**EXPEIMENT 2**

**AIM:** Identify and analyze Uninformed Search Algorithm to solve the problem.

1. BFS

2. DFS

3. BFID

**THEORY:**

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

**BFS:**

BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

The breadth-first search algorithm is an example of a general-graph search algorithm.

Breadth-first search implemented using FIFO queue data structure.

**Time Complexity T(b) :** 1+b2+b3+.......+ bs = O (bs)

Where, s= depth of shallowest solution; b is a node at every state.

**Space Complexity:** O(bd).

**Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

**Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

**Advantages:**

* BFS is comlete.
* If there are more than one solutions for a given problem, then BFS will provide the minimal solution requiring the least number of steps.

**Disadvantages:**

* It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
* BFS needs lots of time if the solution is far away from the root node.

**DFS:**

Depth-first search is a recursive algorithm for traversing a tree or graph data structure.

It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.

DFS uses a stack data structure for its implementation.

**Completeness:** DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

**Time Complexity T(n) =** 1+ n2+ n3 +.........+ nd = O(nd)

where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

**Space Complexity:** O(nd) => DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set ( =n).

**Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

**Advantage:**

* DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.

**Disadvantage:**

* There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
* DFS algorithm goes for deep down searching and sometime it may go to the infinite loop

**DFID**

The iterative deepening algorithm is a combination of DFS and BFS algorithms. DFID search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

It performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

**Completeness:** Generates a complete solution if the branching factor is finite.

**Space Complexity**: O(d).

**Time Complexity:** O(bd)

Where, b is the branching factor and d is the current depth.

**Optimal:** DFID algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

**Advantages:**

* It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

**Disadvantages:**

* The main drawback of DFID is that it repeats all the work of the previous phase.

**CODE:**

**1) BFS & DFS**

#include <conio.h>

#include<stdio.h>

#include<ctype.h>

#include <stdbool.h>

#define MAX 20

int a[MAX][MAX], visited[MAX], dfs\_list[MAX], k = 0, queue[MAX], front = 0, rear = -1, goal[MAX], goal\_pending;

bool isGoal(int s, int len) {

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

if (s == goal[i]) {

return true;

}

}

return false;

}

void dfs(int s, int n, int goal\_num) {

visited[s] = 1; // checked its entry

//dfs\_list[k++] =s ;

if (isGoal(s, goal\_num)) {

printf(" [%d] \t", s);

goal\_pending--;

} else {

printf(" %d \t", s);

}

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

if (a[s][i] == 1 && visited[i] == 0) {

//passing the current node as parent to traverse in depth.

dfs(i, n, goal\_num);

}

}

}

void bfs(int s, int n, int goal\_num) {

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

if (a[s][i] == 1 && visited[i] == 0) {

//adding all the unvisited childs

queue[++rear] = i;

visited[i] = 1; //make it visited

// if(i == goal){

if (isGoal(i, goal\_num)) {

printf(" [%d] ", i);

goal\_pending--;

} else {

printf(" %d ", i);

}

}

}

if (front <= rear) {

bfs(queue[++front], n, goal\_num);

}

}

void main() {

int n, s, key = -1;

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

scanf("%d", & n);

printf("\n Enter the Matrix of Vertices : \n");

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

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

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

}

}

printf("\n The Adjacency Matrix : \n\n");

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

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

printf("%d \t", a[i][j]);

}

printf("\n");

}

do {

// Clearing

for (int j = 0; j < MAX; j++) {

visited[j] = 0;

queue[j] = 0;

goal[j] = 0;

}

front = 0;

rear = -1;

printf("\n \n \*\*\*\*\* MAIN MENU \*\*\*\*\* \n");

printf("\n 1)BFS ");

printf("\n 2)DFS");

printf("\n -1)Exit ");

printf("\n \n Enter your Choice : ");

scanf("%d", & key);

switch (key) {

case 1: {

printf("\n \*\*\*\*\* BFS \*\*\*\*\*");

printf("\n\n Enter Goal Vertex(s) :");

printf("\n To exit -1 ");

int k = 0;

printf("\n\n Enter Goal %d :", k + 1);

scanf("%d", & goal[k]);

while (goal[k] != -1) {

printf(" Enter Goal %d :", k + 2);

scanf("%d", & goal[++k]);

}

int num\_goals = k;

goal\_pending = k;

//Select your Start vertex.

printf("\n Select your Start Vertex : ");

scanf("%d", & s);

queue[++rear] = s;

visited[queue[front]] = 1; //make it visited

printf("\n \*\*\*\*\*BFS TRAVERSAL\*\*\*\*\*\n\n");

if (isGoal(s, num\_goals)) {

printf(" [%d] ", s);

} else {

printf(" %d ", s);

}

bfs(s, n, num\_goals);

if (goal\_pending > 0) {

printf("\n No. of Goal Nodes still pending : %d", goal\_pending);

} else {

printf("\n All Goal Nodes reached!!");

}

printf("\n\n\*\*\*\*\*\*\*\*\*");

printf("\n");

break;

}

case 2: {

printf("\n \*\*\*\*\* DFS \*\*\*\*\*");

printf("\n\n Enter Goal Vertex(s) :");

printf("\n To exit -1 ");

int k = 0;

printf("\n\n Enter Goal %d :", k + 1);

scanf("%d", & goal[k]);

while (goal[k] != -1) {

printf(" Enter Goal %d :", k + 2);

scanf("%d", & goal[++k]);

}

int num\_goals = k;

goal\_pending = k;

//Select your Start vertex.

printf("\n \n Select your Start Vertex : ");

scanf("%d", & s);

printf("\n \*\*\*\*\*DFS TRAVERSAL\*\*\*\*\*\n\n");

dfs(s, n, num\_goals);

if (goal\_pending > 0) {

printf("\n No. of Goal Nodes still pending : %d", goal\_pending);

} else {

printf("\n All Goal Nodes reached!!");

}

printf("\n\n\*\*\*\*\*\*\*\*\*\n");

break;

}

case -1: {

printf("\n \n\*\*\*\*\* END \*\*\*\*\* \n");

break;

}

default: {

printf("\n INVALID KEY!! ");

break;

}

}

} while (key != -1);

}

**2) DFID**

#include <conio.h>

#include<stdio.h>

#include<ctype.h>

#include <stdbool.h>

#define MAX 20

int a[MAX][MAX], visited[MAX], dfs\_list[MAX], k = 0, queue[MAX], front = 0, rear = -1 ,goal[MAX], goal\_pending ;

int MAX\_DEPTH ;

bool isGoal(int s , int len){

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

if(s == goal[i]){

return true;

}

}

return false ;

}

void dfs(int s , int n, int goal\_num , int depth ,int limit) {

visited[s] = 1; // checked its entry

int d= depth ;

// if(s == isGoal(s ,goal\_num)){

// printf(" [%c] \t",(char)(s+65));

// goal\_pending-- ;

// }else{

// printf(" %c \t", (char)(s+65));

if(isGoal(s ,goal\_num)){

printf(" [%d] ", s);

goal\_pending-- ;

}else{

printf(" %d ", s);

}

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

if(depth<=limit){

if (a[s][i] == 1 && visited[i] == 0) {

//passing the current node as parent to traverse in depth.

dfs(i, n, goal\_num , ++d , limit);

}

}

}

}

bool isGoalPending(){

if(goal\_pending >0){

return true ;

}else{

return false;

}

}

void DFID(int s, int n, int num\_goal){

printf("\n Level %d : %d" , 0 , s) ;

printf("\n %d goal(s) found ", (num\_goal- goal\_pending)) ;

printf("\n %d goal(s) pending ", goal\_pending) ;

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

// Clearing

for (int j = 0; j < MAX ; j++) {

visited[j] = 0;

queue[j] = 0;

}

// Restoring total goal before next loop

goal\_pending = num\_goal ;

int depth=0 ;

if(isGoalPending()){

// Next Level

printf("\n\n Level %d : " , i+1) ;

dfs(s, n , num\_goal , depth, i) ;

printf("\n %d goal(s) found ", (num\_goal- goal\_pending)) ;

printf("\n %d goal(s) pending ", goal\_pending) ;

}else{

printf("\n All goals reached!! ") ;

break ;

}

}

void main() {

int n, s,g, key = -1;

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

scanf("%d", &n);

printf("\n Enter the Matrix of Vertices : \n");

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

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

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

}

}

// Clearing

for (int j = 0; j < MAX ; j++) {

visited[j] = 0;

queue[j] = 0;

goal[j] = 0 ;

}

printf("\n Enter the MAX DEPTH : ") ;

scanf("%d", &MAX\_DEPTH) ;

printf("\n Enter the Start Node : ") ;

scanf("%d", &s) ;

printf("\n\n Enter Goal Vertex(s) :");

printf("\n To exit -1 ");

int k =0 ;

printf("\n\n Enter Goal %d :", k+1);

scanf("%d", &goal[k]);

while(goal[k] != -1){

printf(" Enter Goal %d :", k+2);

scanf("%d", &goal[++k]);

}

int num\_goal = k ;

goal\_pending = k ;

printf("\n \*\*\*\*\* DFID\*\*\*\*\* \n");

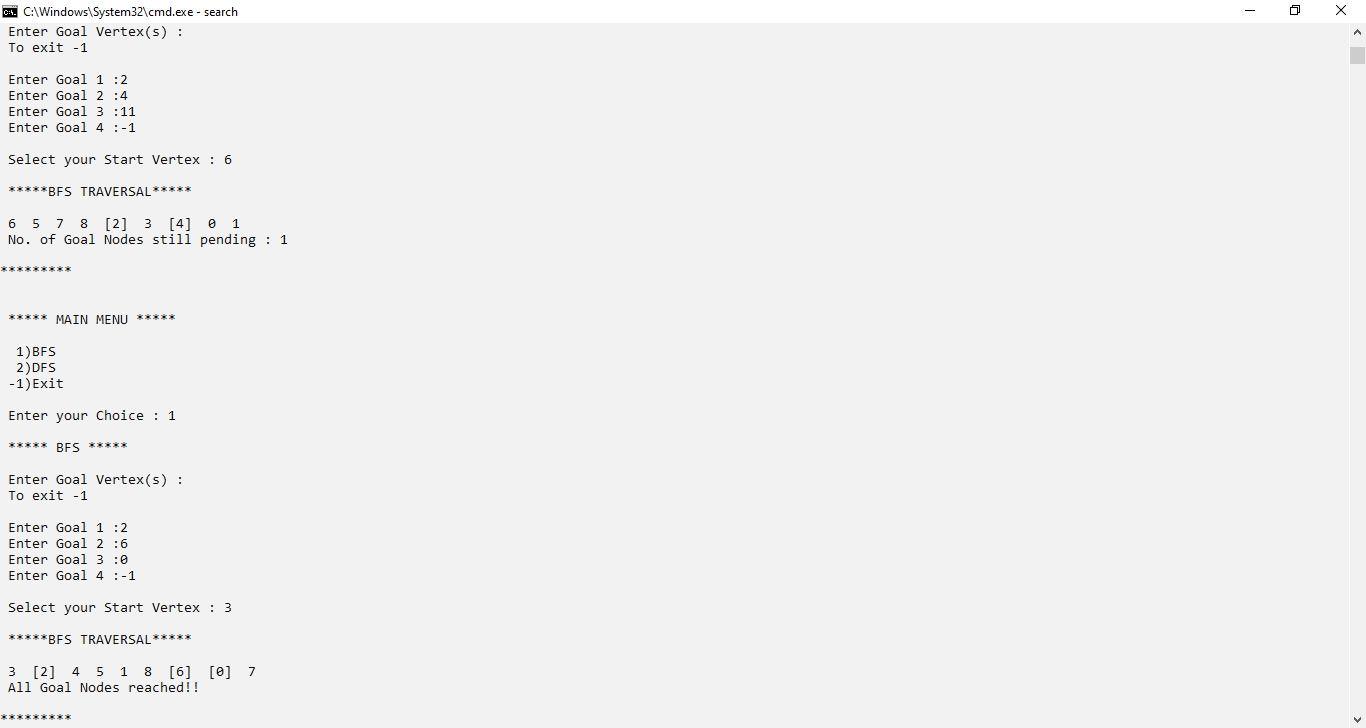
DFID(s, n, num\_goal) ;

}

**OUTPUT**

1. **BFS**

A picture containing table

Description automatically generated

1. **DFS**

Chart

Description automatically generated with low confidenceGraphical user interface, application

Description automatically generated

**3) DFID**

A picture containing background pattern

Description automatically generatedChart, scatter chart

Description automatically generated

**CONCLUSION:** In this experiment I implemented 3 differnet Uninformed Search Algorithms viz, BFS, DFS and DFID. BFS gives a complete solution but has too large space and time complexities. DFS on the other hand, requires less memory space and than BFS, but it is not complete. There’s a possiblity that the states may be re-curring thus not possibility of a solution. Finally, DFID is a combination of BFS and DFS that mutually provides the pros of both the algorithms.