**“Comparative Study and Analysis of**

**Uninformed Search Algorithms”**

***A***

***Project Report***

*submitted in partial fulfilment of the*

*requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE & ENGINEERING**

**With Specialization in**

**Oil and Gas Informatics**

**by**

|  |  |
| --- | --- |
| **Name** | **Roll No.** |
| **Priya Bansal** | **R970217030** |
| Shristi Katiyar | **R970217030** |
| **Siddarth Kumar Jha** | **R970217040** |
| Rahul Yadav | **R970217051** |

***under the guidance of***

**Dr Kingshuk Srivastava**

Assistant Professor (S.G.)

Department of Informatics

upes-new-logo

**Department of Informatics**

**School of Computer Science**

**University of Petroleum & Energy Studies**

**Bidholi, Via Prem Nagar, Dehradun, UK**

**December – 2019**

upes-new-logo

**CANDIDATE’S DECLARATION**

I/We hereby certify that the project work entitled **“ Comparative Study and Analysis of Uninformed search Algorithms”** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in OIL AND GAS INFORMATICS and submitted to the Department of Informatics at School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **August,2019** to **November, 2019** under the supervision of **Dr. Kingshuk Shrivastava, Assistance Professor, Department of Informatics**

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

**Priya Bansal, Shristi Katiyar, Siddarth Kumar Jha, Rahul Yadav**

**(R970217030) (R970217039) (R970217040) (R970217051)**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 28-11-2019 **Dr. Kingshuk Shrivastava**

Project Guide

**Dr Thipendra Pal Singh**

**Professor and H.O.D**

Department of Informatics

School of Computer Science

University of Petroleum & Energy Studies

Dehradun – 248 001 (Uttarakhand)

**ACKNOWLEDGEMENT**

We wish to express our deep gratitude to our guide **Dr Kingshuk Shristava**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank our respected Head of the Department, **Dr Thipendra Pal Singh,** for his great support in doing our minor project in at **SoCS**.

We are also grateful to **Dr Manish Prateek, Dean SoCS**, for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for every support, they have given us.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Priya Bansal** | **Shristi Katiyar** | **Siddarth Kumar Jha** | **Rahul Yadav** |
| **Roll No.** | **R970217030** | **R970217039** | **R970217040** | **R970217051** |

**ABSTRACT**

The search process is a basic task in every computer application wherein the state-space search is one of the approaches to handling the job. It intends to find all possible states for a given problem with the desired properties which acts upon the given set of states. Informed search and Uninformed search are the two broad categories of state-space search, this project is based upon the searching techniques placed under the second category of state-space search, The Un-Informed search. The two most valuable resources for any optimizing algorithms are time and memory. This project will be implemented taking these two as the base and an optimized algorithm will be crafted.

Keywords: state-space search, uninformed search, optimization.

**TABLE OF CONTENTS**

1. **Introduction 1**

1. **Related Work 2-4**

1. **Problem Statement 5**
2. **Objective 5**

1. **Design 6-8**

5.1 Methodology 6

5.2 Algorithms of the project 7-8

1. **Implementation 9-21**

6.1 Pseudocode & Flow Chart 9-12

6.2 SWAT Analysis 13-15 6.2 Output Screen 16-17

6.3 Result Analysis p18-21

1. **Conclusion and Future Scope 21**

1. **Appendix I Project Code 22-31**
2. **References 31**

**LIST OF FIGURES**

**S.No. Figure Page No**

1. **Chapter 1 1 - 3**

Fig. 1.1 DFS Example 1

Fig. 1.2 BFS Example 2

Fig. 1.3 DLS Example 3

Fig. 1.4 IDDFS Example 3

1. **Chapter 6 9 - 12**

Fig 6.1 Flow Chart (DFS) 9

Fig 6.2 Flow Chart (BFS) 10

Fig 6.3 Flow Chart (DLS) 11

Fig 6.4 Flow Chart (IDDFS) 12

Fig 6.5 Output DFS (x64 bits) 16

Fig 6.6 Output BFS (x64 bits) 16

Fig 6.7 Output DLS (x64 bits) 16

Fig 6.8 Output IDDFS (x64 bits) 16

Fig 6.9 Output BFS (x32 bits) 26

Fig 6.10 Output DFS (x32 bits) 26

Fig 6.11 Output DlS (x32 bits) 27

Fig 6.12 Output IDDFS (x32 bits) 27

**LIST OF TABLES**

**S.No. Table Page No**

1. **Chapter 6 13 - 15**

Table 6.2.1 DFS SWAT Analysis 13

Table 6.2.2 BFS SWAT Analysis 13 - 14

Table 6.2.3 DLS SWAT Analysis 14 - 15

Table 6.2.4 IDDFS SWAT Analysis 15

1. **INTRODUCTION**

Addressing the aspect of a problem-solving approach, one is under a particular situation and wants to be in the desired situation, so what comes out as the task is to make a series of decisions or the series of moves which will transform the given situation into the desired situation. A search can be performed in many different spaces as the given state and the goal state likewise, several other states lying in between the two, the given and the goal state. Search is performed over the state spaces to end up at the desired solution when the series of decision is not known. There are two types of state-space searches, informed search and uninformed search. Area of interest of this project lies around uninformed search under state-space search, which is also known as Blind search as it is unaware of the search space and can only distinguish between a goal state or a non-goal state. Under some circumstances, however, only a criterion is Goal which is given as a form of the Boolean objective function. The method other than, that this approach follows, will then not be able to descend a gradient anymore and degenerate to random walks

Here, the uninformed search is a viable alternative since they do not require or take into consideration any knowledge about the special nature of the problem (apart from the knowledge represented by the expand operation, of course). Such algorithms are very general and can be applied to a wide variety of problems. Their common drawback is that search spaces are often very large. Without the incorporation of information in the form of heuristic functions, for example, the search may take very long and quickly becomes infeasible. Various algorithms come under uninformed searches like Breadth-First Search, Depth First Search, Depth-Limited Search, Depth First Iterative Deepening Search, Random Walks, and Bidirectional Search. This Project takes into consideration, four major algorithms that are Breadth-First Search, Depth First Search, Depth-Limited Search, and Depth First search with Iterative Deepening.

1. **Related Study**

Set of all possible states for a given problem is known as **State Space of a problem** and searching is performed over those state spaces to obtain the desired solution. State-space search is implicit that is the goal nodes is not represented as an explicit object but rather are determined algorithmically from some more concise input.

There are two types of state-space searches, informed search and uninformed search. Area of interest of this project lies around uninformed search under state-space search.

**Uninformed search,** which is also known as Blind search as it is unaware of the search space and can only distinguish between a goal state or a non-goal state. It is a class of general-purpose search algorithm which operates in brute force-way. Following are the various types of the uninformed search algorithm.

1. **Depth First Search**: It is a simple search algorithm based on stack implementation. This starts traversing from root explores a path all the way to leaf before backtracking and exploring another path. The solution is optimal in most of the cases. Show in figure 1.1­[8.1]

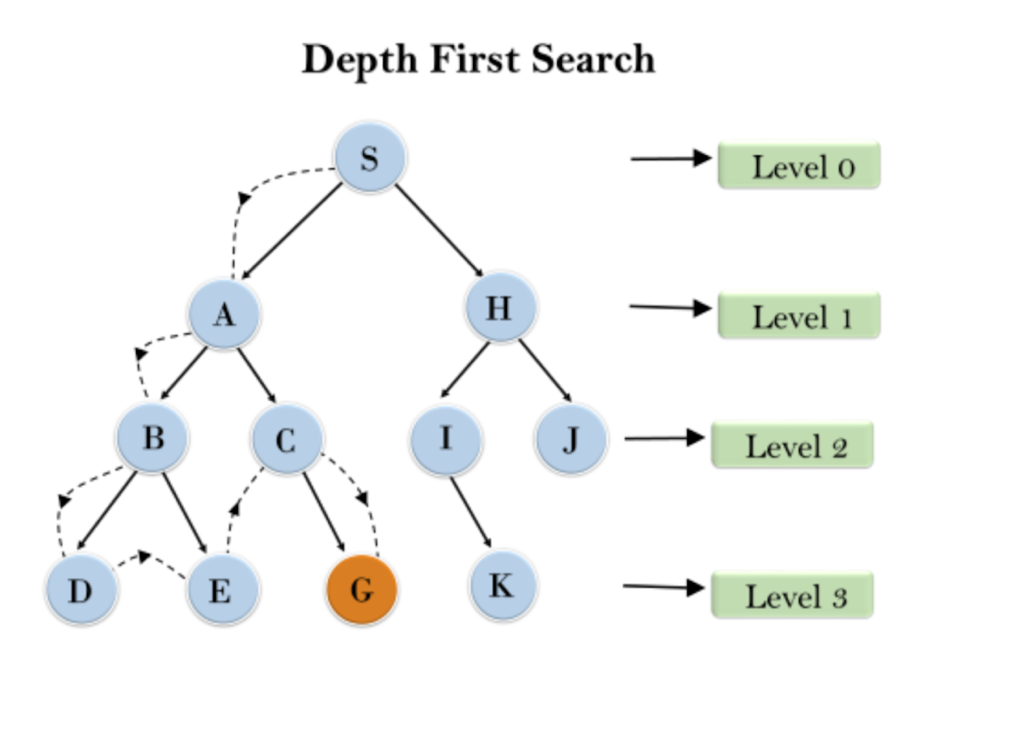


Figure 1.1

1. **Breadth-First Search**: It is a simple search algorithm based on queue implementation with a different logic than depth-first search. This traverses the tree level by level and depth by depth. This starts traversing from root explores the neighbour nodes first, before moving to the next node. The number of nodes to be stored in the queue data structure depends on the branching factor of the tree on a particular tree level. It consumes more memory than a depth-first search. Shown in figure 1.2[8.1]

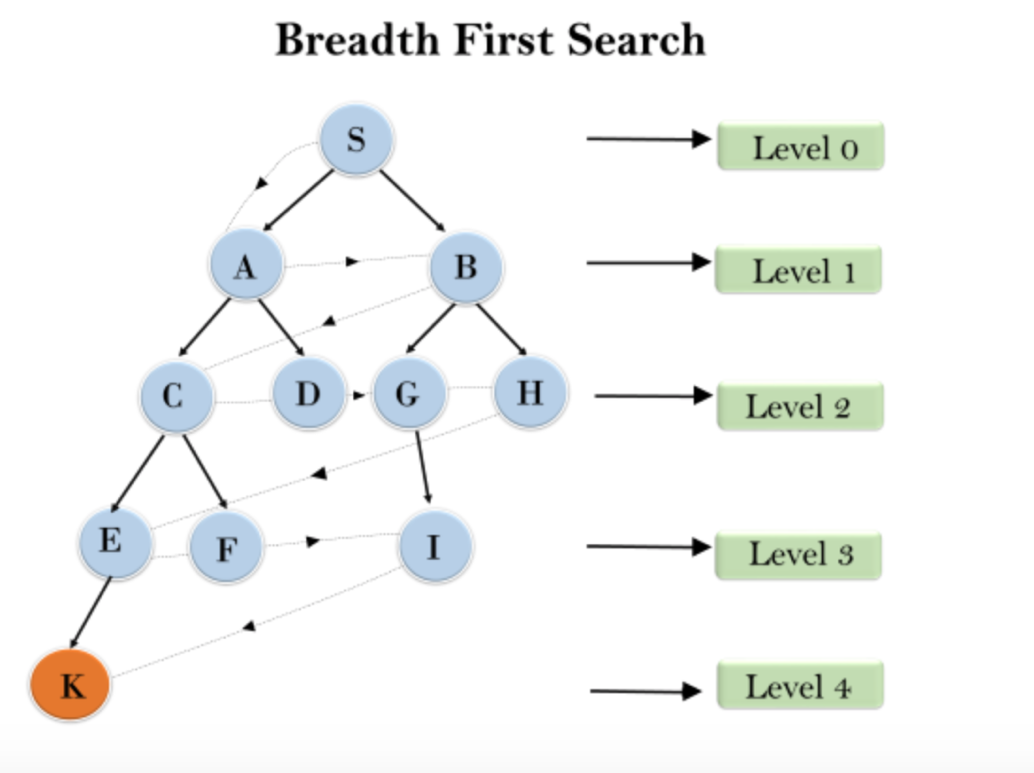


Figure 1.2

1. **Depth Limit Search**: This algorithm is an extension of a depth-first search. This algorithm is implemented in a similar way of depth-first search with a slight difference which is, this will be asking the level of the tree along with the destination. With introducing level, it limits the number of comparisons in the tree which is given in limit. Shown in figure 1.3 with a depth limit as level 2.[8.4]

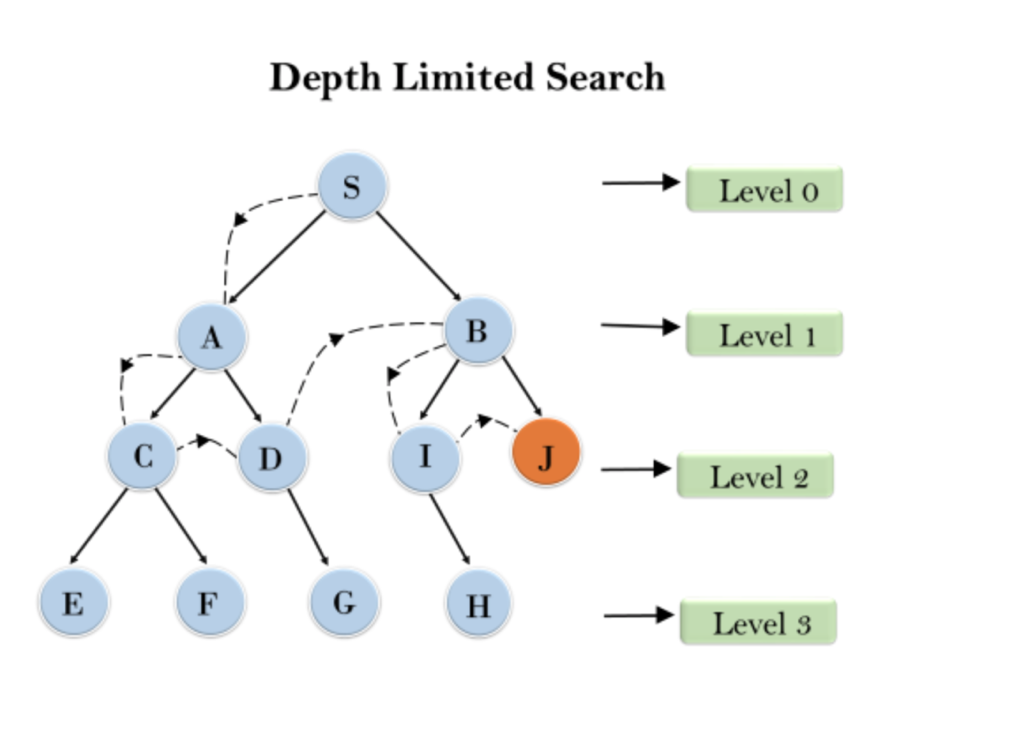


Figure 1.3

1. **Iterative Deepening Depth First Search**: It is a combination of depth-first search and breadth-first search. It selects the benefits of both the algorithms and gives out the best solution compared to previous algorithms. Shown in figure 1.4.[8.4]

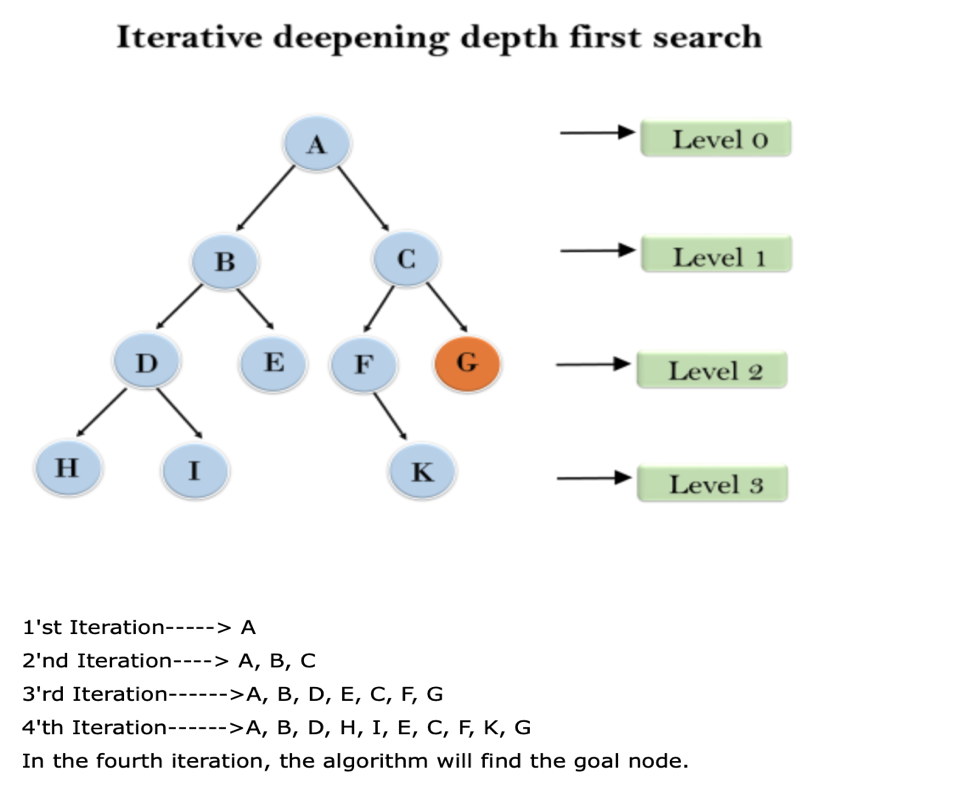


Figure 1.4

1. **Problem Statement**

Searching is carried out to retrieve information stored within the massive data structures, here efficiency becomes a major issue. There exist certain problems which tend to hinder the efficiency of the search algorithms which are taken under consideration in this project. These Problems are: -

1. The searching of the node in DFS algorithm works well but when the target node lies on

the right side of the tree the result will not be optimal.

1. The searching of the node in the BFS algorithm overcomes the problem faced in the DFS algorithm but its memory consumption is much greater than DFS.
2. The searching of a node in DLS algorithm overcomes the problem which is faced in both DFS and BFS algorithm by introducing a limit on the level of search but to find that limit the tree should be solved manually.
3. **Objective**

To perform a comparative study of different performance parameters in Uninformed Search Algorithms listed below to identify and implement the best search approach under different parametric conditions of windows (x32 bit,x64 bit).

1. Breadth-First Search

2. Depth First Search

3. Depth-Limited Search

4. Iterative Deepening Depth First Search

The project aims to overcome the above-stated problems by analysis as well as fair execution. The Iterative Deepening Depth First Search is performed to find the level which then passed to Depth-Limited Search for optimal search

1. **Design**
   1. **Methodology**

The overall study includes:

Step 1. Literature Review.

Step 2. A detailed study of different search algorithms under Uninformed Search.

Step 3. A detailed study of the selected Optimization algorithm.

Step 4. The designing process of flow diagram and algorithms

Step 5. Development and Implementation.

Step 6. Testing

Step 7. Comparative analysis of algorithms on different environments like, windows, Mac, Linux.

Step 8. Report writing

* 1. **Algorithms**
     1. **DFS**

void DFS (tree \*root)

{

If (root! = NULL)

{

Printf ("%d ", root->data);

if(search==root->data)

{

Printf ("\n The Goal node(%d) by DFS algorithm is found.....",search);

Printf ("\n The program is being terminated......");

Exit (0);

}

dfs\_found++;

DFS (root->left);

DFS (root->right);

}

}

* + 1. **BFS**

void enqueue (tree \*root) // \* Queue-ADDING ELEMENTS \*

{

If (root! = NULL)

{

rear++;

queue[rear]=(uintptr\_t)(root);

}

}

void dequeue () // \* DEQUEUE-REMOVING ELEMENTS\*

{

if(rear>=front)

{

tree \*root=(tree \*)(intptr\_t)queue[front];

printNode(root);

front++;

enqueue(root->left);

enqueue(root->right);

}

}

void BFS (tree \*root) // \* BFS Function \*

{

If (root! =NULL)

{

enqueue(root);

do

{

Dequeue ();

} while (front<=rear);

}

}

* + 1. **DLS**

void DLS(searchtree \*root,int8\_t i,int8\_t limit)

{

if(root!=NULL)

{

while(i<limit)

{

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

if(search==root->data)

{

printf("The Goal node(%d) by DLS algo is found.....",search);

printf("\nThe program is being terminated......");

dls\_found=1;

exit(0);

}

DLS(root->left,i,limit-1);

DLS(root->right,i,limit-1);

break;

}

}

}

* + 1. **IDDFS**

void IDDFS(searchtree \*root,int8\_t limit)

{

if(root!=NULL)

{

if(limit==1)

{

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

}

if(search==root->data)

{

printf("\nThe Goal node(%d) is found by IDDFS Algorithm..... ",search);

printf("\nThe program is being terminated.....");

ids\_found=1;

exit(0);

}

else if(limit>1)

{

IDDFS(root->left,limit-1);

IDDFS(root->right,limit-1);

}

}

}

1. **IMPLEMENTATION**
   1. **Pseudo Code & Flow Chart**
      1. **DFS**

Function DFS (struct node \*root):

IF (root! = NULL):

Print (root->data);

IF (search==root->data):

Print (The goal is found);

Exit ();

ENDIF

DFS (root->left);

DFS (root->right);

ENDIF

ENDFUNCTION

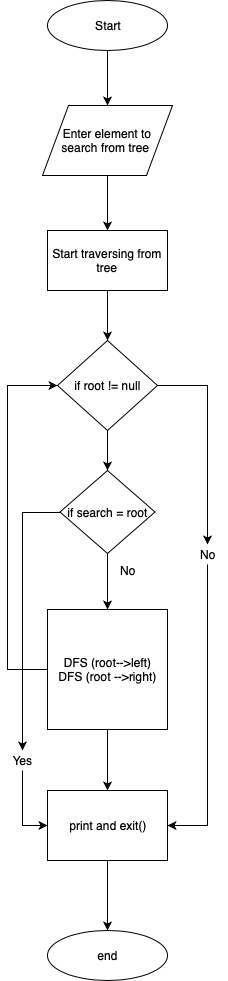


Fig 6.1

* + 1. **BFS**

Function BFS (struct node \*root):

IF (root! = NULL):

Enqueue (root); // Adding elements into the queue

DO:

Dequeue (root); // Removing the elements into the queue

WHILE (front <= rear)

ENDIF

ENDFUNCTION

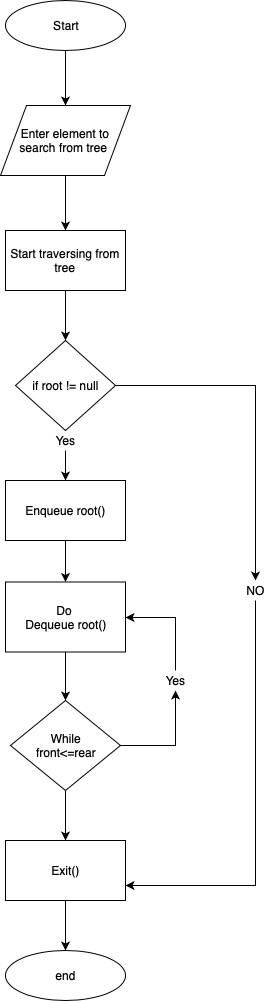
****

Fig 6.2

* + 1. **DLS**

Function DLS (struct node \*root,int8\_t i, int8\_t limit ):

IF (root! = NULL):

WHILE(i <limit):

Print (root->data);

IF (search==root->data):

Print (The goal is found);

Exit ();

ENDIF

DLS (root->left, i, limit-1);

DLS (root->right, i, limit-1);

BREAK;

ENDWHILE

ENDIF

ENDFUNCTION

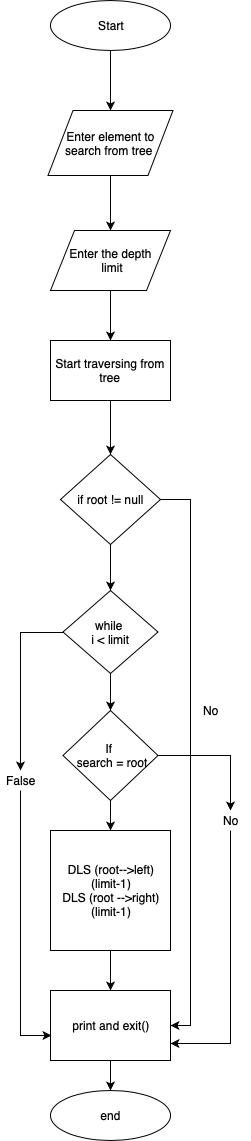
****

Fig 6.3

* + 1. **IDDFS**

Function IDDFS (struct node \*root,int8\_t limit):

IF (root! = NULL):

IF (limit == 1):

Printf(“%d”, root->data);

ENDIF

IF (search == root->data)

Printf(“The goal is found”);

ENDIF

ELSEIF(limit>1)

IDDFS(root->left,limit-1);

IDDFS(root->left,limit-1);

ENDIF

ENDFUNCTION

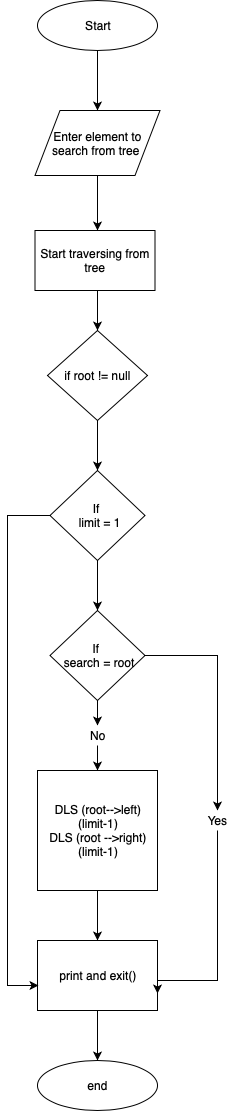
****

Fig 6.4

* 1. **SWAT Analysis**
     1. **DFS**

|  |  |
| --- | --- |
| **INTERNAL FACTORS** | |
| **STRENGTHS (+)** | **WEAKNESSES (–)** |
| 1. The number of nodes to be stored in memory is comparatively less.  2. Less number of nodes are stored so less memory is required.  3. The result obtained through this algorithm is optimized in most of the cases.  4. The algorithm is implemented through the stack. | 1. In some cases, the optimized result is not being obtained through this algorithm.  2. The algorithms start visiting the left child of the tree and explore them and then come to the right child. |
| **EXTERNAL FACTORS** | |
| **OPPORTUNITIES (+)** | **THREATS (–)** |
| 1. The algorithm can be modified in such a way that the control moves to the right child as soon as it visits the left child of the tree. | 1. The algorithm gives un-optimized result when the required node lies on the right child of the tree.  2. When the tree is infinite level(assume) then this algorithm can’t provide the result. |

Table 6.1

* + 1. **BFS**

|  |  |
| --- | --- |
| **INTERNAL FACTORS** | |
| **STRENGTHS (+)** | **WEAKNESSES (–)** |
| 1. The searching style is improvised in this algorithm.  2. This algorithm will always produce the optimized result according to the implementation.  3. The algorithm visits right child immediately after visiting the right child of the tree.  4. The algorithm is implemented through the Queue. | 1. The number of nodes to be stored is increased which will be depending on the branching factor of the tree.  2. The increase in the number of nodes requires more memory. |
| **EXTERNAL FACTORS** | |
| **OPPORTUNITIES (+)** | **THREATS (–)** |
| 1. The memory usage can be controlled by using this logic in the DFS algorithm. | 1. The way of memory usage in this algorithm is not feasible in real life. |

Table 6.2

* + 1. **DLS**

|  |  |
| --- | --- |
| **INTERNAL FACTORS** | |
| **STRENGTHS (+)** | **WEAKNESSES (–)** |
| 1. The number of nodes to be traversed is less.  2. The whole graph need not be traversed like in-depth and breadth-first search.  3. By introducing a limit to the graph increases the speed of execution. | 1. The main problem is to calculate the height of the graph manually.  2. The problem should be solved before to execute depth limit search. |
| **EXTERNAL FACTORS** | |
| **OPPORTUNITIES (+)** | **THREATS (–)** |
| 1. The algorithm must be improved, such that it shouldn’t be needed to solve manually before running the program. | 1. The algorithm doesn’t make easy for the user instead it forces to solve the problem by himself before running program. |

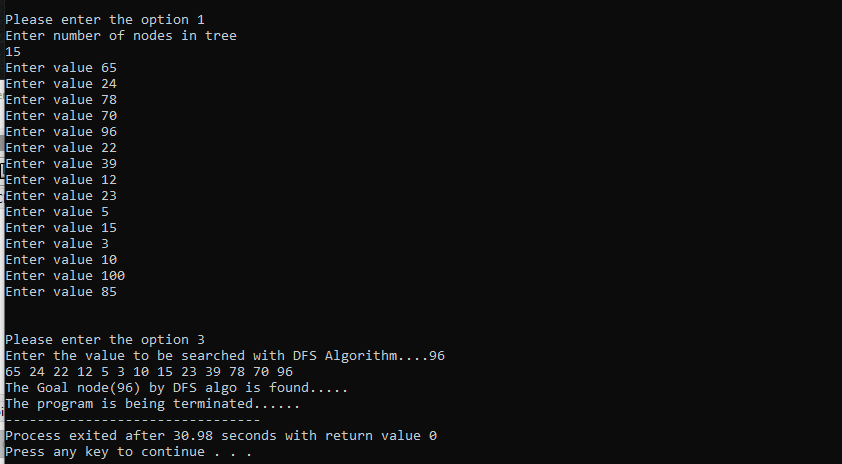
Table 6.3

* + 1. **IDDLS**

|  |  |
| --- | --- |
| **INTERNAL FACTORS** | |
| **STRENGTHS (+)** | **WEAKNESSES (–)** |
| 1. It is a composition of both depth and breadth-first search.  2. This algorithm has overcome the weakness of depth limit search by incrementing the height from start.  3. It takes on advantages of depth and breadth limit search along with the limit of height. | 1. It re-traverses each node repeatedly, after incrementing the height. |
| **EXTERNAL FACTORS** | |
| **OPPORTUNITIES (+)** | **THREATS (–)** |
| 1. The re-traversal of nodes can be avoided which could improvise the search speed. | 1. The repeated traversal of same node is not appreciated.  2. This algorithm will not give optimum result for small size graph. |

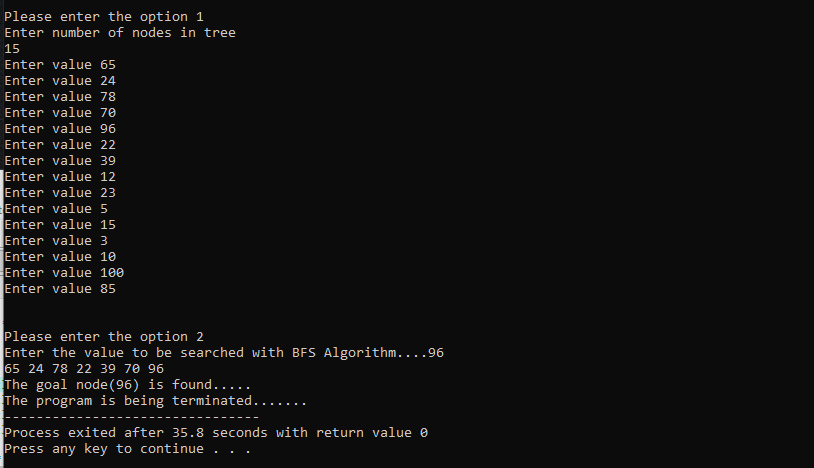
Table 6.4

* 1. **Output Screen**
     1. **Windows (x64 bit)**
        1. **DFS**



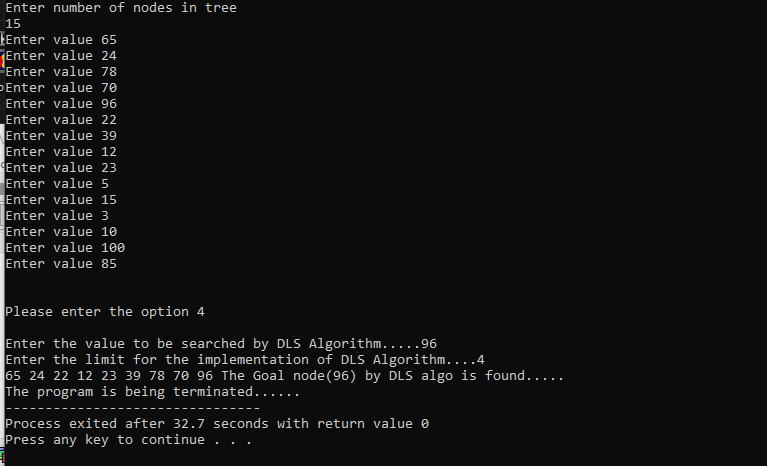
(Fig 6.5)

* + - 1. **BFS**



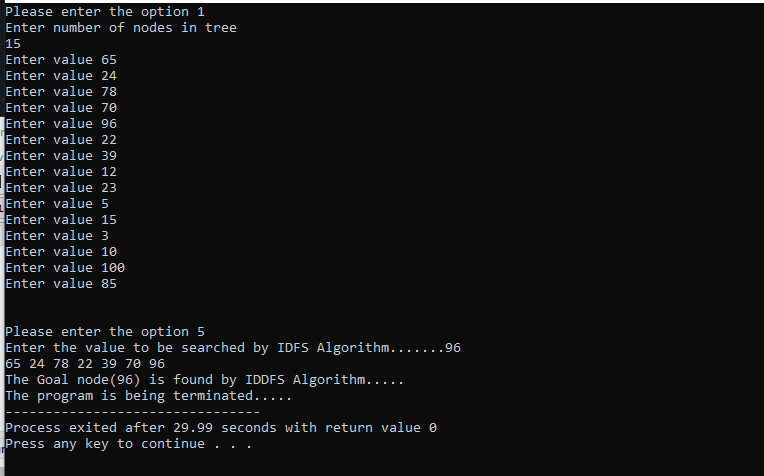
(Fig 6.6)

* + - 1. **DLS**



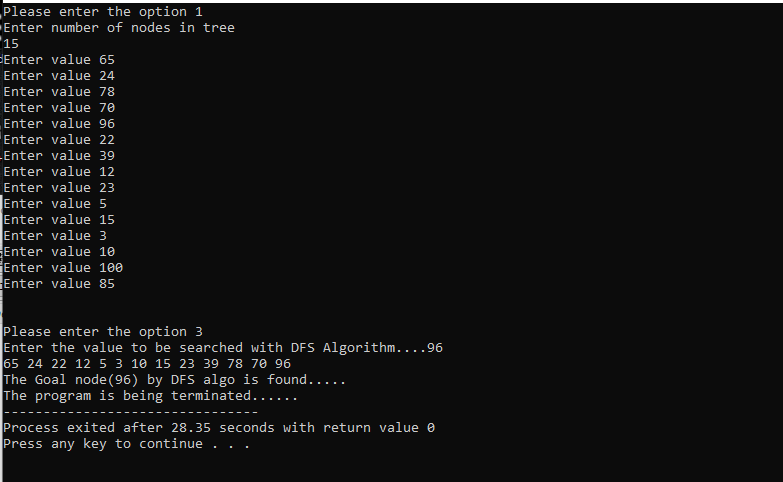
(Fig 6.7)

* + - 1. **IDDFS**



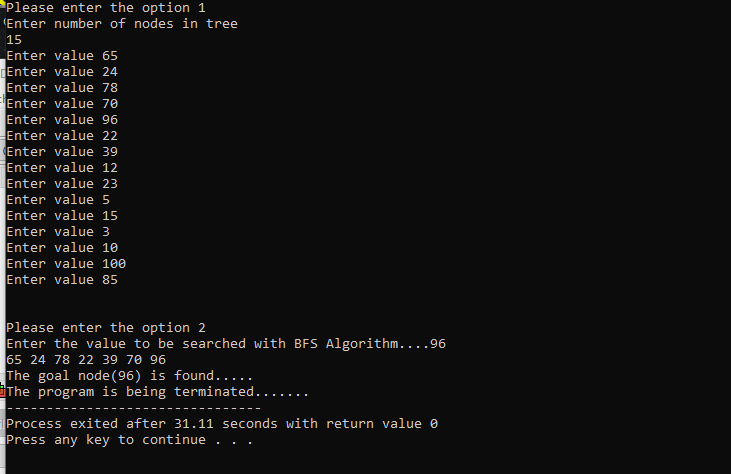
(Fig 6.8)

* + 1. **Windows (x32 bit)**
       1. **DFS**



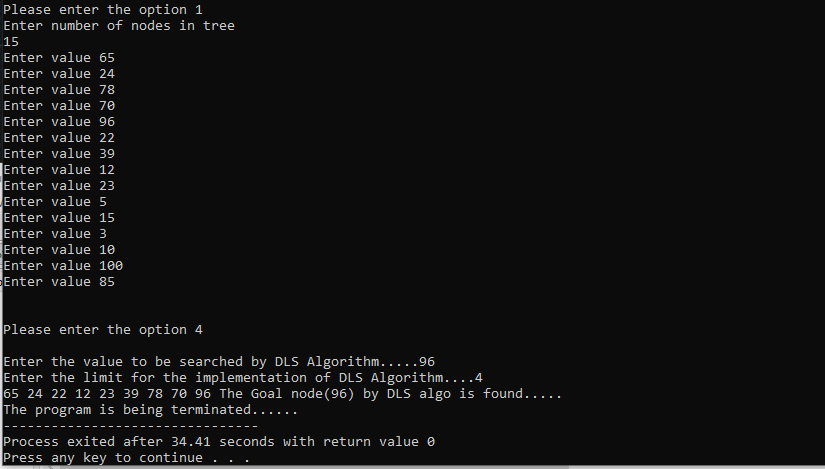
(Fig 6.9)

* + - 1. **BFS**



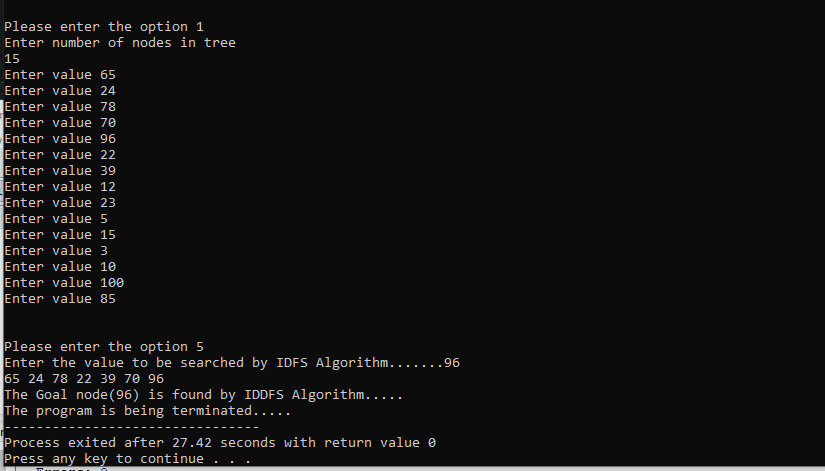
(Fig 6.10)

* + - 1. **DLS**



(Fig 6.11)

* + - 1. **IDDFS**



(Fig 6.12)

* 1. **Result Analysis**

**Case 1: -**When the search algorithms are executed in x64 bits of windows the first set of the result are obtained. For the total 15 entered values if the value to be searched is ‘96’ the time taken by respective algorithm decreases as we move from BFS to DLS to DFS and finally to IDDFS.

**Case 2**: - When the same search algorithms are executed in x32 bits of windows the second set of the result are obtained. For the total 15 entered values if the value to be searched is same as case 1 i.e ‘96’ to provide the same parameter the time taken by respective algorithm again decreases as we move from DLS to BFS to DFS and finally to IDDFS.

After studying the above stated two case we can conclude that the time taken by IDDFS in searching through the tree is always less as compared to the other algorithm. Thus proving us with an optimized search algorithm to traverse through the tree in minimum period.

1. **Conclusion and Future Scope**

We achieved optimization from Depth-First search to Iterative Deepening Depth-First search while taking consideration each algorithm time complexities and found the minimum time search algorithm. This algorithm can be used in many places like: -

1. The algorithm can be used effectively for crawlers in the search engine.
2. The algorithm can also be applied in peer to peer networks (Minimum spanning tree).
3. They can be used for Broadcasting in networks.
4. The algorithm can be used for social websites and GPS systems.
5. **Appendix I Project Code**

/\*

\* IMPLEMENTATION OF DFS BFS DLS and IDFS ALGORITHMS \*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h> // We have imported inttypes to lessen the memory usage as much as possible

#include <stdint.h> // This is imported to type cast the integer to pointer conversion which vary in 32 bit and 64 bit

// \* Variable Declarations \*

int queue[10];

int8\_t front=0,rear=-1;

int search;

static int8\_t n=0,dfs\_found=0,dls\_found=0,ids\_found=0;

static int8\_t temp=0;

// \* Structure Declaration \*

typedef struct node

{

int data;

struct node \*left;

struct node \*right;

}searchtree;

// \* Function Prototype \*

void printNode(searchtree \*root); // We have named the structure with name "searchtree" and will be using this in place of "struct node"

void enqueue(searchtree \*root);

void dequeue();

void BFS(searchtree \*root);

void DFS(searchtree \*root);

void DLS(searchtree \*root,int8\_t i,int8\_t limit);

void IDDS(searchtree \*root,int8\_t limit);

void height(searchtree \*root);

// \* Construction of tree \*

searchtree \*insert(searchtree \*root, int value) // \* Insertion of Nodes value to the Tree \*

{

if(root==NULL)

{

root=(searchtree\*)malloc(sizeof(searchtree));

root->left=root->right=NULL;

root->data=value;

return root;

}

else

{

if(value<=root->data)

{

root->left=insert(root->left,value);

}

else

{

if(value>root->data)

{

root->right=insert(root->right,value);

}

}

return root;

}

};

// \* Printing of Nodes for BFS search \*

void printNode(searchtree \*root) // \* Prints the values of node \*

{

static int bfs\_found=0;

if(root!=NULL)

{

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

if(root->data==search)

{

printf("\nThe goal node(%d) is found.....",search);

printf("\nThe program is being terminated.......");

exit(0);

}

bfs\_found++;

}

if(bfs\_found==temp)

{

printf("\nThe goal node(%d) is NOT found.....",search);

printf("\nYou can try again:)");

bfs\_found=0;

}

}

// \* Queue \*

void enqueue(searchtree \*root) // \* Adding the element into the Queue \*

{

if(root!=NULL)

{

rear++;

queue[rear]=(uintptr\_t)(root);

}

}

void dequeue() // \* Removing Element from the Queue \*

{

if(rear>=front)

{

searchtree \*root=(searchtree \*)(intptr\_t)queue[front];

printNode(root);

front++;

enqueue(root->left);

enqueue(root->right);

}

}

// \* BFS Function \*

void BFS(searchtree \*root)

{

if(root!=NULL)

{

enqueue(root);

do

{

dequeue();

}while(front<=rear);

}

}

// \* DFS Function \*

void DFS(searchtree \*root)

{

if(root!=NULL)

{

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

if(search==root->data)

{

printf("\nThe Goal node(%d) by DFS algo is found.....",search);

printf("\nThe program is being terminated......");

exit(0);

}

dfs\_found++;

DFS(root->left);

DFS(root->right);

}

}

// \* DLS Function \*

void DLS(searchtree \*root,int8\_t i,int8\_t limit)

{

if(root!=NULL)

{

while(i<limit)

{

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

if(search==root->data)

{

printf("The Goal node(%d) by DLS algo is found.....",search);

printf("\nThe program is being terminated......");

dls\_found=1;

exit(0);

}

DLS(root->left,i,limit-1);

DLS(root->right,i,limit-1);

break;

}

}

}

// \* Retrieval of the height of tree for IDDFS Algorithm \*

int level(searchtree \*root)

{

if(root==NULL)

{

return 0;

}

else

{

int lh=level(root->left);

int rh=level(root->right);

if(lh>rh)

return (lh+1);

else

return (rh+1);

}

}

// \* IDDFS Function \*

void IDDFS(searchtree \*root,int8\_t limit)

{

if(root!=NULL)

{

if(limit==1)

{

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

}

if(search==root->data)

{

printf("\nThe Goal node(%d) is found by IDDFS Algorithm..... ",search);

printf("\nThe program is being terminated.....");

ids\_found=1;

exit(0);

}

else if(limit>1)

{

IDDFS(root->left,limit-1);

IDDFS(root->right,limit-1);

}

}

}

// \* Main function \*

int main()

{

searchtree \*root=NULL;

int8\_t select,limit,height;

int i;

int insertnode;

printf("Enter your choice\n1.Enter values\n2.BFS SEARCH\n3.DFS SEARCH\n4.DLS SEARCH\n5.IDDFS SEARCH\n6.Exit");

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

while(1)

{

printf("\n\nPlease enter the option ");

scanf("%d",&select);

switch(select)

{

case 1:

printf("Enter number of nodes in tree \n");

scanf("%d",&n);

temp=temp+n;

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

{

printf("Enter value ");

scanf("%d",&insertnode);

root=insert(root, insertnode);

}

break;

case 2:

printf("Enter the value to be searched with BFS Algorithm....");

scanf("%d",&search);

BFS(root);

break;

case 3:

printf("Enter the value to be searched with DFS Algorithm....");

scanf("%d",&search);

DFS(root);

if(dfs\_found==temp)

{

printf("\nThe goal node(%d) is NOT found by DFS Algorithm.....",search);

printf("\nYou can try again:)");

dfs\_found=0;

}

break;

case 4:

printf("\nEnter the value to be searched by DLS Algorithm.....");

scanf("%d",&search);

printf("Enter the limit for the implementation of DLS Algorithm....");

scanf("%d",&limit);

DLS(root,0,limit);

if(dls\_found==0)

{

printf("\nThe goal node(%d) is NOT found by DLS Algorithm.....",search);

printf("\nYou can try again:)");

dls\_found=0;

}

break;

case 5:

printf("Enter the value to be searched by IDFS Algorithm.......");

scanf("%d",&search);

height=level(root); // \* Height is been retrieved through this function

for(i=1;i<=height;i++)

{

IDDFS(root,i);

}

if(ids\_found==0)

{

printf("\nThe goal node(%d) is NOT found by IDDFS Algorithm.....",search);

printf("\nYou can try again:)");

ids\_found=0;

}

break;

case 6:

printf("Ending the program.......");

exit(0);

default:

printf("\* ERROR CASE...... PLEASE TRY AGAIN.....\*");

break;

}

}

return 0;

}

1. **Reference**
   1. Artificial Intelligence, spring 2013, Peter Ljungl¨of; based on AIMA Slides c Stuart Russel and Peter Norvig, 2004.
   2. Columbia University, New York / CS /lectures /Uninformed search
   3. University of Pennsylvania / CIS /lectures/Uninformed search
   4. George F Luger. 2008. Artificial Intelligence Structures and Strategies for Complex Problem Solving. 6 th Ed. Pearson. ISBN: 0321545893.
   5. <https://www.javatpoint.com/ai-uninformed-search-algorithms>
   6. <http://intelligence.worldofcomputing.net/ai-search/depth-first-search>
   7. Artificial Intelligence by Saroj Kaushik, Cengage.