MC 312: ARTIFICIAL INTELLIGENCE PRACTICAL FILE

DELHI TECHNOLOGICAL UNIVERSITY
(DEPARTMENT OF APPLIED MATHEMATICS)

SUBMITTED TO:
MS. TRASHA GUPTA
DEPARTMENT OF APPLIED MATHEMATICS

SUBMITTED BY:
SIDDHARTH BIHANI
(2K19/MC/125)

PRACTICAL 1

AIM: Write a program to solve the 8-Puzzle problem using Generate and Test Strategy.
(as discussed in the class, given a current state, generate all the possible next states, also known as neighbouring states, and check whether any one of them is final state or not, if not, repeat the procedure with neighbouring states).

You can take the current state as:

8	1	3
4		5
2	7	6

And the final state must be:

1	2	3
8		4
7	6	5

CODE:

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

#define vvi vector<vector<int> >
#define endl "\n"

struct puzzle{
    int loc0x;
    int loc0y;
    vvi currState;
    vector<vvi> movesuntilnow;
    puzzle(){};
    puzzle(vvi &currState){
        this->currState = currState;
        getloc0();
    }
    void getloc0(){
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++){
                if (!currState[i][j]){
                    loc0x = i;
                    loc0y = j;
                    return;
                }
            }
```

```
        }
    }
};

set<vvi> visited;

void inputPuzzle(struct puzzle &p){
    for (int i = 0; i < 3; i++){
        vector<int> temp;
        for (int j = 0; j < 3; j++){
            int x;
            cin >> x;
            temp.push_back(x);
        }
        p.currState.push_back(temp);
    }
    p.getloc0();
    visited.insert(p.currState);
}

void displayPuzzle(struct puzzle &p){
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++)
            cout << p.currState[i][j] << " ";
        cout << endl;
    }
}

void displayMovesTillNow(struct puzzle &p){
    for (int k = 0; k < p.movesuntilnow.size(); ++k){
        cout<<"\nMove "<<k+1<<endl;
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++) cout << p.movesuntilnow[k][i][j] << " ";
            cout << endl;
        }
    }
}

vector<vvi> generateMoves(struct puzzle &final){
    vector<vvi> moves;
    vvi temp = final.currState;
    int x = final.loc0x, y = final.loc0y;
    if(x != 0){
        temp[x][y] = temp[x-1][y];
        temp[x-1][y] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if(x != 2){
        temp[x][y] = temp[x+1][y];
        temp[x+1][y] = 0;
        moves.push_back(temp);
```

```
        temp = final.currState;
    }
    if(y != 0){
        temp[x][y] = temp[x][y-1];
        temp[x][y-1] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if(y != 2){
        temp[x][y] = temp[x][y+1];
        temp[x][y+1] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    return moves;
}

void solvePuzzle(struct puzzle &initial, struct puzzle &final){
    queue<puzzle> q;
    q.push(initial);
    while (!q.empty()){
        puzzle curr = q.front();
        q.pop();
        vector<vvi> moves = generateMoves(curr);
        for (int i = 0; i < moves.size(); i++){
             if(moves[i] == final.currState){
               curr.movesuntilnow.push_back(moves[i]);
               displayMovesTillNow(curr);
               cout<<"Total number of steps to reach the final state:
"<<curr.movesuntilnow.size()<<endl;
               return;
            }
            if (visited.find(moves[i]) == visited.end()){
               puzzle temp(moves[i]);
               temp.movesuntilnow = curr.movesuntilnow;
               temp.movesuntilnow.push_back(moves[i]);
               visited.insert(moves[i]);
               q.push(temp);
            }
        }
    }
}

int main(){
    cout<<endl;
    puzzle initial;
    cout << "Enter the initial puzzle state (Denote blank space using 0):" << endl;
    inputPuzzle(initial);
    cout<<endl;
    puzzle final;
    cout << "Enter the final puzzle state (Denote blank space using 0):" << endl;
    inputPuzzle(final);
```

```
    cout<<endl;
    cout<<"The initial puzzle state is: "<<endl;
    displayPuzzle(initial);
    cout<<endl;
    solvePuzzle(initial, final);
    return 0;
}
```

OUTPUT:

```
siddharth@Siddharths-MacBook-Air lab % cd "/Users/siddharth/Desktop/College/AI/lab/" && g++ Exp1_8PuzzleSolver.cpp -o Exp1_8PuzzleSolver && "/Users/siddhart
h/Desktop/College/AI/lab/"Exp1_8PuzzleSolver

Enter the initial puzzle state (Denote blank space using 0):
8 1 3
4 0 5
2 7 6

Enter the final puzzle state (Denote blank space using 0):
1 2 3
8 0 4
7 6 5
```

```
The initial puzzle state is:
8 1 3
4 0 5
2 7 6


Move 1
8 1 3
0 4 5
2 7 6

Move 2
8 1 3
2 4 5
0 7 6

Move 3
8 1 3
2 4 5
7 0 6

Move 4
8 1 3
2 4 5
7 6 0

Move 5
8 1 3
2 4 0
7 6 5

Move 6
8 1 3
2 0 4
7 6 5

Move 7
8 1 3
0 2 4
7 6 5

Move 8
0 1 3
8 2 4
7 6 5

Move 9
1 0 3
8 2 4
7 6 5

Move 10
1 2 3
8 0 4
7 6 5
Total number of steps to reach the final state: 10
siddharth@Siddharths-MacBook-Air lab %
```

PRACTICAL 2

AIM: Write a program to solve the 8-Puzzle problem using DFID (Depth First Iterative Deepening) Strategy.
You can take the current state as:

8	1	3
4		5
2	7	6

And the final state must be:

1	2	3
8		4
7	6	5

CODE:

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

#define vvi vector<vector<int> >
#define endl "\n"

struct puzzle
{
    int loc0x;
    int loc0y;
    vvi currState;
    vector<vvi> movesuntilnow;
    puzzle(){};
    puzzle(vvi &currState)
    {
        this->currState = currState;
        movesuntilnow.push_back(currState);
        getloc0();
    }
    void getloc0()
    {
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++){
                if (currState[i][j] == 0){
                    loc0x = i;
```

```
                loc0y = j;
                return;
            }
        }
    }
};

set<vvi> visited;

void inputPuzzle(struct puzzle &p)
{
    for (int i = 0; i < 3; i++){
        vector<int> temp;
        for (int j = 0; j < 3; j++){
            int x;
            cin >> x;
            temp.push_back(x);
        }
        p.currState.push_back(temp);
    }
    p.getloc0();
    visited.insert(p.currState);
}

void displayPuzzle(struct puzzle &p){
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++)
            cout << p.currState[i][j] << " ";
        cout << endl;
    }
}

void displayMovesTillNow(struct puzzle &p){
    for (int k = 0; k < p.movesuntilnow.size(); ++k){
        cout<<"\nMove "<<k+1<<endl;
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++) cout << p.movesuntilnow[k][i][j] << " ";
            cout << endl;
        }
    }
}

vector<vvi> generateMoves(struct puzzle &final){
    vector<vvi> moves;
    vvi temp = final.currState;
    int x = final.loc0x, y = final.loc0y;
    if(x != 0){
        temp[x][y] = temp[x-1][y];
        temp[x-1][y] = 0;
        moves.push_back(temp);
        temp = final.currState;
```

```
    }
    if(x != 2){
        temp[x][y] = temp[x+1][y];
        temp[x+1][y] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if(y != 0){
        temp[x][y] = temp[x][y-1];
        temp[x][y-1] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if(y != 2){
        temp[x][y] = temp[x][y+1];
        temp[x][y+1] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    return moves;
}

bool dfs(struct puzzle &final, int d, struct puzzle &asf)
{
    if(d<0)return false;
    if(asf.currState == final.currState){
        displayMovesTillNow(asf);
        cout<<"Total number of steps to reach the final state:
"<<asf.movesuntilnow.size()<<endl;
        return true;
    }
    puzzle curr=asf;
    vector<vvi> moves = generateMoves(asf);
    for (int i = 0; i < moves.size(); i++)
    {
        asf.currState=moves[i];
        asf.getloc0();
        asf.movesuntilnow.push_back(moves[i]);
        if(dfs(final,d-1,asf)){
            return true;
        }
        asf = curr;
    }

    return false;
}

void solvePuzzle(struct puzzle &initial, struct puzzle &final)
{

    int d=0;
    puzzle asf = initial;
```

```
    while(true)
    {
        if(dfs(final,d,asf))
            break;
        d++;
    }
}

int main()
{
    cout<<endl;
    puzzle initial;
    cout << "Enter the initial puzzle state (Denote blank space using 0):" << endl;
    inputPuzzle(initial);
    cout<<endl;
    puzzle final;
    cout << "Enter the final puzzle state (Denote blank space using 0):" << endl;
    inputPuzzle(final);
    cout<<endl;
    cout<<"The initial puzzle state is: "<<endl;
    displayPuzzle(initial);
    cout<<endl;
    solvePuzzle(initial, final);
    return 0;
}
```

OUTPUT:

```
cd "/Users/siddharth/Desktop/College/AI/lab/" && g++ Exp2_8PuzzleSolver_DFID.cpp -o Exp2_8PuzzleSolver_DFID && "/Users/siddharth/Desktop/College/AI/la
b/"Exp2_8PuzzleSolver_DFID
siddharth@Siddharths-MacBook-Air lab % cd "/Users/siddharth/Desktop/College/AI/lab/" && g++ Exp2_8PuzzleSolver_DFID.cpp -o Exp2_8PuzzleSolver_DFID &&
"/Users/siddharth/Desktop/College/AI/lab/"Exp2_8PuzzleSolver_DFID

Enter the initial puzzle state (Denote blank space using 0):
8 1 3
4 0 5
2 7 6

Enter the final puzzle state (Denote blank space using 0):
1 2 3
8 0 4
7 6 5
```

```
The initial puzzle state is:
8 1 3
4 0 5
2 7 6


Move 1
8 1 3
0 4 5
2 7 6

Move 2
8 1 3
2 4 5
0 7 6

Move 3
8 1 3
2 4 5
7 0 6

Move 4
8 1 3
2 4 5
7 6 0

Move 5
8 1 3
2 4 0
7 6 5

Move 6
8 1 3
2 0 4
7 6 5

Move 7
8 1 3
0 2 4
7 6 5

Move 8
0 1 3
8 2 4
7 6 5

Move 9
1 0 3
8 2 4
7 6 5

Move 10
1 2 3
8 0 4
7 6 5
Total number of steps to reach the final state: 10
```

PRACTICAL 3

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

You can take any function in CNF form of your choice or you can take the function F as:
F= (b V c') Λ (c V d') Λ (b') Λ (a' V e') Λ (c' V e) Λ (c' V d').
Let the initial state be {a=1, b=1, c=1, d=1, e=1} and let the heuristic function be number of clauses that are true in any given state.

CODE:

```
#include<bits/stdc++.h>

using namespace std;

vector<vector<char> > simplify(string function) {
    vector<vector<char> > result;
    int i = 0, n = function.size();
    vector<char> temp;
    while (i < n) {
        if (function[i] == ')') {
            result.push_back(temp);
            temp.clear();
            i++;
        }
        else if (function[i] == '^' || function[i] == '(' || function[i] == 'V') {
            i++;
        }
        else {
            if(function[i] == '\'') {
                temp[temp.size()-1] = char(toupper(temp[temp.size()-1]));
            }
            else temp.push_back(function[i]);
            i++;
        }
    }
    return result;
}

int heuristicFunc(vector<bool> &v, vector<vector<char> > simplified_function){
    int count=0;
    for(auto vec: simplified_function){
        bool flag = false;
        for(auto ch: vec){
            if(ch >= 'A' && ch <= 'Z'){
                if(v[ch - 'A'] == false) {
```

```
                flag = true;
                break;
            }
        }
        else if(v[ch - 'a'] == true){
            flag = true;
            break;
        }
    }
    if(flag) count++;
  }
  return count;
}

vector<vector<bool> > moveGen1(vector<bool> &v,int prevHeuCount,
vector<vector<char> > simplified_function)
{
  vector<vector<bool> > moves;
  for(int i=0;i<v.size();i++){
    v[i]=!v[i];
    if(heuristicFunc(v,simplified_function) > prevHeuCount)
      moves.push_back(v);
    v[i]=v[i];
  }
  return moves;
}

vector<vector<bool> > moveGen2(vector<bool> &v,int prevHeuCount,
vector<vector<char> > simplified_function)
{
  vector<vector<bool> > moves;
  for(int i=0;i<v.size();i++){
    v[i]=!v[i];
    for(int j=i+1;j<v.size();j++){
      v[j]=!v[j];
      if(heuristicFunc(v,simplified_function)>prevHeuCount)
        moves.push_back(v);
      v[j]=!v[j];
    }
    v[i]=v[i];
  }
  return moves;
}

void displayMoves(vector<vector<bool> > &asf){
  int sz = asf.size(), x = 0;
  for(auto vec : asf){
    for(auto i : vec) cout<<i<<' ';
    x++;
    if(x == sz) cout<<endl;
    else cout<<"-> ";
  }
```

```
}

bool solve(vector<bool> &v,vector<vector<bool> > &asf, vector<vector<char> >
simplified_function, string function)
{
    asf.push_back(v);
    int currHueCount=heuristicFunc(v, simplified_function);
    if(currHueCount == simplified_function.size()){
        cout<<"\n\nTotal number of steps to reach the final values: "<<asf.size()<<endl;
        cout<<"\nSteps: ";
        displayMoves(asf);
        cout<<"\nFinal values of the variables so that all clauses of the function are true are:
"<<endl;
        for(int i = 0; i < v.size(); i++) cout<<char(i+'a')<<" = "<<asf[asf.size()-1][i]<<endl;
        return true;
    }
    vector<vector<bool> > moves;
    moves=moveGen1(v,currHueCount,simplified_function);
    for(int i=0;i<moves.size();i++)
        if(solve(moves[i],asf, simplified_function, function)==true)return true;
    moves=moveGen2(v,currHueCount,simplified_function);
    for(int i=0;i<moves.size();i++)
        if(solve(moves[i],asf,simplified_function, function)==true)return true;
    asf.pop_back();
    return false;
}

int main(){
    string function;
    cout<<"Enter the function to be solved(without any spaces): ";
    getline(cin, function, '\n');
    int sz = function.size();
    unordered_set<char> variables;
    for(int i = 0; i < sz; i++){
        if((function[i] >= 'a') && (function[i] <= 'z')) variables.insert(function[i]);
    }
    int n = variables.size();
    vector<bool> v(n,true);
    vector<vector<char> > simplified_function = simplify(function);
    cout<<"\n\nPlease enter the Inital values of the variables:- \n";
    for(int i=0; i < n; i++){
        cout<<char(i+97)<<": ";
        int t;
        cin>>t;
        if(t == 1) v[i] = true;
        else v[i] = false;
    }
    vector<vector<bool> > asf;
    solve(v,asf, simplified_function, function);
}
```

OUTPUT:

```
siddharth@Siddharths-MacBook-Air lab % cd "/Users/siddharth/Desktop/College/AI/lab/" && g++ -std=c++17 Exp3_3SATProblem.cpp -o Exp3_3SATProblem && "/U
sers/siddharth/Desktop/College/AI/lab/"Exp3_3SATProblem
Enter the function to be solved(without any spaces): (bVc')^(cVd')^(b')^(a'Ve')^(c'Ve)^(c'Vd')


Please enter the Inital values of the variables:-
a: 1
b: 0
c: 1
d: 1
e: 1


Total number of steps to reach the final values: 4

Steps: 1 0 1 1 1 -> 0 0 1 1 1 -> 1 1 0 0 0 -> 0 0 0 0 0

Final values of the variables so that all clauses of the function are true are:
a = 0
b = 0
c = 0
d = 0
e = 0
siddharth@Siddharths-MacBook-Air lab %
```

PRACTICAL 4

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

You can take any function in CNF form of your choice or you can take the function F as:
F= (b V c') Λ (c V d') Λ (b') Λ (a' V e') Λ (c' V e) Λ (c' V d').
Let the initial state be {a=1, b=1, c=1, d=1, e=1} and let the heuristic function be number of clauses that are true in any given state.
You can use the SIGMOID function as discussed in the class to calculate the probability at every state. Let the value of parameter "T" be 10, and you can choose any probability threshold of your choice for comparison.

CODE:

```
#include<bits/stdc++.h>

using namespace std;

vector<vector<char> > simplify(string function) {
    vector<vector<char> > result;
    int i = 0, n = function.size();
    vector<char> temp;
    while (i < n) {
        if (function[i] == ')') {
            result.push_back(temp);
            temp.clear();
            i++;
        }
        else if (function[i] == '^' || function[i] == '(' || function[i] == 'V') {
            i++;
        }
        else {
            if(function[i] == '\'') {
                temp[temp.size()-1] = char(toupper(temp[temp.size()-1]));
            }
            else temp.push_back(function[i]);
            i++;
        }
    }
    return result;
}

int heuristicFunc(vector<bool> &v, vector<vector<char> > simplified_function){
    int count=0;
    for(auto vec: simplified_function){
        bool flag = false;
        for(auto ch: vec){
```

```
            if(ch >= 'A' && ch <= 'Z'){
                if(v[ch - 'A'] == false) {
                    flag = true;
                    break;
                }
            }
            else if(v[ch - 'a'] == true){
                flag = true;
                break;
            }
        }
        if(flag) count++;
    }
    return count;
}

float prob(vector<bool> &v1, vector<bool> &v2, vector<vector<char> >
simplified_function){
    int T = 10;
    int dE = heuristicFunc(v2, simplified_function)-heuristicFunc(v1, simplified_function);
    float res = 1/(1+exp(-dE*1.0/T));
    return res;
}

pair<int,vector<bool>> generateRand(vector<bool> v){
    int n = v.size(), randInd=rand()%n;
    v[randInd]=v[randInd]^1;
    return {randInd,v};
}

void displayMoves(vector<vector<bool> > &asf){
    int sz = asf.size(), x = 0;
    for(auto vec : asf){
        int n = vec.size(), counter = 0;
        cout<<"(";
        for(auto i : vec){
            counter++;
            cout<<i;
            if(counter == n) cout<<") ";
            else cout<<",";
        }
        x++;
        if(x == sz) cout<<endl;
        else cout<<"-> ";
    }
}

bool solve(vector<bool> &v, vector<vector<bool> > &asf, vector<vector<char> >
simplified_function, string function, float thresholdProb)
{
    asf.push_back(v);
    cout<<"\n\nCurr State: ";
```

```
    for(auto i : v) cout<<i<<' ';
    int currHueCount = heuristicFunc(v, simplified_function);
    if(currHueCount == 6){
        cout<<"\n\nTotal number of steps to reach the final values: "<<asf.size()-1<<endl;
        cout<<"\nSteps: ";
        displayMoves(asf);
        cout<<"\nFinal values of the variables so that all clauses of the function are true are:
"<<endl;
        for(int i = 0; i < v.size(); i++) cout<<char(i+'a')<<" = "<<asf[asf.size()-1][i]<<endl;
        return true;
    }
    set<int> visited;
    while(true){
        pair<int,vector<bool>> resPair=generateRand(v);
        vector<bool> nextRandomState=resPair.second;
        cout<<"\nNext Generated Random State :";
        for(auto i : nextRandomState)cout<<i<<' ';
        cout<<"\nProbability = "<<prob(v, nextRandomState, simplified_function);
        int indexChanged = resPair.first, lastSize = visited.size();
        visited.insert(indexChanged);
        if(visited.size() != lastSize && prob(v, nextRandomState,
simplified_function)>thresholdProb){
            cout<<"\nThe found probability is greater than the threshold probability, hence we
select this and move on to a new state.";
            return solve(nextRandomState, asf, simplified_function, function, thresholdProb);
        }
        if(visited.size() == v.size()) break;
    }
    asf.pop_back();
    return false;
}

int main(){
    string function;
    cout<<"Enter the function to be solved(without any spaces): ";
    getline(cin, function, '\n');
    int sz = function.size();
    unordered_set<char> variables;
    for(int i = 0; i < sz; i++){
        if((function[i] >= 'a') && (function[i] <= 'z')) variables.insert(function[i]);
    }
    int n = variables.size();
    vector<bool> v(n,true);
    vector<vector<char> > simplified_function = simplify(function);
    cout<<"\n\nPlease enter the Inital values of the variables:- \n";
    for(int i = 0; i < n; i++){
        cout<<char(i + 97)<<": ";
        int t;
        cin>>t;
        if(t == 1) v[i] = true;
        else v[i] = false;
    }
```

```
    float thresholdProb;
    cout<<"Enter the threshold probability: ";
    cin>>thresholdProb;
    vector<vector<bool> > asf;
    if(!solve(v, asf, simplified_function, function, thresholdProb))
        cout<<"\n\nProgram Terminated! As there are no further states available with probability greater than the threshold probability, hence we will be unable to reach the goal state.\n";
}
```

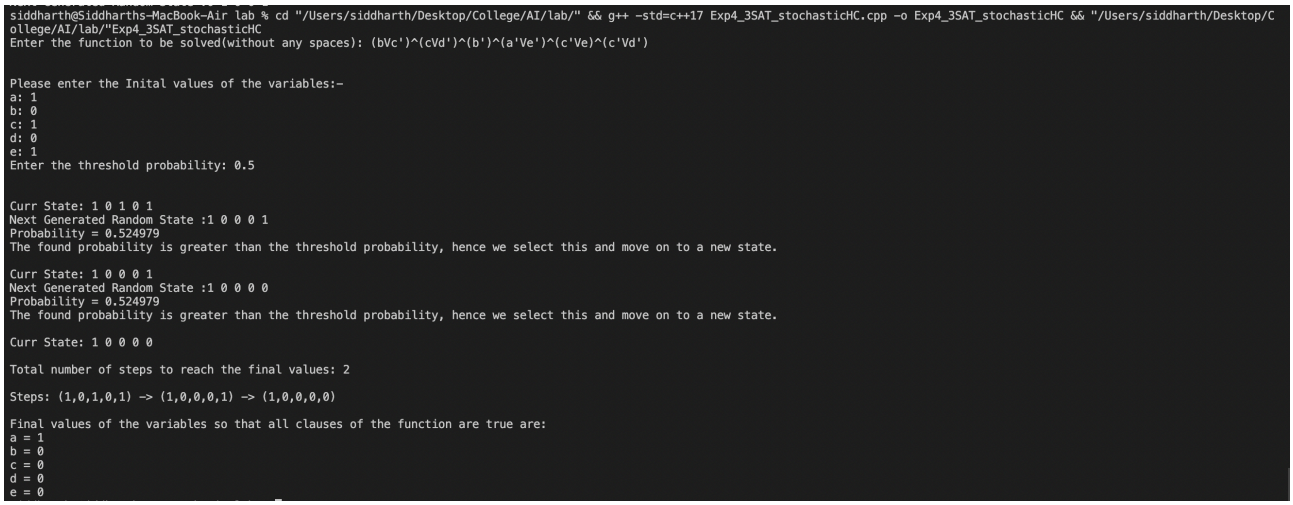
OUTPUT:

For the initial state, {a=1, b=1, c=1, d=1, e=1}:-

```
siddharth@Siddharths-MacBook-Air lab % cd "/Users/siddharth/Desktop/College/AI/lab/" && g++ -std=c++17 Exp4_3SAT_stochasticHC.cpp -o Exp4_3SAT_stochasticHC && "/Users/siddharth/Desktop/College/AI/lab/"Exp4_3SAT_stochasticHC
Enter the function to be solved(without any spaces): (bVc')^(cVd')^(b')^(a'Ve')^(c'Ve)^(c'Vd')


Please enter the Inital values of the variables:-
a: 1
b: 1
c: 1
d: 1
e: 1
Enter the threshold probability: 0.5


Curr State: 1 1 1 1 1
Next Generated Random State :1 1 0 1 1
Probability = 0.5
Next Generated Random State :1 1 1 1 0
Probability = 0.5
Next Generated Random State :1 1 1 0 1
Probability = 0.524979
The found probability is greater than the threshold probability, hence we select this and move on to a new state.

Curr State: 1 1 1 0 1
Next Generated Random State :1 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 1 1 0 1
Probability = 0.524979
The found probability is greater than the threshold probability, hence we select this and move on to a new state.

Curr State: 0 1 1 0 1
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :0 1 1 0 0
Probability = 0.475021
Next Generated Random State :0 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 1 1 0 0
Probability = 0.475021
Next Generated Random State :1 1 1 0 1
Probability = 0.475021
Next Generated Random State :1 1 1 0 1
Probability = 0.475021
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :0 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :0 1 1 0 0
Probability = 0.475021
Next Generated Random State :1 1 1 0 1
Probability = 0.475021
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :0 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 1 1 0 0
Probability = 0.475021
Next Generated Random State :0 1 1 0 0
Probability = 0.475021
Next Generated Random State :0 1 0 0 1
Probability = 0.5
Next Generated Random State :1 1 1 0 1
Probability = 0.475021
Next Generated Random State :0 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 1 1 0 0
Probability = 0.475021
Next Generated Random State :0 1 1 1 1
Probability = 0.475021
Next Generated Random State :0 0 1 0 1
Probability = 0.5

Program Terminated! As there are no further states available with probability greater than the threshold probability, hence we will be unable to reach the goal state.
siddharth@Siddharths-MacBook-Air lab %
```

The program **terminated unsuccessfully**.

Let us take another initial state, {a=1, b=0, c=1, d=0, e=1} :-



The program **terminated successfully**, and the final state was found out to be {a=1, b=0, c=0, d=0, e=0}.

PRACTICAL 5

AIM: Write a program to solve the 8 Puzzle Problem using A* Algorithm.
Given the initial state and final state in 8 puzzle problem , write a program to find the optimal path from initial state to final state using A* algorithm. Consider g(n) = depth of node n and h (n) = number of misplaced tiles

Initial State

Final State

CODE:

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

#define vvi vector<vector<int> >
#define endl "\n"

vvi finalState;

struct puzzle{
    int loc0x;
    int loc0y;
    int gn;
    int hn;
    int fn;
    vvi currState;
    vector<pair<vvi, int> > movestillnow;

    bool compare(vvi &v1, vvi &v2){
        if (v1 == v2) return true;
        else return false;
    }

};

set<vvi> visited;

struct compareFn{
    bool operator()(puzzle &p1, puzzle &p2){
```

```
        return p1.fn > p2.fn;
    }
};

void calcLoc0AndFn(puzzle &p){
    p.hn=0;
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++)
            if(p.currState[i][j]!=finalState[i][j]) p.hn++;
    }
    p.gn=0;
    p.fn=p.hn+p.gn;
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++){
            if (p.currState[i][j] == 0){
                p.loc0x = i;
                p.loc0y = j;
                return;
            }
        }
    }
}

void inputPuzzle(puzzle &p)
{
    cout << "Enter the intial puzzle state(use 0 for empty space):" << endl;
    for (int i = 0; i < 3; i++){
        vector<int> temp(3);
        for (int j = 0; j < 3; j++) cin>>temp[j];
        p.currState.push_back(temp);
    }
    calcLoc0AndFn(p);
    p.movestillnow.push_back({p.currState, p.fn});
}

void displayPuzzle(puzzle &p){
    cout << "The puzzle state is: " << endl;
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++) cout << p.currState[i][j] << " ";
        cout << endl;
    }
}

void displayMovesTillNow(puzzle &p)
{
    for (int k = 0; k < p.movestillnow.size(); ++k)
    {
        if(k==0) cout<<"Intial State, f(n)= "<<p.movestillnow[k].second<<":\n";
        else if(k==p.movestillnow.size()-1) cout<<"Final State, f(n)= "<<p.movestillnow[k].second<<":
\n";
        else cout << "Move " << k  <<", f(n)= "<<p.movestillnow[k].second<<":\n";
        for (int i = 0; i < 3; i++) {
            cout<<setw(25);
            for (int j = 0; j < 3; j++) cout << (p.movestillnow[k].first)[i][j] << " ";
            cout << endl;
        }
        if(k!=p.movestillnow.size()-1){
            cout<<setw(29)<<"  | \n";
            cout<<setw(30)<<"  V  \n";
        }
```

```
    }
}

vector<vvi> generateMoves(puzzle &p)
{
    vector<vvi> moves;
    vvi temp = p.currState;
    int cloc0x = p.loc0x, cloc0y = p.loc0y;
    if (cloc0x != 0){
        temp[cloc0x][cloc0y] = temp[cloc0x - 1][cloc0y];
        temp[cloc0x - 1][cloc0y] = 0;
        moves.push_back(temp);
        temp = p.currState;
    }
    if (cloc0x != 2){
        temp[cloc0x][cloc0y] = temp[cloc0x + 1][cloc0y];
        temp[cloc0x + 1][cloc0y] = 0;
        moves.push_back(temp);
        temp = p.currState;
    }
    if (cloc0y != 0){
        temp[cloc0x][cloc0y] = temp[cloc0x][cloc0y - 1];
        temp[cloc0x][cloc0y - 1] = 0;
        moves.push_back(temp);
        temp = p.currState;
    }
    if (cloc0y != 2){
        temp[cloc0x][cloc0y] = temp[cloc0x][cloc0y + 1];
        temp[cloc0x][cloc0y + 1] = 0;
        moves.push_back(temp);
        temp = p.currState;
    }
    return moves;
}


void solvePuzzle(struct puzzle &p){
    puzzle currentState=p;
    priority_queue<puzzle, vector<puzzle>, compareFn> pq;
    pq.push(currentState);
    while(!pq.empty()){
        currentState=pq.top();
        pq.pop();
        if(visited.find(currentState.currState)!=visited.end()) continue;
        if(currentState.currState==finalState){
            displayMovesTillNow(currentState);
            cout<<"Total number of steps to reach the final state: "<<currentState.movestillnow.size() -
1<<endl;
            return;
        }
        vector<vvi> moves=generateMoves(currentState);
        for(int i=0;i<moves.size();i++){
            puzzle temp = currentState;
            temp.currState=moves[i];
            calcLoc0AndFn(temp);
            temp.movestillnow.push_back({moves[i], temp.fn});
            pq.push(temp);
        }
        visited.insert(currentState.currState);
    }
```

```
}

int main()
{
    cout << "\n\nEnter the final puzzle state(use 0 for empty space):" << endl;
    for (int i = 0; i < 3; i++){
        vector<int> temp;
        for (int j = 0; j < 3; j++){
            int x;
            cin >> x;
            temp.push_back(x);
        }
        finalState.push_back(temp);
    }
    cout<<endl;
    struct puzzle initialState;
    inputPuzzle(initialState);
    cout<<endl;
    solvePuzzle(initialState);
    cout<<endl;
    return 0;
}
```

OUTPUT: