PRACTICAL NO.1

AIM: -Introduction of Artificial Intelligence and its application.

1.1 Artificial Intelligence

According to the father of Artificial Intelligence, John McCarthy, it is "The science and engineering of making intelligent machines, especially intelligent computer programs". Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

1.2 Goals of AI

- **To Create Expert Systems** − The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** − Creating systems that understand, think, learn, and behave like humans.

1.3 Advantages of Artificial Intelligence

- More powerful are more useful computers
- New and improved interfaces
- Solving new problems
- Better handling of information
- Relieves information overload
- Conversion of information into knowledge

1.4 Disadvantages of Artificial Intelligence

- Increased costs
- Difficulty with software development-slow and expensive
- Few experienced programmers
- Few practical products have reached the market as yet

1.5 Foundation of AI: -

Fig.1.1

1.6 Applications of AI: -

AI has been dominant in various fields such as −

- **Gaming** − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.

- **Natural Language Processing** − It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** − These systems understand, interpret, and comprehend visual input on the computer. For example,
  - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
  - Doctors use clinical expert system to diagnose the patient.
  - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
- **Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
- **Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
- **Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

PRACTICAL NO.2

AIM: -Implementation of Depth-First Search(DFS).

2.1 Depth-First Search

DFS is also an important type of uniform search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded nodded. The data structure stack or (LIFO) is used for DFS. One interesting property of DFS is that, the discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

2.2 Advantages of Depth-First Search

- DFS consumes very less memory space.
- It will reach at the goal node in a less time period than BFS if it traverses in a right path.
- It may find a solution without examining much of search because we may get the desired solution in the very first go.
- t takes less memory as compared to BFS as BFS requires entire tree to be stored but this requires only one path.
- Sometimes solution lies in earlier stages then DFS is better.
- If there are multiple solutions then DFS stops when first solution is found. Where as BFS gives all the solutions at the same time.

2.3 Disadvantages of Depth-First Search

- There is a possibility that it may go down the left-most path forever. Even a finite graph can generate an infinite tree.
- It is possible that may states keep reoccurring. There is no guarantee of finding the goal node.
- Sometimes the states may also enter into infinite loops.
- Depth-First Search is not guaranteed to find the solution.
- And there is no guarantee to find a minimal solution, if more than one solution exists.

2.4 Algorithm of Depth-First Search

1: PUSH the starting node into the stack.
2: If the stack is empty then stop and return failure.
3: If the top node of the stack is the goal node, then stop and return success.
4: Else POP the top node from the stack and process it. Find all its neighbours that are in ready state and PUSH them into the stack in any order.
5: Go to step 3.
6: Exit.

2.5 Program: -

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10];
void main()
{
int m;
clrscr();
cout<<"Enter of vertices";
cin>>n;
cout<<"Enter of edges";
cin>>m;
cout<<"EDGES\n";
for(k=1;k<=m;k++)
{
```

```
cin>>i>>j;
cost[i][j]=1;
}
cout<<"Enter initial vertex";
cin>>v;
cout<<"Order of Visited vertices\n";
cout<<v<<" ";
visited[v]=1;
k=1;
while(k<n)
{
for(j=n;j>=1;j--)
if(cost[v][j]!=0&& visited[j]!=1&& visit[j]!=1)
{
visit[j]=1;
stk[top]=j;
top++;}
v=stk[--top];
cout<<v<<" ";
k++;
visit[v]=0;
visited[v]=1;
}
getch();
}
```

Output:-

Fig. 2.1

PRACTICAL NO. 3

AIM: -Write a program to implement water jug problem.

3.1 Water jug problem

Statement : We are given 2 jugs, a 4 liter one and a 3- liter one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can we get exactly 2 liters of water in to the 4-liter jugs?

Solution:-

The state space for this problem can be defined as

{ (i ,j) i = 0,1,2,3,4 j = 0,1,2,3}

'i' represents the number of liters of water in the 4-liter jug and 'j' represents the number of liters of water in the 3-liter jug. The initial state is (0,0) that is no water on each jug. The goal state is to get (2,n) for any value of 'n'.

To solve this we have to make some assumptions not mentioned in the problem. They are

1. We can fill a jug from the pump.
2. We can pour water out of a jug to the ground.
3. We can pour water from one jug to another.
4. There is no measuring device available.

Table No. 3.1

1.	(X, Y) if $X < 4 \rightarrow (4, Y)$	Fill the 4-gallon jug
2.	(X, Y) if $Y < 3 \rightarrow (X, 3)$	Fill the 3-gallon jug
3.	(X, Y) if $X = d \ \& \ d > 0 \rightarrow (X\text{-}d, Y)$	Pour some water out of the 4-gallon jug
4.	(X, Y) if $Y = d \ \& \ d > 0 \rightarrow (X, Y - d)$	Pour some water out of 3-gallon jug
5.	(X, Y) if $X > 0 \rightarrow (0, Y)$	Empty the 4-gallon jug on the ground
6.	(X, Y) if $Y > 0 \rightarrow (X, 0)$	Empty the 3-gallon jug on the ground
7.	(X, Y) if $X + Y \leq 4$ and $Y > 0 \rightarrow 4, (Y - (4 - X))$	Pour water from the 3-gallon jug into the 4-gallon jug until the gallon jug is full.
8.	(X, Y) if $X + Y \geq 3$ and $X > 0 \rightarrow (X - (3 - Y), 3))$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9.	(X, Y) if $X + Y \leq 4$ and $Y > 0 \rightarrow (X + Y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	(X, Y) if $X + Y \leq 3$ and $X > 0 \rightarrow (0, X + Y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	$(0, 2) \rightarrow (2, 0)$	Pour the 2-gallons water from 3-gallon jug into the 4;gallon jug
12.	$(2, Y) \rightarrow (0, Y)$	Empty the 2-gallons in the 4-gallon jug on the ground.

3.1 Program: -

```
#include<iostream.h>
#include<iomanip.h>
#include<math.h>
#include<conio.h>
int xcapacity;
int ycapacity;
void display(int a, int b,int s);
int min(int d, int f)
{if (d < f)
return d;
else
return f;
} int steps(int n)
{
int x = 0, y = 0, step = 0;
```

```
int temp;
cout << setw(55) << " Vessel A Vessel B Steps" << endl; while
(x != n )
{if (x == 0)
{ x = xcapacity;
step += 1;
cout << "Fill X  "; display(x, y,step);
} else if (y ==
ycapacity) { y = 0;
step++;
cout << "Empty Y    "; display(x, y,step);
}else {
temp = min(ycapacity – y, x);
y = y + temp;
x = x – temp;
step++;
cout << "Pour X in Y"; display(x, y, step);
} }
return step;
}void display(int a, int b,int s)
{cout << setw(16) << a << setw(15) << b << setw(15)<<s<<endl;
} void main()
{
int n, ans;
clrscr();
cout << "Enter the liters(GOAL) of water required to be filled in Vessel
1:"; cin >> n;
cout << "Enter the capacity of the vessel:
"; cin >> xcapacity;
cout << "Enter the capacity of the second 7essel: ";
cin >> ycapacity;
ans = steps(n);
cout << "Steps Required: " <<
ans; cout<<"pause";
}
```

Output:-

```
Enter the liters(GOAL) of water required to be filled in Vessel 1:2
Enter the capacity of the vessel: 4
Enter the capacity of the second vesssel: 3
                        Vessel A    Vessel B    Steps
Fill X                  4               0           1
Pour X in Y             1               3           2
Empty Y                 1               0           3
Pour X in Y             0               1           4
Fill X                  4               1           5
Pour X in Y             2               3           6
Steps Required: 6 pause_
```

Fig.3.1

PRACTICAL NO. 4

AIM: -Write a program to implement tic tac toe game for 0 and X.

4.1 Tic Tac Toe

Tic Tac Toe is a child's game played on a 3 by 3 grid. One player, X, starts by placing an X at an unoccupied grid position. Then the other player, O, places an O at an unoccupied grid position. Play alternates between X and O until the grid is filled or one player's symbols occupy an entire line (vertical, horizontal, or diagonal) in the grid.

A tic-tac-toe AI program that never loses. This program uses the minimax algorithm with alpha-beta pruning to reduce the search space.

1. return a value if a terminal state is found (+10, 0, -10)
2. go through available spots on the board
3. call the minimax function on each available spot (recursion)
4. evaluate returning values from function calls
5. and return the best value

4.2 Minimax Algorithm Visualizations:

The entire algorithm has been divided into five parts. We are assuming that the player is playing X and the computer is playing O.

A. Check for winning condition.
B. Playing Defensive.
C. The First Move.
D. The Next Move.
E. Some Special Moves.

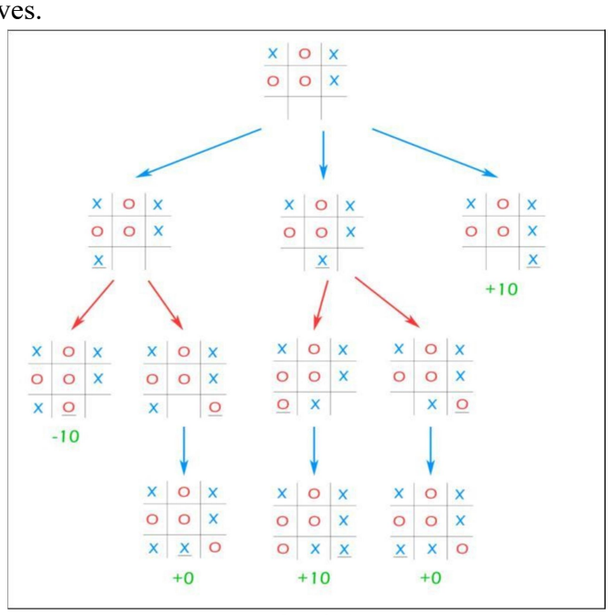

Fig.4.1

4.1 Program: -

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream.h>
char matrix[3][3]; //intitial matrix declaration
char check(void); // declaration of functions
void init_matrix(void);
void get_player_move(void);
void get_computer_move(void);
void disp_matrix(void);
int main(void)
{
clrscr();
```

```
char done;
cout<<"Human vs. AI Tic Tac Toe."<<endl;
cout<<"You will be playing against the computer as
'X'"<<endl; done = ' ';
init_matrix();
do {
disp_matrix();
get_player_move();
done = check(); /* check winner */
if(done!= ' ') break; /* if winner found...*/
get_computer_move();
done = check(); /* check for winner again */
} while(done== ' '); if(done=='X')
cout<<"Human won! (but AI very dumb anyway)\n"; else
cout<<"AI so stupid still can win against you..."<<endl; disp_matrix(); /* show
final positions */
return 0;
}
void init_matrix(void) //matrix intitialisation
{
int i, j;
for(i=0; i<3; i++)
for(j=0; j<3; j++) matrix[i][j] = ' ';
}
void get_player_move(void) //call function for player input
{
int x, y;
cout<<"Enter X,Y coordinates for your move: ";
scanf("%d%*c%d", &x, &y);
x--; y--;
if(matrix[x][y]!= ' ')
{
cout<<"Invalid move, try again.\n";
get_player_move();
}
else matrix[x][y] = 'X';
}
void get_computer_move(void) //AI move input
{
int i, j;
for(i=0; i<3; i++)
{
for(j=0;j<3; j++)
if(matrix[i][j]==' ') break;
if(matrix[i][j]==' ') break;
}
if(i*j==9)
{
cout<<"draw\n";
exit(0);
}
else
matrix[i][j] = 'O';
}
void disp_matrix(void) //matrix display
{
int t;
```

```
for(t=0; t<3; t++)
{printf(" %c | %c | %c ",matrix[t][0],
matrix[t][1], matrix [t][2]);
if(t!=2)
printf("\n---|---|---\n");
}printf("\n");}
char check(void) //used for identifying winner
{ int i;
for(i=0; i<3; i++) /* check rows */
if(matrix[i][0]==matrix[i][1] &&
matrix[i][0]==matrix[i][2]) return matrix[i][0];
for(i=0; i<3; i++) /* check columns */
if(matrix[0][i]==matrix[1][i] &&
matrix[0][i]==matrix[2][i]) return matrix[0][i];
/* test diagonals */
if(matrix[0][0]==matrix[1][1] && matrix[1][1]==matrix[2][2])
return matrix[0][0];
if(matrix[0][2]==matrix[1][1] &&
matrix[1][1]==matrix[2][0])
return matrix[0][2];
return ' ';
}
```

Output:

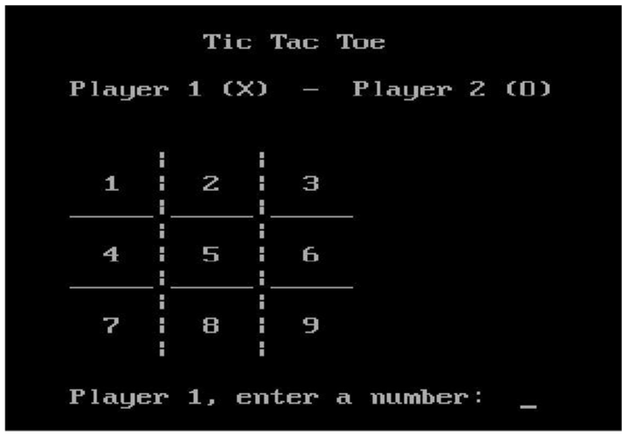

Fig. 4.2

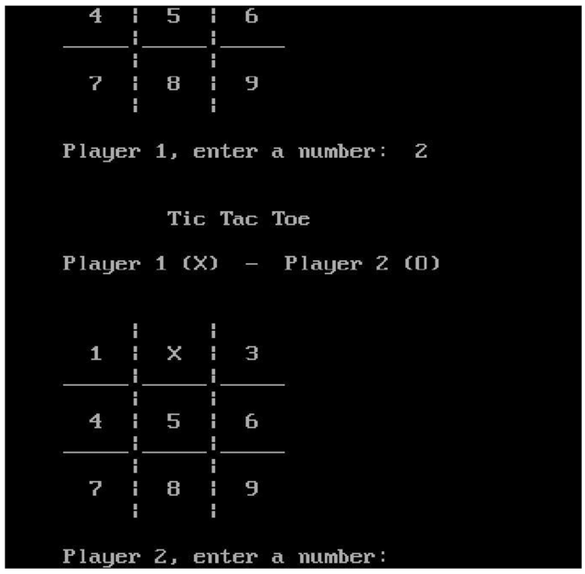

Fig. 4.3

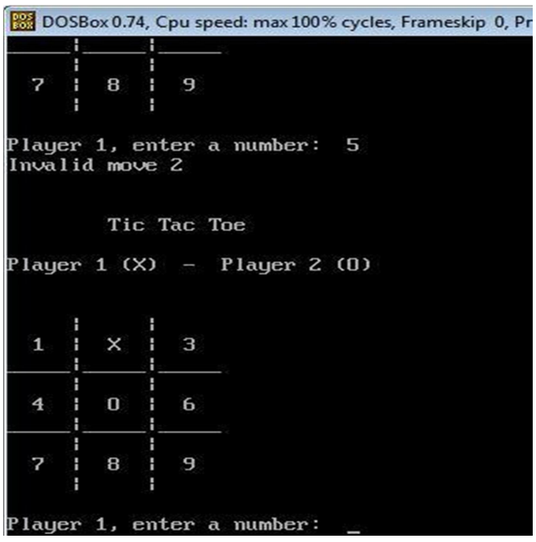

Fig. 4.4

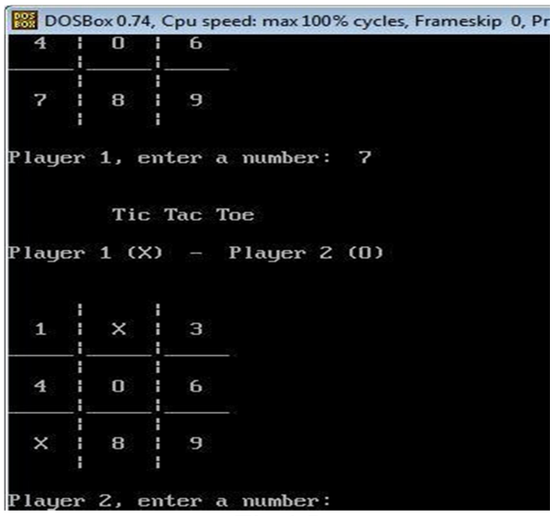

Fig.4.5

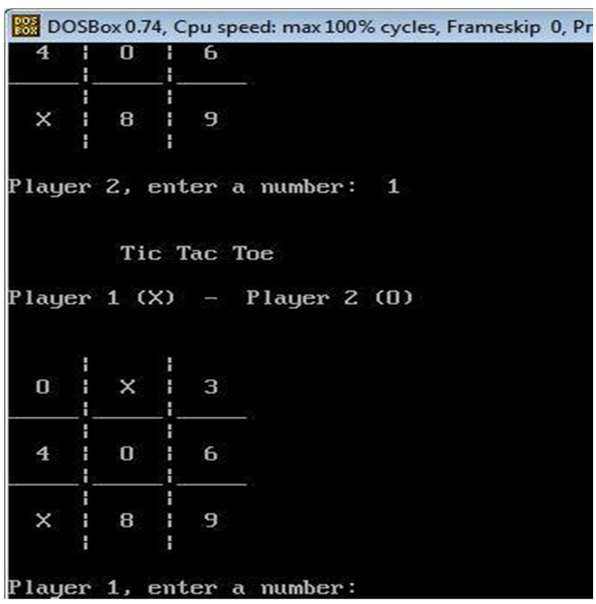

Fig.4.6

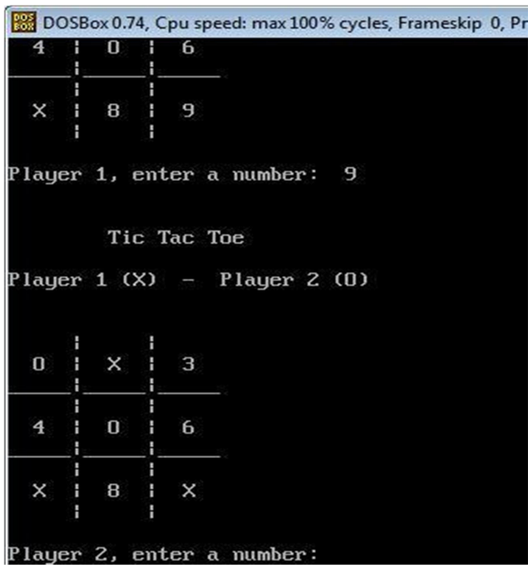

Fig.4.7

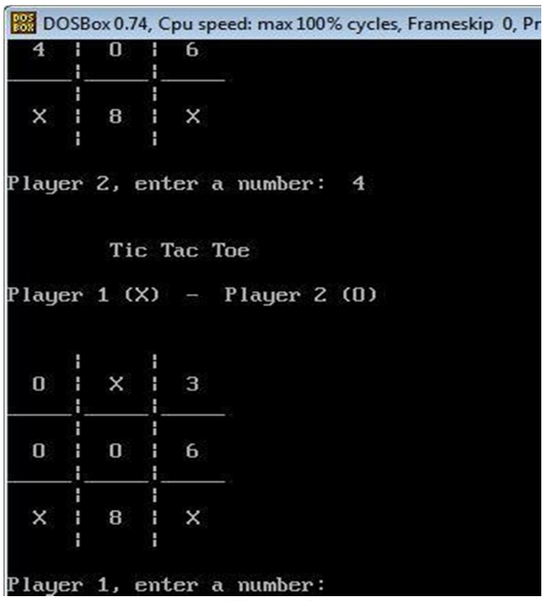

Fig. 4.8

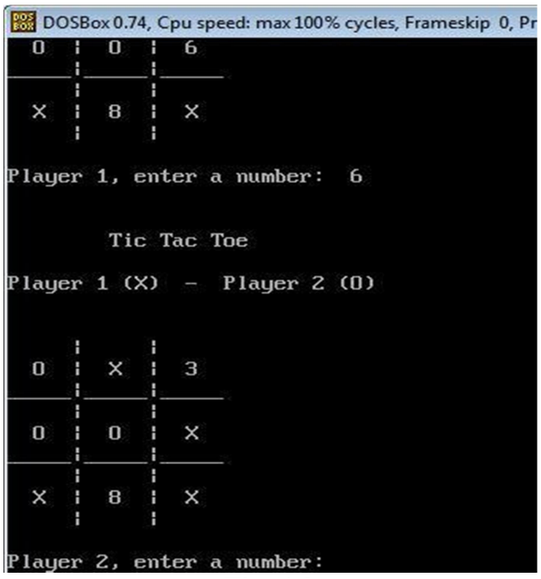

Fig. 4.9

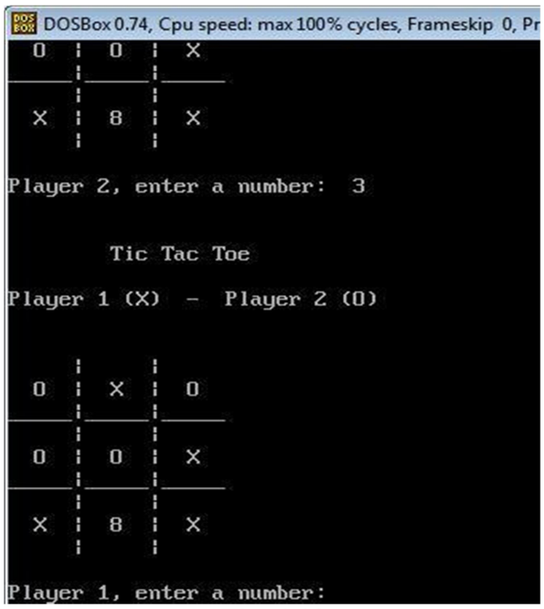

Fig. 4.10

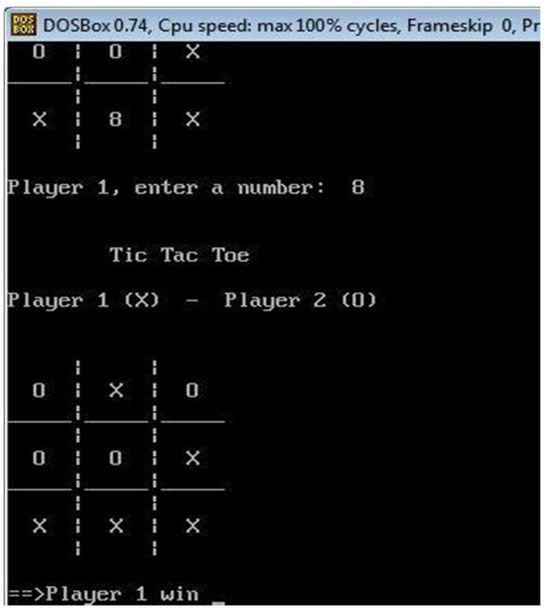

Fig. 4.11

PRACTICAL NO.5

AIM: -Write a program to implement production system.

5.1 Production System:

Search forms the core of many intelligent processes, it is useful to structure AI programs in a way that facilitates describing and performing the search process. Production system provides such structures .In other words the process of solving the problem can usefully be modeled as production system.

The goal database is the central data structure used by an AI production system. The production system. The production rules operate on the global database. Each rule has a precondition that is either satisfied or not by the database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the database. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If several rules are to fire at the same time, the control system resolves the conflicts.

5.2 Components of production system

- A **set of rules** of the form Ci → Ai where Ci is the condition part and Ai is the action part. The condition determines when a given rule is applied, and the action determines what happens when it is applied.
- One or more **knowledge databases** that contain whatever information is relevant for the given problem. Some parts of the database may be permanent, while others may temporary and only exist during the solution of the current problem. The information in the databases may be structured in any appropriate manner.
- **A control strategy** that determines the order in which the rules are applied to the database, and provides a way of resolving any conflicts that can arise when several rules match at once.
- **A rule applier** which is the computational system that implements the control strategy and applies the rules.

5.3 Four classes of production systems:-

1. A monotonic production system
2. A non monotonic production system
3. A partially commutative production system
4. A commutative production system.

5.4 Advantages of production systems:-

1. Production systems provide an excellent tool for structuring AI programs.
2. Production Systems are highly modular because the individual rules can be added, removed or modified independently.
3. The production rules are expressed in a natural form, so the statements contained in the knowledge base should the a recording of an expert thinking out loud.

5.5 Disadvantages of Production Systems:-

1. One important disadvantage is the fact that it may be very difficult analyse the flow of control within a production system because the individual rules don't call each other.
2. Production systems describe the operations that can be performed in a search for a solution to the problem. They can be classified as follows.

5.6 Program

```
#include<iostream.
h>
#include<conio.h>
int main()
{
char answer;
clrscr();
cout<<"Answer the following question to determine whether JOHN should get scholarship or
not?"<<endl;
```

```
cout<<"Q1) Is John a Student?(y/n)\n";
cin>>answer;
if(answer=='y'){
cout<<"John enjoys Student Life"<<endl;
cout<<"John Enjoys Student Life --> John Meets Friends"<<endl;
cout<<"John Enjoys Meets Friends --> John Needs Money"<<endl;
cout<<"Q2) Does John has a job?(y/n)";
cin>>answer;
if(answer=='y')
{
cout<<"John Has Job --> John Has Free Time"<<endl;
cout<<"Since John Works in Free Time--> John Is Not Not Good In Studies "<<endl;
cout<<"John should not get the scholarship";
}
}
else
cout<<"John Will not recieve scholarship as he is not a student"<<endl;
return 0;
}
```

Output

```
Answer the following question to determine whether JOHN should get scholarship o
r not?
Q1) Is John a Student?(y/n)
y
John enjoys Student Life
John Enjoys Student Life --> John Meets Friends
John Enjoys Meets Friends --> John Needs Money
Q2) Does John has a job?(y/n)y
John Has Job --> John Has Free Time
Since John Works in Free Time--> John Is Not Not Good In Studies
John should not get the scholarship
```

Fig.5.1

PRACTICAL NO.6

AIM: - Write a program to implement heuristic search procedure.

Program

```
#include  <iostream>
using namespace std;
int main()
{
  string a,b,c,d,e;
  int t,u,v,w,z,p,q,r;
  cout<< "Enter the root node"<< endl;
  cin>> a;
  cout<<" Enter the value of root node"<< endl;
  cin>>t;
  cout<< "Enter child 1"<< endl;
  cin>>b;
  cout<< "Enter value from root to child 1"<<
  endl; cin>>u;
  cout<< "Enter value from child1 to terminal"<<
  endl; cin>> p;
  cout<< "Enter child 2"<< endl;
  cin>>c;
  cout<< "Enter value of child 2"<< endl;
  cin>>v;
  cout<< "Enter value from child2 to terminal"<<
  endl; cin>> q;
  cout<< "Enter child 3"<< endl;
  cin>>d;
  cout<< "Enter value of child 3"<< endl;
  cin>>w;
  cout<< "Enter value from child3 to terminal"<<
  endl; cin>> r;
  cout<< "Enter terminal node"<< endl;
  cin>> e;
   if((u+p)<(v+q)&& (u+p)<(w+r))
  {
      cout<< "Heuristic path is from a to b to e"<<
      endl; cout<< "Total value is"<< t+u+p;
  }
  else if((v+q)<(u+p)&&(v+p)<(w+r))
  {
     cout<< "Heuristic path is from a to c to e"<<
     endl; cout<< "Total value is"<< t+v+p;
  }
 else if((w+r)<(u+p)&& (w+r)<(v+q))
  {
     cout<< "Heuristic path is from a to d to e"<<
     endl; cout<< "Total value is"<< t+w+r<< endl;
  }
  return 0;
  }
```

OUTPUT:

```
Enter the root node
a
 Enter the value of root node
4
Enter child 1
b
Enter value from root to child 1
1
Enter value from child1 to terminal
2
Enter child 2
c
Enter value of child 2
2
Enter value from child2 to terminal
4
Enter child 3
d
Enter value of child 3
3
Enter value from child3 to terminal
5
Enter terminal node
e
Heuristic path is from a to b to e
Total value is7sh-4.3$
```

Fig.6.1

PRACTICAL NO.7

AIM: - Write a program to implement expert system.

Program

```
#include <iostream>
using namespace std;
void measels(char,char,char,char,char);
void flu(char,char,char,char,char,char,char,char);
void cold(char,char,char,char,char);
void chickenpox(char,char,char,char);
int main()
{
  char name[50];
  char a,b,c,d,e,f,g,h,i,j,k;
  cout << "Please enter your name.. " << endl;
  cin>> name;
  cout << "Do you have fever? (y/n)"<< endl;
  cin>>a;
  cout << "Do you have rashes? (y/n)"<< endl;
  cin>>b;
  cout << "Do you have headache? (y/n)"<< endl;
  cin>>c;
  cout << "Do you have running nose? (y/n)"<<
  endl; cin>>d;
  cout << "Do you have conjunctivities? (y/n)"<<
  endl; cin>>e;
  cout << "Do you have cough? (y/n)"<< endl;
  cin>>f;
  cout << "Do you have ache? (y/n)"<< endl;
  cin>>g;
  cout << "Do you have chills? (y/n)"<< endl;
  cin>>h;
  cout << "Do you have swollen glands? (y/n)"<<
  endl; cin>>i;
  cout << "Do you have snezzing? (y/n)"<< endl;
  cin>>j;
  cout << "Do you have sore throat? (y/n)"<< endl;
  cin>>k;
  measels(a,f,e,d,b);
  flu(a,c,g,e,h,k,f,d);
  cold(c,j,k,d,h);
  chickenpox(a,h,g,b);
  return 0;
} void measels(char q,char w,char r,char t,char y)
{
 if(q=='y'&&w=='y'&& r=='y' && t=='y' && y==
    'y') cout<< "You may have measels."<< endl;
    else
    cout<< "";
}
void flu(char q,char w,char r,char t,char y,char p,char l,char x)
{
   if(q=='y'&&w=='y'&& r=='y' && t=='y' && y== 'y'&& p=='y' && l=='y' &&
    x=='y') cout<< "You may have flu."<< endl;
    else
    cout<< "";
}
```

OUTPUT:

```
sh-4.3$ g++ -o main *.cpp
sh-4.3$ main
Please enter your name..
shiffali
Do you have fever? (y/n)
y
Do you have rashes? (y/n)
y
Do you have headache? (y/n)
n
Do you have running nose? (y/n)
n
Do you have conjunctivities? (y/n)
y
Do you have cough? (y/n)
n
Do you have ache? (y/n)
y
Do you have chills? (y/n)
y
Do you have swollen glands? (y/n)
n
Do you have snezzing? (y/n)
n
Do you have sore throat? (y/n)
n
You may have chicken-pox.
sh-4.3$
```

Fig.7.1

PRACTICAL NO.8

AIM: - WAP to implement search problem of 3 x 3 puzzle.

Program

```
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;
void printarray(int num[3][3])
{
   for(int i = 0;i < 3; i++)
   {
      for(int j = 0; j< 3; j++)
      {
         cout << num[i][j] << " ";
      }
      cout << "\n";
   }
}

void swapValues(int (&array)[3][3], int i1,int j1,int i2,int j2)
{
   int temp = array[i1][j1];
   array[i1][j1] = array[i2][j2];
   array[i2][j2] = temp;
}

void moveleft(int (&array)[3][3], int i, int j)
{
   if(i==1&&j==1)
   {
      swapValues(array, i, j, i-1, j);
      swapValues(array, i, j, i+1, j);
   }
   else if(i==1&&j==2)
   {
      swapValues(array, i, j, i, j-1);
      swapValues(array, i, j, i, j-2);
   }
   else if(i==0&&j==0)
   {
      swapValues(array, i, j+2, i, j);
      swapValues(array, i, j+1, i, j);
   }
   else if(i==0&&j==2)
   {
      swapValues(array, i, j, i, j);
   }
   else
      swapValues(array, i, j, i-1, j);
      swapValues(array, i, j, i+1, j);

}

void movedown(int (&array)[3][3], int i, int j)
{
   if(i==0&&j==0)
```

```
    {
      swapValues(array, i, j, i+1, j);
      swapValues(array, i, j, i+2, j);
    }
    else if(i==1&&j==1)
    {
      swapValues(array, i+1, j, i, j);
      swapValues(array, i-1, j, i, j);
    }
    else if(i==0&&j==2)
    {
      swapValues(array, i, j, i+1, j);
    }
    else
      swapValues(array, i, j, i-1, j);
      swapValues(array, i, j, i+1, j);
}

void moveright(int (&array)[3][3], int i, int j)
{
    if(i==0&&j==0)
    {
      swapValues(array, i, j+1, i, j);
      swapValues(array, i, j+2, i, j);
    }
    else if(i==1&&j==1)
    {
      swapValues(array, i, j+1, i, j);
      swapValues(array, i, j-1, i, j);
    }
    else if(i==1&&j==2)
    {
      swapValues(array, i, j-2, i, j);
      swapValues(array, i, j-1, i, j);
    }
    else
      swapValues(array, i, j, i-1, j);
      swapValues(array, i, j, i+1, j);
}

void moveup(int (&array)[3][3], int i, int j)
{
    if(i==0&&j==0)
    {
      swapValues(array, i, j, i+2, j);
      swapValues(array, i, j, i+1, j);
    }
    else if(i==1&&j==1)
    {
      swapValues(array, i, j, i-1, j);
      swapValues(array, i, j, i+1, j);
    }
    else if(i==0&&j==2)
    {
      swapValues(array, i, j, i+1, j);
      swapValues(array, i, j, i+2, j);
```

```
        swapValues(array, i, j, i+1, j);
    }
    else
        swapValues(array, i, j, i-1, j);
        swapValues(array, i, j, i+2, j);
    //int coord1d = j * 3 + i;
}
class Puzzle
{
public:
    Puzzle();
    void swapValues(int num[3][3], int i1, int j1, int i2, int
    j2); //void moveleft(int[3][3], int start1, int start2);
    void moveup(int (&array)[3][3], int i, int j);
    void moveright(int (&array)[3][3], int i, int j);
    void moveleft(int (&array)[3][3], int i, int j);
    void movedown(int (&array)[3][3], int i, int j);
    void printarray(int[3][3]);
};
int main()
{
    int state1[3][3] = {{2, 8, 3}, {1, 6, 4}, {7, 0, 5}};
    int state2[3][3] = {{2, 8, 1}, {4, 6, 3}, {0, 7, 5}};
    int gstate[3][3] = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}};
    cout << "this is state 1" << endl;
    printarray(state1);
    cout << "\n\n";
    cout << "now this is state 2" << endl;
    printarray(state2);
    cout << "\n\n";
    cout << "now this is the goal state" << endl;;
    printarray(gstate);
    int start1, start2;
    cout << endl << endl << "please enter your starting point in the states listed above going up or
down" << endl;
    cin >> start1;
    cout << "now enter the point going left to right" <<
    endl; cin >> start2;
    //moveleft(state1, start1, start2);
    cout << "this is moving coordinate (1, 1) upwards" <<
    endl; moveup(state1, start1, start2);
    printarray(state1);
    cout << endl;
    cout << "this is moving coordinate (1, 1) to the left" <<
    endl; moveleft(state1, start1, start2);
    printarray(state1);
    cout << endl;
    cout << "this is moving coordinate (1, 1) downwards" <<
    endl; movedown(state1, start1, start2);
    printarray(state1);
    cout << endl;
    cout << "this is moving coordinate (1, 1) to the right" << endl;
    moveright(state1, start1, start2);
    printarray(state1);
    cout << endl;
    cin.get();
    cin.get();
    return 0;
```

}

OUTPUT:

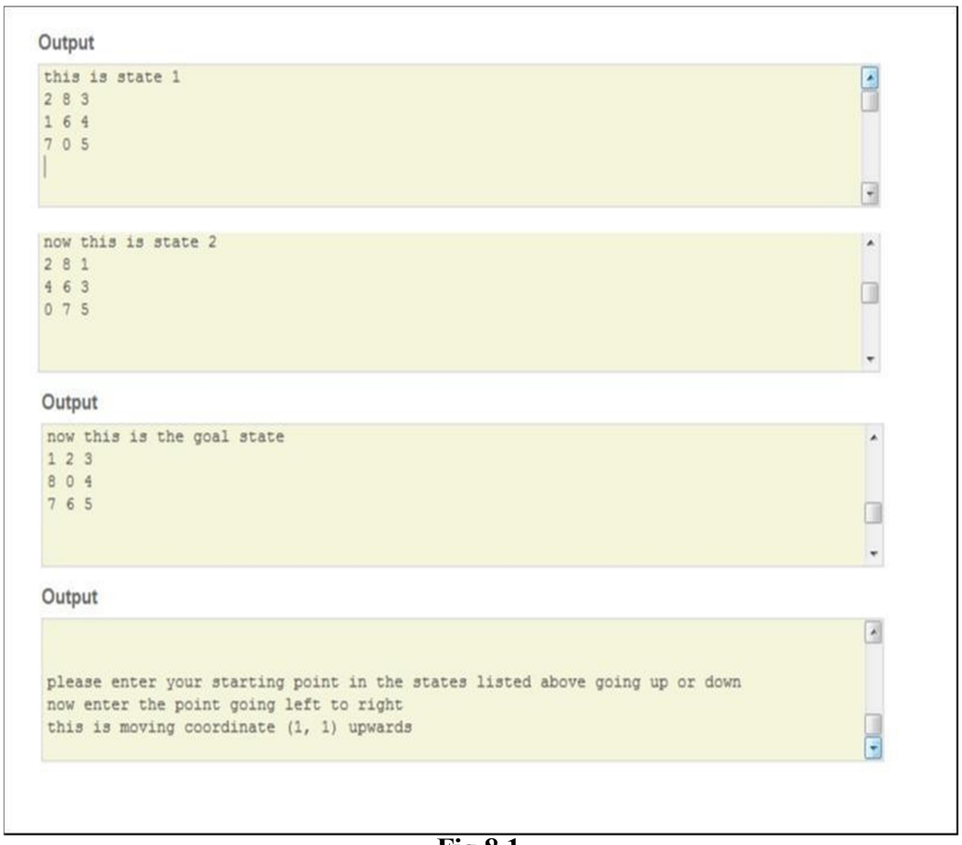

Fig 8.1

PRACTICAL NO.9

AIM: - Program to implement a* algorithm.

Program

```
#include <iostream>
#include <iomanip>
#include <queue>
#include <string>
#include <math.h>
#include <ctime>
using namespace std;
const int n=60; // horizontal size of the map
const int m=60; // vertical size size of the map
static int map[n][m];
static int closed_nodes_map[n][m]; // map of closed (tried-out) nodes
static int open_nodes_map[n][m]; // map of open (not-yet-tried)
nodes static int dir_map[n][m]; // map of directions
const int dir=8; // number of possible directions to go at any
position // if dir==4
//static int dx[dir]={1, 0, -1, 0};
//static int dy[dir]={0, 1, 0, -1};
// if dir==8
static int dx[dir]={1, 1, 0, -1, -1, -1, 0, 1};
static int dy[dir]={0, 1, 1, 1, 0, -1, -1, -1};
class node
{
   3.current
   position int xPos;
   int yPos;
   4.total distance already travelled to reach the node
   int level;
   5.priority=level+remaining distance estimate
   int priority; // smaller: higher priority

   public:
      node(int xp, int yp, int d, int p)
         {xPos=xp; yPos=yp; level=d; priority=p;}

      int getxPos() const {return xPos;}
      int getyPos() const {return yPos;}
      int getLevel() const {return level;}
      int getPriority() const {return priority;}

      void updatePriority(const int & xDest, const int & yDest)
      {
         priority= level+estimate(xDest, yDest)*10; //A*
      }

      // give better priority to going strait instead of diagonally
      void nextLevel(const int & i) // i: direction
      {
         level+=(dir==8?(i%2==0?10:14):10);
      }

      // Estimation function for the remaining distance to the goal. const
      int & estimate(const int & xDest, const int & yDest) const
      {
```

```
            static int xd, yd, d;
            xd=xDest-xPos;
            yd=yDest-yPos;

            // Euclidian Distance
            d=static_cast<int>(sqrt(xd*xd+yd*yd));

            // Manhattan distance
            //d=abs(xd)+abs(yd);

            // Chebyshev distance
            //d=max(abs(xd), abs(yd));

            return(d);
        }
};

// Determine priority (in the priority queue) bool
operator<(const node & a, const node & b)
{
  return a.getPriority() > b.getPriority();
}

// A-star algorithm.
// The route returned is a string of direction digits.
string pathFind( const int & xStart, const int & yStart,
                 const int & xFinish, const int & yFinish )
{
    static priority_queue<node> pq[2]; // list of open (not-yet-tried) nodes
    static int pqi; // pq index
    static node* n0;
    static node* m0;
    static int i, j, x, y, xdx, ydy;
    static char c;
    pqi=0;

    // reset the node maps
    for(y=0;y<m;y++)
    {
        for(x=0;x<n;x++)
        {
            closed_nodes_map[x][y]=0;
            open_nodes_map[x][y]=0;
        }
    }

    // create the start node and push into list of open
    nodes n0=new node(xStart, yStart, 0, 0);
    n0->updatePriority(xFinish,
    yFinish); pq[pqi].push(*n0);
    open_nodes_map[x][y]=n0->getPriority(); // mark it on the open nodes map




    // A* search
    while(!pq[pqi].empty())
```

```
{
   // get the current node w/ the highest priority
   // from the list of open nodes
   n0=new node( pq[pqi].top().getxPos(), pq[pqi].top().getyPos(),

            pq[pqi].top().getLevel(), pq[pqi].top().getPriority());

   x=n0->getxPos(); y=n0->getyPos();

   pq[pqi].pop(); // remove the node from the open
   list open_nodes_map[x][y]=0;
   // mark it on the closed nodes
   map closed_nodes_map[x][y]=1;

   // quit searching when the goal state is reached
   //if((*n0).estimate(xFinish, yFinish) == 0)
   if(x==xFinish && y==yFinish)
   {
      // generate the path from finish to start
      // by following the directions
      string path="";
      while(!(x==xStart && y==yStart))
      {
         j=dir_map[x][y];
         c='0'+(j+dir/2)%dir;
         path=c+path;
         x+=dx[j];
         y+=dy[j];
      }

      // garbage collection
      delete n0;
      // empty the leftover nodes
      while(!pq[pqi].empty()) pq[pqi].pop();
      return path;
   }

   // generate moves (child nodes) in all possible
   directions for(i=0;i<dir;i++)
   {
      xdx=x+dx[i]; ydy=y+dy[i];

      if(!(xdx<0 || xdx>n-1 || ydy<0 || ydy>m-1 ||
         map[xdx][ydy]==1 || closed_nodes_map[xdx][ydy]==1))
      {
         // generate a child node
         m0=new node( xdx, ydy, n0->getLevel(),
                  n0->getPriority());
         m0->nextLevel(i);
         m0->updatePriority(xFinish, yFinish);

         // if it is not in the open list then add into
         that if(open_nodes_map[xdx][ydy]==0)
         {
            open_nodes_map[xdx][ydy]=m0-
            >getPriority(); pq[pqi].push(*m0);
            // mark its parent node direction
            dir_map[xdx][ydy]=(i+dir/2)%dir;
         }
         else if(open_nodes_map[xdx][ydy]>m0->getPriority())
         {
```

```
                // update the priority info
                open_nodes_map[xdx][ydy]=m0->getPriority();

                // update the parent direction info
                dir_map[xdx][ydy]=(i+dir/2)%dir;

                // replace the node
                // by emptying one pq to the other one
                // except the node to be replaced will be ignored
                // and the new node will be pushed in instead
                while(!(pq[pqi].top().getxPos()==xdx &&
                      pq[pqi].top().getyPos()==ydy))
                {
                   pq[1-pqi].push(pq[pqi].top());
                   pq[pqi].pop();
                }
                pq[pqi].pop(); // remove the wanted node

                // empty the larger size pq to the smaller one
                if(pq[pqi].size()>pq[1-pqi].size()) pqi=1-pqi;
                while(!pq[pqi].empty())
                {
                   pq[1-pqi].push(pq[pqi].top());
                   pq[pqi].pop();
                }
                pqi=1-pqi;
                pq[pqi].push(*m0); // add the better node instead
             }
             else delete m0; // garbage collection
          }
       }
       delete n0; // garbage collection
    }
    return ""; // no route found
}

int main()
{
    srand(time(NULL));

    // create empty map
    for(int y=0;y<m;y++)
    {
       for(int x=0;x<n;x++) map[x][y]=0;
    }

    // fillout the map matrix with a '+' pattern
    for(int x=n/8;x<n*7/8;x++)
    {
       map[x][m/2]=1;
    }
    for(int y=m/8;y<m*7/8;y++)
    {
       map[n/2][y]=1;
    }

    // randomly select start and finish
    locations int xA, yA, xB, yB;
    switch(rand()%8)
    {
```

```
        case 0: xA=0;yA=0;xB=n-1;yB=m-1; break;
        case 1: xA=0;yA=m-1;xB=n-1;yB=0; break;
        case 2: xA=n/2-1;yA=m/2-1;xB=n/2+1;yB=m/2+1; break;
        case 3: xA=n/2-1;yA=m/2+1;xB=n/2+1;yB=m/2-1; break;
        case 4: xA=n/2-1;yA=0;xB=n/2+1;yB=m-1; break;
        case 5: xA=n/2+1;yA=m-1;xB=n/2-1;yB=0; break;
        case 6: xA=0;yA=m/2-1;xB=n-1;yB=m/2+1; break;
        case 7: xA=n-1;yA=m/2+1;xB=0;yB=m/2-1; break;
    }
    cout<<"Map Size (X,Y): "<<n<<","<<m<<endl;
    cout<<"Start: "<<xA<<","<<yA<<endl;
    cout<<"Finish: "<<xB<<","<<yB<<endl;
    // get the route
    clock_t start = clock();
    string route=pathFind(xA, yA, xB, yB);
    if(route=="") cout<<"An empty route
    generated!"<<endl; clock_t end = clock();
    double time_elapsed = double(end - start);
    cout<<"Time to calculate the route (ms):
    "<<time_elapsed<<endl; cout<<"Route:"<<endl;
    cout<<route<<endl<<endl;

    // follow the route on the map and display
    it if(route.length()>0)
    {
        int j; char c;
        int x=xA; int
        y=yA;
        map[x][y]=2;
        for(int i=0;i<route.length();i++)
        {
            c =route.at(i);
            j=atoi(&c);
            x=x+dx[j];
            y=y+dy[j];
            map[x][y]=3;
        }
        map[x][y]=4;

        // display the map with the
        route for(int y=0;y<m;y++)
        {
for(int x=0;x<n;x++)
            if(map[x][y]==0)
                cout<<".";
            else if(map[x][y]==1)
                cout<<"O"; //obstacle
            else if(map[x][y]==2)
                cout<<"S";  //start
            else if(map[x][y]==3)
                 cout<<"R"; //route
            else if(map[x][y]==4)
                 cout<<"F"; //finish
          cout<<endl;
        }
    }
    getchar(); // wait for a (Enter) keypress
    return(0);
}
```

OUTPUT:

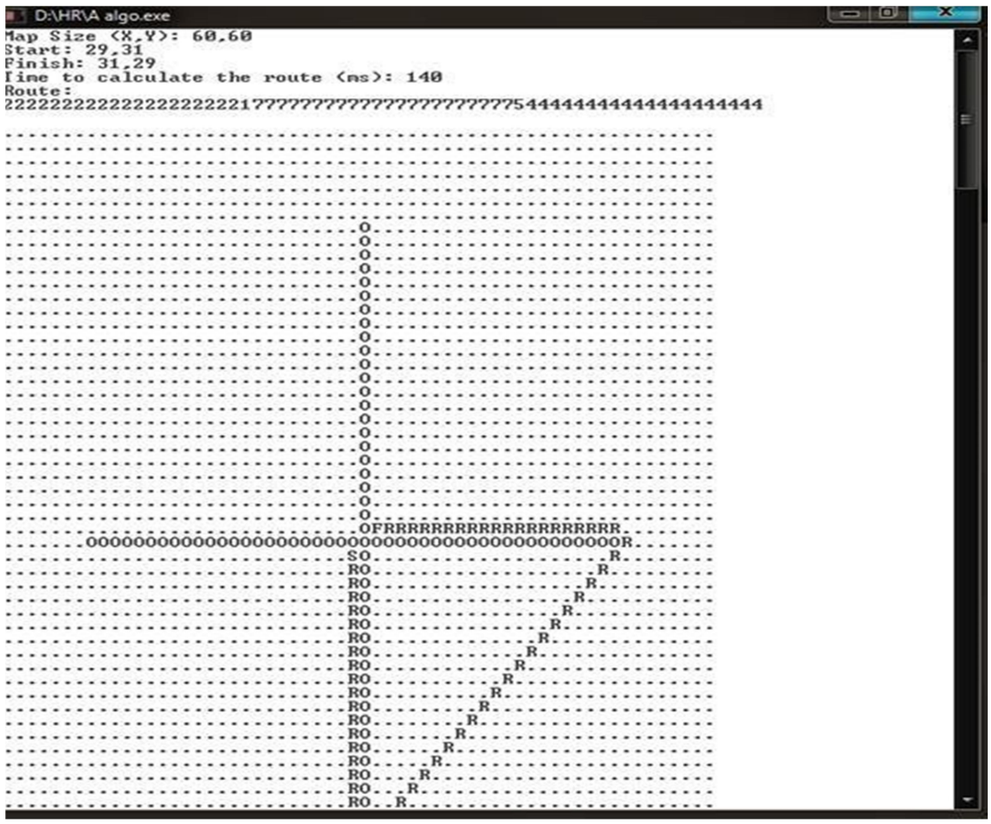

Fig.9 .1