## Practical: 04

<u>Aim:</u> Write a program to implement Heuristic(Steepest Ascent)Search for 8 puzzle game problem.

## Theory:

In artificial intelligence, a **heuristic** is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

A **heuristic function**, also called simply a **heuristic**, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution.

**Steepest-Ascent Hill climbing:** It first examines all the neighboring nodes and then selects the node closest to the solution state as of next node.

Heuristic is a technique designed to solve a problem quickly, when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- o Hill Climbing is mostly used when a good heuristic is available.
- only keeps a single current state.

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

 Step 1: Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.

o **Step 2:** Loop until a solution is found or the current state does not change.

a. Let SUCC be a state such that any successor of the current state will be better than it.

b. For each operator that applies to the current state:

a. Apply the new operator and generate a new state.

b. Evaluate the new state.

c. If it is goal state, then return it and quit, else compare it to the SUCC.

d. If it is better than SUCC, then set new state as SUCC.

e. If the SUCC is better than the current state, then set current state to SUCC.

Step 5: Exit.

Program:

```
1
      #include <stdio.h>
 2
      int final[3][3];
 3
      int track[3][3];
      void compare();
 4
 5
      int start[3][3],goal[3][3];
      int copy[3][3];
int i,j,l,m,k,flag=0,max=9999,h=0,count=0;
 6
    int main(void) {
 8
        printf("Enter the start state\n");
 9
10
        for(i=0;i<3;i++)
     - (
11
12
          for(j=0;j<3;j++)
13
14
            scanf("%d", &start[i][j]);
15
          }
16
         for(i=0;i<3;i++)
17
18
19
          for(j=0;j<3;j++)
20
            copy[i][j]=start[i][j];
21
22
            final[i][j]=start[i][j];
23
24
25
         printf("Enter the goal state\n");
         for(i=0;i<3;i++)
26
27
          for(j=0;j<3;j++)
28
29
30
            scanf("%d", &goal[i][j]);
31
          }
32
33
         printf("\nThe path is\n");
         int t=0;
34
35
         int temp;
36
         while (flag!=1)
37
          for (k=0; k<4; k++)
38
```

```
39
40
             t=0;
             h=0;
41
42
             for(i=0;i<3;i++)
43
44
               for(j=0;j<3;j++)
45
46
                 copy[i][j]=final[i][j];
47
               }
             }
48
49
50
51
             if(k==0)//up
52
53
                for(i=0;i<3;i++)
54
55
                  for(j=0;j<3;j++)
56
57
                    if(copy[i][j]==0)
58
59
                     l=i;
60
                      m=j;
61
                      t=1;
62
                    if(t==1)
63
64
65
                      break;
66
67
                  1
68
69
                if(1-1>=0)
70
71
                  temp=copy[1][m];
72
                  copy[1][m]=copy[1-1][m];
                  copy[1-1][m]=temp;
73
74
```

```
75
 76
                 for(i=0;i<3;i++)
 77
 78
                   for(j=0;j<3;j++)
 79
 80
                     if(goal[i][j]!=copy[i][j])
 81
 82
                     h++;
 83
 84
 85
 86
                 if (max>h)
 87
 88
                  max=h;
 89
                  h=0;
 90
                  for(i=0;i<3;i++)
 91
 92
                     for(j=0;j<3;j++)
 93
 94
                     track[i][j]=copy[i][j];
 95
 96
                  }
 97
                 }
 98
             }
 99
              if(k==1)//down
100
101
102
                 for(i=0;i<3;i++)
103
104
                  for(j=0;j<3;j++)
105
106
                     if(copy[i][j]==0)
107
108
                      1=i;
109
                      m=j;
110
                      t=1;
111
                     if(t==1)
112
```

```
113
                   -
114
                      break;
115
116
                   }
117
                 }
118
                 if(1+1<=2)
119
                 1
120
                  temp=copy[1][m];
121
                   copy[1][m]=copy[1+1][m];
122
                   copy[1+1][m]=temp;
123
124
                 for(i=0;i<3;i++)
125
126
127
                   for(j=0;j<3;j++)
128
129
                    if(goal[i][j]!=copy[i][j])
130
131
                      h++;
132
                    }
133
134
                 }
135
                 if (max>h)
136
137
                   max=h;
138
                   h=0;
139
                   for(i=0;i<3;i++)
140
141
                    for(j=0;j<3;j++)
142
143
                       track[i][j]=copy[i][j];
144
145
                  1
                 }
146
147
              }
148
              if(k==2)//left
149
150
                 for(i=0;i<3;i++)
```

```
152
                    for(j=0;j<3;j++)
153
154
                      if(copy[i][j]==0)
155
156
                        l=i;
157
                        m=j;
158
                        t=1;
159
160
                      if(t==1)
161
162
                        break;
163
164
                    }
165
                  }
166
                  if(m-1>=0)
167
168
                    temp=copy[1][m];
169
                    copy[1][m]=copy[1][m-1];
170
                    copy[1][m-1]=temp;
171
172
173
                 for(i=0;i<3;i++)
174
175
                    for(j=0;j<3;j++)
176
177
                      if(goal[i][j]!=copy[i][j])
178
179
                        h++;
180
181
182
                 }
183
                 if (max>h)
184
185
                    max=h;
186
                    h=0;
187
                    for (i=0; i<3; i++)
188
189
                      for(j=0;j<3;j++)
```

```
190
                     -
191
                        track[i][j]=copy[i][j];
192
193
                   }
                  }
194
195
196
               if(k==3)//right
197
198
                  for(i=0;i<3;i++)
199
200
                    for(j=0;j<3;j++)
201
202
                      if(copy[i][j]==0)
203
204
                        1=i;
205
                        m=j;
206
                        t=1;
207
                      }
208
                      if(t==1)
209
210
                        break;
211
                      }
212
                    }
                  }
213
214
                  if(m+1<=2)
215
216
                    temp=copy[1][m];
217
                    copy[1][m]=copy[1][m+1];
218
                    copy[1][m+1]=temp;
219
                  }
220
221
                  for(i=0;i<3;i++)
222
223
                    for(j=0;j<3;j++)
224
                      if(goal[i][j]!=copy[i][j])
225
226
227
                        h++;
```

```
227
                          h++;
 228
                        }
 229
                     1
 230
                   }
 231
                    if (max>h)
 232
                   -
 233
                     max=h;
 234
                     h=0;
 235
                      for(i=0;i<3;i++)
 236
 237
                        for(j=0;j<3;j++)
 238
 239
                          track[i][j]=copy[i][j];
 240
 241
 242
 243
 244
 245
              printf("\n");
 246
 247
              for(i=0;i<3;i++)
 248
 249
                for(j=0;j<3;j++)
 250
 251
                  final[i][j]=track[i][j];
 252
                  printf("%d ",final[i][j]);
 253
 254
                printf("\n");
 255
 256
              for(i=0;i<3;i++)
 257
 258
                for(j=0;j<3;j++)
 259
 260
                  if(goal[i][j] == final[i][j])
 261
 262
                    count++;
 263
                  }
 264
265
                if (count==9)
 266
 267
                  flag=1;
 268
 269
              }
 270
              count=0;
 271
 272
 273
 274
          }
 275
```

## Output:

■ Select "C:\Users\LENOVO\Desktop\AI Practical\heuristic.exe"

```
Enter the start state
1 2 3
5 6 0
7 8 4
Enter the goal state
1 2 3
5 8 6
074
The path is
1 2 3
5 0 6
7 8 4
1 2 3
5 8 6
7 0 4
1 2 3
5 8 6
074
                           execution time : 32.196 s
Process returned 0 (0x0)
Press any key to continue.
```

Output: : Successfully Implemented Heuristic (Steepest Ascent) Search for 8 puzzle problem game problem.