Solving Genshin's "Bunkoku Enigma" Puzzles Using Hill Climbing Algorithm for 8 Puzzles

By

Ferry Nur Alfian Eka Putra (5025201214)

Reza Maranelo Alifiansyah (50252010)

1. What is Genshin Impact?

Genshin Impact is an open-world, action RPG developed and published by HoYoverse for PC, iOS/Android, and PS4/PS5 platforms. The game is free-to-play with a gacha monetization system in the form of Wishes.

url: https://genshin-impact.fandom.com/wiki/Genshin_Impact_Wiki

2. What is Bunkoku Enigma?

Bunkoku means Provinces that in the Heian Period were governed by court nobles. Enigma means a person or thing that is mysterious, puzzling, or difficult to understand. The Bunkoku Enigma is a quest for Genshin Impact's event. This Quest has added on February 17, 2022, and ends on March 30, 2022, within The Three Realms Gateway Offering in-game event.

url: https://genshin-impact.fandom.com/wiki/The_Bunkoku_Enigma

3. Why this Puzzles can be solved using hill climbing algorithm for 8 puzzles?

Because this puzzle has the same mechanism as the 8 puzzles problem, but with the different final state. For 8 puzzles, the hollow piece is placed after number 8 (at the bottom-right corner), but the Bunkoku Enigma Puzzles' hollow piece is place at the middle of the puzzle.

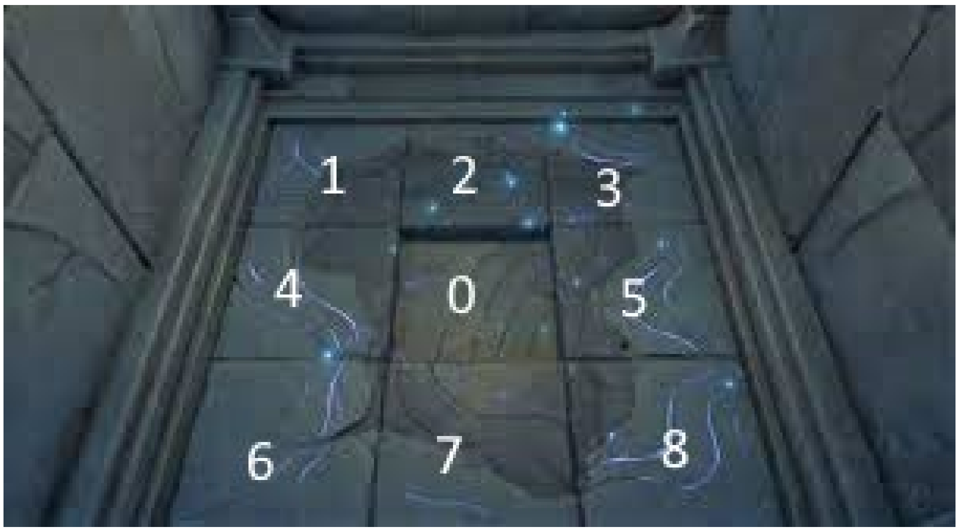


Image: final state of Bunkoku Enigma Puzzle

4. What is Hill Climbing Algorithm?

In short, Hill Climb Algorithm is Algorithm that count every Manhattan Distance for possibility which in this case, is 0 tile movements. Then, the Algorithm will compare each possibility and choose which one that has the least total Manhattan Distance cost.

4.1. What is Manhattan Distance?
Manhattan Distance is a distance metric between two points in N dimensional vector space, which in this case 2 dimensions.

4.2. How Manhattan Distance works on Hill Climb Algorithm?
Every Manhattan Distance of the tiles is counted based on the final state. So when the state has 0 Manhattan Distance, it means that the algorithm is over (coming to an end).

5. How Hill Climbing Algorithm works?
Using BFS (Breadth First Search), it will see every possibility for the 0 tiles to move. Then, Hill Climbing Algorithm will count every possibility's Manhattan cost. Then it will choose the least Manhattan cost of moves. Then, BFS will search for the next possibility. This phase will repeat again and again until the Manhattan cost reach 0;

The Program :

```
// Solving 8-Puzzle Using Steepest Ascent Hill Climbing Algorithm

#include <bits/stdc++.h>

using namespace std;

const char UP ='1';

const char DOWN= '2';

const char LEFT= '3';

const char RIGHT= '4';


void printArray(int** array)

{

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

  {

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

    {

            cout<< setw(8) << array[a][b];

    }

    cout<<endl;

  }
```

```
    cout<<endl;
}

// Calculate Manhattan distance
int manhattan_distance(int** start_state,int** goal_state)
{
    int manhattan_distance=0;
    for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                if(start_state[i][j] > 0)
                {
                    for(int k=0;k<3;k++)
                    {
                        for(int l=0;l<3;l++)
                        {
                            if (goal_state[k][l] == start_state[i][j])
                            {
                                manhattan_distance=manhattan_distance + (abs(i-k)+abs(j-l));


                            }
                        }
                    }
                }
            }
        }
return manhattan_distance;
}
```

```
void makeMove(int** temp,int move)
{
   int flag=0;
   for(int i=0;i<3;i++)
   {
      for(int j=0;j<3;j++)
      {
         if(temp[i][j] == 0)
         {
            if(move==1)
            {
               temp[i][j] = temp[i-1][j];
                  temp[i-1][j] = 0;
               flag=1;
               break;
            }
            else if(move==2)
            {
               temp[i][j] = temp[i+1][j];
                  temp[i+1][j] = 0;
               flag=1;
               break;
            }
            else if(move==3)

            {
               temp[i][j] = temp[i][j-1];
                  temp[i][j-1] = 0;
               flag=1;
               break;
            }
            else
            {
                temp[i][j] = temp[i][j+1];
                temp[i][j+1] = 0;
              flag=1;
              break;
            }
          }
        }
        if(flag==1){break;}
    }
}

//------------------------------------------------------------------------------------------------------------
int tile_Ordering(int** current_state,int** goal_state,int move)
{
    int** temp = new int*[3]; // allocate an array of 3 int pointers - these represents rows
    for(int i=0;i<3;i++)
    {
        temp[i]=new int[3]; // these represents columns
        for(int j=0;j<3;j++)
        {
            temp[i][j] = current_state[i][j];
        }
    }
    makeMove(temp,move);
    printArray(temp);
    int manhattan=manhattan_distance(temp,goal_state);
    cout<<"Current Manhattan number :"<<manhattan<<endl<<endl<<endl;
    for(int i=0;i<3;i++)
    {
        delete temp[i];
    }
    delete temp;
    return manhattan;
}

//---------------------------------------------------------------------------------------------------------
void steepestAscentHillClimbing( int** start_state, int** goal_state,int former_move)
{
            int arr[4] = {100,100,100,100};
    cout<<"-------------------------------------------------------------------------------"<<endl;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            if (start_state[i][j] == 0)
            {
                if(i>0 && former_move!=2)
                {
                    cout<<"Checking child (moving 0 up) "<<endl;
                    arr[0]=tile_Ordering(start_state,goal_state,1);
                }
                if(i<2 && former_move!=1)
                {
                    cout<<"Checking child (moving 0 down) "<<endl;
                    arr[1]=tile_Ordering(start_state,goal_state,2);
                }
                if(j>0 && former_move!=4)
            {
               cout<<"Checking child (moving 0 left) "<<endl;
               arr[2]=tile_Ordering(start_state,goal_state,3);
            }
            if(j<2 && former_move!=3)
            {
               cout<<"Checking child (moving 0 right) "<<endl;
               arr[3]=tile_Ordering(start_state,goal_state,4);
            }
         }
      }
      cout<<endl;
   }
   int localOptimum=99;
   int index=0;
   for(int i=0;i<4;i++)
   {
      if(arr[i]<localOptimum)
      {
         localOptimum=arr[i];
         index=i+1;
      }
   }
   makeMove(start_state,index);
   cout<<"Next state = minimum Manhattan number :"<<endl;
   printArray(start_state);
   if(localOptimum==0)
   {
      cout<<"goal state reached"<<endl;
```

```
        return;
    }
    else
    steepestAscentHillClimbing(start_state,goal_state,index);
}

int main()
{
    int** initial = new int*[3]; // allocate an array of 3 int pointers - these represents rows
    for(int i=0;i<3;i++)
    {
        initial[i]=new int[3]; // these represents columns
    }

    int** final = new int*[3]; // allocate an array of 3 int pointers - these represents rows
    for(int i=0;i<3;i++)
    {
        final[i]=new int[3]; // these represents columns
    }

    int player_Input;
    cout << "Enter initial board configuration - 0 denotes empty position" << endl;
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            cout<<"Enter input A["<<i<<"]["<<j<<"]"<<endl;
            cin>>player_Input;
            initial[i][j]=player_Input;
        }
```

```
}

cout<<"------------------------------------------------------------------------------"<<endl;
cout << "Enter final board configuration - 0 denotes empty position" << endl;
for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
    {
        cout<<"Enter input A["<<i<<"]["<<j<<"]"<<endl;
        cin>>player_Input;
        final[i][j]=player_Input;
    }

cout<<"\n--------------------------Your  initial  matrix  is-------------------------------\n"<<endl;



for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        cout << setw(8) << initial[i][j];
    }
    cout<<endl;
}

cout<<"\n--------------------------Your final matrix is--------------------------------\n"<<endl;



for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
```

```
        {
           cout << setw(8) << final[i][j];
        }
        cout<<endl;
    }


           cout<<"\n-------------------------------------------------------------------------------"<<endl;
           cout<<"\n-------------------------------------------------------------------------------"<<endl;
    cout<<"\nCalling Steepest Ascent Hill Climbing function\n"<<endl;
   steepestAscentHillClimbing(initial,final,0);
    for(int i=0;i<3;i++)
    {
        delete initial[i];
        delete final[i];
    }
    delete initial;
    delete final;
    return 0;
}
```

The 1st input: 0 2 3 1 5 8 4 6 7 (the initial State of Bunkoku Enigma Puzzle)

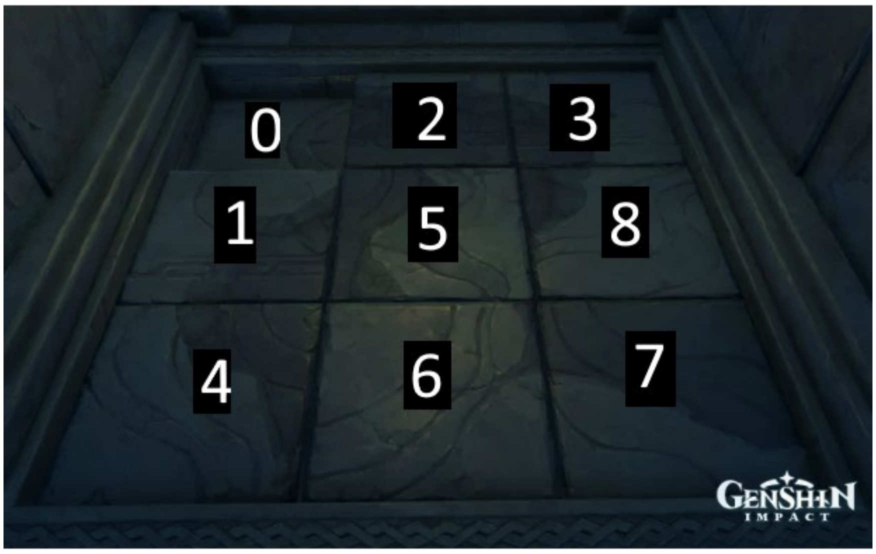


Image: initial state of Bunkoku Enigma Puzzle

The 2nd input: 1 2 3 4 0 5 6 7 8 (the final State of Bunkoku Enigma Puzzle)

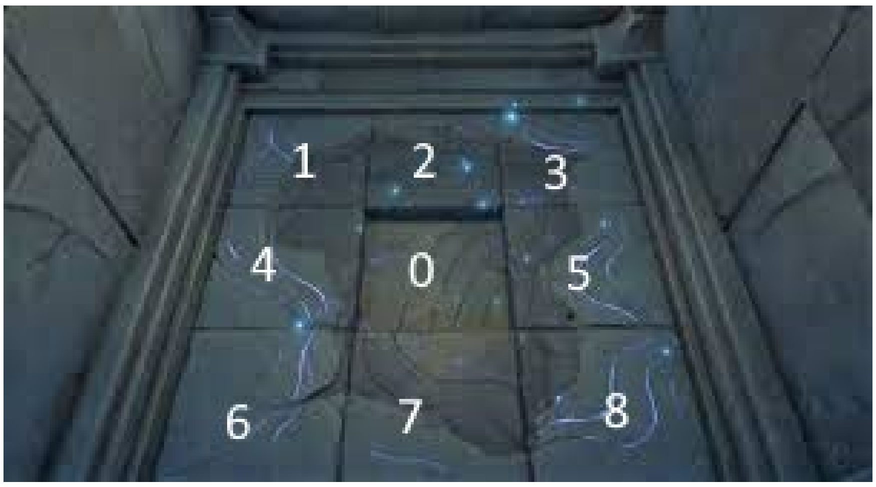
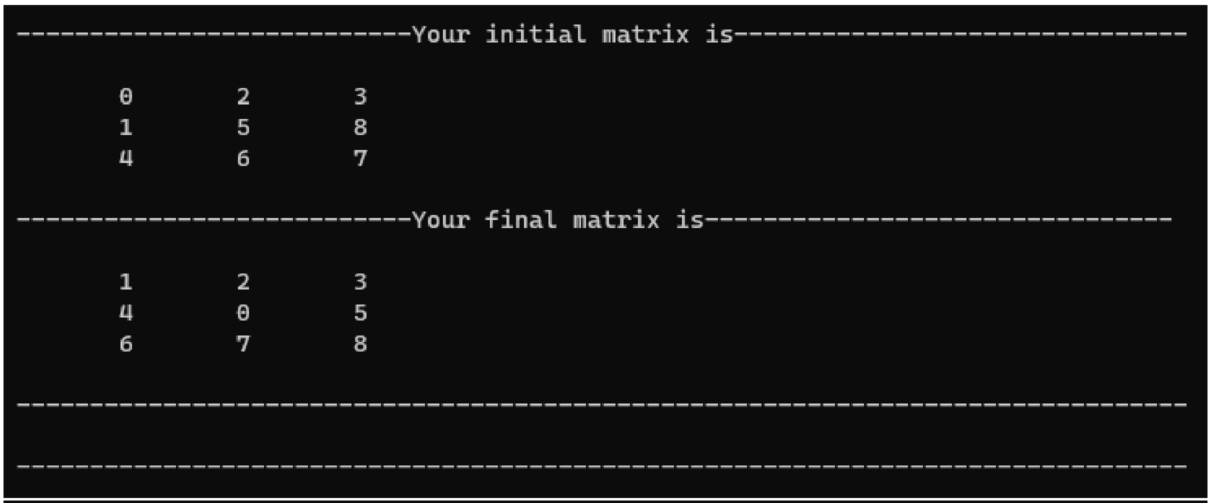
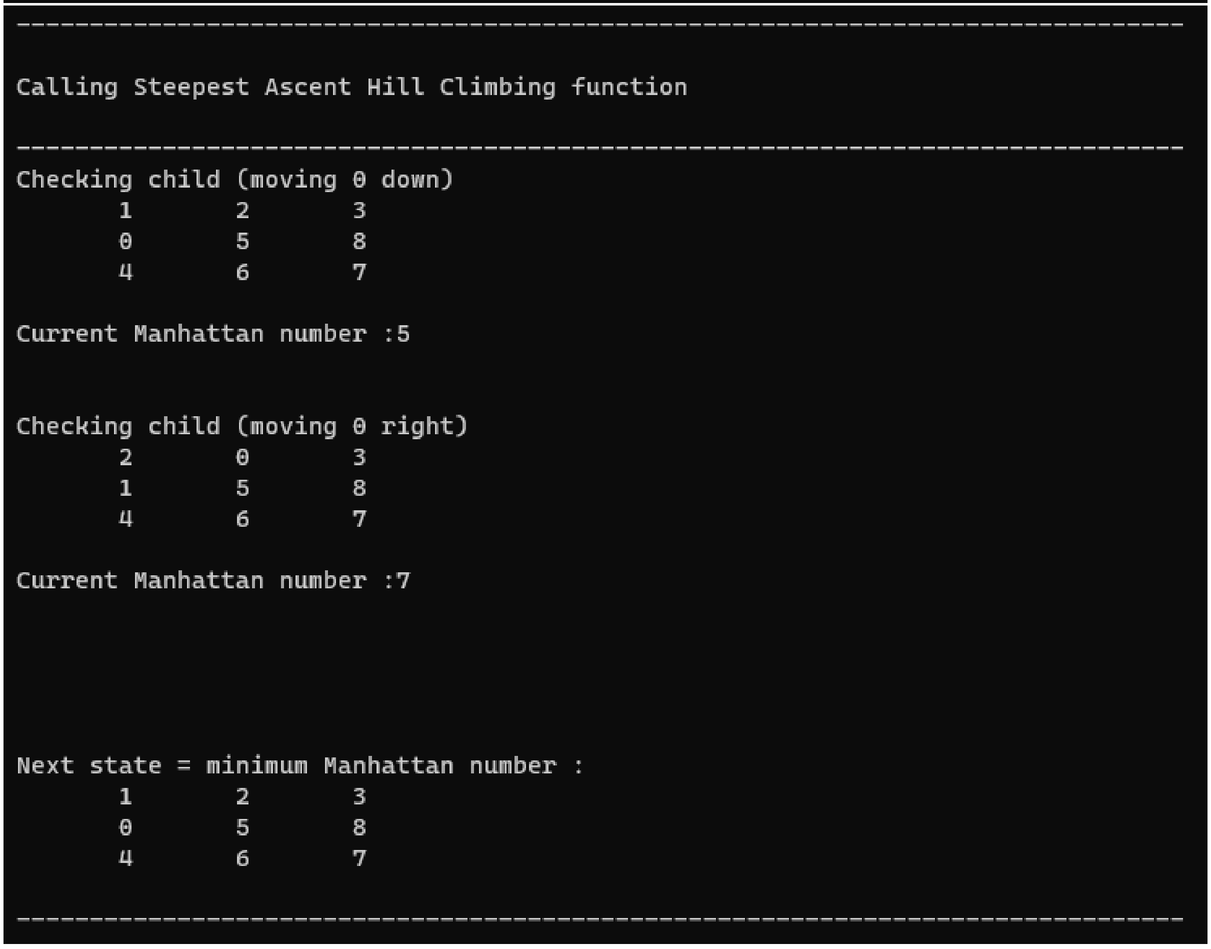


Image : final state of Bunkoku Enigma Puzzle

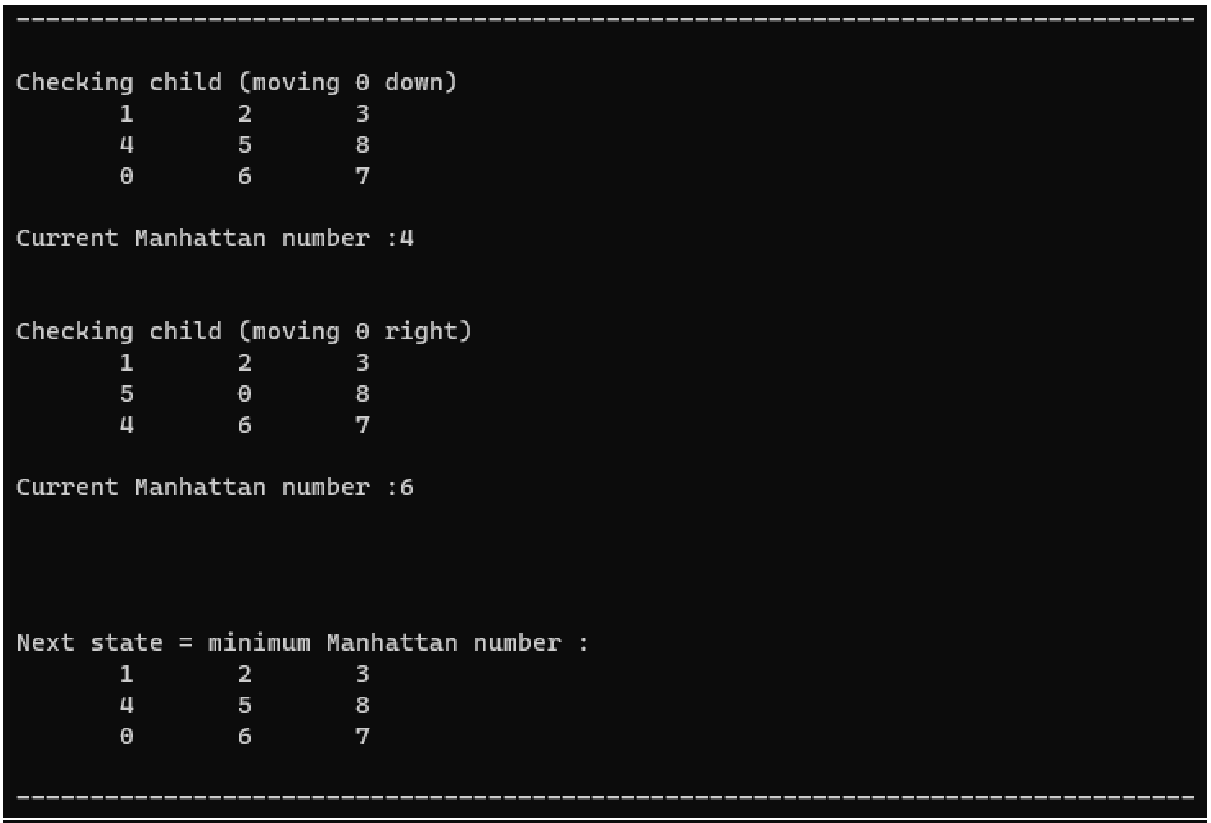
Output:



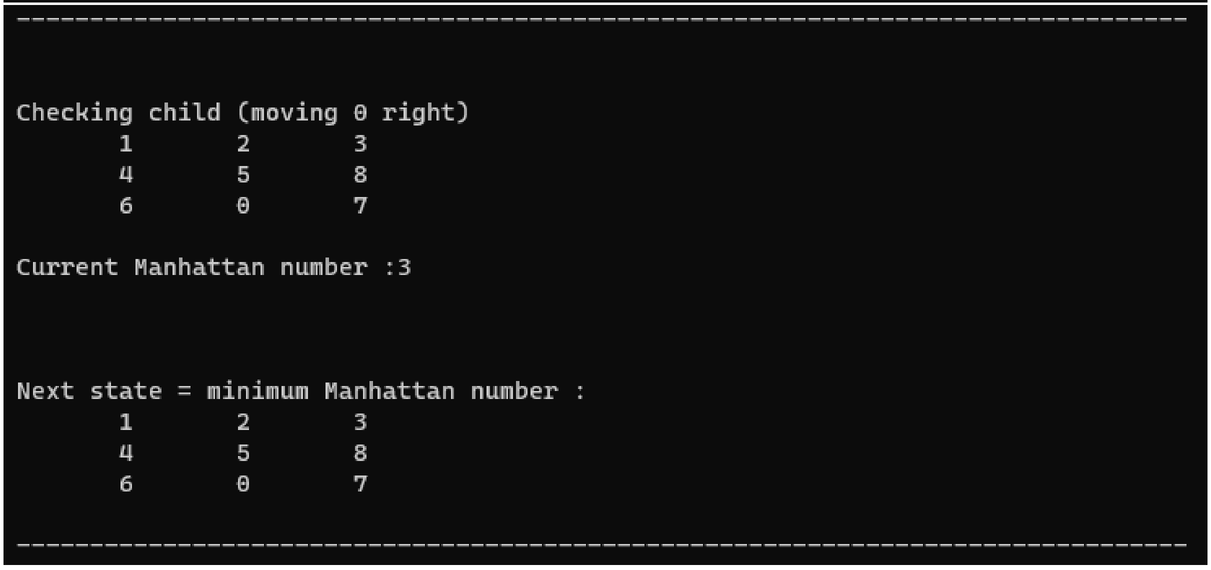
```
---------------------------Your initial matrix is-------------------------------

       0       2       3
       1       5       8
       4       6       7

---------------------------Your final matrix is--------------------------------

       1       2       3
       4       0       5
       6       7       8

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
```



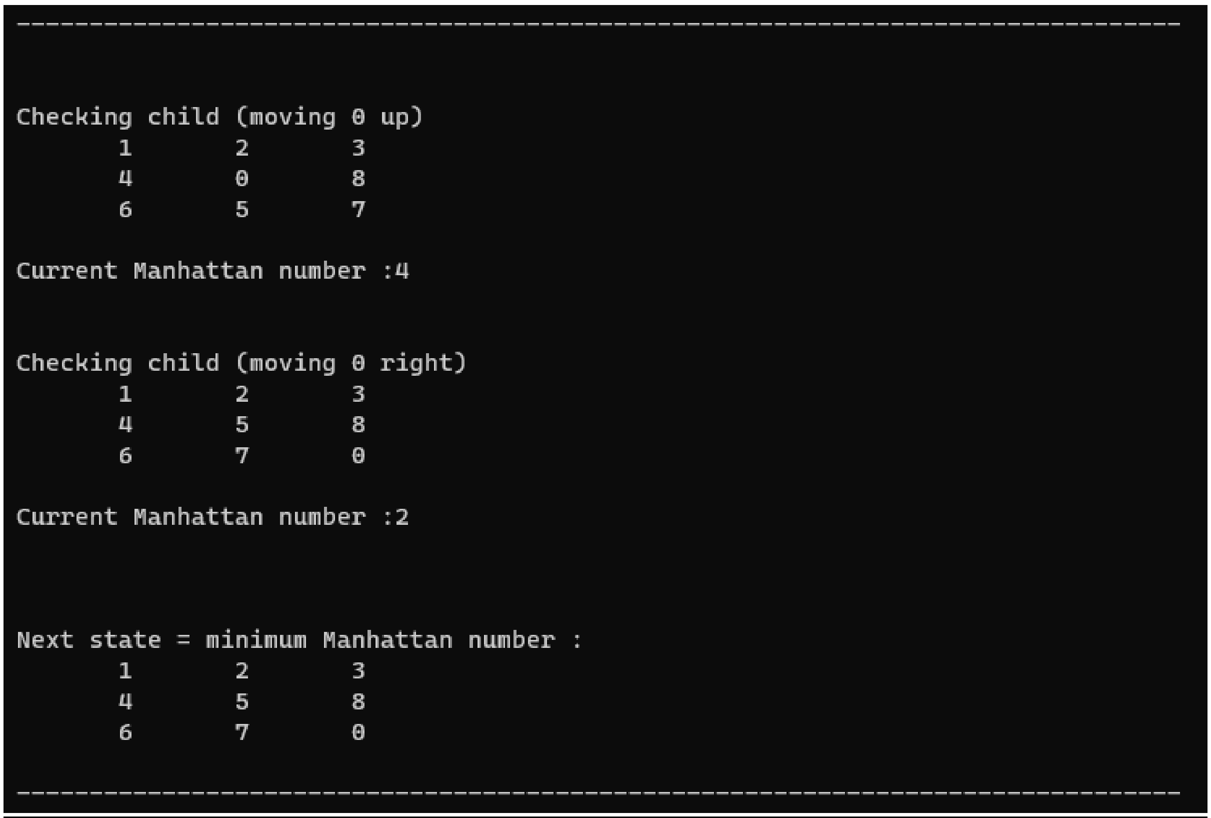
```
--------------------------------------------------------------------------------

Calling Steepest Ascent Hill Climbing function

--------------------------------------------------------------------------------
Checking child (moving 0 down)
       1       2       3
       0       5       8
       4       6       7

Current Manhattan number :5


Checking child (moving 0 right)
       2       0       3
       1       5       8
       4       6       7

Current Manhattan number :7





Next state = minimum Manhattan number :
       1       2       3
       0       5       8
       4       6       7

--------------------------------------------------------------------------------
```



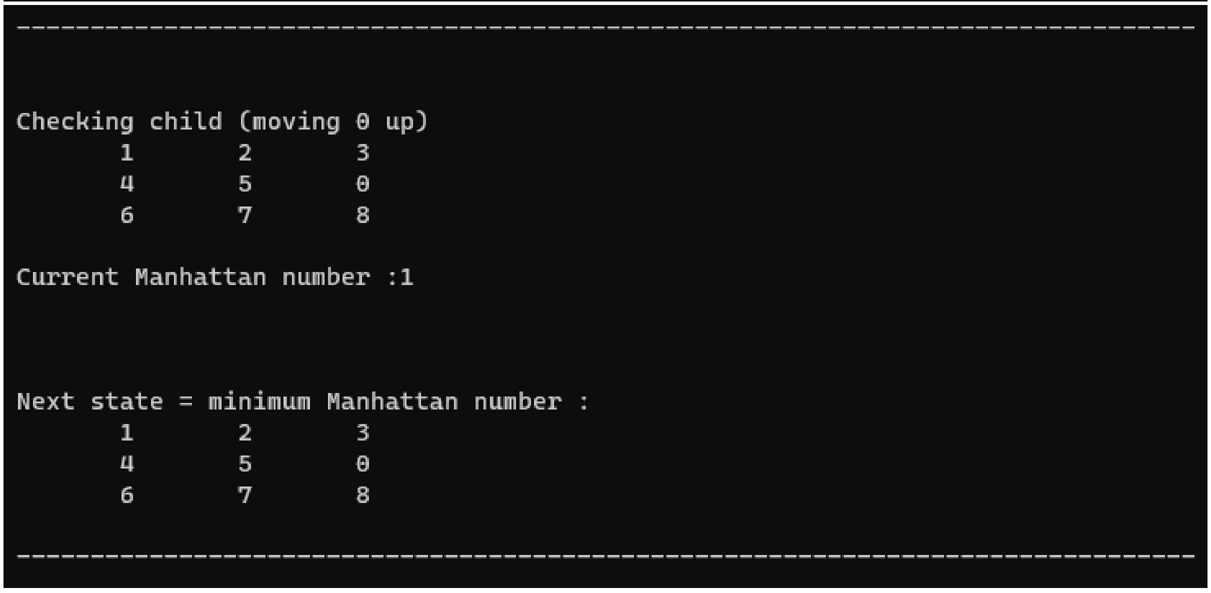
```
---------------------------------------------------------------------------


Checking child (moving 0 down)
       1       2       3
       4       5       8
       0       6       7

Current Manhattan number :4


Checking child (moving 0 right)
       1       2       3
       5       0       8
       4       6       7

Current Manhattan number :6




Next state = minimum Manhattan number :
       1       2       3
       4       5       8
       0       6       7

---------------------------------------------------------------------------
```



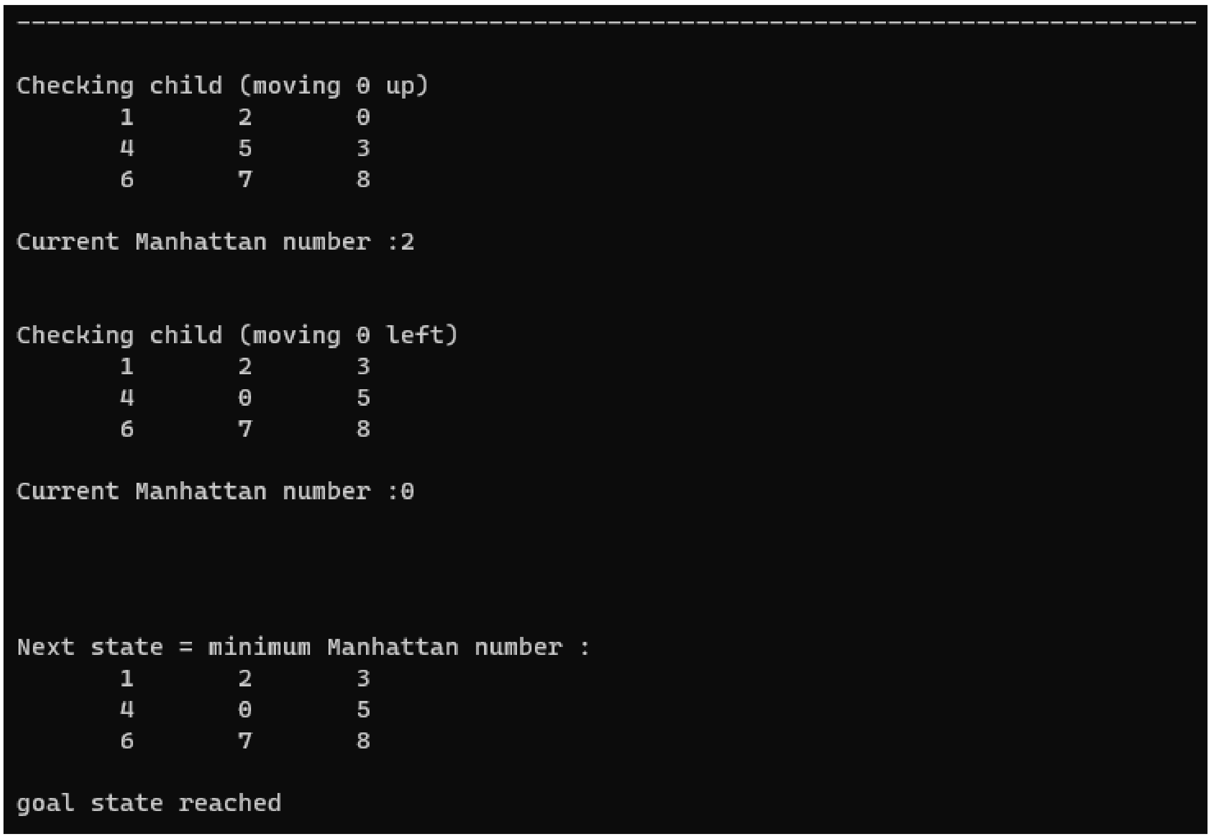
```
---------------------------------------------------------------------------



Checking child (moving 0 right)
       1       2       3
       4       5       8
       6       0       7

Current Manhattan number :3



Next state = minimum Manhattan number :
       1       2       3
       4       5       8
       6       0       7

---------------------------------------------------------------------------
```



```
--------------------------------------------------------------------------------



Checking child (moving 0 up)
        1       2       3
        4       0       8
        6       5       7

Current Manhattan number :4


Checking child (moving 0 right)
        1       2       3
        4       5       8
        6       7       0

Current Manhattan number :2




Next state = minimum Manhattan number :
        1       2       3
        4       5       8
        6       7       0

--------------------------------------------------------------------------------
```



```
--------------------------------------------------------------------------------



Checking child (moving 0 up)
        1       2       3
        4       5       0
        6       7       8

Current Manhattan number :1




Next state = minimum Manhattan number :
        1       2       3
        4       5       0
        6       7       8

--------------------------------------------------------------------------------
```



```
--------------------------------------------------------------------------------

Checking child (moving 0 up)
       1       2       0
       4       5       3
       6       7       8

Current Manhattan number :2


Checking child (moving 0 left)
       1       2       3
       4       0       5
       6       7       8

Current Manhattan number :0




Next state = minimum Manhattan number :
       1       2       3
       4       0       5
       6       7       8

goal state reached
```