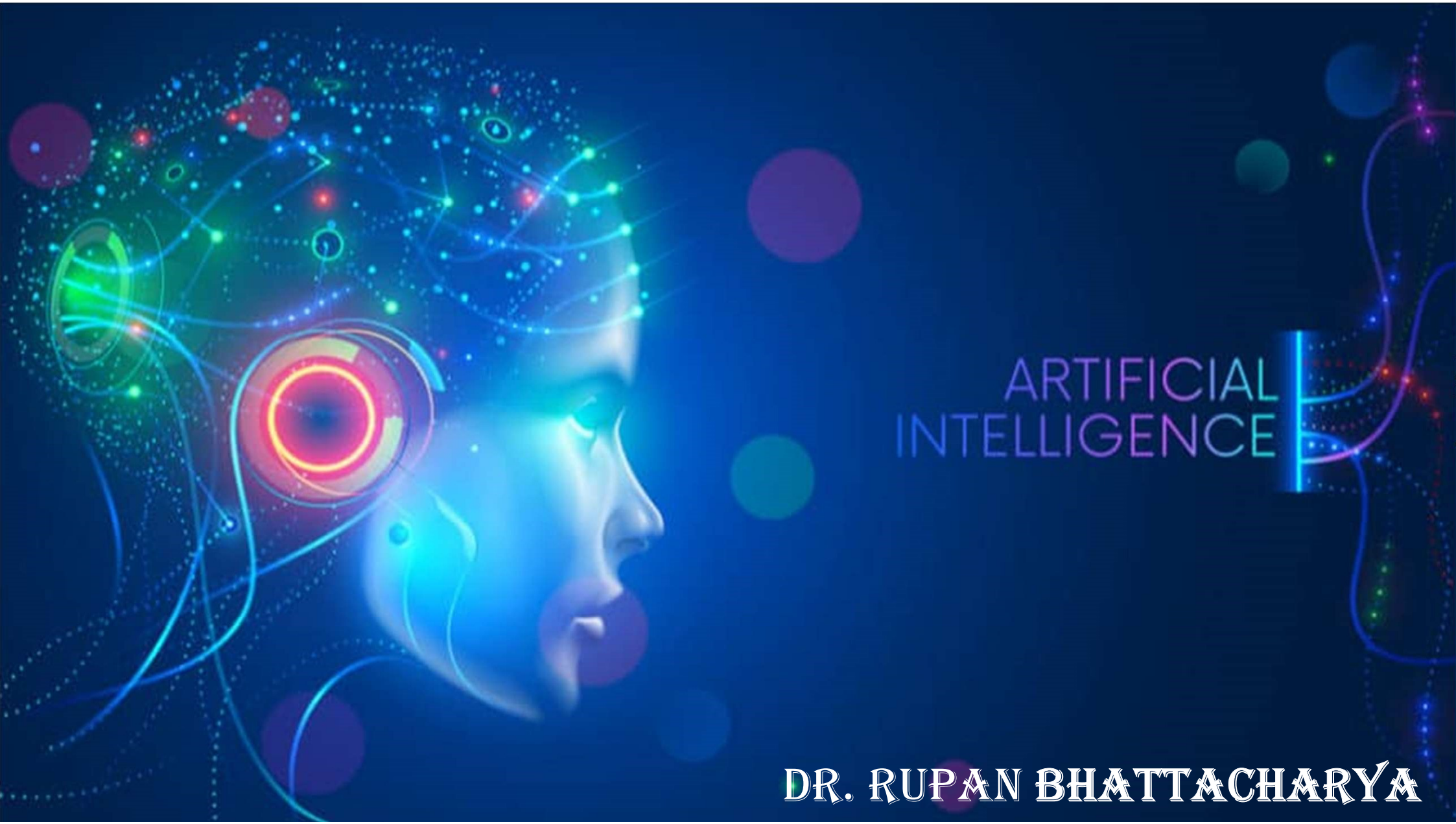
भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHAGALPUR

An Institute of National Importance Under Act of Parliament

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING


ARTIFICIAL
INTELLIGENCE
DR. RUPAN BHATTACHARYA

Submitted by: (3rd year)

1.Karmaveer Kumar (180101023)

2.Dhanu Kumar (180101014)

3.Krishnakant Kumar (180101025)

4.Kajal Rani (180102016)

5. Shreya Sharma (180101042)

6. Mayur Sakher Patel (180101029)

7. Ambati Hari Prasad (180101003)

8. Bammidi Vihar (180103008)

9. Rohit Kumar (180102030)

10. Anupam Dubey (180101009)

CERTIFICATE

This is to certify that the project entitled 8-Puzzle Problem using A* algorithm by the group is a record of their work carried out under my supervision in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering.

Dr. Rupam Bhattacharya

Assistant Professor

Department of CSE

ACKNOWLEDGMENT

We have taken a lot of deliberations in this venture. But it wouldn't have been conceivable without the help and backing of numerous people. We want to enlarge our true appreciation and thank them.

We take this opportunity to express our profound gratitude and deep regards to our guide Dr. Rupam Bhattacharya sir for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessings, help and guidance given by him time to time shall carry us a long way in the journey of life on which we are about to embark.

We are obliged to all the professors of the Department of Computer Science and Engineering, IIIT Bhagalpur for instilling in us the basic knowledge about the field that greatly benefitted us while carrying out the project and achieving the goal.

ABSTRACT

N-puzzle problem has been one of the basic problem since the beginning of artificial intelligence. The most popular version of n-puzzle among people is 8- puzzle problem. It consists of an area divided into 3x3 grid containing 8 numbered (to identify) tiles and one empty grid. We are given an initial state and we have to reach the goal state which is also specified. Various heuristic involved in the informed search like number of misplaced tiles, Manhattan distance were analysed; Manhattan distance being one of the most popular ones. Drawbacks of the heuristics are mentioned and an improvement in Manhattan distance heuristic is implemented.

TABLE OF CONTENTS

8-Puzzzle Problem Using A* Algorithm using h(n) manhattan distance

INITIAL STATE:

4	3	
6	7	2
8	1	5

FINAL STATE:

	1	2
3	4	5
6	7	8

Conditions:

Move can be done only

- **Up**
- **Down**
- **Left**
- **Right**

Function:

- **h(n) = number of misplaced tiles by comparing the current state and the goal state**

Initial state

4	3	
6	7	2
8	1	5

Manhattan distance=2+1+2+2+1+1+1+2=12
State 1

4	3	2
6	7	
8	1	5

Manhattan distance = 11

4		3
6	7	2
8	1	5

Manhattan distance = 13
.....
And so, on until goal state is reached
.....
Final state

0	1	2
3	4	5
6	7	8

Manhattan distance = 0

C-CODE:

Some functions used in c code:

- **For finding Manhattan distance: manhattan()**
- **For finding zero and moving zero: alter()**
- **If we move down, then for finding h(n): diffdown()**
- **If we move left, then for finding h(n): diffleft()**
- **If we move right, then for finding h(n): diffright()**
- **If we move up, then for finding h(n): diffup()**
- **For finding minimum Manhattan distance: minimum()**
- **For printing puzzle: display()**

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int manhattan(int A[][3], int B[][3]);
5   int alter(int A[][3], int B[][3],int steps);
6   int diffup(int A[][3], int B[][3]);
7   int diffdown(int A[][3], int B[][3]);
8   int diffleft(int A[][3], int B[][3]);
9   int diffright(int A[][3], int B[][3]);
10  int minimum(int a, int b, int c, int d);
11  void display(int A[][3]);
```

IN MAIN FUNCTION WE ARE TAKING INPUT OF INITIAL STATE AND GOAL STATE,

WHILE PROCESSING IF PUZZLE OF NODE IS EQUAL GOAL STATE THEN PRINT THE ALL PATH SOLUTION REACHED TO GOAL STATE

```
12  int main(int argc, char *argv[])
13  {
14        int A[3][3] = {{4,3,0}, {6,7,2}, {8,1,5}};
15          int B[3][3] =  {{0,1,2}, {3,4,5}, {6,7,8}};
16           int i, j,d;
17             int steps =0 ;
18             printf("   Target: \n");
19             display(B);
20             printf("   Original: \n");
21             display(A);
22              printf("\ninitial manhattan distance =  %d    \n",manhattan(B,A));
23              while(1)
24              {
25                                d = manhattan(A, B);
26                                          if(d==0)      {  printf("\n\n\n puzzle solved in
    steps : %d\n\n", steps);     return 0;  }
27
28
29                                          steps++;
30
31                                          printf("\nStep: %d \n", steps);
32
33                                          alter(A, B,steps);
34
35                                          display(A);
36
37
38
39              }
40
41              return 0;
42  }
```

- Printing puzzle

```
44  void display(int A[][3])
45  {
46              int i,j;
47                    for(i=0;i<3;i++)
48                                {
49                                                for(j=0;j<3;j++)
50                                                          printf(" %d ", A[i][j]);
51                                                      printf("\n");
52                                                        }
53                      }
54
```

Finding Manhattan distance

```
55  int manhattan(int A[][3], int B[][3])
56  {
57
58      int counter =0 ,i,j,k,l,m,n,p,s=0;
59
60      for(i=0;i<3; i++)
61
62          for( j=0;j<3;j++)
63
64              if(A[i][j] != B[i][j])
65
66              {     p=A[i][j];
67
68                      if((p!=0)&&(B!=0))     {    for(k=0;k<3;k++)
69                         for(l=0;l<3;l++)
70
71                              if(B[k][l]==p){
72                              if(k>i)
73                                  m=k-i;
74                              else
75                                  m=i-k;
76                              if(l>j)
77                                  n=l-j;
78                              else
79                                  n=j-l;
80                              s=n+m;
81                              }
82                  }
83
84                          counter=counter+s;
85                  }
86
87                return counter;
88  }
```

- Finding in which direction Manhattan is minimum

```
90  int alter(int A[][3], int B[][3],int steps)
91  {
92
93          int dup, ddown, dleft, dright;
94               int temp, i , j, flag=0, serial=0,q=steps-1;
95                char ran[4],kch[9],change;
96
97                      dup = diffup(A, B);
98                   ddown = diffdown(A, B);
99                   dleft = diffleft(A, B);
100                 dright = diffright(A, B);
101
102               int min = minimum(dup, ddown, dleft, dright);
103               if(steps==2)
104                     change ='l';
105                else if (steps==3)
106                     change='d';
107
108                  else if (steps==9)
109                       change='r';
110               else
111               {   if (min == dright)
112                                 ran[serial++] = 'r';
113                    if (min == dleft)
114                                 ran[serial++] = 'l';
115                    if (min == dup)
116                                  ran[serial++] = 'u';
117                    if (min == ddown)
118                                            ran[serial++] = 'd';
119
120
121                           int sel = rand()%serial;
122
123                            change = ran[sel];
124
125               }
```

- Move right

```
125                     }
126
127                             if(change == 'r')
128
129                             {for(i=0;i<3;i++)
130
131                                 for (j=0;j<2;j++)
132
133                                     if(A[i][j]==0)
134                                         {       A[i][j] = A[i][j+1];       A[i][j+1] = 0;
135                                     printf("\n minimum manhattan distance =  %d
       \n",manhattan(B,A));
136                                      printf(" right\n"); return 0;   }
137                                                                                          }
138
139
```

- Move left

```
139
140
141                             else if(change == 'l')
142
143                             {for(i=0;i<3;i++)
144
145                                 for (j=1;j<3;j++)
146
147                                     if(A[i][j]==0)       {       A[i][j] = A[i][j-1];
       A[i][j-1] = 0;
148                                     printf("\n minimum  manhattan distance =  %d
       \n",manhattan(B,A));
149                                       printf(" left\n");return 0; }
150
151                             }
152
```

- Move up

```
153
154
155                                 else if(change == 'u')
156
157                                 {for(i=1;i<3;i++)
158
159                                     for (j=0;j<3;j++)
160
161                                         if(A[i][j]==0)
162                                             {       A[i][j] = A[i-1][j];        A[i-1][j] = 0;
163                                             printf("\n minimum manhattan distance =  %d
    \n",manhattan(B,A));
```

```
164                                         printf(" up\n");   return 0;}
165
166                                 }
167
168
```

- Move down

```
169
170                                 else if(change == 'd')
171
172                                 {for(i=0;i<2;i++)
173
174                                     for (j=0;j<3;j++)
175
176                                         if(A[i][j]==0)
177                                             {       A[i][j] = A[i+1][j];        A[i+1][j] = 0;
178                                             printf("\n minimum  manhattan distance =  %d
    \n",manhattan(B,A));
179                                           printf(" down\n"); return 0; }
180
181                                 }
182
183
184
185                                 return 0;
186
187 }
```

- Check Manhattan distance if we will move up

```
189
190   int diffup(int A[][3], int B[][3])
191   {
192           int temp[3][3], i, j;
193
194                 for(i=0;i<3;i++)
195                       for (j=0;j<3;j++)
196                             temp[i][j] = A[i][j];
197
198                  for(i=1;i<3;i++)
199                             for (j=0;j<3;j++)
200
201                             if(A[i][j]==0)
202
203                             {
204
205                                  temp[i-1][j] = 0;
206
207                                  temp[i][j] = A[i-1][j];
208
209                             }
210                       return manhattan(temp, B);
211   }
212
```

- Check Manhattan distance if we will move down

```
212
213   int diffdown(int A[][3], int B[][3])
214   {
215           int temp[3][3], i, j;
216
217
218
219           for(i=0;i<3;i++)
220
221               for (j=0;j<3;j++)
222
223                   temp[i][j] = A[i][j];
224
225
226
227           for(i=0;i<2;i++)
228
229               for (j=0;j<3;j++)
230
231                   if(A[i][j]==0)
232
233                   {
234
235                       temp[i+1][j] = 0;
236
237                       temp[i][j] = A[i+1][j];
238
239                   }
240                     return manhattan(temp, B);
241   }
242
```

- Check Manhattan distance if we will move left

```
242
243   int diffleft(int A[][3], int B[][3])
244   {
245             int temp[3][3], i, j;
246             for(i=0;i<3;i++)
247           for (j=0;j<3;j++)
248
249                     temp[i][j] = A[i][j];
250
251            for(i=0;i<3;i++)
252
253                 for (j=1;j<3;j++)
254
255                     if(A[i][j]==0)
256
257                     {
258
259                         temp[i][j-1] = 0;
260
261                         temp[i][j] = A[i][j-1];
262
263                       return manhattan(temp, B);
264   }
265
```

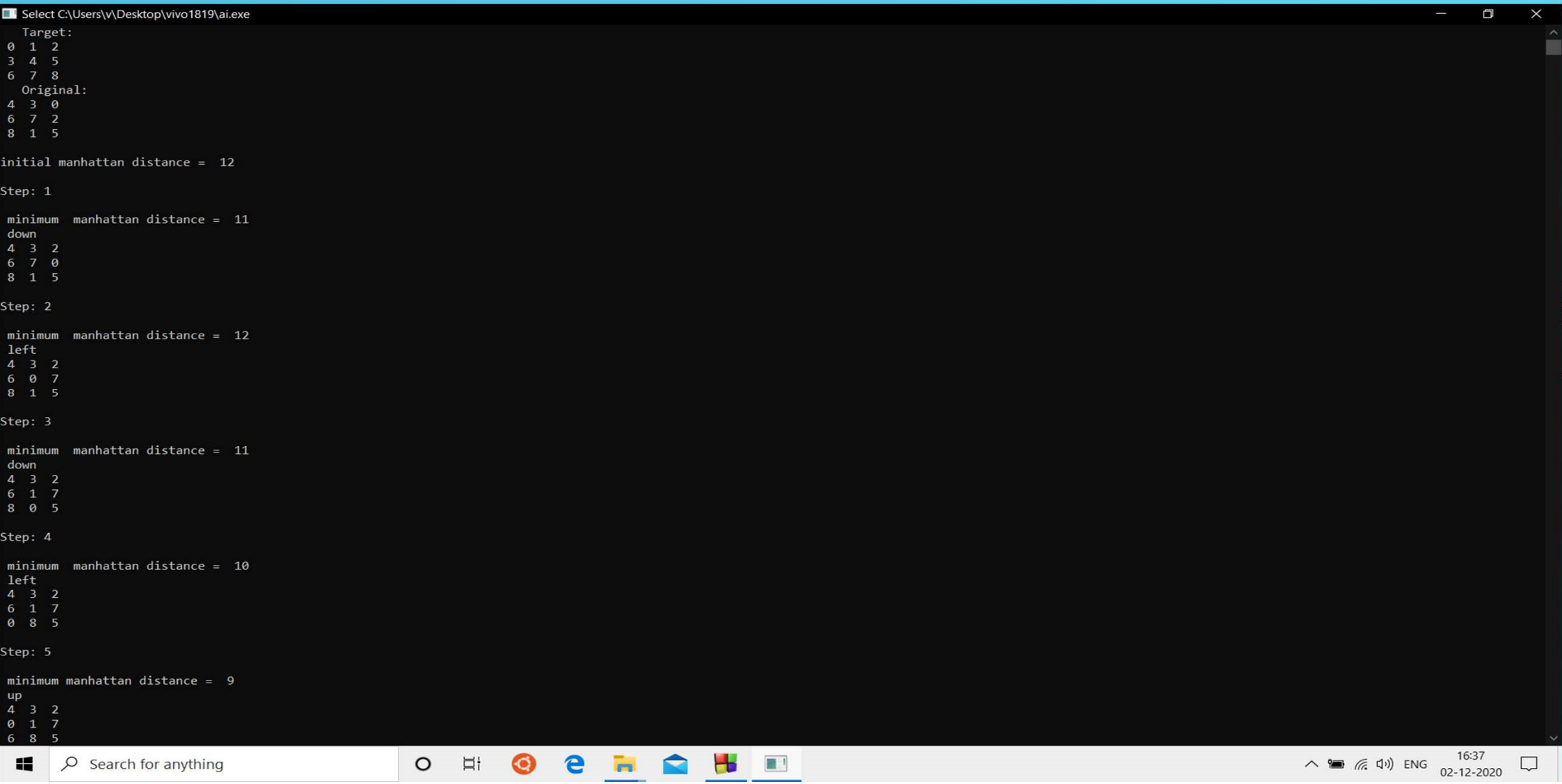
- Check Manhattan distance if we will move right

```
266   int diffright(int A[][3], int B[][3])
267   {
268           int temp[3][3], i, j;
269           for(i=0;i<3;i++)
270
271               for (j=0;j<3;j++)
272                   temp[i][j] = A[i][j];
273
274           for(i=0;i<3;i++)
275               for (j=0;j<2;j++)
276                   if(A[i][j]==0)
277
278                   {
279                       temp[i][j+1] = 0;
280                       temp[i][j] = A[i][j+1];
281
282                   }
283           return manhattan(temp, B);
284
285   }
286
```

- Finding minimum manhattan

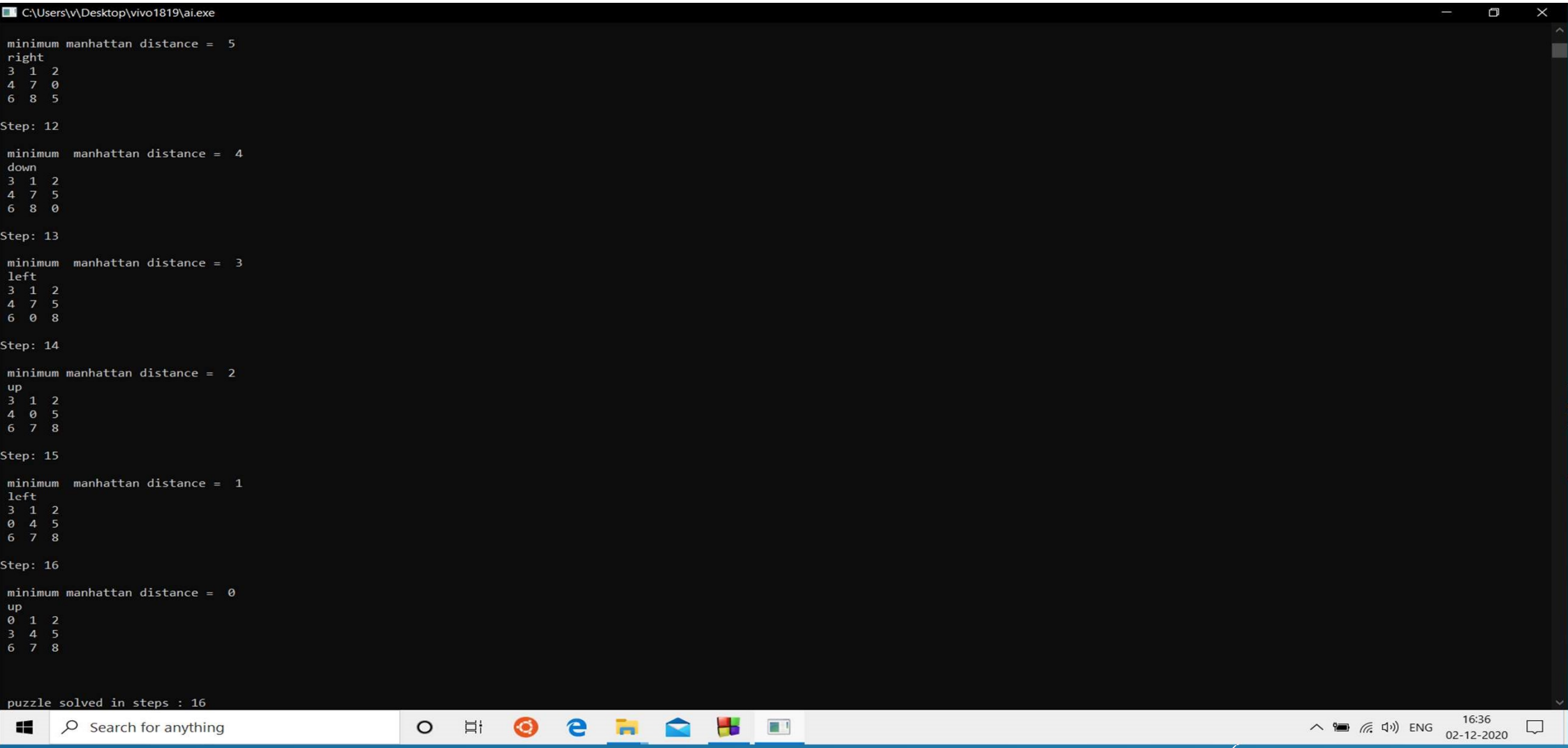
```
286
287
288   int minimum (int a, int b, int c , int d)
289   {
290       int min = a;
291       if(b<min)
292           min= b;
293       if(c<min)
294           min = c;
295       if(d<min)
296           min = d;
297
298
299
300       return min;
301   }
302
```

▶ OUTPUT:


Select C:\Users\v\Desktop\vivo1819\ai.exe
Target:
0 1 2
3 4 5
6 7 8
Original:
4 3 0
6 7 2
8 1 5
initial manhattan distance = 12
Step: 1
minimum manhattan distance = 11
down
4 3 2
6 7 0
8 1 5
Step: 2
minimum manhattan distance = 12
left
4 3 2
6 0 7
8 1 5
Step: 3
minimum manhattan distance = 11
down
4 3 2
6 1 7
8 0 5
Step: 4
minimum manhattan distance = 10
left
4 3 2
6 1 7
0 8 5
Step: 5
minimum manhattan distance = 9
up
4 3 2
0 1 7
6 8 5
Search for anything
ENG
16:37
02-12-2020

Select C:\Users\v\Desktop\vivo1819\ai.exe

```
Step: 5

 minimum manhattan distance =  9
 up
 4  3  2
 0  1  7
 6  8  5

Step: 6

 minimum manhattan distance =  8
 up
 0  3  2
 4  1  7
 6  8  5

Step: 7
 0  3  2
 4  1  7
 6  8  5

Step: 8
 0  3  2
 4  1  7
 6  8  5

Step: 9

 minimum manhattan distance =  7
 right
 3  0  2
 4  1  7
 6  8  5

Step: 10

 minimum  manhattan distance =  6
 down
 3  1  2
 4  0  7
 6  8  5

Step: 11

 minimum manhattan distance =  5
 right
 3  1  2
 4  7  0
 6  8  5
```



NO. OF STEPS REQUIRED TO SOLVE THE PUZZLE: 16 STEPS

ANALYSIS OF A* :

- A* is optimally efficient. i.e. if there exists a path from start to goal node then a* guarantees to find the optimal path.
- It is complete i.e., if a solution exists then it is found.
- Complexity-As the algorithm is optimally efficient so other algorithm can guarantee to examine fewer nodes. However
- **Time complexity** –exponential O(b^d)
- where b=branching factor
- d= depth of the tree.
- **space complexity**- exponential O(b^d)
- It stores all the nodes generated in the open list.