**Write a C program to create a graph and perform a Breadth First Search and Depth First Search.**

**Algorithm:**

CODE:-

//BFS

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

typedef struct Node node;

node \*create(int data) {

node \*N = malloc(sizeof(node));

N->data = data;

N->next = NULL;

return N;

}

struct Queue {

int ele;

struct Queue \*next;

};

typedef struct Queue q;

q \*f = NULL;

q \*r = NULL;

void enqueue(int ele) {

q \*newnode = malloc(sizeof(q));

newnode->ele = ele;

newnode->next = NULL;

if (f == NULL && r == NULL) {

f = r = newnode;

return;

}

r->next = newnode;

r = newnode;

}

int dequeue() {

if (f == NULL) {

return -1; // Return -1 if the queue is empty

}

q \*temp = f;

f = f->next;

int s = temp->ele;

free(temp);

if (f == NULL) {

r = NULL; // Update rear pointer if the queue becomes empty

}

return s;

}

void addedge(node \*adj[], int u, int v) {

node \*newnode = create(v);

newnode->next = adj[u];

adj[u] = newnode;

}

void bfs(node \*adj[], int si, int v) {

int visited[v];

for (int i = 0; i < v; ++i) {

visited[i] = 0;

}

enqueue(si);

visited[si] = 1;

while (f != NULL) {

int u = dequeue();

printf("%d ", u);

node \*temp = adj[u];

while (temp != NULL) {

int d = temp->data;

if (!visited[d]) {

visited[d] = 1;

enqueue(d);

}

temp = temp->next;

}

}

printf("\n");

}

int main() {

int vertices = 5;

// Adjacency list representation of the graph

node \*adjList[vertices];

for (int i = 0; i < vertices; ++i)

adjList[i] = NULL;

// Add edges to the graph

addedge(adjList, 0, 1);

addedge(adjList, 0, 2);

addedge(adjList, 1, 3);

addedge(adjList, 1, 4);

addedge(adjList, 2, 4);

// Perform BFS traversal starting from vertex 0

printf("Breadth First Traversal starting from vertex 0: ");

bfs(adjList, 0, vertices);

return 0;}

OUTPUT FOR BFS:-

Breadth First Traversal starting from vertex 0: 0 2 1 4 3

CODE:-

//DFS

#include <stdio.h>

#include <stdlib.h>

// Node structure for adjacency list

struct Node {

int data;

struct Node \*next;

};

typedef struct Node node;

// Create a new node

node \*create(int data) {

node \*N = malloc(sizeof(node));

N->data = data;

N->next = NULL;

return N;

}

// Add edge to the graph

void addedge(node \*adj[], int u, int v) {

node \*newnode = create(v);

newnode->next = adj[u];

adj[u] = newnode;

}

// Recursive DFS function

void dfsUtil(node \*adj[], int v, int visited[]) {

visited[v] = 1;

printf("%d ", v);

node \*temp = adj[v];

while (temp != NULL) {

int d = temp->data;

if (!visited[d]) {

dfsUtil(adj, d, visited);

}

temp = temp->next;

}

}

// DFS traversal

void dfs(node \*adj[], int si, int vertices) {

int visited[vertices];

for (int i = 0; i < vertices; ++i) {

visited[i] = 0;

}

dfsUtil(adj, si, visited);

}

int main() {

int vertices = 5;

// Adjacency list representation of the graph

node \*adjList[vertices];

for (int i = 0; i < vertices; ++i)

adjList[i] = NULL;

// Add edges to the graph

addedge(adjList, 0, 1);

addedge(adjList, 0, 2);

addedge(adjList, 1, 3);

addedge(adjList, 1, 4);

addedge(adjList, 2, 4);

// Perform DFS traversal starting from vertex 0

printf("Depth First Traversal starting from vertex 0: ");

dfs(adjList, 0, vertices);

return 0;

}

OUTPUT FOR DFS:-

Depth First Traversal starting from vertex 0: 0 2 4 1 3