmptyStack(Stack s)

{

if(s.size==0) return 1;

else return 0;

}

int peek(Stack s){

return s.head->vertex;

}

void removeAllItemsFromStack(Stack \*sPtr)

{

    while(pop(sPtr));

}

Q7

Text

Description automatically generated

5

1 2

1 4

2 3

3 4

4 5

a

1 5

Enter the number of vertices:

Enter two vertices which are adjacent to each other:

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices which are adjacent to each other: (press a to stop)

Enter two vertices for finding their shortest distance: (press a to stop)

The shortest distance is 2

#include <stdio.h>

#include <stdlib.h>

typedef struct \_listnode

{

int vertex;

    struct \_listnode \*next;

} ListNode;

typedef struct \_graph{

int V;

int E;

int \*visited;

int \*\*matrix;

}Graph;

typedef ListNode QueueNode;

typedef struct \_queue{

int size;

QueueNode \*head;

QueueNode \*tail;

} Queue;

int SD (Graph G, int v, int w);

void printGraphMatrix(Graph );

// You should not change the prototypes of these functions

void enqueue(Queue \*qPtr, int item);

int dequeue(Queue \*qPtr);

int getFront(Queue q);

int isEmptyQueue(Queue q);

void removeAllItemsFromQueue(Queue \*qPtr);

int main()

{

Graph g;

int i,j;

printf("Enter the number of vertices:\n");

scanf("%d",&g.V);

g.E = 0;

g.matrix = (int \*\*)malloc(g.V\*sizeof(int \*));

for(i=0;i<g.V;i++)

g.matrix[i] = (int \*)malloc(g.V\*sizeof(int));

for(i=0;i<g.V;i++)

for(j=0;j<g.V;j++)

g.matrix[i][j] = 0;

g.visited = (int \*) malloc(sizeof(int)\*g.V);

for(i=0;i<g.V;i++) g.visited[i] = 0;

int V1, V2;

printf("Enter two vertices which are adjacent to each other:\n");

while(scanf("%d %d",&V1,&V2)==2)

{

if(V1>0 && V1<=g.V && V2>0 && V2<=g.V)

{

g.matrix[V1-1][V2-1] = 1;

g.matrix[V2-1][V1-1] = 1;

g.E++;

}

else

break;

printf("Enter two vertices which are adjacent to each other: (press a to stop)\n");

}

scanf("%\*c");

// printGraphMatrix(g);

printf("Enter two vertices for finding their shortest distance: (press a to stop)\n");

scanf("%d %d", &i, &j);

int d = SD(g,i,j);

if(d==-1)

printf("%d and %d are unconnected.\n",i,j);

else

printf("The shortest distance is %d\n",d);

return 0;

}

int SD(Graph g, int v, int z){

// Write your code here

    Queue q;

    q.size = 0;

    q.head = q.tail = NULL;

    enqueue(&q, v);

    g.visited[v - 1] = 1;

    int dist = 0;

    while (isEmptyQueue(q) != 1)

    {

        dist++;

        int j = dequeue(&q);

        for (int i = 0; i < g.V; i++)

        {

            if ( g.visited[i] == 0 && g.matrix[j - 1][i] == 1 )

            {

                if (i == z - 1)

                {

                 // double check

                 removeAllItemsFromQueue(&q);

                 //return g.visited[j];

return g.visited[j-1];

                }

                enqueue(&q, i + 1);

                g.visited[i] = g.visited[j - 1] + 1;

            }

        }

    }

    return -1;

}

void printGraphMatrix(Graph g)

{

int i,j;

for(i=0;i<g.V;i++){

for(j=0;j<g.V;j++)

printf("%d\t",g.matrix[i][j]);

printf("\n");

}

}

void enqueue(Queue \*qPtr, int item) {

QueueNode \*newNode;

newNode = malloc(sizeof(QueueNode));

if(newNode==NULL) exit(0);

newNode->vertex = item;

newNode->next = NULL;

if(isEmptyQueue(\*qPtr))

qPtr->head=newNode;

else

qPtr->tail->next = newNode;

qPtr->tail = newNode;

qPtr->size++;

}

int dequeue(Queue \*qPtr) {

if(qPtr==NULL || qPtr->head==NULL){ //Queue is empty or NULL pointer

return 0;

}

else{

int headValue = qPtr->head->vertex;

QueueNode \*temp = qPtr->head;

qPtr->head = qPtr->head->next;

if(qPtr->head == NULL) //Queue is emptied

qPtr->tail = NULL;

free(temp);

qPtr->size--;

return headValue;

}

}

int getFront(Queue q){

return q.head->vertex;

}

int isEmptyQueue(Queue q) {

if(q.size==0) return 1;

else return 0;

}

void removeAllItemsFromQueue(Queue \*qPtr)

{

    while(dequeue(qPtr));

}

Q8

Graphical user interface, text, application

Description automatically generated

5

1 2

1 4

2 3

3 4

4 1

4 5

A

Enter the number of vertices:

Enter a directed edge: (press a to stop)

Enter a directed edge: (press a to stop)

Enter a directed edge: (press a to stop)

Enter a directed edge: (press a to stop)

Enter a directed edge: (press a to stop)

Enter a directed edge: (press a to stop)

Enter a directed edge: (press a to stop)

The graph is not strongly connected.

#include <stdio.h>

#include <stdlib.h>

typedef struct \_listnode

{

int vertex;

    struct \_listnode \*next;

} ListNode;

typedef ListNode StackNode;

typedef struct \_graph{

int V;

int E;

int \*visited; //store the visited vertices

int \*\*matrix;

}Graph;

typedef struct \_linkedlist

{

    int size;

    ListNode \*head;

} Stack;

// You should not change the prototypes of these functions

//////STACK///////////////////////////////////////////

void push(Stack \*sPtr, int vertex);

int pop(Stack \*sPtr);

int peek(Stack s);

int isEmptyStack(Stack s);

void removeAllItemsFromStack(Stack \*sPtr);

////GRAPH////////////////////////////////////////////

void printGraphMatrix(Graph );

////////////////////////////////////////////

int Connected (Graph);

int main()

{

Graph g;

int i,j;

printf("Enter the number of vertices:\n");

scanf("%d",&g.V);

g.E = 0;

g.matrix = (int \*\*)malloc(g.V\*sizeof(int \*));

for(i=0;i<g.V;i++)

g.matrix[i] = (int \*)malloc(g.V\*sizeof(int));

for(i=0;i<g.V;i++)

for(j=0;j<g.V;j++)

g.matrix[i][j] = 0;

    g.visited = (int \*) malloc(sizeof(int)\*g.V);

for(i=0;i<g.V;i++) g.visited[i] = 0;

int V1, V2;

printf("Enter a directed edge: (press a to stop)\n");

while(scanf("%d %d",&V1,&V2)==2)

{

if(V1>0 && V1<=g.V && V2>0 && V2<=g.V)

{

g.matrix[V1-1][V2-1] = 1;

g.E++;

}

else

break;

printf("Enter a directed edge: (press a to stop)\n");

}

scanf("%\*c");

int res = Connected(g);

if(res == 1)

printf("The graph is strongly connected.\n");

else

printf("The graph is not strongly connected.\n");

return 0;

}

int Connected (Graph g)

{

//write your code here

int longest\_path, vertice;

int \*count = (int \*) malloc(sizeof(int)\*g.V);

Stack s;

s.size = 0;

s.head = NULL;

//int i;

// all zero

for(int i=0; i<g.V; i++)

{

count[i] = 0;

}

for(int i=0; i<g.V; i++)

{

//count[i] = 0; //not working try initialise first

push(&s, i+1);

for(int longest\_path=0; longest\_path<g.V; longest\_path++)

        {

g.visited[longest\_path]=0;

        }

        g.visited[i] = 1;

        count[i]++;

while(isEmptyStack(s) != 1)

{

int j = peek(s);

int stop\_Node = 0;

for(vertice=1; vertice<=g.V; vertice++)

{

if( g.visited[vertice-1] == 0 && g.matrix[j-1][vertice-1] == 1)

{

push(&s, vertice);

g.visited[vertice-1] = 1;

count[i]++;

stop\_Node = 1;

break;

}

}

if(stop\_Node == 0)

{

pop(&s);

}

}

}

for(int i=0; i<g.V; i++)

{

// no ha

if(count[i]!=g.V)

{

return 0;

}

}

return 1;

}

void printGraphMatrix(Graph g)

{

int i,j;

for(i=0;i<g.V;i++){

for(j=0;j<g.V;j++)

printf("%d\t",g.matrix[i][j]);

printf("\n");

}

}

void push(Stack \*sPtr, int vertex)

{

    StackNode \*newNode;

newNode= malloc(sizeof(StackNode));

newNode->vertex = vertex;

newNode->next = sPtr->head;

sPtr->head = newNode;

sPtr->size++;

}

int pop(Stack \*sPtr)

{

if(sPtr==NULL || sPtr->head==NULL){

return 0;

}

else{

StackNode \*temp = sPtr->head;

sPtr->head = sPtr->head->next;

free(temp);

sPtr->size--;

return 1;

}

}

int isEmptyStack(Stack s)

{

if(s.size==0) return 1;

else return 0;

}

int peek(Stack s){

return s.head->vertex;

}

void removeAllItemsFromStack(Stack \*sPtr)

{

    while(pop(sPtr));

}