**LIST OF PROGRAMS**

1. Graph Coloring
2. Water Jug Problem using BFS and DFS
3. Heuristic Algorithms - A\* algorithm
4. Best First Search(BFS)
5. Unification
6. Minmax Algorithm
7. Blocks word problem
8. Prepositional Logic
9. Forward chain and backward chain
10. Development of Expert System
11. Expert System for Medical Diagnosis
12. Count the Vowels in a List
13. Guess the word

**Graph Coloring Problem**

**AIM:**

To develop a program for Graph Coloring Problem

**ALGORITHM**

1. Color first vertex with first color.

2. Do following for remaining V-1 vertices.

3. Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it.

4. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

**SOURCE CODE:**

**/\*\***

\*\* Java Program to Implement Graph Coloring Algorithm

\*\*/

import java.util.Scanner;

/\*\* Class GraphColoring \*\*/

public class GraphColoring

{

private int V, numOfColors;

private int[] color;

private int[][] graph;

/\*\* Function to assign color \*\*/

public void graphColor(int[][] g, int noc)

{

V = g.length;

numOfColors = noc;

color = new int[V];

graph = g;

try

{

solve(0);

System.out.println("No solution");

}

catch (Exception e)

{

System.out.println("\nSolution exists ");

display();

}

}

/\*\* function to assign colors recursively \*\*/

public void solve(int v) throws Exception

{

/\*\* base case - solution found \*\*/

if (v == V)

throw new Exception("Solution found");

/\*\* try all colours \*\*/

for (int c = 1; c <= numOfColors; c++)

{

if (isPossible(v, c))

{

/\*\* assign and proceed with next vertex \*\*/

color[v] = c;

solve(v + 1);

/\*\* wrong assignement \*\*/

color[v] = 0;

}

} }

/\*\* function to check if it is valid to allot that color to vertex \*\*/

public boolean isPossible(int v, int c)

{

for (int i = 0; i < V; i++)

if (graph[v][i] == 1 && c == color[i])

return false;

return true;

}

/\*\* display solution \*\*/

public void display()

{

System.out.print("\nColors : ");

for (int i = 0; i < V; i++)

System.out.print(color[i] +" ");

System.out.println();

}

/\*\* Main function \*\*/

public static void main (String[] args)

{

Scanner scan = new Scanner(System.in);

System.out.println("Graph Coloring Algorithm Test\n");

/\*\* Make an object of GraphColoring class \*\*/

GraphColoring gc = new GraphColoring();

/\*\* Accept number of vertices \*\*/

System.out.println("Enter number of verticesz\n");

int V = scan.nextInt();

/\*\* get graph \*\*/

System.out.println("\nEnter matrix\n");

int[][] graph = new int[V][V];

for (int i = 0; i < V; i++)

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

graph[i][j] = scan.nextInt();

System.out.println("\nEnter number of colors");

int c = scan.nextInt();

gc.graphColor(graph, c);

}

}

**SAMPLE OUTPUT:**

Graph Coloring Algorithm Test

Enter number of vertices

10

Enter matrix

0 1 0 0 0 1 0 0 0 0

1 0 1 0 0 0 1 0 0 0

0 1 0 1 0 0 0 1 0 0

**0 0 1 0 1 0 0 0 1 0**

1 0 0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 1 1 0

0 1 0 0 0 0 0 0 1 1

0 0 1 0 0 1 0 0 0 1

0 0 0 1 0 1 1 0 0 0

0 0 0 0 1 0 1 1 0 0

Enter number of colors

3

Solution exists

Colors : 1 2 1 2 3 2 1 3 3 2

**RESULT:**

Thus the Program for Graph coloring problem was implemented and executed.

**Water Jug Problem using DFS,BFS**

**AIM:**

To develop a program for Water Jug Problem

**Description :**

The Water Jug problem can be stated as follows: “*Given two unmarked jugs having capacities ‘a’ and ‘b’ liters respectively and a target volume ‘t1’ and ‘t2’ liters, find the moves that get exactly ‘t1’ and ‘t2’ liters in the two jugs.”There are* two search procedures namely BFS (Breadth First Search) and DFS (Depth First Search) to work in a way to generate new states (target ie. t1 and t2) from a given state(initial ie. a and b).

The operations you can perform are:

1. Empty a Jug
2. Fill a Jug
3. Pour water from one jug to the other until one of the jugs is either empty or full.

**Procedure :**

**Solution 1 (Always pour from ‘a’ litre jug into ‘b’ litre jug)**

1. Fill the ‘a’ litre jug and empty it into ‘b’ litre jug.
2. Whenever the ‘a’ litre jug becomes empty fill it.
3. Whenever the ‘b’ litre jug becomes full empty it.
4. Repeat steps 1,2,3 till either n litre jug or the m litre jug contains d litres of water.

**Solution 2 (Always pour from ‘b’ litre jug into ‘a’ litre jug)**

1. Fill the ‘b’ litre jug and empty it into ‘a’ litre jug.
2. Whenever the ‘a’ litre jug becomes empty fill it.
3. Whenever the ‘a’ litre jug becomes full empty it.
4. Repeat steps 1, 2 and 3 till either n litre jug or the m litre jug contains d litres of water.

**SOURCE CODE:**

#include<stdio.h>

#include<process.h>

#include<iostream.h>

#include<conio.h>

void proces(int,int);

void bfs();

void disstack();

void add(int,int);

void delet();

void push(int,int);

void pop();

void dstack();

void dfs();

int a,b;

int t1,t2,z,ch,top=-1,backtrack=-1,f=0;

int ch1,x,y;

int front=0,rear=-1;

struct queue

{

int i,j,level;

}s1[40];

struct stack

{

int i,j;

int level;

}s[40];

void main()

{

clrscr();

cout<<"\n\t\t **WATER JUG PROBLEM USING BFS AND DFS";**

cout<<"\n\t\t---------------------------------------------------";

while(ch1!=3)

{

cout<<"\n1.BFS";

cout<<"\n2.DFS";

cout<<"\n3.EXIT";

cout<<"\nEnter your choice: ";

cin>>ch1;

switch(ch1)

{

case 1:

cout<<"\n\t\t\t **WATER JUG PROBLEM USING BFS";**

cout<<"\n\t\t\t----------------------------------------------";

cout<<"\nEnter the goal state:\n";

cin>>t1;

cin>>t2;

x=0;

y=0;

add(0,0);

bfs();

while(backtrack==1)

{

delet();

bfs();

}

break;

case 2:

cout<<"\n\t\**t WATER JUG PROBLEM USING DFS**";

cout<<"\n\t\t------------------------------------------------";

cout<<"\nEnter the Goal state:\n";

cin>>t1;

cin>>t2;

x=0;

y=0;

push(x,y);

dfs();

while(backtrack==1)

{

pop();

dfs();

}

break;

case 3:

cout<<"EXIT";

exit(0);

break;

}

}

getch();

}

void bfs()

{

int flag=0;

while(x!=t1 || y!=t2)

{

x=s1[front].i;

y=s1[front].j;

x=s1[rear].i;

y=s1[rear].j;

if(y<3)

{

y=3;

flag=0;

for(int f=0;f<=front;f++)

{

if(s1[f].i==x && s1[f].j==y)

{

flag=1;

break;

}

}

if(flag==0)

add(x,y);

}

front++;

x=s1[front].i;

y=s1[front].j;

if(x+y<=4 && y>=0)

{

x=x+y;

y=0;

flag=0;

for(int f=0;f<=front;f++)

{

if(s1[f].i==x && s1[f].j==y)

{

flag=1;

break;

}

}

if(flag==0)

add(x,y);

}

front++;

x=s1[front].i;

y=s1[front].j;

if(x+y>=3 && x>=0)

{

y=3;

x=x+y-3;

flag=0;

for(int f=0;f<=front;f++)

{

if(s1[f].i==x&&s1[f].j==y)

{

flag=1;

break;

}

}

if(flag==0)

add(x,y);

}

front++;

x=s1[front].i;

y=s1[front].j;

if(x+y>=4 && y>=0)

{

y=x+y-4;

x=4;

flag=0;

for(int f=0;f<=front;f++)

{

if(s1[f].i==x&&s1[f].j==y)

{

flag=1;

break;

}

}

if(flag==0)

add(x,y);

}

front++;

x=s1[front].i;

y=s1[front].j;

if(x>0)

{

x=0;

flag=0;

for(int f=0;f<=front;f++)

{

if(s1[f].i==x && s1[f].j==y)

{

flag=1;

break;

}

}

if(flag==0)

add(x,y);

}

}

}

void add(int m,int n)

{

rear++;

s1[rear].i=m;

s1[rear].j=n;

s1[rear].level=rear;

if(m==t1 && n==t2)

{

disstack();

backtrack=0;

getch();

}

}

void delet()

{

front--;

}

void disstack()

{

cout<<"\n **SOLUTION**: \n";

cout<<"\n4G,3G\n\n";

for(int e=0;e<=rear;e++)

cout<<"("<<s1[e].i<<","<<s1[e].j<<")\n";

}

void dfs()

{

int flag=0;

while(x!=t1||y!=t2)

{

x = s[top].i;

y = s[top].j;

**// Fill the 4 gallon jug**

if(x < 4)

{

x = 4;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**// Fill the 3 Gallon jug**

if(y < 3)

{

y = 3;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x && s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**// Empty the 4 Gallon jug**

if(x > 0)

{

x = 0;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**// Empty the 3 Gallon jug**

if(y > 0)

{

x = 0;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**// Pour some water from the 4 Gallon jug into 3 Gallon jug until 3 Gallon jug is full**

if(x+y >= 3 && x >= 0)

{

y = 3;

x = x+y-3;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**// Pour water from the 3 Gallon jug into the 4 Gallon jug until 4 Gallon jug is full**

if(x+y >= 4 && y >= 0)

{

y = x+y-4;

x = 4;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**// Pour all water from 4 Gallon jug into the 3 Gallon jug**

if(x+y <= 3 && x >= 0)

{

y = x+y;

x = 0;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

}

}

if(flag==0)

{

push(x,y);

}

}

x = s[top].i;

y = s[top].j;

**//Pour all water from 3 Gallon jug into the 4 Gallon jug**

if(x+y <= 4 && y >= 0)

{

x = x+y;

y = 0;

flag = 0;

for(f=0;f<=top;f++)

{

if(s[f].i==x&&s[f].j==y)

{

flag = 1;

break;

} }

if(flag==0)

{

push(x,y);

}

x = s[top].i;

y = s[top].j;

}

}

}

void push(int m,int n)

{

top++;

s[top].i = m;

s[top].j = n;

s[top].level = top;

if(m==t1 && n==t2)

{

dstack();

backtrack = 0;

getch();

}

}

void pop(void)

{

top--;

}

void dstack()

{

cout<<"\n **SOLUTION**: \n\n";

cout<<"4G,3G\n\n";

for(z=0;z<=top;z++)

{

cout<<"("<<s[z].i<<","<<s[z].j<<")"<<"\n";

}

}

**SAMPLE OUTPUT:**

**WATER JUG PROBLEM USING BFS AND DFS**

--------------------------------------------------------------

1. BFS

2. DFS

3. EXIT

Enter your choice: 1

**WATER JUG PROBLEM USING BFS**

--------------------------------------------------------------

Enter the goal state:

2

3

**SOLUTION:**

4G, 3G

(0, 0)

(0, 3)

(3, 0)

(3, 3)

(4, 2)

(0, 2)

(2, 0)

(2, 3)

**WATER JUG PROBLEM USING BFS AND DFS**

--------------------------------------------------------------

1. BFS

2. DFS

3. EXIT

Enter your choice: 2

**WATER JUG PROBLEM USING DFS**

------------------------------------------------------

Enter the Goal state:

2

3

**SOLUTION:**

4G, 3G

(0, 0)

(4, 0)

(4, 3)

(0, 3)

(3, 0)

(3, 3)

(4, 2)

(0, 2)

(2, 0)

(2, 3)

**RESULT:**

Thus the program for Water Jug Problem was implemented and executed.

**Heuristic Algorithms - A\* algorithm**

**AIM:**

To develop a program to implement A\* Algorithm.

**ALGORITHM:**

1. Start the program by creating the class A star.
2. Get the number of nodes using the scanner class.
3. Get the edges from the number of nodes entered.
4. If the node is visited, it will be indicated as “Reached”.
5. And the estimate distance for each node is entered and the path will be generated from cost matrix.
6. When the estimation cost is entered, the cost matrix will be generated.
7. End the program.

**SOURCE CODE:**

import java.util.Scanner;

public class Astar{

static int INFINITY =99;

static int n,front,rear,v;

static Scanner in = new Scanner(System.in);

static int estimate[]=new int[10];

static int cost[][]=new int[10][10];

static void visit(int v,int n)

{

int i,q,alt;

System.out.print(v + "->");

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

{

v=nearest(v);

System.out.print(v);

if(v == n)

{

System.out.print("Reached \n");

return;

}

System.out.print("->");

}

System.out.println("No Path \n");

}

static int nearest(int v)

{

int i,node, weight;

weight = INFINITY;

node = v;

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

{

if(cost[v][i] == INFINITY) continue;

if(estimate[i] < weight)

{

weight = estimate[i];

node=i;

}

}

return node;

}

static void display()

{

int i,j;

System.out.println("Cost Matrix \n");

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

{

for(j=1;j<=n;j++)

{

System.out.print( cost[i][j] + "\t");

}

System.out.println("\n");

}

}

public static void main(String[] args)

{

**// n = 4**

**// 1->2 = 2**

**// 1->3 = 3**

**// 1->4 = 99**

**// 2->3 = 99**

**// 2->4 = 4**

**// 3->4 = 2**

**//**

**// Enter the Estimate distance of 1 = 5**

**// Enter the Estimate distance of 2 = 4**

**// Enter the Estimate distance of 3 = 2**

**// Enter the Estimate distance of 4 = 0**

int v,u,i,j,k;

System.out.println("Enter the number of nodes \n");

n = in.nextInt();

System.out.println("Press 99 for no edge \n");

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

{

cost[i][i] = 99;

for(j=i+1;j<=n;j++)

{

System.out.println("\nEnter the edge " + i + "->" + j + " value \n");

cost[i][j] = in.nextInt();

cost[j][i] = cost[i][j];

}

}

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

{

System.out.println("\nEnter the Estimate distance of " + i + " ");

estimate[i] = in.nextInt();

}

display();

visit(1,n); // source = 1; destination = n

}

}

**SAMPLE OUTPUT:**

**Enter the number of nodes**

4

**Press 99 for no edge**

Enter the edge 1->2 value

2

Enter the edge 1->3 value

3

Enter the edge 1->4 value

99

Enter the edge 2->3 value

99

Enter the edge 2->4 value

4

Enter the edge 3->4 value

2

Enter the Estimate distance of 1

5

Enter the Estimate distance of 2

4

Enter the Estimate distance of 3

2

Enter the Estimate distance of 4

0

**Cost Matrix**

99 2 3 99

2 99 99 4

3 99 99 2

99 4 2 99

**1->3->4->Reached**

**RESULT:**

Thus the Program for A\* algorithm was implemented and executed

**BEST FIRST SEARCH (BFS)**

**AIM:**

To develop the program for Best First Search

**ALGORITHM:**

1. Start at some node ‘i’. This is now our current node.
2. State that our current node is ‘visited’.
3. Now look at all nodes adjacent to our current node.
4. If we see an adjacent node that has not been ‘visited’, add it to the queue.
5. Then pull out the first node on from the queue and traverse to it.
6. And go back to step 1.

**SOURCE CODE:**

import java.util.Comparator;

import java.util.InputMismatchException;

import java.util.PriorityQueue;

import java.util.Scanner;

public class BestFirstSearch

{

private PriorityQueue<Vertex> priorityQueue;

private int heuristicvalues[];

private int numberOfNodes;

public static final int MAX\_VALUE = 999;

public BestFirstSearch(int numberOfNodes)

{

this.numberOfNodes = numberOfNodes;

this.priorityQueue = new PriorityQueue<Vertex>(this.numberOfNodes,

new Vertex());

}

public void bestFirstSearch(int adjacencyMatrix[][], int[] heuristicvalues,int source)

{

int evaluationNode;

int destinationNode;

int visited[] = new int [numberOfNodes + 1];

this.heuristicvalues = heuristicvalues;

priorityQueue.add(new Vertex(source, this.heuristicvalues[source]));

visited[source] = 1;

while (!priorityQueue.isEmpty())

{

evaluationNode = getNodeWithMinimumHeuristicValue();

destinationNode = 1;

System.out.print(evaluationNode + "\t");

while (destinationNode <= numberOfNodes)

{

Vertex vertex = new Vertex(destinationNode,this.heuristicvalues[destinationNode]);

if ((adjacencyMatrix[evaluationNode][destinationNode] != MAX\_VALUE

&& evaluationNode != destinationNode)&& visited[destinationNode] == 0)

{

priorityQueue.add(vertex);

visited[destinationNode] = 1;

}

destinationNode++;

}

}

}

private int getNodeWithMinimumHeuristicValue()

{

Vertex vertex = priorityQueue.remove();

return vertex.node;

}

public static void main(String... arg)

{

int adjacency\_matrix[][];

int number\_of\_vertices;

int source = 0;

int heuristicvalues[];

Scanner scan = new Scanner(System.in);

try

{

System.out.println("Enter the number of vertices: ");

number\_of\_vertices = scan.nextInt();

adjacency\_matrix = new int[number\_of\_vertices + 1][number\_of\_vertices + 1];

heuristicvalues = new int[number\_of\_vertices + 1];

System.out.println("Enter the Weighted Matrix for the graph: ");

for (int i = 1; i <= number\_of\_vertices; i++)

{

for (int j = 1; j <= number\_of\_vertices; j++)

{

adjacency\_matrix[i][j] = scan.nextInt();

if (i == j)

{

adjacency\_matrix[i][j] = 0;

continue;

}

if (adjacency\_matrix[i][j] == 0)

{

adjacency\_matrix[i][j] = MAX\_VALUE;

}

}

}

for (int i = 1; i <= number\_of\_vertices; i++)

{

for (int j = 1; j <= number\_of\_vertices; j++)

{

if (adjacency\_matrix[i][j] == 1 && adjacency\_matrix[j][i] == 0)

{

adjacency\_matrix[j][i] = 1;

}

}

}

System.out.println("Enter the heuristic values of the nodes: ");

for (int vertex = 1; vertex <= number\_of\_vertices; vertex++)

{

System.out.print(vertex + ".");

heuristicvalues[vertex] = scan.nextInt();

System.out.println();

}

System.out.println("Enter the source: ");

source = scan.nextInt();

System.out.println("The graph is explored as follows ");

BestFirstSearch bestFirstSearch = new BestFirstSearch(number\_of\_vertices);

bestFirstSearch.bestFirstSearch(adjacency\_matrix, heuristicvalues,source);

}

catch (InputMismatchException inputMismatch)

{

System.out.println("Wrong Input Format");

}

scan.close();

}

}

class Vertex implements Comparator<Vertex>

{

public int heuristicvalue;

public int node;

public Vertex(int node, int heuristicvalue)

{

this.heuristicvalue = heuristicvalue;

this.node = node;

}

public Vertex()

{

}

@Override

public int compare(Vertex vertex1, Vertex vertex2)

{

if (vertex1.heuristicvalue < vertex2.heuristicvalue)

return -1;

if (vertex1.heuristicvalue > vertex2.heuristicvalue)

return 1;

return 0;

}

@Override

public boolean equals(Object obj)

{

if (obj instanceof Vertex)

{

Vertex node = (Vertex) obj;

if (this.node == node.node)

{

return true;

}

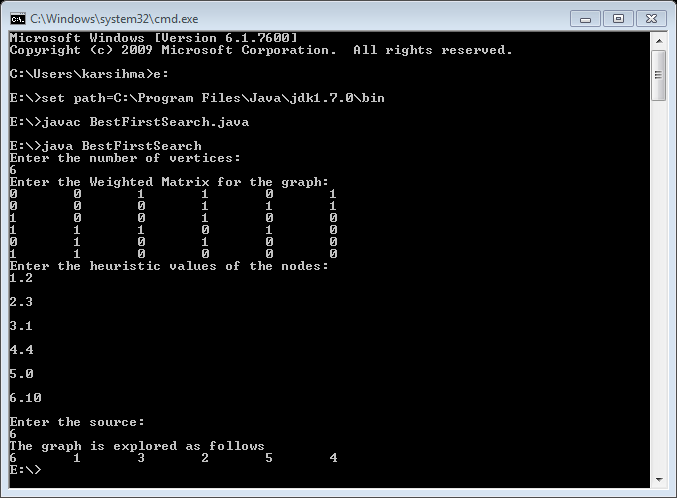
}

return false;

}

}

**OUTPUT:**

****

**RESULT:**

Thus the program for Best First Search was implemented and executed.

**UNIFICATION**

**AIM:**

To Develop a Program for Unification

**ALGORITHM:**

1. Start at some node ‘i’. This is now our predicate.
2. Checking Predicate.
3. Checking arguement.
4. Unify function is applied.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

int no\_of\_pred;

int no\_of\_arg[10];

int i,j;

char nouse;

char predicate[10];

char argument[10][10];

void unify();

void display();

void chk\_arg\_pred();

   void main()

   {

   char ch;

   do{

   clrscr();  
  
             printf("\t=========PROGRAM FOR UNIFICATION=========\n");

            printf("\nEnter Number of Predicates:- [ ]\b\b");

            scanf("%d",&no\_of\_pred);

            for(i=0;i<no\_of\_pred;i++)

            {

            scanf("%c",&nouse);    //to accept "Enter" as a character

            printf("\nEnter Predicate %d:-[ ]\b\b",i+1);

            scanf("%c",&predicate[i]);

            printf("\n\tEnter No.of Arguments for Predicate %c:-[ ]\b\b",predicate[i]);

            scanf("%d",&no\_of\_arg[i]);

                        for(j=0;j<no\_of\_arg[i];j++)

                        {

                         scanf("%c",&nouse);

                         printf("\n\tEnter argument %d:( )\b\b",j+1);

                         scanf("%c",&argument[i][j]);

                        }

            }

            display();

            chk\_arg\_pred();

            getch();

            flushall();

            printf("Do you want to continue(y/n): ");

            scanf("%c",&ch);

       }while(ch=='y');

   }

   void display()

   {

       printf("\n\t=======PREDICATES ARE======");

            for(i=0;i<no\_of\_pred;i++)

            {

             printf("\n\t%c(",predicate[i]);

                        for(j=0;j<no\_of\_arg[i];j++)

                        {

                        printf("%c",argument[i][j]);

                        if(j!=no\_of\_arg[i]-1)

                                    printf(",");

                        }

             printf(")");

            }

   }

   void chk\_arg\_pred()

   {

   int pred\_flag=0;

   int arg\_flag=0;

   /\*======Checking Prediactes========\*/

            for(i=0;i<no\_of\_pred-1;i++)

            {

                        if(predicate[i]!=predicate[i+1])

                        {

                        printf("\nPredicates not same..");

                        printf("\nUnification cannot progress!");

                        pred\_flag=1;

                        break;

                        }

            }

   /\*=====Chking No of Arguments====\*/

     if(pred\_flag!=1)

     {

            for(i=0;i<no\_of\_arg[i]-1;i++)

            {

                        if(no\_of\_arg[i]!=no\_of\_arg[i+1])

                        {

                        printf("\nArguments Not Same..!");

                        arg\_flag=1;

                        break;

                        }

            }

     }

            if(arg\_flag==0&&pred\_flag!=1)

                        unify();

   }

/\*==========UNIFY FUNCTION=========\*/

   void unify()

   {

            int flag=0;

            for(i=0;i<no\_of\_pred-1;i++)

            {

                 for(j=0;j<no\_of\_arg[i];j++)

                 {

                        if(argument[i][j]!=argument[i+1][j])

                        {

                          if(flag==0)

                          printf("\n\t======SUBSTITUTION IS======");

                        printf("\n\t%c/%c",argument[i+1][j],argument[i][j]);

                         flag++;

                        }

                }

            }

            if(flag==0)

            {          printf("\nArguments are Identical...");

                        printf("\nNo need of Substitution\n");

            }

   }

**OUTPUT:**

        =========PROGRAM FOR UNIFICATION=========

Enter Number of Predicates:- [2]

Enter Predicate 1:-[P]

        Enter No.of Arguments for Predicate P:-[2]

        Enter argument 1:(a)

        Enter argument 2:(b)

Enter Predicate 2:-[P]

        Enter No.of Arguments for Predicate P:-[2]

        Enter argument 1:(c)

        Enter argument 2:(b)

        =======PREDICATES ARE======

        P(a,b)

        P(c,b)

        ======SUBSTITUTION IS======

        c/a

Do you want to continue(y/n): y

        =========PROGRAM FOR UNIFICATION=========

Enter Number of Predicates:- [2]

Enter Predicate 1:-[P]

        Enter No.of Arguments for Predicate P:-[3]

        Enter argument 1:(a)

        Enter argument 2:(b)

        Enter argument 3:(c)

Enter Predicate 2:-[P]

        Enter No.of Arguments for Predicate P:-[2]

        Enter argument 1:(d)

        Enter argument 2:(r)

        =======PREDICATES ARE======

        P(a,b,c)

        P(d,r)

Arguments Not Same..!Do you want to continue(y/n): y

        =========PROGRAM FOR UNIFICATION=========

Enter Number of Predicates:- [2]

Enter Predicate 1:-[P]

        Enter No.of Arguments for Predicate P:-[2]

        Enter argument 1:(1)

        Enter argument 2:(3)

       Enter Predicate 2:-[Q]

        Enter No.of Arguments for Predicate Q:-[2]

        Enter argument 1:(3)

        Enter argument 2:(4)

        =======PREDICATES ARE======

        P(1,3)

        Q(3,4)

Predicates not same..

Unification cannot progress!

**RESULT:**

Thus the program for Unification was implemented and executed.

**MINIMAX ALGORITHM**

**AIM:**

To Develop a Program for Minimax Algorithm

**Algorithm:**

1. Start.
2. Import util.Scanner
3. Declare a class ‘tictac’ as public.
4. Represent the seeds and contents in the cell by ‘public’.
5. Initialize ‘x’,’y’ as two players.
6. Initialize various stages of the game such as playing,draw,cross\_won,nought\_won.
7. Represent the status of the game board by matrix form.
8. Inside the main method, perform functions of the game.
9. Initialize the status of game.
10. Update the current status of the board and the player.
11. Stop.

**SOURCE CODE:**

import java.util.Scanner;

public class TicTac

{

public static final int EMPTY = 0;

public static final int CROSS = 1;

public static final int NOUGHT = 2;

public static final int PLAYING = 0;

public static final int DRAW = 1;

public static final int CROSS\_WON = 2;

public static final int NOUGHT\_WON = 3;

public static final int ROWS = 3, COLS = 3;

public static int[][] board = new int[ROWS+1][COLS+1];

public static int currentState;

public static int currentPlayer;

public static int currentRow, currentCol;

public static Scanner in = new Scanner(System.in);

public static void main(String[] args)

{

initGame();

do

{

playerMove();

updateGame();

printBoard();

if (currentState == CROSS\_WON)

{

System.out.println("'X' won! Bye!");

}

else if (currentState == NOUGHT\_WON)

{

System.out.println("'O' won! Bye!");

}

else if (currentState == DRAW)

{

System.out.println("It's a Draw! Bye!");

}

currentPlayer = (currentPlayer == CROSS) ? NOUGHT : CROSS;

} while (currentState == PLAYING);

}

public static void initGame()

{

for (int row = 1; row <= ROWS; ++row)

{

for (int col = 1; col <= COLS; ++col)

{

board[row][col] = EMPTY;

}

}

currentState = PLAYING;

currentPlayer = CROSS;

}

public static void playerMove()

{

boolean validInput = false;

do {

if (currentPlayer == CROSS)

{

System.out.print("Player 'X', enter your move (row[1-3] column[1-3]): ");

}

else

{

System.out.print("Player 'O', enter your move (row[1-3] column[1-3]): ");

}

int row = in.nextInt() ;

int col = in.nextInt() ;

if (row >= 1 && row <= ROWS && col >= 1 && col <= COLS && board[row][col] == EMPTY)

{

currentRow = row;

currentCol = col;

board[currentRow][currentCol] = currentPlayer;

validInput = true; // input okay, exit loop

}

else

{

System.out.println("This move at (" + (row) + "," + (col)+ ") is not valid. Try again...");

}

} while (!validInput);

}

public static void updateGame()

{

if (hasWon())

{

currentState = (currentPlayer == CROSS) ? CROSS\_WON : NOUGHT\_WON;

}

else if (isDraw())

{

currentState = DRAW;

}

}

public static boolean isDraw()

{

for (int row = 1; row <= ROWS; ++row)

{

for (int col = 1; col <= COLS; ++col)

{

if (board[row][col] == EMPTY)

{

return false;

}

}

}

return true;

}

public static boolean hasWon()

{

return (board[currentRow][1] == currentPlayer

&& board[currentRow][2] == currentPlayer

&& board[currentRow][3] == currentPlayer

|| board[1][currentCol] == currentPlayer

&& board[2][currentCol] == currentPlayer

&& board[3][currentCol] == currentPlayer

|| currentRow == currentCol

&& board[1][1] == currentPlayer

&& board[2][2] == currentPlayer

&& board[3][3] == currentPlayer

|| currentRow + currentCol == 4

&& board[1][3] == currentPlayer

&& board[2][2] == currentPlayer

&& board[3][1] == currentPlayer);

}

public static void printBoard()

{

for (int row = 1; row <= ROWS; ++row)

{

for (int col = 1; col <= COLS; ++col)

{

printCell(board[row][col]);

if (col != COLS)

{

System.out.print("|");

}

}

System.out.println();

if (row != ROWS)

{

System.out.println("-----------");

}

}

System.out.println();

}

public static void printCell(int content)

{

switch (content)

{

case EMPTY: System.out.print(" "); break;

case CROSS: System.out.print(" X "); break;

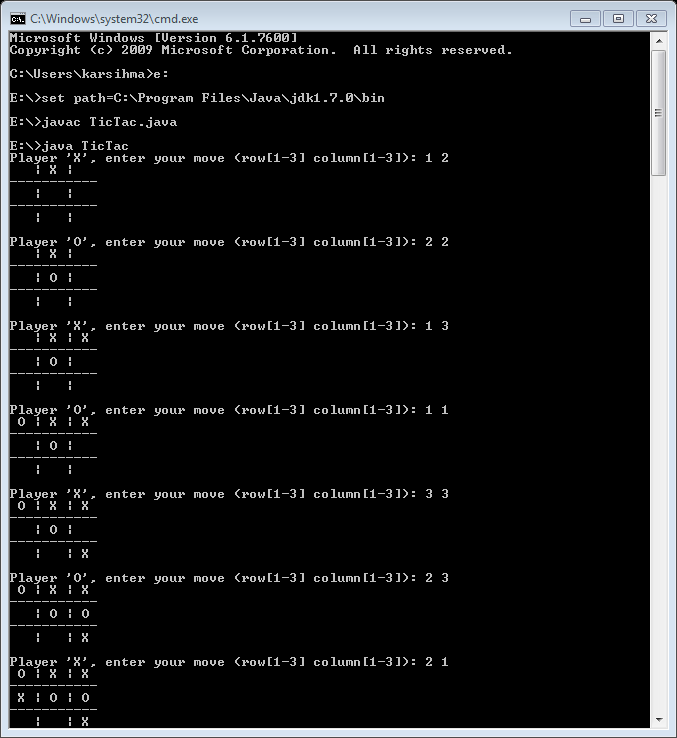
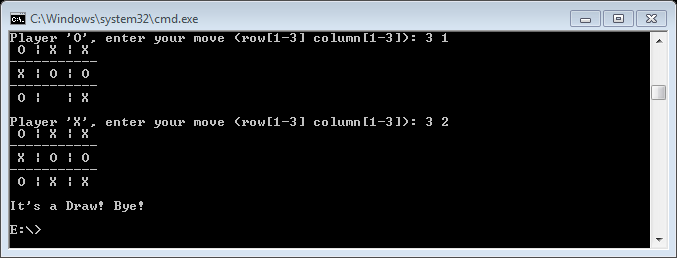
case NOUGHT: System.out.print(" O "); break;

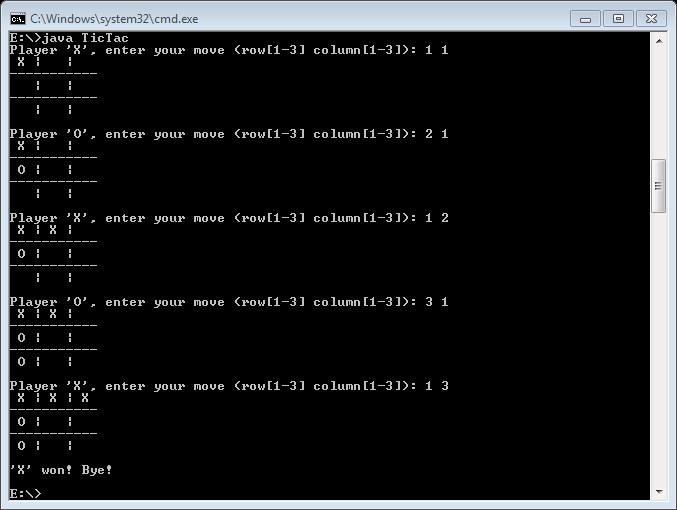
}

}

}

**OUTPUT:**

** **

****

**RESULT:**

Thus the program for Minimax algorithm was implemented and executed

**BLOCK WORLD PROBLEM**

**AIM:**

To develop a program for Blocks Word Problem

**ALGORITHM:**

1. Start the program by creating the class “Example1”.
2. Declare the class PL plan and create the object for it.
3. Define the functions “Facts” and add the facts by using the object “planner”.
4. Define the function “Goal” and add the goals using the object “planner”.
5. Declare the array list for each function and add the facts, goals and operators to the list.
6. Add the facts to the list and assign operators and remove the facts from the list once the fact operation is over.
7. Finally, print the result to find the print and display.
8. End the program.

**SOURCE CODE:**

public class Example1 {

public static void main(String[] args) throws PlPlanException {

System.out.println(&quot;-- TEST1 --&quot;);<br />

// Create a planning problem, and then find a plan<br />

// This problem is based on Blockworld<br />

PLPlan planner = new PLPlan();<br />

planner.setAlgorithm(EnumAlgorithm.GRAPHPLAN);

// Facts (Initial state) //td, ocd, obc, oab, na

planner.addFact("td");

planner.addFact("ocd");

planner.addFact("obc");

planner.addFact("oab");

planner.addFact("na");

// Goal //( oca, odb, ta, tb, nc, nd)

planner.addGoalFact("oca");

planner.addGoalFact("odb");

planner.addGoalFact("ta");

planner.addGoalFact("tb");

planner.addGoalFact("nc");

planner.addGoalFact("nd");

// --- Operators ---

// (op uAB

// (na, oab)

// (ta, nb)

// (oab))

List<String> precond = new ArrayList<String>();

precond.add("na");

precond.add("oab");

List<String> neg = new ArrayList<String>();

neg.add("oab");

List<String> pos = new ArrayList<String>();

pos.add("ta");

pos.add("nb");

planner.addOperator("uAB",precond, neg, pos);

// (op uBC // operator name : "uBC" unstack block b from block c

// (nb, obc) // predonditions : the b Block is on block c (obc) and

// there is no block on block b (nb)

// (tb, nc) // add facts : the block b is on table (tb)

// there is no block on block c (nc)

// (obc)) // remove facts : we remove the fact that

// the b Block is on block c (ocd)

precond = new ArrayList<String>();

precond.add("nb");

precond.add("obc");

neg = new ArrayList<String>();

neg.add("obc");

pos = new ArrayList<String>();

pos.add("tb");

pos.add("nc");

planner.addOperator("uBC",precond, neg, pos);

// (op uCD // operator name : "uCD" unstack block c from block d

// (nc, ocd) // predonditions : the c Block is on block D (ocd) and

// there is no block on block c (nc)

// (tc, nd) // add facts : the block c is on table (tc)

// there is no block on block d (nd)

// (ocd)) // remove facts : we remove the fact that

// the c Block is on block d (ocd)

precond = new ArrayList<String>();

precond.add("nc");

precond.add("ocd");

neg = new ArrayList<String>();

neg.add("ocd");

pos = new ArrayList<String>();

pos.add("tc");

pos.add("nd");

planner.addOperator("uCD",precond, neg, pos);

// (op sCA // operator name : "sCA" stack block c on block a

// (na, tc, nc) // predonditions : there is no block on block a (na)

// the block c is on table (tc)

// there is no block on block c (nc)

// (oca) // add facts : the block c is on block a (oca)

// (na, tc)) // remove facts : we remove the facts that

// - there is no block on block a (na),

// - block c is on table (tc).

precond = new ArrayList<String>();

precond.add("na");

precond.add("tc");

precond.add("nc");

neg = new ArrayList<String>();

neg.add("na");

neg.add("tc");

pos = new ArrayList<String>();

pos.add("oca");

planner.addOperator("sCA", precond, neg, pos);

// (op sDB // operator name : "sDB" stack block d on block b

// (nd, tb, nb) // predonditions : there is no block on block d (nd)

// the block b is on table (tb)

// there is no block on block b (nb)

// (odb) // add facts : the block d is on block b (odb)

// (nb, td)) // remove facts : we remove the facts that

// - there is no block on block b (nb),

// - block d is on table (td).

precond = new ArrayList<String>();

precond.add("nb");

precond.add("td");

precond.add("nd");

neg = new ArrayList<String>();

neg.add("nb");

neg.add("td");

pos = new ArrayList<String>();

pos.add("odb");

planner.addOperator("sDB", precond, neg, pos);

List resultats = planner.findPlan();

System.out.println(resultats);

}

}

**SAMPLE OUTPUT:**

The solution : [uBA, uAG, uGC, uCH, sCA, sFC, sBF]

**RESULT:**

Thus the program for Blocks world Problem was implemented and executed.

**PREPOSITIONAL LOGIC**

**AIM:**

To develop the program for Representation of Knowledge using Propositional Logic and Querying

**ALGORITHM:**

1. Start the program by creating the class ‘P1’.
2. Create the Array List and add the Query and facts to solve the logic.
3. The Task List are ‘forward chain’, backward chain , resolution.
4. In the resolution function , check for the CN and iterate the loop.
5. The result is true , it is yes or it is no.
6. If the result is true , it is ‘yes’ or it is ‘no’.
7. In the forward chain, the array list is created and the rules are entered.
8. If the rules are satisfied, it is ‘yes’ or ‘no’.

**SOURCE CODE:**

Pl.java

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.PrintWriter;

import java.io.UnsupportedEncodingException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

import java.util.LinkedList;

public class pl {

/\*\*

\* @param args

\*/

static ArrayList<Rule> knowledgeBase = new ArrayList<Rule>();

static ArrayList<String> query=new ArrayList<String>();

static ArrayList<String> fact=new ArrayList<String>();

static PrintWriter writerLog;

static PrintWriter writerOutput;

public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException {

// TODO Auto-generated method stub

int task=Integer.parseInt(args[1]);

String kbfile=args[3];

String queryfile=args[5];

String outputentail=args[7];

String outputlog=args[9];

writerLog = new PrintWriter(outputlog, "UTF-8");

writerOutput= new PrintWriter(outputentail, "UTF-8");

processKBfile(kbfile);

processquery(queryfile);

switch (task)

{

case 1:

forwardchain();

break;

case 2:

backwardchain();

break;

case 3:

resolution();

}

}

static void resolution()

{

writerLog.println("Resolving clause 1#Resolving clause 2#Added clause");

/\* Conversion in CNF\*/

ArrayList<String> clauses\_CNF = new ArrayList<String>();

for(String entail:query)

{

clauses\_CNF=convertToCNF();

clauses\_CNF.add(negation(entail));

boolean result=checkresolution(clauses\_CNF,entail);

if(result==true)

writerOutput.println("YES");

else

writerOutput.println("NO");

writerLog.println("------------------------------------------------------------------");

}

writerLog.close();

writerOutput.close();

}

static boolean checkresolution(ArrayList<String> clauses\_CNF,String entail)

{

ArrayList<String> newClauses=new ArrayList<String>();

int iteration=1;

//start the loop

while(iteration<=6)

{

writerLog.println("ITERATION = "+iteration);

for(int i=0; i<clauses\_CNF.size(); i++) // selecting the first clause

{

for(int j=i+1; j<clauses\_CNF.size(); j++)

{

String resolvedClause=null;

String clause1=clauses\_CNF.get(i);

String clause2=clauses\_CNF.get(j);

if(!clause1.equals(clause2))

{

resolvedClause=resolve(clause1,clause2);

if(resolvedClause!=null)

writerLog.println(clause1+" # "+clause2+" # "+resolvedClause);

if(resolvedClause!=null && !resolvedClause.equals("Empty"))

newClauses.add(resolvedClause);

if(resolvedClause!=null && resolvedClause.equals("Empty"))

return true;

}

}

}

for(String claus:newClauses)

{

if(!checkifPresent(claus,clauses\_CNF))

{

clauses\_CNF.add(claus);

}

}

iteration++;

newClauses.clear();

}

return false;

}

static boolean checkifPresent(String newclause,ArrayList<String> clauses\_CNF)

{

String[] newclausetokens=newclause.split(" OR ");

Arrays.sort(newclausetokens);

String newc="";

int flag=0;

for(String b:newclausetokens)

{

newc+=b;

}

for(String clause:clauses\_CNF)

{

String[] clausetokens=clause.split(" OR ");

Arrays.sort(clausetokens);

String oldc="";

for(String a:clausetokens)

{

oldc+=a;

}

if(oldc.equals(newc))

flag=1;

}

if(flag==1)

return true;

else

return false;

}

static String resolve(String clause1,String clause2)

{

int flag=0;

String result="";

String[] clause1\_tokens=clause1.split(" ");

String[] clause2\_tokens=clause2.split(" ");

ArrayList<String> clause1tok=new ArrayList<String>();

ArrayList<String> clause2tok=new ArrayList<String>();

//removing OR's

for(String c1:clause1\_tokens)

{

if(!c1.equals("OR"))

clause1tok.add(c1);

}

for(String c2:clause2\_tokens)

{

if(!c2.equals("OR"))

clause2tok.add(c2);

}

Iterator<String> it = clause1tok.iterator();

while(it.hasNext())

{

String klaus=it.next();

String neg=negation(klaus);

if(clause2tok.contains(neg)) //only resolve if the negation exist in other clause

{

clause2tok.remove(neg);

it.remove();

flag++;

}

}

if(flag>1)

{

return null;

}

if(clause1tok.isEmpty() && clause2tok.isEmpty())

return "Empty";

if(flag==1)

{

ArrayList<String> finalList=new ArrayList<String>();

Iterator<String> it1 = clause1tok.iterator();

Iterator<String> it2 = clause2tok.iterator();

while(it1.hasNext())

{ String str=it1.next();

if(!finalList.contains(str))

finalList.add(str);

}

while(it2.hasNext())

{ String str=it2.next();

if(!finalList.contains(str))

finalList.add(str);

}

int i=1;

for(String str:finalList)

{

result+=str;

if(i<finalList.size())

result+=" OR ";

i++;

}

return result;

}

return null;

}

static String negation(String str)

{

String result;

if(str.charAt(0)!='-')

result="-"+str;

else

result=Character.toString(str.charAt(1));

return result;

}

static ArrayList<String> convertToCNF()

{

ArrayList<String> clauses\_CNF = new ArrayList<String>();

for(Rule entry: knowledgeBase)

{

String cnf="";

if(entry.consequent!=null)

{

String prec= entry.precedent;

int len=prec.length()-1;

int i=0;

while(i<=len)

{

cnf+="-"+prec.charAt(i)+" OR ";

i++;

}

cnf+=entry.consequent;

}

else

{

cnf=entry.precedent;

}

clauses\_CNF.add(cnf);

}

return clauses\_CNF;

}

static void backwardchain()

{

writerLog.println("<Queue of Goals>#Relevant Rules/Fact#New Goal Introduced");

for(String entail:query)

{

ArrayList<String> facts= new ArrayList<String>();

ArrayList<Rule> clauses = new ArrayList<Rule>();

LinkedList<String> goals=new LinkedList<String>();

//get the facts from knowledge base

for(Rule entry: knowledgeBase)

{

if(entry.consequent==null)

{

facts.add(entry.precedent);

}

else

{

clauses.add(entry);

}

}

goals.add(entail); // add to the goal queue

int flag=0;

boolean result = backchain(facts,clauses,goals,entail,flag);

if(result == true)

{

writerOutput.println("YES");

}

else if(result == false)

{

writerOutput.println("NO");

}

writerLog.println("-------------------------------------------------------------");

}

writerOutput.close();

writerLog.close();

}

static boolean backchain(ArrayList<String> facts,ArrayList<Rule> clauses,LinkedList<String> goals,String entail,int flag)

{

if(goals.isEmpty())

return true;

else

{

String topOfQueue=goals.remove();

/\*Cycle detection starts \*/

if((topOfQueue.equals(entail)) && flag !=0)

{

writerLog.println(topOfQueue+" # CYCLE DETECTED # "+"N/A");

return false;

}

flag = 1;

/\*Cycle detection ends \*/

// check if topofQueue is a fact

if(facts.contains(topOfQueue))

{

writerLog.println(topOfQueue+" # "+topOfQueue+" # N/A");

return backchain(facts,clauses,goals,entail,flag);

}

else

{ int flag2=0;

for (Iterator<Rule> iterator = clauses.iterator(); iterator.hasNext(); ) {

Rule entry=iterator.next();

if(entry.consequent.equals(topOfQueue))

{

flag2=1;

String prec=entry.precedent;

char[] precs=prec.toCharArray();

String newgoals="";

writerLog.print(topOfQueue+" # "+topOfQueue+" :- ");

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

{

writerLog.print(precs[i]);

newgoals=newgoals+precs[i];

if(i<precs.length-1)

{

writerLog.print(",");

newgoals=newgoals+", ";

}

}

writerLog.println(" # "+newgoals);

for(int i=precs.length-1;i>=0;i--)

{

goals.addFirst(Character.toString(precs[i]));

}

boolean result = backchain(facts,clauses,goals,entail,flag);

if(result == true)

{

return true;

}

else

{

goals.clear();

}

}

}

if(flag2==0)

{

writerLog.println(topOfQueue+" # N/A # N/A");

return false;

}

return false;

}

}

}

static void forwardchain()

{

int flag=0;

int flag2=0;

writerLog.println("<Known/Deducted facts>#Rules Fires#NewlyEntailedFacts");

// for each string in query do the forward chain

for(String entail:query)

{

// add to the facts

String facts="";

ArrayList<Rule> localknowledgeBase = new ArrayList<Rule>();

for (Rule entry : knowledgeBase) {

localknowledgeBase.add(entry);

if(entry.consequent==null)

{

facts=facts+""+entry.precedent;

}

}

// check knowledge base for fired rules

while(!facts.contains(entail))

{

//get the different form of facts

comb2(facts); //it gives all the combination of facts into fact

for(String f: fact) //for every fact

{

flag=0;

for (Iterator<Rule> iterator = localknowledgeBase.iterator(); iterator.hasNext(); ) {

Rule entry=iterator.next();

//do not check for null values keys

if(entry.consequent!=null)

{

String prec=entry.precedent;

/\*check if the rule exist\*/

//check if different form of facts exist as prec if not then say it is not entailed

if(myequal(f, prec))

{

flag=1;

char[] fac=facts.toCharArray();

int start=0;

while(start<fac.length)

{

writerLog.print(fac[start]);

start++;

if(start<fac.length)

writerLog.print(", ");

}

writerLog.print("#"+entry.consequent+" :- ");

char[] precs=prec.toCharArray();

int index=0;

while(index<precs.length)

{

writerLog.print(precs[index]);

index++;

if(index<precs.length)

writerLog.print(",");

}

writerLog.println(" # "+entry.consequent);

facts=facts+entry.consequent;

iterator.remove();

}

}

}

if(flag==1)

break;

}

if(flag==0)

{

flag2=1;

break;

}

}

if(flag2==1)

writerOutput.println("NO");

else

writerOutput.println("YES");

flag2=0;

writerLog.println("---------------------------------------------------------");

}

writerOutput.close();

writerLog.close();

}

static boolean myequal(String a, String b)

{

if(a.length()!=b.length())

return false;

else

{

int i=0;

while(i<a.length())

{

String str=Character.toString(b.charAt(i));

if(!a.contains(str))

return false;

i++;

}

}

return true;

}

static void comb2(String s)

{ comb2("", s); }

static void comb2(String prefix, String s) {

if(prefix!="")

fact.add(prefix);

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

comb2(prefix + s.charAt(i), s.substring(i + 1));

}

static void processquery(String path)

{

BufferedReader br;

try{

br= new BufferedReader( new FileReader(path));

String line;

while ((line = br.readLine()) != null) {

query.add(line);

}

}

catch(Exception e)

{

System.out.println(e);

}

}

static void processKBfile(String path)

{

BufferedReader br;

try{

br= new BufferedReader( new FileReader(path));

String line,consequent,prec;

while ((line = br.readLine()) != null) {

ArrayList<String> precedent=new ArrayList<String>();

if(line.contains(":-"))

{

String[] tokens=line.split(":-");

consequent=tokens[0].replaceAll("\\s+","");

prec=tokens[1].replaceAll("\\s+","");

if(prec.contains(","))

{

String[] tok=prec.split(",");

for(String t: tok)

{

precedent.add(t);

}

}

else

{

precedent.add(prec);

}

String preced="";

for(String p:precedent)

preced=preced+p;

Rule r=new Rule(preced,consequent);

knowledgeBase.add(r);

}

else //facts

{

precedent.add(line.replaceAll("\\s+",""));

String preced="";

for(String p:precedent)

preced=preced+p;

Rule r=new Rule(preced,null);

knowledgeBase.add(r);

}

}

br.close();

}catch (Exception e)

{

System.out.println(e);

}

}

}

Rule.java

public class Rule {

String precedent;

String consequent;

int used;

public Rule(String p, String c)

{

this.precedent=p;

this.consequent=c;

used=0;

}

}

**SAMPLE OUTPUT:**

YES

NO

YES

NO

NO

YES

NO

YES

NO

NO

YES

NO

YES

NO

NO

**RESULT:**

Thus the program for Representation of Knowledge using Propositional Logic and Querying was implemented and executed.

**Forward Chaining and Backward Chaining**

**AIM:**

To develop the program for Forward chaining and Backward chaining.

**ALGORITHM:**

1. Start the program by creating the class ‘P1’.
2. Create the Array List and add the Query and facts to solve the logic.
3. The Task List are ‘forward chain’, backward chain , resolution.
4. In the resolution function , check for the CN and iterate the loop.
5. The result is true , it is yes or it is no.
6. If the result is true , it is ‘yes’ or it is ‘no’.
7. In the forward chain, the array list is created and the rules are entered.
8. If the rules are satisfied, it is ‘yes’ or ‘no’.

**SOURCE CODE**

Pl.java

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.PrintWriter;

import java.io.UnsupportedEncodingException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

import java.util.LinkedList;

public class pl {

/\*\*

\* @param args

\*/

static ArrayList<Rule> knowledgeBase = new ArrayList<Rule>();

static ArrayList<String> query=new ArrayList<String>();

static ArrayList<String> fact=new ArrayList<String>();

static PrintWriter writerLog;

static PrintWriter writerOutput;

public static void main(String[] args) throws FileNotFoundException, UnsupportedEncodingException {

// TODO Auto-generated method stub

int task=Integer.parseInt(args[1]);

String kbfile=args[3];

String queryfile=args[5];

String outputentail=args[7];

String outputlog=args[9];

writerLog = new PrintWriter(outputlog, "UTF-8");

writerOutput= new PrintWriter(outputentail, "UTF-8");

processKBfile(kbfile);

processquery(queryfile);

switch (task)

{

case 1:

forwardchain();

break;

case 2:

backwardchain();

break;

case 3:

resolution();

}

}

static void resolution()

{

writerLog.println("Resolving clause 1#Resolving clause 2#Added clause");

/\* Conversion in CNF\*/

ArrayList<String> clauses\_CNF = new ArrayList<String>();

for(String entail:query)

{

clauses\_CNF=convertToCNF();

clauses\_CNF.add(negation(entail));

boolean result=checkresolution(clauses\_CNF,entail);

if(result==true)

writerOutput.println("YES");

else

writerOutput.println("NO");

writerLog.println("------------------------------------------------------------------");

}

writerLog.close();

writerOutput.close();

}

static boolean checkresolution(ArrayList<String> clauses\_CNF,String entail)

{

ArrayList<String> newClauses=new ArrayList<String>();

int iteration=1;

//start the loop

while(iteration<=6)

{

writerLog.println("ITERATION = "+iteration);

for(int i=0; i<clauses\_CNF.size(); i++) // selecting the first clause

{

for(int j=i+1; j<clauses\_CNF.size(); j++)

{

String resolvedClause=null;

String clause1=clauses\_CNF.get(i);

String clause2=clauses\_CNF.get(j);

if(!clause1.equals(clause2))

{

resolvedClause=resolve(clause1,clause2);

if(resolvedClause!=null)

writerLog.println(clause1+" # "+clause2+" # "+resolvedClause);

if(resolvedClause!=null && !resolvedClause.equals("Empty"))

newClauses.add(resolvedClause);

if(resolvedClause!=null && resolvedClause.equals("Empty"))

return true;

}

}

}

for(String claus:newClauses)

{

if(!checkifPresent(claus,clauses\_CNF))

{

clauses\_CNF.add(claus);

}

}

iteration++;

newClauses.clear();

}

return false;

}

static boolean checkifPresent(String newclause,ArrayList<String> clauses\_CNF)

{

String[] newclausetokens=newclause.split(" OR ");

Arrays.sort(newclausetokens);

String newc="";

int flag=0;

for(String b:newclausetokens)

{

newc+=b;

}

for(String clause:clauses\_CNF)

{

String[] clausetokens=clause.split(" OR ");

Arrays.sort(clausetokens);

String oldc="";

for(String a:clausetokens)

{

oldc+=a;

}

if(oldc.equals(newc))

flag=1;

}

if(flag==1)

return true;

else

return false;

}

static String resolve(String clause1,String clause2)

{

int flag=0;

String result="";

String[] clause1\_tokens=clause1.split(" ");

String[] clause2\_tokens=clause2.split(" ");

ArrayList<String> clause1tok=new ArrayList<String>();

ArrayList<String> clause2tok=new ArrayList<String>();

//removing OR's

for(String c1:clause1\_tokens)

{

if(!c1.equals("OR"))

clause1tok.add(c1);

}

for(String c2:clause2\_tokens)

{

if(!c2.equals("OR"))

clause2tok.add(c2);

}

Iterator<String> it = clause1tok.iterator();

while(it.hasNext())

{

String klaus=it.next();

String neg=negation(klaus);

if(clause2tok.contains(neg)) //only resolve if the negation exist in other clause

{

clause2tok.remove(neg);

it.remove();

flag++;

}

}

if(flag>1)

{

return null;

}

if(clause1tok.isEmpty() && clause2tok.isEmpty())

return "Empty";

if(flag==1)

{

ArrayList<String> finalList=new ArrayList<String>();

Iterator<String> it1 = clause1tok.iterator();

Iterator<String> it2 = clause2tok.iterator();

while(it1.hasNext())

{ String str=it1.next();

if(!finalList.contains(str))

finalList.add(str);

}

while(it2.hasNext())

{ String str=it2.next();

if(!finalList.contains(str))

finalList.add(str);

}

int i=1;

for(String str:finalList)

{

result+=str;

if(i<finalList.size())

result+=" OR ";

i++;

}

return result;

}

return null;

}

static String negation(String str)

{

String result;

if(str.charAt(0)!='-')

result="-"+str;

else

result=Character.toString(str.charAt(1));

return result;

}

static ArrayList<String> convertToCNF()

{

ArrayList<String> clauses\_CNF = new ArrayList<String>();

for(Rule entry: knowledgeBase)

{

String cnf="";

if(entry.consequent!=null)

{

String prec= entry.precedent;

int len=prec.length()-1;

int i=0;

while(i<=len)

{

cnf+="-"+prec.charAt(i)+" OR ";

i++;

}

cnf+=entry.consequent;

}

else

{

cnf=entry.precedent;

}

clauses\_CNF.add(cnf);

}

return clauses\_CNF;

}

static void backwardchain()

{

writerLog.println("<Queue of Goals>#Relevant Rules/Fact#New Goal Introduced");

for(String entail:query)

{

ArrayList<String> facts= new ArrayList<String>();

ArrayList<Rule> clauses = new ArrayList<Rule>();

LinkedList<String> goals=new LinkedList<String>();

//get the facts from knowledge base

for(Rule entry: knowledgeBase)

{

if(entry.consequent==null)

{

facts.add(entry.precedent);

}

else

{

clauses.add(entry);

}

}

goals.add(entail); // add to the goal queue

int flag=0;

boolean result = backchain(facts,clauses,goals,entail,flag);

if(result == true)

{

writerOutput.println("YES");

}

else if(result == false)

{

writerOutput.println("NO");

}

writerLog.println("-------------------------------------------------------------");

}

writerOutput.close();

writerLog.close();

}

static boolean backchain(ArrayList<String> facts,ArrayList<Rule> clauses,LinkedList<String> goals,String entail,int flag)

{

if(goals.isEmpty())

return true;

else

{

String topOfQueue=goals.remove();

/\*Cycle detection starts \*/

if((topOfQueue.equals(entail)) && flag !=0)

{

writerLog.println(topOfQueue+" # CYCLE DETECTED # "+"N/A");

return false;

}

flag = 1;

/\*Cycle detection ends \*/

// check if topofQueue is a fact

if(facts.contains(topOfQueue))

{

writerLog.println(topOfQueue+" # "+topOfQueue+" # N/A");

return backchain(facts,clauses,goals,entail,flag);

}

else

{ int flag2=0;

for (Iterator<Rule> iterator = clauses.iterator(); iterator.hasNext(); ) {

Rule entry=iterator.next();

if(entry.consequent.equals(topOfQueue))

{

flag2=1;

String prec=entry.precedent;

char[] precs=prec.toCharArray();

String newgoals="";

writerLog.print(topOfQueue+" # "+topOfQueue+" :- ");

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

{

writerLog.print(precs[i]);

newgoals=newgoals+precs[i];

if(i<precs.length-1)

{

writerLog.print(",");

newgoals=newgoals+", ";

}

}

writerLog.println(" # "+newgoals);

for(int i=precs.length-1;i>=0;i--)

{

goals.addFirst(Character.toString(precs[i]));

}

boolean result = backchain(facts,clauses,goals,entail,flag);

if(result == true)

{

return true;

}

else

{

goals.clear();

}

}

}

if(flag2==0)

{

writerLog.println(topOfQueue+" # N/A # N/A");

return false;

}

return false;

}

}

}

static void forwardchain()

{

int flag=0;

int flag2=0;

writerLog.println("<Known/Deducted facts>#Rules Fires#NewlyEntailedFacts");

// for each string in query do the forward chain

for(String entail:query)

{

// add to the facts

String facts="";

ArrayList<Rule> localknowledgeBase = new ArrayList<Rule>();

for (Rule entry : knowledgeBase) {

localknowledgeBase.add(entry);

if(entry.consequent==null)

{

facts=facts+""+entry.precedent;

}

}

// check knowledge base for fired rules

while(!facts.contains(entail))

{

//get the different form of facts

comb2(facts); //it gives all the combination of facts into fact

for(String f: fact) //for every fact

{

flag=0;

for (Iterator<Rule> iterator = localknowledgeBase.iterator(); iterator.hasNext(); ) {

Rule entry=iterator.next();

//do not check for null values keys

if(entry.consequent!=null)

{

String prec=entry.precedent;

/\*check if the rule exist\*/

//check if different form of facts exist as prec if not then say it is not entailed

if(myequal(f, prec))

{

flag=1;

char[] fac=facts.toCharArray();

int start=0;

while(start<fac.length)

{

writerLog.print(fac[start]);

start++;

if(start<fac.length)

writerLog.print(", ");

}

writerLog.print("#"+entry.consequent+" :- ");

char[] precs=prec.toCharArray();

int index=0;

while(index<precs.length)

{

writerLog.print(precs[index]);

index++;

if(index<precs.length)

writerLog.print(",");

}

writerLog.println(" # "+entry.consequent);

facts=facts+entry.consequent;

iterator.remove();

}

}

}

if(flag==1)

break;

}

if(flag==0)

{

flag2=1;

break;

}

}

if(flag2==1)

writerOutput.println("NO");

else

writerOutput.println("YES");

flag2=0;

writerLog.println("---------------------------------------------------------");

}

writerOutput.close();

writerLog.close();

}

static boolean myequal(String a, String b)

{

if(a.length()!=b.length())

return false;

else

{

int i=0;

while(i<a.length())

{

String str=Character.toString(b.charAt(i));

if(!a.contains(str))

return false;

i++;

}

}

return true;

}

static void comb2(String s)

{ comb2("", s); }

static void comb2(String prefix, String s) {

if(prefix!="")

fact.add(prefix);

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

comb2(prefix + s.charAt(i), s.substring(i + 1));

}

static void processquery(String path)

{

BufferedReader br;

try{

br= new BufferedReader( new FileReader(path));

String line;

while ((line = br.readLine()) != null) {

query.add(line);

}

}

catch(Exception e)

{

System.out.println(e);

}

}

static void processKBfile(String path)

{

BufferedReader br;

try{

br= new BufferedReader( new FileReader(path));

String line,consequent,prec;

while ((line = br.readLine()) != null) {

ArrayList<String> precedent=new ArrayList<String>();

if(line.contains(":-"))

{

String[] tokens=line.split(":-");

consequent=tokens[0].replaceAll("\\s+","");

prec=tokens[1].replaceAll("\\s+","");

if(prec.contains(","))

{

String[] tok=prec.split(",");

for(String t: tok)

{

precedent.add(t);

}

}

else

{

precedent.add(prec);

}

String preced="";

for(String p:precedent)

preced=preced+p;

Rule r=new Rule(preced,consequent);

knowledgeBase.add(r);

}

else //facts

{

precedent.add(line.replaceAll("\\s+",""));

String preced="";

for(String p:precedent)

preced=preced+p;

Rule r=new Rule(preced,null);

knowledgeBase.add(r);

}

}

br.close();

}catch (Exception e)

{

System.out.println(e);

}

}

}

Rule.java

public class Rule {

String precedent;

String consequent;

int used;

public Rule(String p, String c)

{

this.precedent=p;

this.consequent=c;

used=0;

}

}

**SAMPLE OUTPUT:**

YES

NO

YES

NO

NO

YES

NO

YES

NO

NO

YES

NO

YES

NO

NO

**RESULT:**

Thus the program for forward and Backward Chaining was implemented and executed.

**Development of Expert System**

**AIM:**

To develop a program for an Expert System

**ALGORITHM:**

1. Start the program by creating the class compiler.
2. Enter the string in the stack using scanner class.
3. The string will be passed to be hash table and special character will be stored in table.
4. The string will be pushed into the stack when the hash table has the same string.
5. Or else, it will be popped from the stack and print as “  
   error”.
6. If the size of the string is greater than 0, it will print of compilation or good job.
7. End the program.

**SOURCE CODE:**

**// Input: { a < b > }**

import java.util.Hashtable;

import java.util.Scanner;

import java.util.Stack;

public class Compiler {

public static void main(String[] args)

{

Scanner in = new Scanner(System.in);

System.out.println("ENTER THE STRING");

Stack<String> stk = new Stack<String>();

Hashtable<String,String> ht = new Hashtable<String,String>();

ht.put("}", "{");

ht.put(")", "(");

ht.put(">", "<");

String input;

input= in.nextLine();

String[] ch = new String[1000];

try

{

ch = input.split(" ");

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

{

if(ht.containsValue(ch[i]))

{

stk.push(ch[i]);

}

else if(ht.containsKey(ch[i]))

{

String st = stk.peek();

String kv = ht.get(ch[i]);

if (st.equals(kv))

{

stk.pop();

}

else

{

System.out.println("Compilation Errors");

return;

}

}

}

if(stk.size() > 0)

{

System.out.println("Compilation Errors");

}

else

{

System.out.println("Good Job");

}

}

catch (Exception e)

{

System.out.println("Compilation Errors");

}

}

}

**SAMPLE OUTPUT:**

ENTER THE STRING

{<>}

Good Job

ENTER THE STRING

{

Compilation Errors

**RESULT:**

Thus an Expert System was implemented and executed successfully.

**EXPERT SYSTEM FOR MEDICAL DIAGNOSIS**

**AIM:**

To develop a program for an Expert System in prolog

**ALGORITHM:**

1. Create a new program

A text editor window will open. You can type in the program.

* + Click on File
  + Click on Save

1. If the program exists
   * Click on File
   * Click on Open
   * Choose the program

A text editor window with the program will open.

1. Loading the program
   * Click on Listener
   * Click on Start

The Listener window will open. This is the window to enter queries and to obtain the answers.

* + Click on Listener
  + Click on Consult

The program will be loaded (Opening the program in the text editor window does not make the program available for the interpreter. That is why it has to be "consulted"

1. Normal run

In the Listener window after the Prolog prompt '?-' type in the goal you want to prove.   
Be sure to type the exact name of the predicate as it is in the program, with same number of arguments.   
End the query with a period.

1. Debugging

**CODING:**

disease(y,y,y,y):-write('You are suffering from VIRAL FEVER').

disease(y,y,y,n):-write('You are suffering from COMMON FEVER').

disease(y,y,n,y):-write('You are suffering from VIRAL FEVER').

disease(y,y,n,n):-write('You maybe OVERHOOKED').

disease(y,n,y,y):-write('You are suffering from COLD').

disease(y,n,y,n):-write('You might develop THROAT INFECTION').

disease(y,n,n,y):-write('You might have SEDENTARY for a long spell').

disease(y,n,n,n):-write('You are suffering from HEADACHE').

disease(n,y,y,y):-write('You are suffering from FEVER').

disease(n,y,y,n):-write('You are experiencing initial stages of COMMON COLD').

disease(n,y,n,y):-write('You are suffering from DEHYDRATION').

disease(n,y,n,n):-write('You are suffering from DENGU FEVER').

disease(n,n,y,y):-write('You are suffering from CHICKENGUNYA').

disease(n,n,y,n):-write('You are suffering from SORE THROAT').

disease(n,n,n,y):-write('You can relax, U are ENERVATED').

disease(n,n,n,n):-write('You are ALRIGHT').

run:-write('Do you have HEADACHE? '),

read(HA),

write('DO you have TEMPERATURE? '),

read(TP),

write('Do you have a SORE THROAT? '),

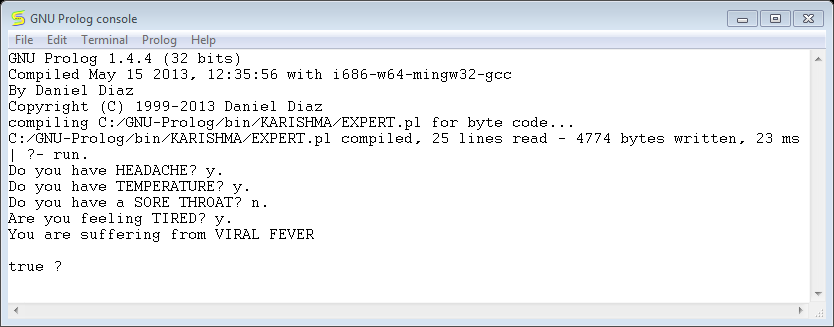
read(ST),

write('Are you feeling TIRED? '),

read(TR),

disease(HA,TP,ST,TR).

**OUTPUT:**



**RESULT:**

Thus an Expert System was implemented and executed successfully.

**COUNT THE VOWELS IN A LIST**

**AIM:**

To develop a program for an Count the vowels in a list in prolog

**ALGORITHM:**

1. Create a new program

A text editor window will open. You can type in the program.

* + Click on File
  + Click on Save

1. If the program exists
   * Click on File
   * Click on Open
   * Choose the program

A text editor window with the program will open.

1. Loading the program
   * Click on Listener
   * Click on Start

The Listener window will open. This is the window to enter queries and to obtain the answers.

* + Click on Listener
  + Click on Consult

The program will be loaded (Opening the program in the text editor window does not make the program available for the interpreter. That is why it has to be "consulted"

1. Normal run

In the Listener window after the Prolog prompt '?-' type in the goal you want to prove.   
Be sure to type the exact name of the predicate as it is in the program, with same number of arguments.   
End the query with a period.

1. Debugging

**CODING:**

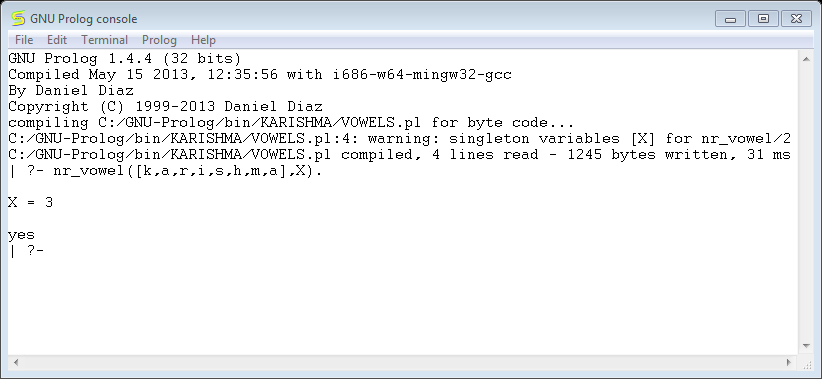
vowel(X):- member(X,[a,e,i,o,u]).

nr\_vowel([],0).

nr\_vowel([X|T],N):-vowel(X),nr\_vowel(T,N1),N is N1+1,!.

nr\_vowel([X|T],N):-nr\_vowel(T,N).

**OUTPUT:**



**RESULT:**

Thus a Count vowels in a list was implemented and executed successfully.

**GUESS THE WORD**

**AIM:**

To develop a program for guess the word in prolog

**ALGORITHM:**

1. Create a new program

A text editor window will open. You can type in the program.

* + Click on File
  + Click on Save

1. If the program exists
   * Click on File
   * Click on Open
   * Choose the program

A text editor window with the program will open.

1. Loading the program
   * Click on Listener
   * Click on Start

The Listener window will open. This is the window to enter queries and to obtain the answers.

* + Click on Listener
  + Click on Consult

The program will be loaded (Opening the program in the text editor window does not make the program available for the interpreter. That is why it has to be "consulted"

1. Normal run

In the Listener window after the Prolog prompt '?-' type in the goal you want to prove.   
Be sure to type the exact name of the predicate as it is in the program, with same number of arguments.   
End the query with a period.

1. Debugging

**CODING:**

in\_mind([h,a,r,d,w,o,r,k]).

start:-write('Guess First Letter : '),read(X),

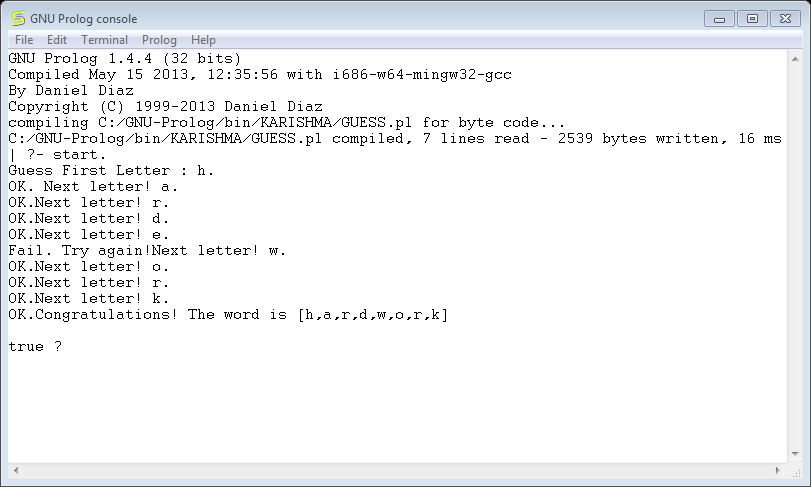
in\_mind([X|T]),write('OK.'),guess(T).

guess([]):-write('Congratulations! The word is '),in\_mind(W),write(W),!.

guess(L):-repeat,write('Next letter! '),read(X),

((L=[X|T1],write('OK. '),guess(T1));

(write('Fail. Try again! '),guess(L))).

**OUTPUT:** 

**RESULT:**

Thus Guess a word was implemented and executed successfully.