PRACTICAL FILE

COURSE : ARTIFICIAL INTELLIGENCE

SUBJECT CODE : MC 307

B.TECH SEMESTER-V

DEPARTMENT OF APPLIED MATHEMATICS

DELHI TECHNOLOGICAL UNIVERSITY

(FORMELY DELHI COLLEGE OF ENGINEERING)

BAWANA ROAD , DELHI – 110042

SUBMITED BY : LOKESH GODARA

ROLL NO . : 23/MC/082

SUBMITED TO : Ms. REETIKA SINGH

Ms. DRISHTI

INDEX

S.No.	Experiments	Date	Remarks
1.	Write a program to solve the 8-Puzzle problem using Generate and Test Strategy.		
2.	Write a program to solve the 8-Puzzle problem using DFID Technique.		
3.	Write a program to solve the 3-SAT Problem using Variable Neighbourhood Descent Algorithm.		
4.	Write a program to solve the 3- SAT Problem using Stochastic Hill Climbing Algorithm.		
5.	Write a program to solve the 8-Puzzle problem using A* algorithm.		
6.	Write a program to solve AND OR Graph using AO* Search Algorithm.		
7.	WAP to find maximum of two/three numbers.		
8.	WAP to find factorial of a number.		
9.	WAP to find sum of first N numbers.		
10.	Write a program to find Fibonacci sequence upto Nth term.		
11	Write a program to implement decision tree and naive bayes algorithm and produce the training and testing performance in the form of accuracy and F-score on the IRIS.csv dataset		

Experiment 1

Aim: Write a program to solve the 8-Puzzle problem using Generate and Test Strategy.

Code:

```
#include<bits/stdc++.h>
using namespace std;

int solve8Puzzle(vector<vector<int>> &grid){
   vector<vector<int>> dir={{1,3},{0,2,4},{1,5},{0,4,6},{1,3,5,7},{2,4,8},{3,7},{4,6,8},{5,7}};
string start="", goal="123804765";
   int zeroPos;

   for(int i=0;i<3;i++){        for(int
j=0;j<3;j++){
start+=to_string(grid[i][j]);
          if(grid[i][j]==0) zeroPos=3*i+j;
      }
   }

   queue<pair<string,int>> q;
unordered_set<string> visited;
q.emplace(start,zeroPos);     visited.insert(start);
   int moves=0, sz;

   while(!q.empty()){        sz=q.size();
while(sz--){            auto ele=q.front();
q.pop();            string state=ele.first;
int pos=ele.second;            if(state==goal)
return moves;            for(int
next:dir[pos]){
swap(state[pos],state[next]);
if(!visited.count(state)){
visited.insert(state);
                q.emplace(state,next);
             }
             swap(state[pos],state[next]);
          }
       }
       moves++;
   }
   return -1;
}

int main(){
   vector<vector<int>> grid(3,vector<int>(3));     cout<<"Enter
the initial 3x3 grid (use 0 for blank):\n";     for(int i=0;i<3;i++)
      for(int j=0;j<3;j++) cin>>grid[i][j];
```

```
    int result=solve8Puzzle(grid);
    if(result!=-1) cout<<"Solved in "<<result<<" moves.\n";     else
cout<<"Unsolvable puzzle.\n";

    return 0;
}
```

Output:

```
Enter the initial 3x3 grid (use 0 for blank):
1 2 4
3 0 8
5 7 6
Solved in 57 moves.
```

Experiment 2

Aim: Write a program to solve the 8-Puzzle problem using DFID Technique.

Code:

```
#include<bits/stdc++.h>
using namespace std;

vector<vector<int>> dir={{1,3},{0,2,4},{1,5},{0,4,6},{1,3,5,7},{2,4,8},{3,7},{4,6,8},{5,7}}; string
goal="123804765";

bool DLS(string state, int pos, int depth, unordered_set<string> &visited){     if(state==goal) return true;
   if(depth==0) return false;

   visited.insert(state);     for(int next:dir[pos]){
swap(state[pos],state[next]);
if(!visited.count(state)){
         if(DLS(state, next, depth-1, visited)) return true;
      }
      swap(state[pos],state[next]);
   }
   visited.erase(state);
   return false;
}

int DFID(vector<vector<int>> &grid){     string
start="";     int zeroPos;     for(int i=0;i<3;i++)
for(int j=0;j<3;j++){            start +=
to_string(grid[i][j]);
         if(grid[i][j]==0) zeroPos = 3*i + j;
      }

   for(int depth=0; depth<=50; depth++){        unordered_set<string>
visited;        if(DLS(start, zeroPos, depth, visited)) return depth;
   }
   return -1;
}

int main(){
   vector<vector<int>> grid(3,vector<int>(3));     cout<<"Enter the initial 3x3
grid (use 0 for blank):\n";     for(int i=0;i<3;i++)
      for(int j=0;j<3;j++) cin>>grid[i][j];

   int result = DFID(grid);     if(result != -1) cout<<"Solved in "<<result<<" moves using
DFID.\n";     else cout<<"Unsolvable puzzle (or exceeds depth limit).\n";

   return 0;
}
```

Output

```
Enter the initial 3x3 grid (use 0 for blank):
1 2 4
3 0 8
5 7 6
Solved in 33 moves using DFID.
```

Experiment 3

Aim: Write a program to solve the 3-SAT Problem using Variable Neighbourhood Descent Algorithm.

Code:

```
#include<bits/stdc++.h>
using namespace std;

using Clause = vector<int>;
using Formula = vector<Clause>;

int evaluate(Formula &formula, vector<bool> &assignment) {     int
satisfied = 0;
   for(const auto &clause : formula){         for(int lit : clause){
int var = abs(lit) - 1;             bool val = (lit > 0) ? assignment[var] :
!assignment[var];             if(val){ satisfied++; break; }
       }
   }
   return satisfied;
}

bool VND(const Formula &formula, int nVars, vector<bool> &assignment){
int totalClauses = formula.size();     int bestScore = evaluate(formula,
assignment);

   while(true){         bool improved =
false;         for(int i=0;i<nVars;i++){
assignment[i] = !assignment[i];
          int newScore = evaluate(formula, assignment);
if(newScore > bestScore){                bestScore =
newScore;                improved = true;
             break;
          } else assignment[i] = !assignment[i];
       }
       if(!improved) break;
   }

   return bestScore == totalClauses;
}

int main(){
   int nVars=3, nClauses;
   cout << "Enter number of clauses: ";     cin
>> nClauses;

   Formula formula(nClauses);     cout << "Enter clauses
(use negative for negation):\n";     for(int
i=0;i<nClauses;i++){         Clause clause(3);
       for(int j=0;j<3;j++) cin >> clause[j];
formula[i] = clause;
```

```
    }

    for(int attempt=0;attempt<1000;attempt++){          vector<bool> assignment(nVars);
        for(int i=0;i<nVars;i++) assignment[i] = rand()%2;

        if(VND(formula, nVars, assignment)){                cout << "Satisfiable
assignment found:\n";
            for(int i=0;i<nVars;i++) cout << "x" << (i+1) << " = " << assignment[i] << "\n";              return 0;
        }
    }

    cout << "No satisfying assignment found (may be unsatisfiable).\n";      return 0;
}
```

Output:

```
Enter number of clauses: 5
Enter clauses (use negative for negation):
3 -1 2
2 3 -1
-2 1 3
-3 -1 -2
-2 -3 1
Satisfiable assignment found:
x1 = 1
x2 = 1
x3 = 0
```

Experiment-4

Aim: Write a program to solve the 3- SAT Problem using Stochastic Hill Climbing Algorithm.

Code:

```
#include<bits/stdc++.h>
using namespace std;

using Clause = vector<int>;
using Formula = vector<Clause>;

int evaluate(Formula &formula, vector<bool> &assignment) {     int
satisfied = 0;
   for(const auto &clause : formula){
for(int lit : clause){             int var =
abs(lit) - 1;
          bool val = (lit > 0) ? assignment[var] : !assignment[var];
if(val){ satisfied++; break; }
       }
   }
   return satisfied;
}

bool stochasticHillClimbing(Formula &formula, int nVars, vector<bool> &assignment, int maxIter = 10000){
int totalClauses = formula.size();
   int bestScore = evaluate(formula, assignment);

   for(int iter=0;iter<maxIter;iter++){
int var = rand() % nVars;
      assignment[var] = !assignment[var];

      int newScore = evaluate(formula, assignment);
if(newScore >= bestScore){             bestScore =
newScore;
         if(bestScore == totalClauses) return true;
      } else {
         assignment[var] = !assignment[var];
      }
   }

   return false;
}

int main(){
   srand(time(0));

   int nVars, nClauses;        cout << "Enter number of
variables and clauses: ";     cin >> nVars >> nClauses;

   Formula formula(nClauses);     cout << "Enter clauses
(use negative for negation):\n";     for(int
```

```
i=0;i<nClauses;i++){        Clause clause(3);        for(int
j=0;j<3;j++) cin >> clause[j];        formula[i] = clause;
    }

    for(int attempt=0;attempt<100;attempt++){        vector<bool> assignment(nVars);
        for(int i=0;i<nVars;i++) assignment[i] = rand() % 2;

        if(stochasticHillClimbing(formula, nVars, assignment)){            cout << "Satisfiable assignment
found:\n";
            for(int i=0;i<nVars;i++)
                cout << "x" << (i+1) << " = " << assignment[i] << "\n";            return 0;
        }
    }

    cout << "No satisfying assignment found (may be unsatisfiable).\n";    return 0;
}
```

Output:

```
Enter number of clauses: 5
Enter clauses (use negative for negation):
3 -1 2
2 3 -1
-2 1 3
-3 -1 -2
-2 -3 1
Satisfiable assignment found:
x1 = 0
x2 = 0
x3 = 1
```

Experiment-5

Aim: Write a program to solve the 8-Puzzle problem using A* algorithm.

Code:

```
#include<bits/stdc++.h>
using namespace std;

int heuristic(string &state, string &goal){     int cnt=0;
   for(int i=0;i<9;i++) cnt+=(state[i]!=goal[i]);     return cnt;
}

int solve8PuzzleAStar(vector<vector<int>> &grid){     vector<vector<int>>
dir={{1,3},{0,2,4},{1,5},{0,4,6},{1,3,5,7},{2,4,8},{3,7},{4,6,8},{5,7}};     string start="", goal="123804765";
   int zeroPos;

   for(int i=0;i<3;i++)         for(int j=0;j<3;j++){
start+=to_string(grid[i][j]);
         if(grid[i][j]==0) zeroPos=3*i+j;
      }

   using State=tuple<int,int,string,int>; // f, g, state, pos
priority_queue<State,vector<State>,greater<State>> pq;     unordered_set<string> visited;

   int h=heuristic(start,goal);     pq.emplace(h,0,start,zeroPos);
   visited.insert(start);

   while(!pq.empty()){
      State S=pq.top(); pq.pop();
      int f=get<0>(S), g=get<1>(S), pos=get<3>(S);
      string state=get<2>(S);         if(state==goal) return g;
for(int next:dir[pos]){             swap(state[pos],state[next]);
if(!visited.count(state)){                int h=heuristic(state,goal);
pq.emplace(g+1+h,g+1,state,next);
            visited.insert(state);
         }
         swap(state[pos],state[next]);
      }
   }
   return -1;
}

int main(){
   vector<vector<int>> grid(3,vector<int>(3));     cout<<"Enter the initial 3x3 grid
(use 0 for blank):\n";     for(int i=0;i<3;i++)
      for(int j=0;j<3;j++) cin>>grid[i][j];

   int result=solve8PuzzleAStar(grid);
   if(result!=-1) cout<<"Solved in "<<result<<" moves using A*.\n";
   else cout<<"Unsolvable puzzle.\n";

   return 0;
```

}

Output:

```
Enter the initial 3x3 grid (use 0 for blank):
1 2 4
3 0 8
5 7 6
Solved in 22 moves using A*.
```

Experiment 6

Aim: Write a program to solve AND OR Graph using AO* Search Algorithm.

Code:

```
#include <bits/stdc++.h>
using namespace std;

struct Node {     string name;
   vector<vector<string>> children;     vector<int> costs;     bool
solved = false;     int finalCost = INT_MAX;
   vector<string> solution;
};

unordered_map<string, Node> graph;

// Function to recursively apply AO* Search pair<vector<string>, int> aoStar(string nodeName)
{
   Node &node = graph[nodeName];

   if (node.solved) return {node.solution, node.finalCost};

   // Goal node (no children)        if (node.children.empty()) {
node.solved = true;         node.finalCost = 0;         node.solution
= {nodeName};
      return {node.solution, 0};
   }

   int minCost = INT_MAX;
   vector<string> bestSol;

   for (int i = 0; i < node.children.size(); ++i) {         int cost =
node.costs[i];         vector<string> tempSol = {nodeName};
      int subCost = 0;
      bool allSolved = true;

      for (const string &child : node.children[i]) {                  auto ele = aoStar(child);
vector<string> sol=ele.first;                int c=ele.second;                if (graph[child].solved) {
tempSol.insert(tempSol.end(), sol.begin(), sol.end());              subCost += c;
         } else {               allSolved = false;
            break;
         }
      }

      if (allSolved && cost + subCost < minCost) {            minCost = cost + subCost;
         bestSol = tempSol;
      }
   }

   node.solved = true;     node.finalCost = minCost;     node.solution =
bestSol;
   return {bestSol, minCost};
}
```

```
int main() {
    graph["g1"] = {"g1", {}, {}, true, 0, {"g1"}};     graph["g2"] = {"g2", {}, {}, true, 0,
{"g2"}};     graph["g3"] = {"g3", {}, {}, true, 0, {"g3"}};

    graph["b"] = {"b", {{"g1"}}, {1}};
    graph["c"] = {"c", {{"g2", "g3"}}, {2}};

    graph["a"] = {"a", {{"b"}, {"c"}}, {1, 2}};

    auto ele = aoStar("a");      vector<string> solution=ele.first;      int
cost=ele.second;

    cout << "AO* Solution Path: ";     for (auto &node : solution) cout << node << "
";
    cout << "\nTotal Cost: " << cost << endl;

    return 0;
}
```

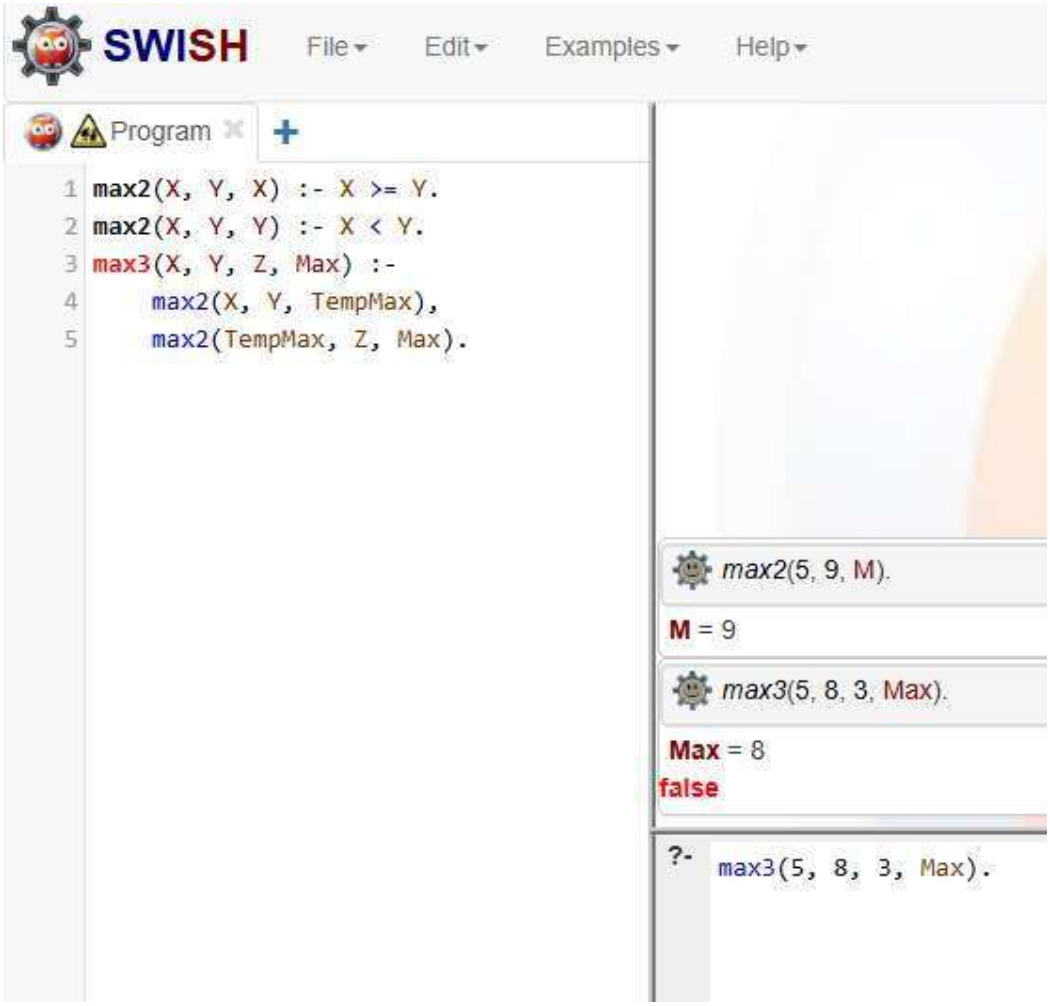
Output:

Experiment-7

Aim: WAP to find maximum of two/three numbers. Code:

```
max2(X, Y, X) :- X >= Y.
max2(X, Y, Y) :- X < Y.
max3(X, Y, Z, Max) :-
max2(X, Y, TempMax),
max2(TempMax, Z, Max).
```

Output:



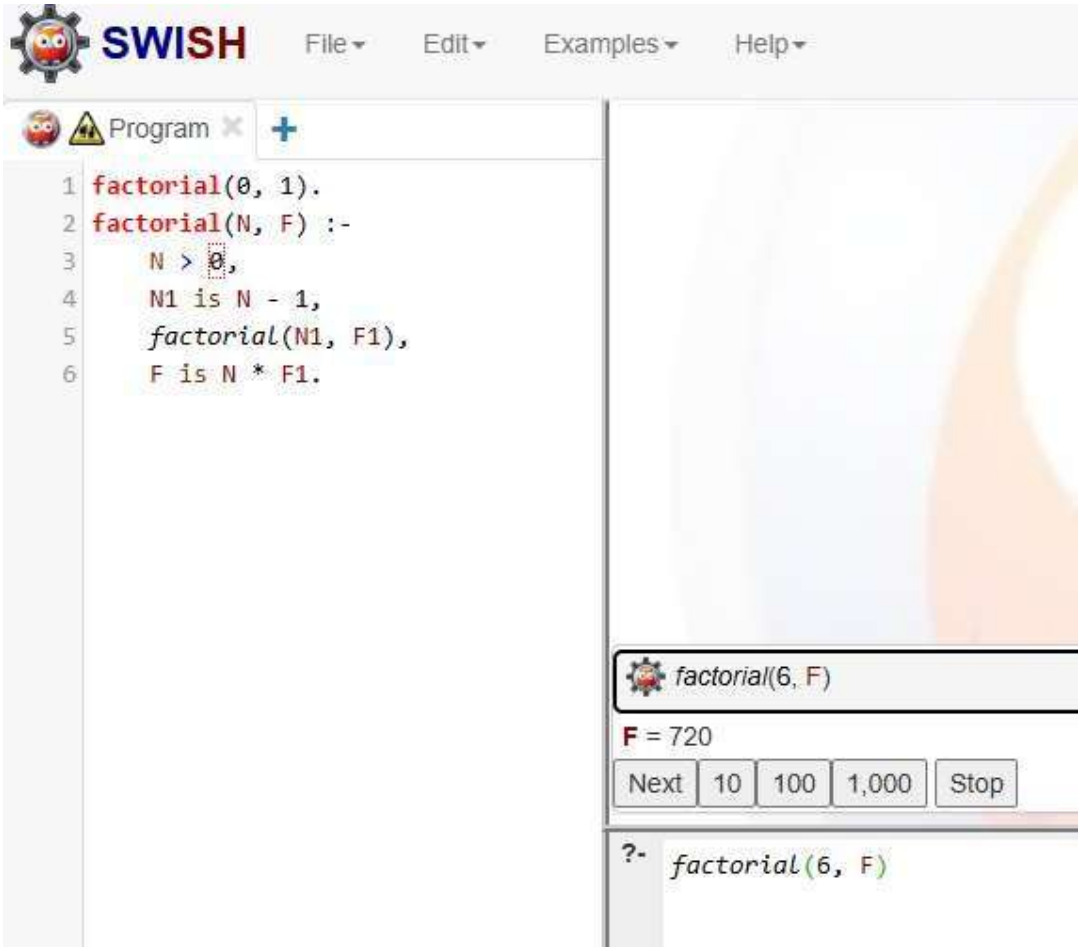
Experiment 8

Aim: WAP to find factorial of a number.

Code:

```
factorial(0, 1). factorial(N, F) :-     N > 0,     N1 is N - 1,     factorial(N1, F1),
F is N * F1.
```

Output:



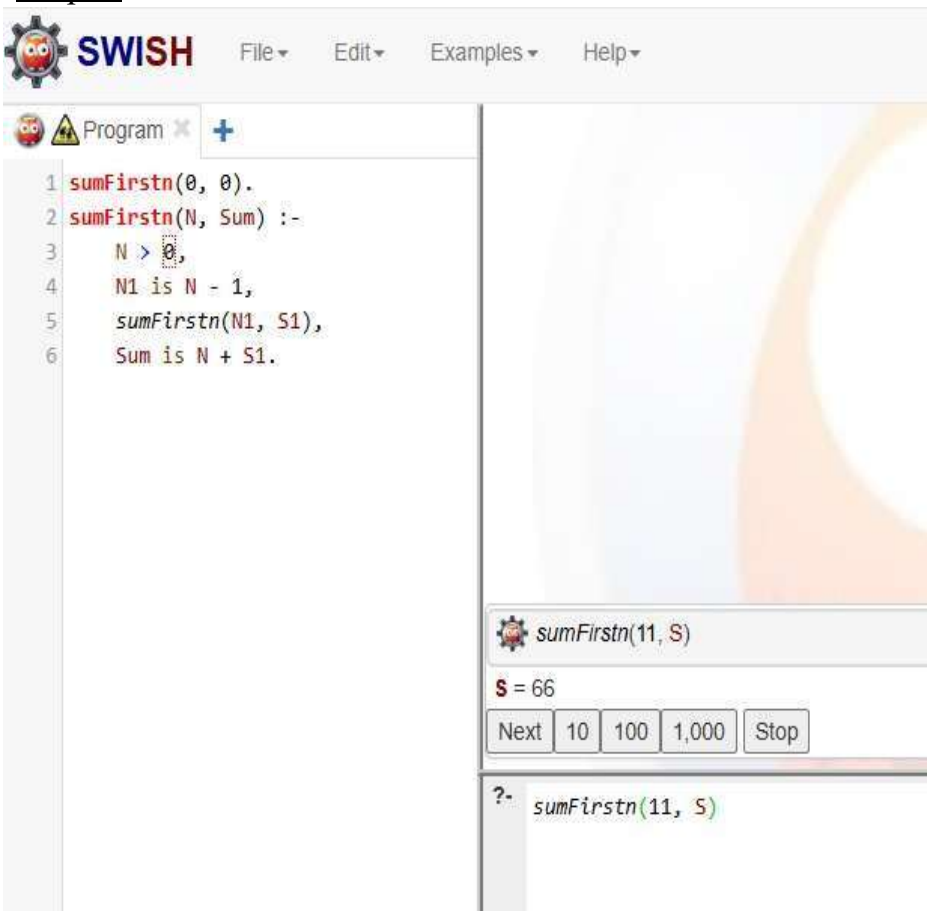
Experiment-9

Aim: WAP to find sum of first N numbers.

Code:

```
sumFirstn(0, 0).
sumFirstn(N, Sum) :-    N > 0,
   N1 is N - 1,    sumFirstn(N1, S1),
Sum is N + S1.
```

Output:



Experiment-10

Aim: Write a program to find Fibonacci sequence upto Nth term.

Code:

```
fibonacci(0, [0]). fibonacci(1, [0,
1]). fibonacci(N, Seq) :-     N > 1,
fibonacci_helper(2, N, [1, 0], SeqRev),
reverse(SeqRev, Seq).

fibonacci_helper(I, N, Acc, Acc) :- I > N.
fibonacci_helper(I, N, [A, B | Rest], Seq) :-     C is
A + B,     I1 is I + 1,     fibonacci_helper(I1, N, [C,
A, B | Rest], Seq).
```

Output: