```python
import heapq

class Node:
    def __init__(self, state, parent, g, h):
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

    def __lt__(self, other):
        return self.f < other.f

def a_star_search(start, goal, heuristic,
neighbors):
    open_list = []
    closed_set = set()
    node_map = {}

    start_node = Node(start, None, 0, heuristic(start
 goal))
    heapq.heappush(open_list, start_node)
    node_map[start] = start_node

    while open_list:
        current_node = heapq.heappop(open_list)

        if current_node.state == goal:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
            return path[::-1]

        closed_set.add(current_node.state)

```

```
35          for neighbor, cost in neighbors(current_node.
    state):
36              if neighbor in closed_set:
37                  continue
38
39              g = current_node.g + cost
40              h = heuristic(neighbor, goal)
41              neighbor_node = Node(neighbor,
    current_node, g, h)
42
43              if neighbor not in node_map or g <
    node_map[neighbor].g:
44                  heapq.heappush(open_list,
    neighbor_node)
45                  node_map[neighbor] = neighbor_node
46
47      return None
48
49  def heuristic(state, goal):
50      return abs(state[0] - goal[0]) + abs(state[1] -
    goal[1])
51
52  def neighbors(state):
53      x, y = state
54      return [
55          ((x + 1, y), 1),
56          ((x - 1, y), 1),
57          ((x, y + 1), 1),
58          ((x, y - 1), 1)
59      ]
60
61  start = (0, 0)
62  goal = (3, 3)
63
64  path = a_star_search(start, goal, heuristic,
    neighbors)
65  print("Path from start to goal:", path)
```

```
Path from start to goal: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (
3, 2), (3, 3)]

[Program finished]
```