```
"""
Brian Tremaine  April 8, 2019
Python implementation of the A* algorithm using my code for Dijkstra as
a starting point.

TremaineConsultingGroup.com

pseudo code:
function astar(Graph, source):
       create list of all vertices Q
       create list of all visited vertices, S

       for each vertex v in Graph:
           dist[v] ← INFINITY
           prev[v] ← UNDEFINED
           add v to Q
      dist[source] ← 0

      while Q is not empty:
          u ← vertex in Q with min dist[u]
          S ← add u to S
          remove u from Q

          for each neighbor v of u:       // only v that are still in Q
              alt ← dist[u] + w(u, v)
              if alt < dist[v]:
                  dist[v] ← alt
                  prev[v] ← u

      return dist[], prev[]
  end function
"""
import numpy as np

def astar(A, start, end):
    """ 'A' is a list of rows with filled in occupied sites
        'start' and 'end' are the respective start & end grids locations.
        Note: With 'A' mxn there are n*m grids.

        The list of unvisited nodes, Q, and visited nodes, S, will
        have up to n*m entries for the above example. Q and S are arranged
        as a list of rows.
    """
    INFINITY = 9999
    nodes = len(A)*len(A[0])  # number of nodes
    N= len(A)
    M= len(A[0])

    total = 0    # keep track of # iterations
    # initialize dict
    dist = {}
    prev = {}
    cost = {}
    for i in range(nodes):
        dist.update({i:INFINITY})   # 'cost' function for algorithm
        cost.update({i:INFINITY})   # length from start to node
        prev.update({i:-1})

    dist[vertex(start,A)] = 0
    prev[vertex(start,A)] = 0
    cost[vertex(start,A)] = 0

    Q= []  # list of unvisited nodes
    S= []  # list of visited nodes
    for v in range(nodes):
        Q.append(v)

```

```
68      neighbors = [[-1,0],[1,0],[0,1],[0,-1],[1,1],[1,-1],[-1,-1],[-1,1]]
69      while Q:
70         # find vertex in Q with minimum dist and remove from list
71         # ======================================================
72         u = minVertex(Q, dist)  # u is str label
73         if u == vertex(end,A): break  # break when reached end node
74         Q.remove(u)
75         S.append(u)
76
77         # search neighbors of u in A, updating dist and prev
78         #[dy][dx]
79         x1 = u % (N)      # u = [y1][x1]
80         y1 = int(u/M)
81         for point in neighbors:
82           # redefine v as neighbor vertex to u
83           x2 = x1 + point[1]
84           y2 = y1 + point[0]
85           if (x2>=0) & (x2<M) & (y2>=0) & (y2<N) :
86              v = x2 + y2 * M  # new node
87
88              if A[y2][x2] !=  'x':
89                 # cost function update
90                 if dist[v] > dist[u] + weight(point,x2,y2,end):
91                    dist[v] = dist[u] + weight(point,x2,y2,end)
92                    cost[v] = cost[u] + distance(point)
93
94                    # print(u, v, x1, y1, x2, y2)
95                    prev[v] = u
96                    total = total + 1
97
98         # update output return variables
99         path= prev
100        length = cost[vertex(end,A)]
101     # ============================================================
102
103     print('total steps', total)
104
105     return path, length, dist, Q, S
106
107  def nodeNum(Map, coord):
108     # calculate node # from (y,x) point
109     num = coord[1] + len(Map[0])*coord[0]
110
111     return num
112
113  def minVertex(Q, dst):
114     # return vertex str in Q with minimum distance
115     temp = {}
116     for j in Q:
117         temp.update({j:dst[j]})
118
119     return min(temp, key=temp.get)
120
121  def weight(p,x2,y2,end):
122     # cost function g(n) + h(n)
123     val = np.sqrt(p[0]**2 + p[1]**2) + \
124             np.sqrt((end[0]-y2)**2 + (end[1]-x2)**2)
125
126     return val
127
128  def distance(p):
129     # distance start to node p, g(n)
130     val = np.sqrt(p[0]**2 + p[1]**2)
131
132     return val
133
134  def vertex(coord, A):
```

```
135       # convert [y][x] to vertex #
136       ver = coord[0]*len(A)+coord[1]
137
138       return ver
139
140   def show_path(A, path, start, end):
141       # display path
142       # calculate nodes
143       nstart = nodeNum(A, start)
144       nend = nodeNum(A,end)
145
146       A[start[0]][start[1]] = 'S'
147
148       s = nend
149       while s != nstart:
150           coord = [int(s/len(A)), s % len(A[0])]   # [y][x]
151           A[coord[0]][coord[1]] = 1
152           s = path[s]
153
154       A[end[0]][end[1]] = 'F'
155       U  = A
156
157       return U
158
159   # ========================================================================
160   if __name__ == '__main__':
161       # initialize grid
162       # ========================================================================
163       # ========================================================================
164       # graph for specific problem goes next, list of rows
165       # 0 denotes vacant, 'x' denotes occupied
166
167     if 1:
168       A=        [[0, 0, 0,  0, 'x', 0, 0,  0,  0, 0],
169                  [0, 0, 0,  0, 'x', 0, 0,  0,  0, 0],
170                  [0, 0, 0,  0, 'x', 0, 0,  0,  0, 0],
171                  [0, 0, 0,  0, 'x', 0, 0,  0,  0, 0],
172                  [0, 0, 0,  0, 'x', 0, 0,  0,  0, 0],
173                  [0, 0, 0,  0,  0,  0, 0,  0,  0, 0],
174                  [0, 0, 0, 'x', 0,  0, 0,  0,  0, 0],
175                  [0, 0, 0, 'x', 0,  0, 0, 'x', 0, 0],
176                  [0, 0, 0, 'x', 0,  0, 0,  0,  0, 0],
177                  [0, 0, 0, 'x', 0,  0, 0,  0,  0, 0]]
178
179       start = [0,0]
180       end = [8,8]
181
182     else:
183     # if redefined above A not used
184       A=       [[0, 0, 0, 0, 0],
185                 [0, 0, 0, 0, 0],
186                 [0, 0,'x',0, 0],
187                 [0, 0,'x',0, 0],
188                 [0, 0, 0, 0, 0]]
189
190       start = [3,0]
191       end = [1,4]
192
193     # ========================================================================
194     path,length,dist, Q, S= astar(A,start,end)
195     U = show_path(A, path, start, end)
196
197     print('distance', length)
198
199
200
```