**Name : Tanishq Thuse**

**Branch : CS(AI)**

**Year : SY**

**Div : B**

**Roll no. : 60**

**PRN : 12310237**

**Title : AI Assignment-2 : Implement Uninformed Search Algorithms**

**Q1) Implementation of BFS.**

**Code :**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

struct Queue {

int items[MAX\_VERTICES];

int front;

int rear;

};

int graph[MAX\_VERTICES][MAX\_VERTICES];

int visited[MAX\_VERTICES];

int vertices, edges;

struct Queue\* createQueue() {

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

q->front = -1;

q->rear = -1;

return q;

}

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

if (q->rear == MAX\_VERTICES - 1)

printf("Queue is full");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct Queue\* q) {

int item;

if (q->front == -1)

printf("Queue is empty");

else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

q->front = q->rear = -1;

}

return item;

}

return -1;

}

// BFS algorithm

void bfs(int startVertex) {

struct Queue\* q = createQueue();

visited[startVertex] = 1;

enqueue(q, startVertex);

printf("Breadth-First Traversal starting from vertex %d: ", startVertex);

while (q->front != -1) {

int currentVertex = dequeue(q);

printf("%d ", currentVertex);

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

if (graph[currentVertex][i] == 1 && !visited[i]) {

visited[i] = 1;

enqueue(q, i);

}

}

}

}

int main() {

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

scanf("%d", &vertices);

printf("Enter the number of edges: ");

scanf("%d", &edges);

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

visited[i] = 0;

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

graph[i][j] = 0;

}

printf("Enter the edges (format: vertex1 vertex2):\n");

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

int vertex1, vertex2;

scanf("%d %d", &vertex1, &vertex2);

graph[vertex1][vertex2] = 1;

graph[vertex2][vertex1] = 1;

}

int startVertex;

printf("Enter the starting vertex for BFS: ");

scanf("%d", &startVertex);

bfs(startVertex);

printf("\n");

return 0;

/\* Visualization

0

/ \

1 2

/ \

3 4

\*/

/\*\*Example run :

\* Enter the number of vertices: 5

Enter the number of edges: 4

Enter the edges (format: vertex1 vertex2):

0 1

0 2

1 3

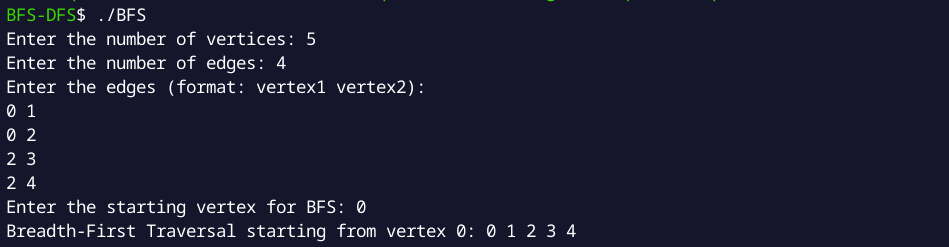
1 4

Enter the starting vertex for BFS: 0

\*/

}

**Output** :

****

**Q2) Implementation of DFS**

**Code :**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

int graph[MAX\_VERTICES][MAX\_VERTICES];

int visited[MAX\_VERTICES];

int vertices, edges;

struct Stack {

int items[MAX\_VERTICES];

int top;

};

void initializeStack(struct Stack\* stack) {

stack->top = -1;

}

void push(struct Stack\* stack, int value) {

stack->top++;

stack->items[stack->top] = value;

}

int pop(struct Stack\* stack) {

int item = stack->items[stack->top];

stack->top--;

return item;

}

void dfs(int startVertex) {

struct Stack stack;

initializeStack(&stack);

push(&stack, startVertex);

visited[startVertex] = 1;

printf("Depth-First Traversal starting from vertex %d: ", startVertex);

while (stack.top != -1) {

int currentVertex = pop(&stack);

printf("%d ", currentVertex);

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

if (graph[currentVertex][i] == 1 && !visited[i]) {

push(&stack, i);

visited[i] = 1;

}

}

}

}

int main() {

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

scanf("%d", &vertices);

printf("Enter the number of edges: ");

scanf("%d", &edges);

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

visited[i] = 0;

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

graph[i][j] = 0;

}

printf("Enter the edges (format: vertex1 vertex2):\n");

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

int vertex1, vertex2;

scanf("%d %d", &vertex1, &vertex2);

graph[vertex1][vertex2] = 1;

graph[vertex2][vertex1] = 1;

}

int startVertex;

printf("Enter the starting vertex for DFS: ");

scanf("%d", &startVertex);

dfs(startVertex);

printf("\n");

return 0;

/\* Visualization

0

/ \

1 2

/ \

3 4

\*/

/\*\*Example run :

\* Enter the number of vertices: 5

Enter the number of edges: 4

Enter the edges (format: vertex1 vertex2):

0 1

0 2

1 3

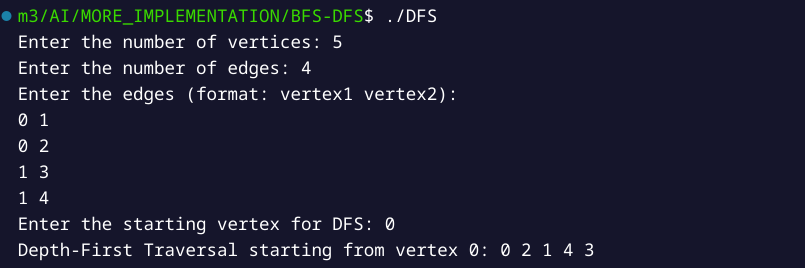
1 4

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

\*/

}

**Output :**

****

**Q3) Bidirectional Search with Limited Depth(BLFS) :**

**Code** :

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

struct Queue {

int items[MAX\_VERTICES];

int front, rear;

};

int graph[MAX\_VERTICES][MAX\_VERTICES];

int visited\_start[MAX\_VERTICES];

int visited\_goal[MAX\_VERTICES];

int vertices, edges;

struct Queue\* createQueue() {

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

q->front = -1;

q->rear = -1;

return q;

}

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

if (q->rear == MAX\_VERTICES - 1)

printf("Queue is full\n");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct Queue\* q) {

int item;

if (q->front == -1) {

printf("Queue is empty\n");

return -1;

} else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

q->front = q->rear = -1;

}

return item;

}

}

// Function to check if any common vertex is visited by both searches

int isIntersecting() {

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

if (visited\_start[i] && visited\_goal[i])

return i; // Return the intersecting vertex

}

return -1;

}

// Bidirectional BFS algorithm

void bidirectionalBFS(int startVertex, int goalVertex) {

struct Queue\* q\_start = createQueue();

struct Queue\* q\_goal = createQueue();

// Initialize visited arrays for both searches

visited\_start[startVertex] = 1;

visited\_goal[goalVertex] = 1;

enqueue(q\_start, startVertex);

enqueue(q\_goal, goalVertex);

printf("Bidirectional BFS from vertex %d to vertex %d:\n", startVertex, goalVertex);

while (q\_start->front != -1 && q\_goal->front != -1) {

// BFS from start vertex

int current\_start = dequeue(q\_start);

printf("Start search visiting: %d\n", current\_start);

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

if (graph[current\_start][i] == 1 && !visited\_start[i]) {

visited\_start[i] = 1;

enqueue(q\_start, i);

}

}

// BFS from goal vertex

int current\_goal = dequeue(q\_goal);

printf("Goal search visiting: %d\n", current\_goal);

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

if (graph[current\_goal][i] == 1 && !visited\_goal[i]) {

visited\_goal[i] = 1;

enqueue(q\_goal, i);

}

}

// Check for intersection between the two searches

int intersecting\_vertex = isIntersecting();

if (intersecting\_vertex != -1) {

printf("Path found! The searches intersect at vertex %d\n", intersecting\_vertex);

return;

}

}

printf("No path found between vertex %d and vertex %d\n", startVertex, goalVertex);

}

int main() {

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

scanf("%d", &vertices);

printf("Enter the number of edges: ");

scanf("%d", &edges);

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

visited\_start[i] = 0;

visited\_goal[i] = 0;

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

graph[i][j] = 0;

}

printf("Enter the edges (format: vertex1 vertex2):\n");

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

int vertex1, vertex2;

scanf("%d %d", &vertex1, &vertex2);

graph[vertex1][vertex2] = 1;

graph[vertex2][vertex1] = 1;

}

int startVertex, goalVertex;

printf("Enter the starting vertex for BLFS: ");

scanf("%d", &startVertex);

printf("Enter the goal vertex for BLFS: ");

scanf("%d", &goalVertex);

bidirectionalBFS(startVertex, goalVertex);

printf("\n");

return 0;

/\*\*Visualization :

\* 0

/ \

1 2

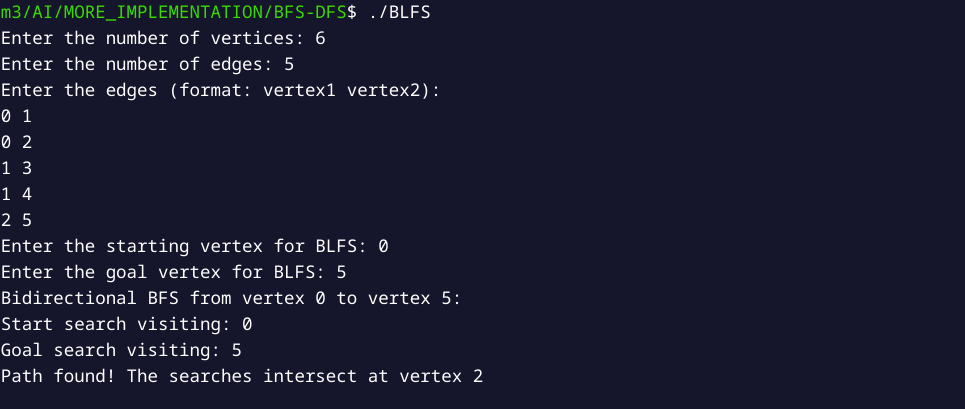
/ \ \

3 4 5

\*/

}

**Output :**

 **Q4) Depth-Limited Search**

**Code :**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

int graph[MAX\_VERTICES][MAX\_VERTICES];

int visited[MAX\_VERTICES];

int vertices, edges;

// Stack implementation for DFS

struct Stack {

int items[MAX\_VERTICES];

int top;

};

void initializeStack(struct Stack\* stack) {

stack->top = -1;

}

void push(struct Stack\* stack, int value) {

stack->top++;

stack->items[stack->top] = value;

}

int pop(struct Stack\* stack) {

int item = stack->items[stack->top];

stack->top--;

return item;

}

bool isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

// Depth-Limited Search algorithm

void depthLimitedSearch(int startVertex, int depthLimit) {

struct Stack stack;

initializeStack(&stack);

push(&stack, startVertex);

visited[startVertex] = 1;

printf("Depth-Limited Search starting from vertex %d with depth limit %d:\n", startVertex, depthLimit);

while (!isEmpty(&stack)) {

int currentVertex = pop(&stack);

printf("%d ", currentVertex);

if (currentVertex == depthLimit) {

printf("\nReached the depth limit %d\n", depthLimit);

return;

}

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

if (graph[currentVertex][i] == 1 && !visited[i]) {

push(&stack, i);

visited[i] = 1;

}

}

}

printf("\nNo further nodes to explore.\n");

}

int main() {

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

scanf("%d", &vertices);

printf("Enter the number of edges: ");

scanf("%d", &edges);

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

visited[i] = 0;

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

graph[i][j] = 0;

}

printf("Enter the edges (format: vertex1 vertex2):\n");

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

int vertex1, vertex2;

scanf("%d %d", &vertex1, &vertex2);

graph[vertex1][vertex2] = 1;

graph[vertex2][vertex1] = 1;

}

int startVertex, depthLimit;

printf("Enter the starting vertex for DLFS: ");

scanf("%d", &startVertex);

printf("Enter the depth limit for DLFS: ");

scanf("%d", &depthLimit);

depthLimitedSearch(startVertex, depthLimit);

printf("\n");

return 0;

/\*\* Visualization

0

/ \

1 2

/ \ \

3 4 5

\*/

/\*\*Enter the number of vertices: 6

Enter the number of edges: 5

Enter the edges (format: vertex1 vertex2):

0 1

0 2

1 3

1 4

2 5

Enter the starting vertex for DLFS: 0

Enter the depth limit for DLFS: 2

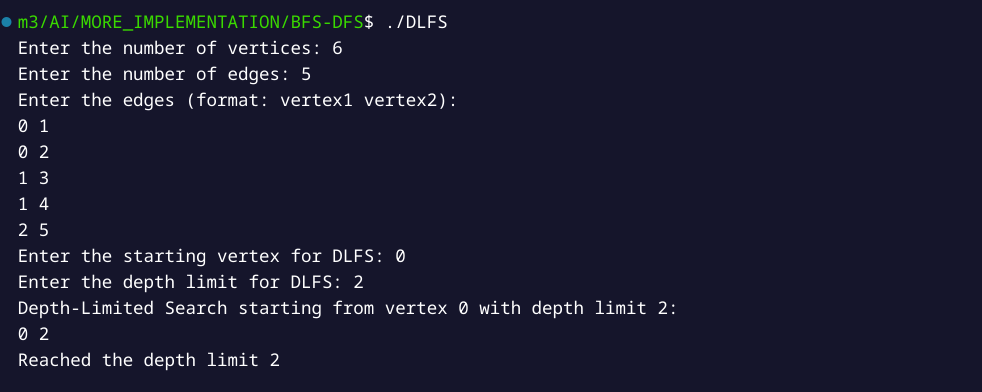
Depth-Limited Search starting from vertex 0 with depth limit 2:

0 2 1

Reached the depth limit 2 \*/

}

**Output :**

****