EXPERIMENT-04

Aim: Implement any uniformed searching technique - BFS & IDS in any programming language.

Theory:

Searching algorithms are one of the most important areas in areas in artificial intelligence since there can be more that one solution to a problem these agents search space for all combinations and was approaches to find the shortest path or a suitable way to reach the final goal. Hence search plays a key role in A I. It consists of various search algorithm which are used in process of problem solving & finding a solution, how agents perceive the world & make assumptions.

properties of search algorithms.

- complete: A search algorithm is termed complete as it ensures a return of solution that exists for any random input.

- optimal
It helps deliver optimal solutions. as it to provide the best solution among all the available alternatives.

- Time complexity: It is the measure of time taken to complete a task by a search algorithm. as well as the maximum number of nodes created.

Sunderam

FOR EDUCATIONAL USE

Uninformed search	Informed search
1) Uninformed search does not require any additional information to reach solution	1) Informed search requires and knowledge to traverse & find the solution.
2) Other name: Blind / Brute search	2) Other name heuristic search
3) Compared to inform the search, the cost of problem solving is high here.	3) This cost of problem solving is low or optimal issue.
4) It is comparatively less efficient.	4) Highly efficient as it is time & cost effective.
5) Can handles large search problems.	5) ~~Cannot~~ Cannot handle large search problems.
6) Slower than an informed search in finding a solution & uses more computation	6) finds solutions quickly & uses less computation
7) categorized as - Breadth first search - Uniform cost search - Depth first search - Iterative deepening	7) categorized into - greedy search - A tree search - A graph search

Sundaram
FOR EDUCATIONAL USE

Criterian	Breadth first search	uniform last	Depth first	Depth limited	Iterative Deepening
Complete	yes	yes	No	NO	yes
optimal	yes	yes	NO	NO	yes
last	yes	yes			
time	$O(b^a)$	$O(b^{1+[c+1\varepsilon]})$	$O(b^m)$	$O(b^2)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+[c-\varepsilon]})$	$O(b^a)$	$O(b^2)$	$O(b^d)$

Conclusion:

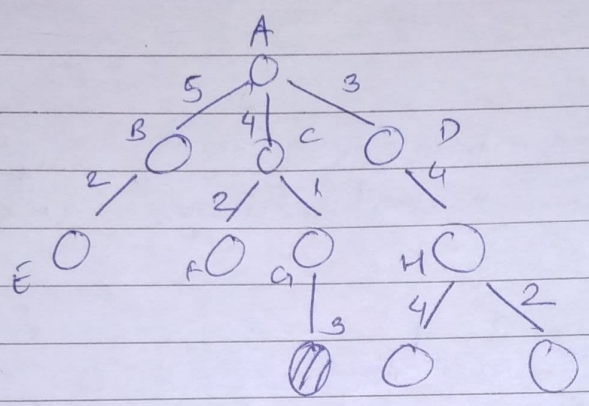
In this experiment we learned about different uninformed searching techniques & compared it with informed searching.

Sundaram®

FOR EDUCATIONAL USE

Assignment 1b

Q.



1) BFS

Start from root node A and traverse its neighbouring nodes since its level 0 it does not have any neighbours but we add it to queue.

| A | | | | |

Queue

display : A

Next we traverse to level 1 and traverse B and its neighbour.

| B | C | D | | |

display : A

Step 2: Now we reach level 2, so we traverse E & its neighbour and remove B from queue and display.

| C | D | E |

Queue.

display : AB

Step 3: Now, we traverse F & neighbour G and add it queue and remove and display C node.

| D | E | F | G |

display : ABC

Step 4: | E | F | G | H |

display : ABCD

Step 5: level 3

| F | G | H |

display : ABCDE

Step 6 : | G | H |

display : ABCDEF

Step 7:

| H | I |

display : ABCDEFG

Step 8:

| I |

display : ABCDEFGH

As I is Goal state, we stop process

∴ Final path :-

ABCDEFGH.

2) DFS:

Step 1: Add root to stack.

display : —
stack : A

Step 2: Add all neighbours of A to stack and print A.

display : A
stack : BCD ← top.

Step 3: Print D and add neighbours to stack.

display : AD
stack : BCH ← top.

Step 4: Repeat above procedure.

display : ADH
stack : BCJK ← top.

Step 5:

display : ADHK
stack : B C J ← top.

Step 6: display : ADHKJ
stack : BC ← top.

Step 7:

display : A D H K J C
stack : B F G ← top.

Step 8:

display : A D H K J C G
stack : B F

reached goal state so final path:
A → D → H → K → J → C → G → (I).

* Depth Limited Search:

Step 1: Add A to stack
display : —
stack : A

Step 2 :

display : A
stack : D C B ← top

Step3 : child of B is leaf nod and not goal state
display : A B
stack : D C ← top.

Step 4:

display : A B C
stack : D G ← top :: F is leaf node & not goal state.

Step 5:

display: A B C G

stack: D I ← top

)

Step 6:

display: A B C G I

stack: D

∴ Final path is:

$A \rightarrow B \rightarrow C \rightarrow G \rightarrow I$.

* Uniform Cost Search:

Step 1: Start from root and add to priority Queue.

| A | | |

P. Queue.

display: —

Step 2:

| B | C | D |

P. Queue.

display: A

Step 3:

| B | C | H |

P. Queue.

display: A D

Step 4:

| B | C | J | K |

P. Queue.

display: A D H.

Step 5:

| B | C | J |

display : A D H K

Step 6:

| B | C |

display : A D H K J

Step 7:

| B | F | G |

display : A D H K J C

Step 8:

| B | F | I |

display : A D H K J C G

Step 9:

Reached goal state

A → D → H → K → J → C → G → (I).

* Iterative Depth Search:

The IDDFS calls DFS for different depths.

Depth	Iterative Deepining Depth First Search
0	A
1	A B C D
2	A B E C F G D H
3	A B E C F G I D H J K

∴ Final path is : A → B → E → C → F → G → (I).

AI – Experiment 4: Program on uninformed search methods

Code :

```
import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
public class BFS
{ private Queue<Integer> queue;
  public BFS()
  {queue = new LinkedList<Integer>();          }
  public void bfs(int adjacency_matrix[][], int source)
  {   int number_of_nodes = adjacency_matrix[source].length - 1;
    int[] visited = new int[number_of_nodes + 1];
    int i, element;
    visited[source] = 1;
    queue.add(source);
    while (!queue.isEmpty())
    { element = queue.remove();
      i = element;
      System.out.print(i + "\t");
      while (i <= number_of_nodes)
      {       if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
        {     queue.add(i);
          visited[i] = 1;}
        i++;  }         }         }
  public static void main(String... arg)
  {   int number_no_nodes, source;
    Scanner scanner = null;
    try
    { System.out.println("Enter the number of nodes in the graph");
      scanner = new Scanner(System.in);
      number_no_nodes = scanner.nextInt();
      int adjacency_matrix[][] = new int[number_no_nodes + 1][number_no_nodes + 1];
      System.out.println("Enter the adjacency matrix");
      for (int i = 1; i <= number_no_nodes; i++)
        for (int j = 1; j <= number_no_nodes; j++)
          adjacency_matrix[i][j] = scanner.nextInt();
      System.out.println("Enter the source for the graph");
      source = scanner.nextInt();
      System.out.println("The BFS traversal of the graph is ");
      BFS bfs = new BFS();
      bfs.bfs(adjacency_matrix, source);
    } catch (InputMismatchException inputMismatch)
    { System.out.println("Wrong Input Format");       }
    scanner.close();   }         }
```

Output: Using cmd