**Heuristic Search**

* **Aim:**

Use heuristic search techniques to implement best first search and A\* algorithm.

* **Software Requirements:**

NetBeans IDE

* **Program Execution:**

**Using BFS**

run:

Enter No of nodes : 5

Enter the name of node 1 : A

Enter the name of node 2 : B

Enter the name of node 3 : C

Enter the name of node 4 : D

Enter the name of node 5 : E

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : B

Enter distance between nodes A and B : 3

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : C

Enter distance between nodes A and C : 1

Do you want to add any adjacent node to node A : n

Do you want to add any adjacent node to node B : y

Enter the name of adjacent node of B : D

Enter distance between nodes B and D : 3

Do you want to add any adjacent node to node B : y

Enter the name of adjacent node of B : E

Enter distance between nodes B and E : 2

Do you want to add any adjacent node to node B : n

Do you want to add any adjacent node to node C : n

Do you want to add any adjacent node to node D : n

Do you want to add any adjacent node to node E : n

A : (B,3), (C,1)

B : (D,3), (E,2)

C :

D :

E :

Priority queue contents :

A

C B

B

B

E D

D

D

Goal node 'D' found

Path :

A, B, D

BUILD SUCCESSFUL (total time: 1 minute 8 seconds)

**Using A\* algorithm**

run:

Enter No of nodes : 4

Enter the name of node 1 : A

Enter the heuristic value of node A : 6

Enter the name of node 2 : B

Enter the heuristic value of node B : 4

Enter the name of node 3 : C

Enter the heuristic value of node C : 3

Enter the name of node 4 : D

Enter the heuristic value of node D : 1

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : B

Enter distance between nodes A and B : 1

Do you want to add any adjacent node to node A : y

Enter the name of adjacent node of A : C

Enter distance between nodes A and C : 3

Do you want to add any adjacent node to node A : n

Do you want to add any adjacent node to node B : y

Enter the name of adjacent node of B : D

Enter distance between nodes B and D : 2

Do you want to add any adjacent node to node B : n

Do you want to add any adjacent node to node C : y

Enter the name of adjacent node of C : D

Enter distance between nodes C and D : 5

Do you want to add any adjacent node to node C : n

Do you want to add any adjacent node to node D : n

A (hx = 6) : (B,1), (C,3)

B (hx = 4) : (D,2)

C (hx = 3) : (D,5)

D (hx = 1) :

Fx of node A = 6

Open List : A

Closed List : Empty

Open List : Empty

Closed List : A

Fx of node B = 5

Fx of node C = 6

Open List : B C

Open List : C

Closed List : A B

Fx of node D = 4

Open List : D C

Open List : C

Closed List : A B D

Path :

A, B, D

Goal node 'D' found

BUILD SUCCESSFUL (total time: 1 minute 37 seconds)

**Eight Puzzle Problem**

* **Aim:**

Solve 8-puzzle problem using A\* algorithm. Assume any initial configuration and define goal configuration clearly.

* **Software Requirements:**

NetBeans IDE

* **Program Execution:**

run:

Enter start Board :

Enter one tile as '-' ie. Blank tile

Enter the value of tile [0][0] : -

Enter the value of tile [0][1] : a

Enter the value of tile [0][2] : c

Enter the value of tile [1][0] : h

Enter the value of tile [1][1] : b

Enter the value of tile [1][2] : d

Enter the value of tile [2][0] : g

Enter the value of tile [2][1] : f

Enter the value of tile [2][2] : e

The given start board is :

- a c

h b d

g f e

Enter goal Board :

Enter one tile as '-' ie. Blank tile

Enter the value of tile [0][0] : a

Enter the value of tile [0][1] : b

Enter the value of tile [0][2] : c

Enter the value of tile [1][0] : h

Enter the value of tile [1][1] : -

Enter the value of tile [1][2] : d

Enter the value of tile [2][0] : g

Enter the value of tile [2][1] : f

Enter the value of tile [2][2] : e

The given goal board is :

a b c

h - d

g f e

The board is solved as :

Board after 0 moves :

- a c

h b d

g f e

Possible moves are :

For Fn = 3 :

a - c

h b d

g f e

For Fn = 5 :

h a c

- b d

g f e

Board after 1 moves :

a - c

h b d

g f e

Possible moves are :

For Fn = 5 :

- a c

h b d

g f e

For Fn = 5 :

a c -

h b d

g f e

For Fn = 2 :

a b c

h - d

g f e

Board after 2 moves :

a b c

h - d

g f e

Goal state achieved.

BUILD SUCCESSFUL (total time: 36 seconds)

**Medical Expert System**

* **Aim:**

Implement expert system for medical diagnosis of diseases based on adequate symptoms.

* **Software Requirements:**

SWI-Prolog for Windows, Editor.

* **Theory:**

A system that uses human expertise to make complicated decisions. Simulates reasoning by applying knowledge and interfaces. Uses expert’s knowledge as rules and data within the system. Models the problem solving ability of a human expert.

Components of an ES:

1. Knowledge Base
2. Represents all the data and information imputed by experts in the field.
3. Stores the data as a set of rules that the system must follow to make decisions.
4. Reasoning or Inference Engine
5. Asks the user questions about what they are looking for.
6. Applies the knowledge and the rules held in the knowledge base.
7. Appropriately uses this information to arrive at a decision.
8. User Interface
9. Allows the expert system and the user to communicate.
10. Finds out what it is that the system needs to answer.
11. Sends the user questions or answers and receives their response.
12. Explanation Facility
13. Explains the systems reasoning and justifies its conclusions.

* **Program Execution:**

?- go.

Does the patient has the symptom headache? : y.

Does the patient has the symptom runny\_nose? : |: n.

Does the patient has the symptom sore\_throat? : |: n.

Does the patient has the symptom abdominal\_pain? : |: y.

Does the patient has the symptom poor\_appetite? : |: y.

Does the patient has the symptom fever? : |: y.

Advices and Sugestions:

1: Chloramphenicol

2: Amoxicillin

3: Ciprofloxacin

4: Azithromycin

Please do complete bed rest and take soft diet because

It is suggested that the patient has typhoid

true .

**ChatBot**

* **Aim:**

Develop elementary chatbot for suggesting investment as per the customer needs.

* **Software Requirements:**

NetBeans, program-ab library.

* **Set-up steps:**

Download the program-ab file.

Copy the bots folder into project folder.

Add the Ab.jar file to the project libraries.

Create the .aiml file in bots/super/aiml folder.

* **Program Execution:**

**Execute a ChatBot Jar file using following command**

**Java –jar dist/ChatBot.jar (Due to execution of jar file, aimlif folder updated automatically)**

**Aimlif contains csv(comma separated files) which are the filed used for early execution**

Human : Hello

Robot : Hi nice to see you!

Human : What are different investment options

Robot : Some of the investment options are Public provident fund Mutual fund Equity shares Real estate investment etc.

Human : I am interested in mutual fund

Robot : Yes we can discuss mutual fund

Human : Minimum investment period

Robot : Only applicable in case of close-ended and ELSS schemes

Human : Let us discuss equity shares

Robot : Yes we can discuss equity shares

Human : Maximum investment amount

Robot : No limit

Human : Bye

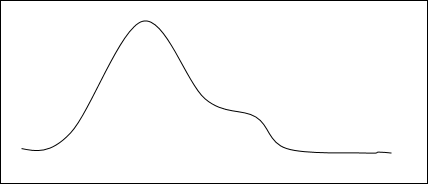
Robot : Goodbye ! It was nice to talk to you

**Hill Climbing Algorithm**

**AIM :** To solve 8 queens problem using Hill Climbing Algorithm.

**Theory :**

Hill Climbing is a technique to solve certain optimization problems. In this technique, we start with a suboptimal solution and the solution is improved repeatedly until some condition is maximized.



The idea of starting with a sub-optimal solution is compared to starting from the base of the hill, improving the solution is compared to walking up the hill, and finally maximizing some condition is compared to reaching the top of the hill.

Hence, the hill climbing technique can be considered as the following phases −

* Constructing a sub-optimal solution obeying the constraints of the problem
* Improving the solution step-by-step
* Improving the solution until no more improvement is possible

Input :

As we are using Random function in java , we will get different input states each time we execute the code.

Queens are placed on the board depending upon the generated random number.

Function for placing Queens:

public void placeQueens()

{ queenPositions = generateQueens();

for (int i = 0; i < board.length; i++) {

board[queenPositions[i]][i] = 1;

}

}

queenPositions[i] is the random number generated.

Therefore, for the following random numbers, the queens will be placed on the board like this :

Random Numbers generated:

7   
3  
5  
7  
1  
5  
1  
1

Corresponding Board:

0 0 0 0 0 0 0 0   
0 0 0 0 1 0 1 1   
0 0 0 0 0 0 0 0   
0 1 0 0 0 0 0 0   
0 0 0 0 0 0 0 0   
0 0 1 0 0 1 0 0   
0 0 0 0 0 0 0 0   
1 0 0 1 0 0 0 0

**Goal Stack Planning**

**Title :** Implement goal stack planning for the following configurations from the blocks world.

**Aim:** To implement goal stack planning for given the blocks world configurations.

**Requirement:** Java Editor (Eclipse/Net beans)

**Theory:**

Planning is process of determining various actions that often lead to a solution.

Planning is useful for non-decomposable problems where subgoals often interact.

Goal Stack Planning (in short GSP) is the one of the simplest planning algorithm that is designed to handle problems having compound goals. And it utilizes STRIP as a formal language for specifying and manipulating the world with which it is working.

This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order.

**Algorithm:**

Push the Goal state in to the Stack

Push the individual Predicates of the Goal State into the Stack

Loop till the Stack is empty

Pop an element E from the stack

IF E is a Predicate

IF E is True then

Do Nothing

ELSE

Push the relevant action into the Stack

Push the individual predicates of the Precondition of the action into the Stack

Else IF E is an Action

Apply the action to the current State.

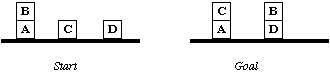
Add the action ‘a’ to the plan

The Goal Stack Planning Algorithms works will the stack. It starts by pushing the unsatisfied goals into the stack. Then it pushes the individual subgoals into the stack and its pops an element out of the stack. When popping an element out of the stack the element could be either a predicate describing a situation about our world or it could be an action that can be applied to our world under consideration. So based on the kind of element we are popping out from the stack a decision has to be made. If it is a Predicate. Then compares it with the description of the current world, if it is satisfied or is already present in our current situation then there is nothing to do because already its true. On the contrary if the Predicate is not true then we have to select and push relevant action satisfying the predicate to the Stack.

So after pushing the relevant action into the stack its precondition should also has to be pushed into the stack. In order to apply an operation its precondition has to be satisfied. In other words the present situation of the world should be suitable enough to apply an operation. For that, the preconditions are pushed into the stack once after an action is pushed.

**Input:**

Consider the following where wish to proceed from the start to goal state.



No of Blocks : 4

Initial stage : (on b a)^(ontable c)^(ontable a)^(ontable d)^(clear b)^(clear c)^(clear d)^(AE)

Final stage : (on c a)^(on b d)^(ontable a)^(ontable d)^(clear c)^(clear b)^(AE)

**Output:**

Set of actions to be taken:

1. (unstack b d)
2. (stack b d)
3. (pick c)
4. (stack c a)