//Vishal Singh

//Batch 2

//roll no 63

#include<iostream>

#include<vector>

#include<math.h>

using namespace std;

int globalId = 0;

int main(){

    vector<int> goalState;

    goalState = {1,2,3,8,0,4,7,6,5};

    cout<<"Please Enter Initial Board Position ->>\n";

    vector<vector<int>> initState;

    int t;

    for(int i=0; i<3; i++){

        vector<int> temp;

        cin>>t;

        temp.push\_back(t);

        cin >> t;

        temp.push\_back(t);

        cin >> t;

        temp.push\_back(t);

        initState.push\_back(temp);

    }

    NodeStruct \*open=NULL, \*close=NULL;

    double topH\_val, h\_val = countEucledianDist(initState);

    //cout<<"h\_val: "<<h\_val;

    open = insertOPEN(open, close, initState, h\_val, 0);

    vector<vector<vector<int>>> nextMoves;

    int parentId;

    topH\_val = h\_val;

    int moveCount=0;

    while (topH\_val != 0){

        nextMoves = moveGenerator(open->boardPosition);

        parentId = open->id;

        topH\_val = open->hVal;

        close = insertCLOSE(close, open);

        open = removeTop(open);

        if(topH\_val == 0){

            break;

        }

        cout << "\nStep: " << ++moveCount;

        for (auto i = nextMoves.rbegin(); i != nextMoves.rend(); ++i)

        {

            h\_val = countEucledianDist(\*i);

            //cout<<h\_val;

            open = insertOPEN(open, close, \*i, h\_val, parentId);

        }

        cout << "\nPushing Current state to close linked list.......";

        printList(close);

        cout << "\n\nPushing next stated to open linked list........";

        printList(open);

    }

    cout << "\n\n---------End:---------\n";

    cout << "\nClosed linked list.....";

    printList(close);

    cout << "\n\nOpen linked list.....";

    printList(open);

    int trialId;

    NodeStruct \*temp = close;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    trialId = temp->id;

    cout<<"\n\n\*\*\*\*\*\*\*Final Move Sequence\*\*\*\*\*\*\*\n\n";

    cout << trialId;

    printFinalMoves(close, trialId);

    return 0;

}

struct NodeStruct

{

    vector<vector<int>> boardPosition;

    double hVal;

    NodeStruct \*next;

    int id, parentId;

    NodeStruct(vector<vector<int>> v, double h\_val, int parent, int newId)

    {

        boardPosition = v;

        parentId = parent;

        hVal = h\_val;

        next = NULL;

        id = newId;

    }

};

double countEucledianDist(vector<vector<int>> initialState)

{

    //printMove(initialState);

    vector<vector<int>> goalVector = {{1,2,3},{8,0,4},{7,6,5}};

    int count = 0;

    double distance = 0;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < 3; j++){

            if (goalVector[i][j] - initialState[i][j] != 0)

                distance = distance + (goalVector[i][j] - initialState[i][j]) \* (goalVector[i][j] - initialState[i][j]);

        }

    }

    //cout<<distance;

    return sqrt(distance);

}

vector<vector<int>> moveTile(vector<vector<int>> state, int z\_i, int z\_j, int i, int j)

{

    state[z\_i][z\_j] = state[z\_i][z\_j] + state[i][j];

    state[i][j] = state[z\_i][z\_j] - state[i][j];

    state[z\_i][z\_j] = state[z\_i][z\_j] - state[i][j];

    return state;

}

void printMove(vector<vector<int>> state)

{

    cout << "\n";

    for (int j = 0; j < 3; j++)

    {

        for (int k = 0; k < 3; k++)

        {

            cout << state[j][k] << " ";

        }

        cout << "\n";

    }

}

vector<vector<vector<int>>> moveGenerator(vector<vector<int>> initState)

{

    vector<vector<vector<int>>> nextMoves;

    //cout<<"\ninto\_move generator";

    int z\_i, z\_j;

    for(z\_i=0; z\_i<3; z\_i++){

        for(z\_j=0; z\_j<3; z\_j++){

            if(initState[z\_i][z\_j]==0)

                break;

        }

        if(z\_j<=2)

            break;

    }

    //cout<<z\_i<<" "<<z\_j;

    vector<vector<int>> temp;

    double dist;

    int i=-1;

    while(i < 2)

    {

        if (((z\_i + i) >= 0) && ((z\_j + i) >= 0) && ((z\_i + i) <= 2) && ((z\_j + i) <= 2)){

            temp = moveTile(initState, z\_i, z\_j, z\_i + i, z\_j);

            nextMoves.push\_back(temp);

            temp = moveTile(initState, z\_i, z\_j, z\_i, z\_j + i);

            nextMoves.push\_back(temp);

        }else if (((z\_i + i) >= 0) && ((z\_i + i) <= 2)){

            temp = moveTile(initState, z\_i, z\_j, z\_i + i, z\_j);

            nextMoves.push\_back(temp);

        }else if (((z\_j + i) >= 0) && ((z\_j + i) <= 2)){

            temp = moveTile(initState, z\_i, z\_j, z\_i, z\_j + i);

            nextMoves.push\_back(temp);

        }

        i = i+2;

    }

    return nextMoves;

}

bool isPresent(NodeStruct \*open, NodeStruct \*close, NodeStruct \*node){

    NodeStruct \*temp1 = open;

    NodeStruct \*temp2 = close;

    while(temp1 != NULL){

        if(temp1->id == node->id)

            return true;

        temp1 = temp1->next;

    }

    while (temp2 != NULL)

    {

        if (temp2->id == node->id)

            return true;

        temp2 = temp2->next;

    }

    return false;

}

NodeStruct \*insertOPEN(NodeStruct \*head, NodeStruct \*close, vector<vector<int>> initialState, double h\_val, int parentId)

{

    NodeStruct \*node = new NodeStruct(initialState, h\_val, parentId, ++globalId);

    NodeStruct \*temp = head;

    cout<<"\nMove id "<<node->id;

    printMove(node->boardPosition);

    if(isPresent(head, close, node)){

        return temp;

    }

    if(temp == NULL){

        return node;

    }

    if(h\_val <= temp->hVal){

        node->next = temp;

        return node;

    }

    while(h\_val <= temp->hVal && temp->next != NULL){

        temp = temp->next;

    }

    node->next = temp->next;

    temp->next = node;

    return temp;

}

NodeStruct \*removeTop(NodeStruct \*head){

    NodeStruct \*temp = head;

    return temp->next;

}

NodeStruct \*insertCLOSE(NodeStruct \*head, NodeStruct \*node){

    //cout<<"closed\n";

    NodeStruct \*temp = head, \*temp2;

    temp2 = new NodeStruct(node->boardPosition, node->hVal, node->parentId, node->id);

    if (temp == NULL)

    {

        return temp2;

    }

    while(temp->next != NULL){

        temp = temp->next;

    }

    temp->next = temp2;

    return head;

}

void printList(NodeStruct \*head)

{

    NodeStruct \*temp = head;

    while (temp != NULL)

    {

        cout << "\nval: " << temp->hVal << "  id:" << temp->id << "  Parent id: " << temp->parentId;

        temp = temp->next;

    }

}

void printFinalMoves(NodeStruct \*head, int trialId){

    NodeStruct \*temp = head;

    if(trialId == 1){

        cout << "\nBoard Position:";

        printMove(temp->boardPosition);

        cout << "\nh\_val: " << temp->hVal;

        return;

    }

    while(temp->id != trialId){

        temp = temp->next;

    }

    printFinalMoves(head, temp->parentId);

    cout << "\nBoard Position:";

    printMove(temp->boardPosition);

    cout << "\nh\_val: " << temp->hVal;

}

Output:

Text

Description automatically generated