|  |
| --- |
| **NAME:** GOKULRAJ V |
| **ROLL\_NO:** 1905014 |

|  |
| --- |
| **DATE:** 05/08/2022 |
| **EXP\_NO:** 02 |

|  |
| --- |
| **8-PUZZLES** |

**AIM:**

To solve 8 puzzle problem using iterative deepening search & depth limited dfs.

**Problem Statement:**

Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match the final configuration using the empty space. We can slide four adjacent (left, right, above, and below) tiles into the empty space.

**Depth-Limited Search Algorithm:**

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depthlimited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further. Depth-limited search can be terminated with two Conditions of failure:

* Standard failure value: It indicates that problem does not have any solution.
* Cutoff failure value: It defines no solution for the problem within a given depth limit. Completeness: DLS search algorithm is complete if the solution is above the depth-limit. Time Complexity: Time complexity of DLS algorithm is O(bℓ ).
* Space Complexity: Space complexity of DLS algorithm is O(b×ℓ).
* Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

**Iterative deepening depth-first Search:** The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found. This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found

* Completeness: This algorithm is complete is if the branching factor is finite.
* Time Complexity: Let's suppose b is the branching factor and depth is d then the worst-case time complexity is O(bd ).
* Space Complexity: The space complexity of IDDFS will be O(bd).
* Optimal: IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

**CODE:**

#include<bits/stdc++.h>

using namespace std;

vector<vector<int>> states = {{0,1},{0,-1},{1,0},{-1,0}};

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

void printv(vector<vector<int>> v, vector<int> l)

{

cout<<"------ "<<l[2]<<" -----\n";

for(auto i:v)

{

for(auto j:i)

{

cout<<j<<" ";

}

cout<<"\n";

}

cout<<"\n";

}

void printv2(vector<vector<int>> v)

{

for(auto i:v)

{

for(auto j:i)

{

cout<<j<<" ";

}

cout<<"\n";

}

cout<<"\n";

}

bool isFinal(vector<vector<int>> v)

{

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

{

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

{

if(v[j]!=fin[j])

return false;

}

}

return true;

}

bool nxtState(vector<vector<int>> &s, vector<int> ns, vector<int> &emt, set<vector<vector<int>>> hmap)

{

int i=emt[0] + ns[0];

int j =emt[1] + ns[1];

if(i<0 || i>=3 || j<0 || j>=3)

{

return false;

}

swap(s[i][j], s[emt[0]][emt[1]]);

if(hmap.count(s)!=0)

{

swap(s[i][j], s[emt[0]][emt[1]]);

return false;

}

emt[0] = i;

emt[1] = j;

emt[2]++;

return true;

}

bool ids(vector<vector<int>> b, vector<int> emt, int max\_d )

{

cout<<"++++++++ max\_d = "<<max\_d<<" +++++++++++\n";

stack<pair<vector<vector<int>>,vector<int>>> s;

stack<vector<vector<int>>> bt;

set<vector<vector<int>>> hmap;

int pred=0;

s.push({b,emt});

bt.push(b);

while(!s.empty())

{

pair<vector<vector<int>>,vector<int>> tmp = s.top();

vector<vector<int>> state = tmp.first;

vector<int> e = tmp.second;

s.pop();

if(e[2]>pred)

{

bt.push(state);

}

else if(e[2]==pred){

bt.pop();

bt.push(state);

}

else{

bt.pop();

}

if(e[2]>max\_d) continue;

if(hmap.count(tmp.first)==0)

hmap.insert(tmp.first);

if(isFinal(tmp.first))

{

while(!bt.empty())

{

printv2(bt.top());

bt.pop();

}

return true;

}

for(int i=0;i<4;i++)

{

if(nxtState(state,states[i],e, hmap))

{

s.push({state,e});

state = tmp.first;

e = tmp.second;

}

}

pred = e[2];

}

while(!bt.empty())

{

printv2(bt.top());

bt.pop();

}

return false;

}

int main()

{

vector<vector<int>> b(3,vector<int>(3,0));

vector<int> emt(3,0);

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

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

{

cin>>b[i][j];

if(b[i][j]==-1)

{

emt[0] = i;

emt[1] = j;

}

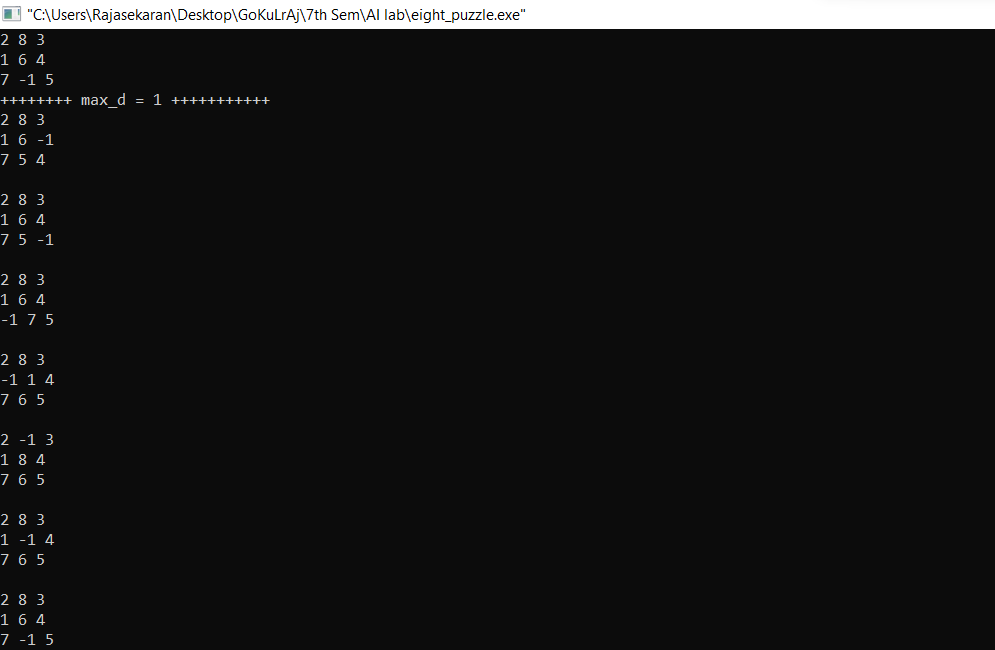
}

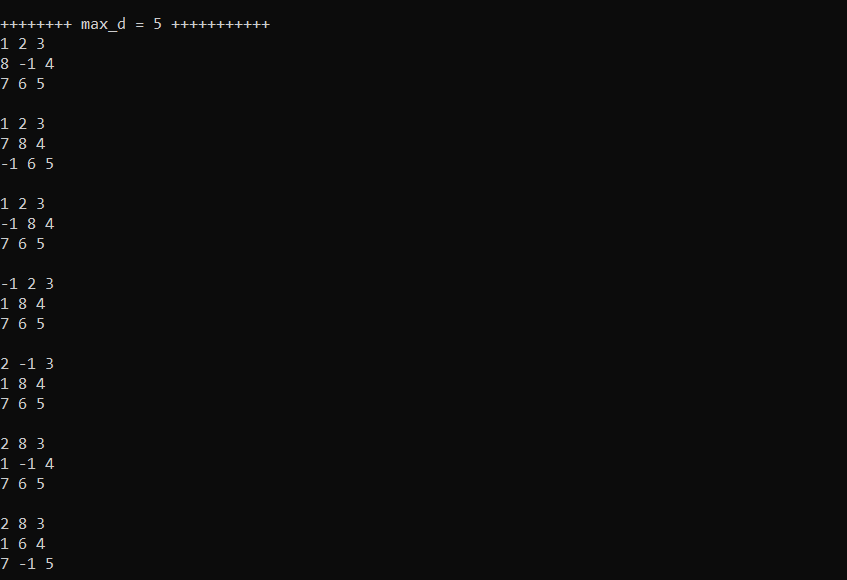
int i=0;

while(!ids(b,emt,++i));

}

**OUTPUT:**





**RESULT:**

Thus, the code has been successfully executed.