**Experiment Number: 02**

**Problem Statement: 8-puzzle using A\* algorithm.**

**NAME: Priti Patil ROLLNO: 80**

# **CLASS: TY(IT) BATCH: 3**

**DATE OF PERFORMANCE: 15/ 10 / 2022**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Code-**

#include <bits/stdc++.h>

using namespace std;

//defining node structure

struct node

{

vector<vector<int>> arr; // matrix

int level; // current state..g value

int h; // f value

node \*prev; // father node

node()

{

level = 0;

h = 0;

prev = NULL;

}

};

void printMat(vector<vector<int>> v)

{

cout << endl;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

cout << v[i][j] << " ";

}

cout << endl;

}

}

// calculate heuristic value..h'

int getScore(vector<vector<int>> &ans, vector<vector<int>> v)

{

int count = 0;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (v[i][j] != ans[i][j])

count++;

}

}

return count;

}

// to compare which node is better based on the heuristic value

bool comp(node a, node b)

{

return a.h < b.h;

}

//check if it is in set

bool isinset(node a, vector<node> b)

{

for (int i = 0; i < b.size(); i++)

{

if (a.arr == b[i].arr)

{

return true;

}

}

return false;

}

//here we are generating a move based on where we got 0(vacant pos) in the matrix

void addmove(node current, vector<vector<int>> &goal, int i, int j, int posi, int posj, vector<node> &openset, vector<node> closet)

{

node newState;

newState = current;

swap(newState.arr[i][j], newState.arr[posi][posj]);

if (!isinset(newState, closet) && !isinset(newState, openset)) // if the node is not in the open and closed set/list..

{

// if yes then create a new ode and insert it into the open sets

newState.level = current.level + 1;

newState.h = newState.level + getScore(goal, newState.arr);

cout<<"Value of the node(f') is : "<<newState.h<<endl;

node \*temp = new node();

\*temp = current;

newState.prev = temp;

openset.push\_back(newState);

}

}

void possibleMove(node current, vector<vector<int>> goal, vector<node> &openset, vector<node> &closet)

{

// this is to generate all possible moves

int i, j, posi, posj, val;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

val = current.arr[i][j];

// val=0 means we have a vacant position there..and now we generate possible moves

// on getting a 0(vacant pos) save that position and generate possible moves

// by calling the addmove function

if (val == 0)

{

// cout<<"found"<<endl;

posi = i;

posj = j;

break;

}

}

}

i = posi;

j = posj;

//take it within the bound

if (i - 1 >= 0)

addmove(current, goal, i - 1, j, posi, posj, openset, closet);

if (i + 1 < 3)

addmove(current, goal, i + 1, j, posi, posj, openset, closet);

if (j - 1 >= 0)

addmove(current, goal, i, j - 1, posi, posj, openset, closet);

if (j + 1 < 3)

addmove(current, goal, i, j + 1, posi, posj, openset, closet);

}

void getpath(node curr, vector<node> &ans)

{

node \*temp = &curr;

try

{

while (temp != NULL)

{

ans.push\_back(\*temp);

temp = temp->prev;

}

}

catch (const bad\_alloc &e)

{

cout << "failed in while loop" << e.what() << 'n';

}

}

void printList(vector<node> open)

{

for (auto it : open)

{

printMat(it.arr);

}

}

bool astar(vector<vector<int>> goal, vector<vector<int>> start)

{

// putting into the open and closed state

vector<node> openset;

vector<node> closet;

node current;

current.arr = start;

current.level = 0; // g value..

current.h = current.level + getScore(goal, current.arr);

//cout<<"Value of the node put in the open set(f') is : "<<current.h<<endl;

openset.push\_back(current);

while (openset.size() > 0)

{

// sort the nodes by comparing them based on their f value

sort(openset.begin(), openset.end(), comp);

node temp = openset[0];

// printMat(temp.arr);

cout<<"Printing the open set"<<endl;

printList(openset);

// cout<<"Printing the close set"<<endl;

// printList(closet);//intially closed set will be empty..

if (temp.arr == goal)

{

vector<node> ans;

getpath(temp, ans);

for (int i = ans.size() - 1; i >= 0; i--)

{

printMat(ans[i].arr);

}

return true;

}

openset.erase(openset.begin()); // remove the node from open set

cout<<"Printing open set after removing best node"<<endl;

printList(openset);

cout<<"Printing the close set after removing the best node from open and putting it into the closed : "<<endl;

printList(closet);

closet.push\_back(temp); // put into teh close set

// generate possible moves for that particular best node

possibleMove(temp, goal, openset, closet);

}

return false;

}

int main()

{

vector<vector<int>> ans(3, vector<int>(3));

ans[0][0] = 1;

ans[0][1] = 2;

ans[0][2] = 3;

ans[1][0] = 8;

ans[1][1] = 0;

ans[1][2] = 4;

ans[2][0] = 7;

ans[2][1] = 6;

ans[2][2] = 5;

vector<vector<int>> v(3, vector<int>(3));

int posx, posy;

// vector<int>valid(9,-1);

int sum = 36;

cout << "Please enter input matrix elements : " << endl;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

cin >> v[i][j];

sum -= v[i][j];

}

}

if (sum != 0)

{

cout << "Invalid input::";

return 0;

}

if (astar(ans, v))

{

cout << "success";

}

else

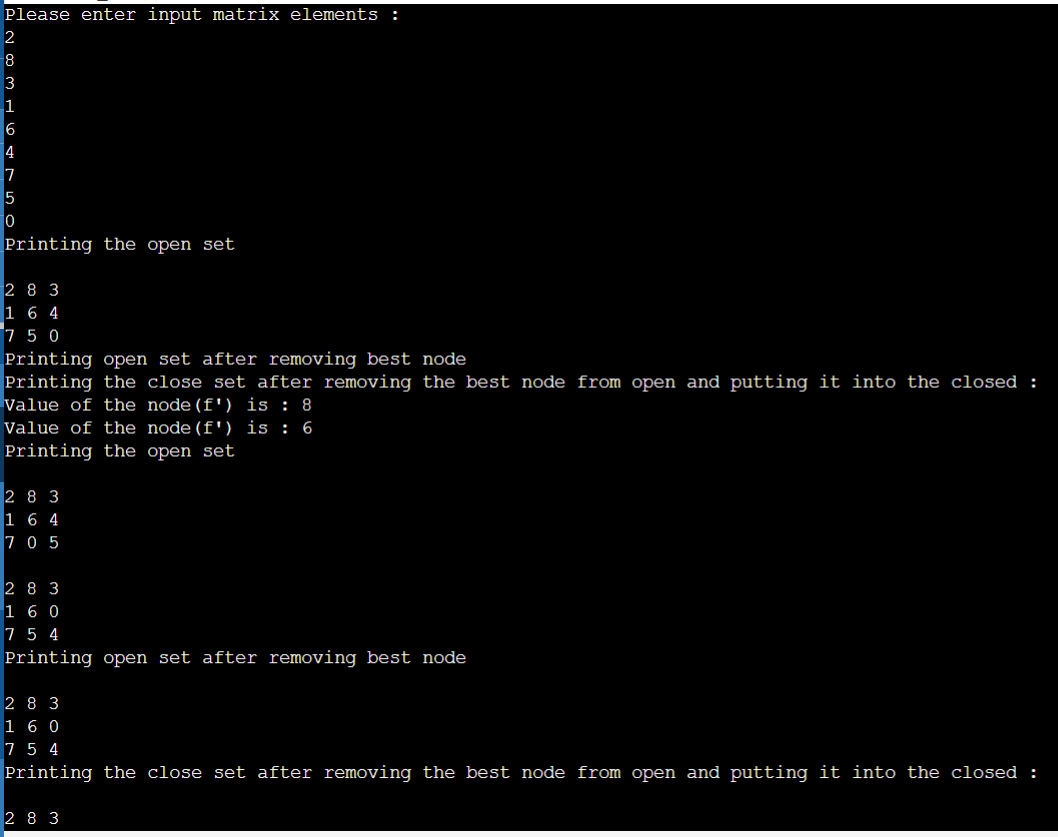
{

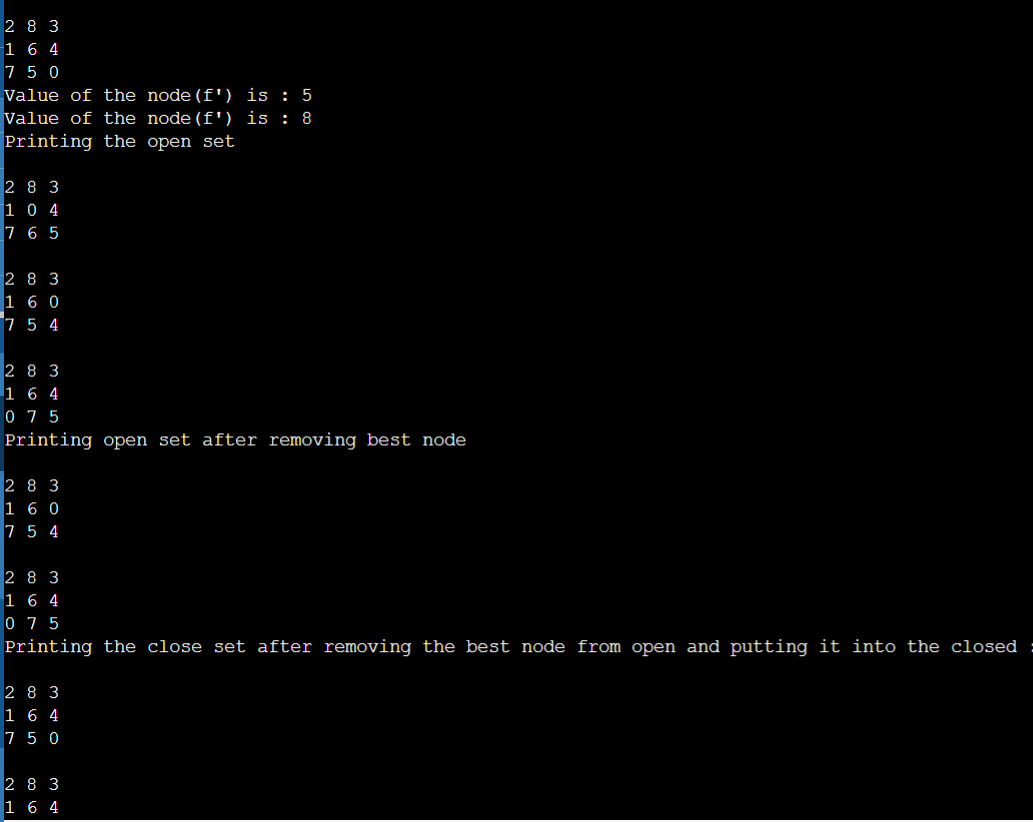
cout << "Fail";

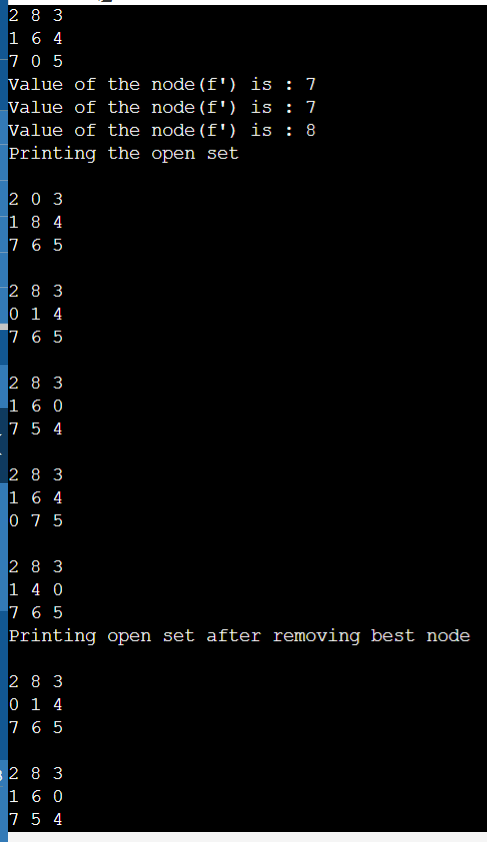
}

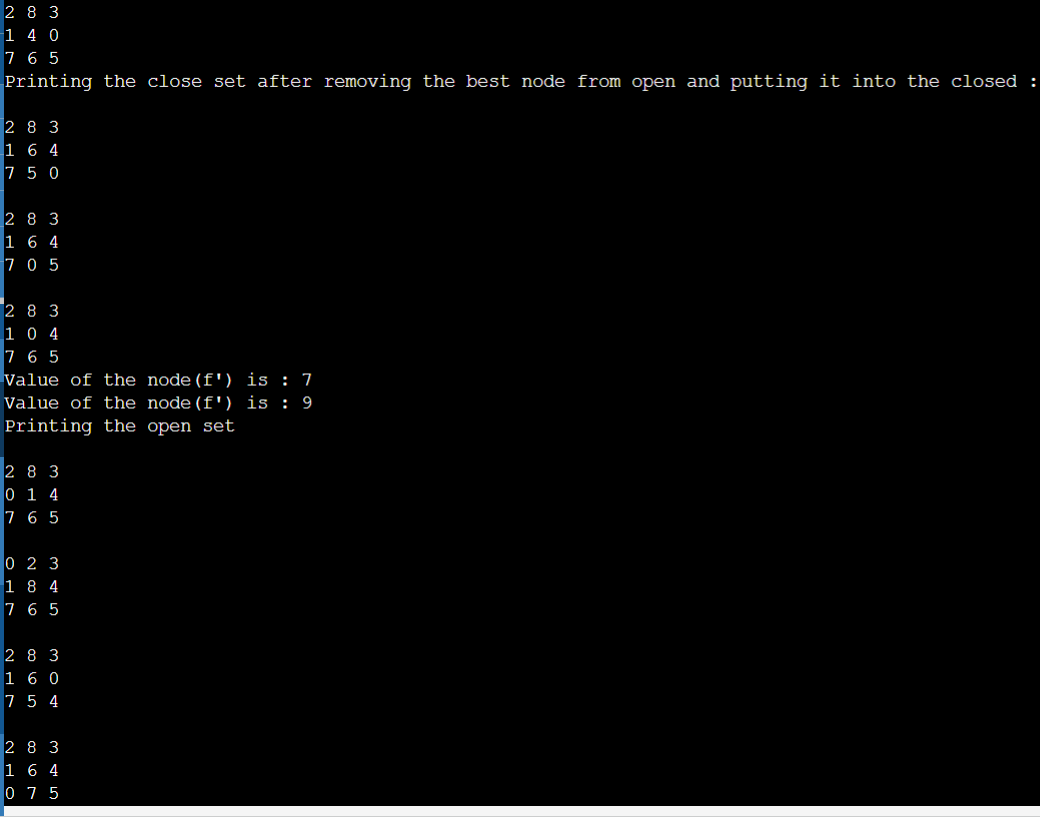
}

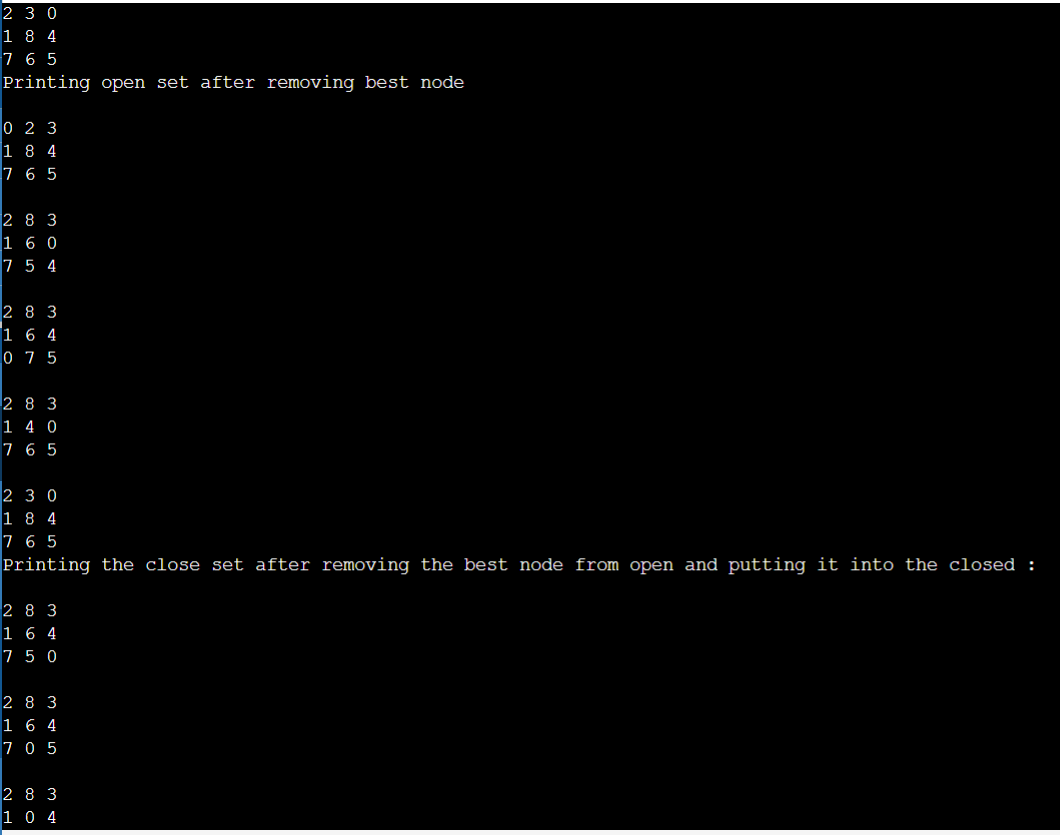
**Output-**

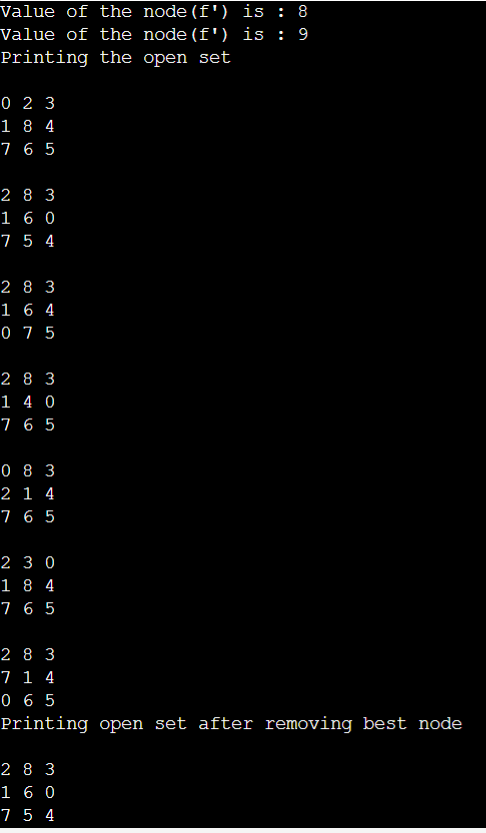
****

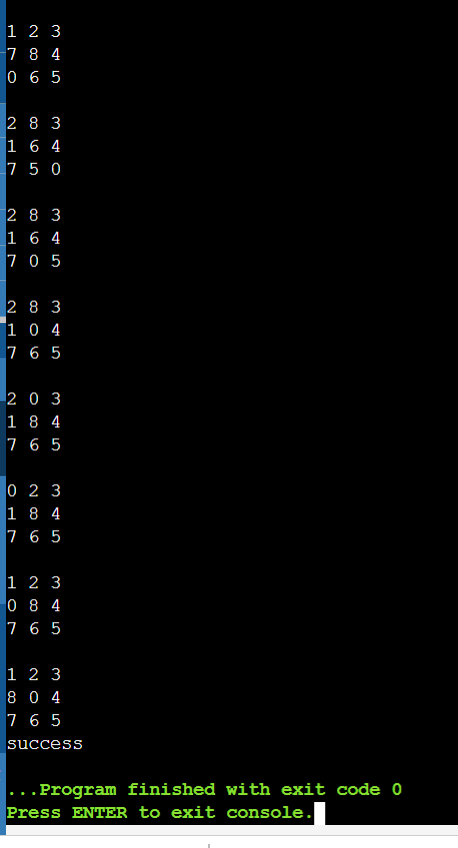
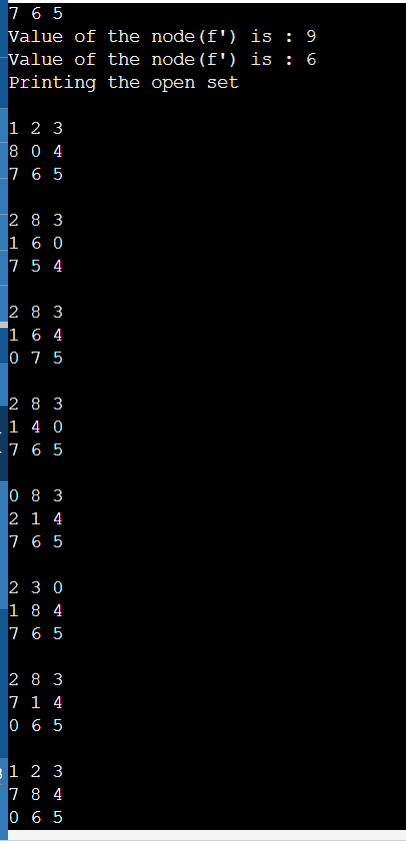
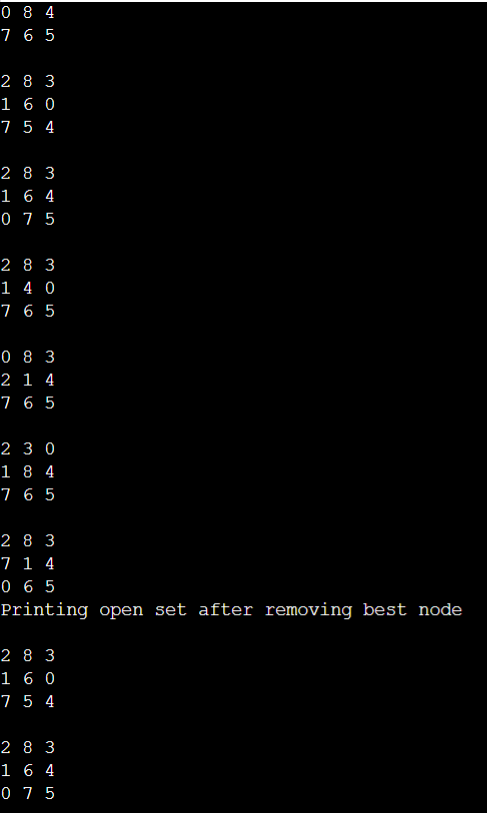
****

****

****

****

****

****