```
/// 31444- Mehul Oswal
// A* algorithm for 8 puzzle

// f(x) = g(x) + h(x)
// g(x) -> level order (distance from current node
to root node)
// h(x) -> manhattan distance(number of misplaced
tiles by comparing the current state and the goal
state)

// could be solved using hamming or manhattan
distance.
#include <bits/stdc++.h>
using namespace std;

map<vector<vector<int>>, bool> visited;
map<vector<vector<int>>, vector<vector<int>>>
parent; // 2-d to 2-d
vector<vector<int>> goal(3, vector<int>(3));
// initializing the matrix

bool visit(vector<vector<int>> a)
{
    if (visited[a] == true)
        return true;
    else
        return false;
}

int manhattan(vector<vector<int>> a, int moves)
{
    int dist = moves;
    for (int i = 0; i < 3; i++) // for initial-
state
    {
        for (int j = 0; j < 3; j++)
        {
            if (a[i][j] != 0)
            {
                for (int k = 0; k < 3; k++) // for
goal-state
                {
                    for (int l = 0; l < 3; l++)
                    {
                        if (a[i][j] == goal[k][l])
                            dist += abs(i - k) +
abs(j - l);
                    }
                }
            }
        }
    }

    return dist; // f(x)
    // f(x) -> manhattan distance(number of
misplaced tiles by comparing the current state and
the goal state)
}

bool isGoal(vector<vector<int>> a) // to check
whether the matrix is final matrix or not.
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (a[i][j] != goal[i][j])
                return false;
        }
    }

    return true;
}

bool check(int i, int j) // finds whether the
missing place could be swapped to its neighbors or
not
{
    if (i >= 0 && i <= 2 && j >= 0 && j <= 2)
```

```
        return true;
    else
        return false;
}
int dx[] = {-1, +1, 0, 0}; // two arrays used for
restrict the motion between rows and columns
int dy[] = {0, 0, -1, +1};
vector<vector<vector<int>>>
neighbours(vector<vector<int>> a) // Collection of
neighbours(multiple 2D arrays)
{
    pair<int, int> pos; // to maintain coordinates
the positiion of the blank space(0)
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (a[i][j] == 0)
            {
                pos.first = i;  // 0
                pos.second = j; // 0
                break;
            }
        }
    }
    vector<vector<vector<int>>> ans;
    for (int k = 0; k < 4; k++) // getting all the
possible neighbours where the blank space could be
shifted
    {
        int cx = pos.first;  // 0// assigning the
missing place
        int cy = pos.second; // 0
        vector<vector<int>> n = a;
        if (check(cx + dx[k], cy + dy[k]))
        {
            swap(n[cx + dx[k]][cy + dy[k]], n[cx]
[cy]);
            ans.push_back(n); // adding the 2-d
array to the 3-d array
```

```cpp
        }
    }

    return ans;
}

void print_path(vector<vector<int>> s)
{
    if (parent.count(s))
        print_path(parent[s]);

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            printf("%d ", s[i][j]);
        }
        cout << endl;
    }

    cout << "|" << endl;
    cout << "v" << endl;
    cout << "\n";

    return;
}

void solve(vector<vector<int>> a, int moves) //
params-> (initial-state, g(x))
{

    if (isGoal(a))
    {
        print_path(a);
        cout << "Done\n"
             << "-------------------\n";
        return;
    }

    vector<vector<int>> test;
    vector<vector<vector<int>>> ns = neighbours(a);
// getting the neighbours of that state
    vector<vector<vector<int>>>::iterator it;
// iterator to surf across the neighbours
    visited[a] = true;
    int cost = 10000;
    for (it = ns.begin(); it != ns.end(); it++)
    {

        vector<vector<int>> temp = *it;
        if (!visit(temp))
        {
            if (manhattan(temp, moves + 1) < cost)
            {
                cost = manhattan(temp, moves + 1);
                test = temp;
            }
            parent.insert(pair<vector<vector<int>>,
vector<vector<int>>>(temp, a)); // next state,
current state
        }
    }
    solve(test, moves + 1);
}

int main()
{
    vector<vector<int>> a(3, vector<int>(3));
    cout << "Enter the initial state of 8-puzzle
game:[ Example Below ] "
         << "\n\t\t\t\t 1 2 3 \n\t\t\t\t 4 0 5
\n\t\t\t\t 6 7 8" << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cin >> a[i][j];
        }
    }
    cout << 
"==================================================
=======" << endl;
    cout << "Enter the goal state of 8-puzzle game:
[ Example Below ]"
         << "\n\t\t\t\t 1 2 3 \n\t\t\t\t 4 0 5 
\n\t\t\t\t 6 7 8" << endl;

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cin >> goal[i][j];
        }
    }

    cout << "\n\n\nSolution...\n\n";

    solve(a, 0);
}
```

OUTPUT

```
/*
Enter the initial state of 8-puzzle game:[ Example 
Below ]
                                 1 2 3 
                                 4 0 5 
                                 6 7 8
1 2 3
0 4 6
7 5 8
===================================================
======
Enter the goal state of 8-puzzle game:[ Example
Below ]
                                 1 2 3
                                 4 0 5
                                 6 7 8
1 2 3
4 5 6
7 8 0



Solution...

1 2 3
0 4 6
7 5 8
|
v

1 2 3
4 0 6
7 5 8
|
v

1 2 3
4 5 6
7 0 8
|
v

1 2 3
4 5 6
7 8 0
|
v

Done
*/
```