```
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_list = set()
  start_node = Node(start, None, 0, heuristic(start, goal))
  heapq.heappush(open_list, start_node)


  while open_list:
      current = heapq.heappop(open_list)
      if current.state == goal:
          path = []
          while current:
              path.append(current.state)
              current = current.parent
          return path[::-1]
      closed_list.add(current.state)
      for neighbor, cost in neighbors(current.state):
          if neighbor in closed_list:
              continue
          g = current.g + cost
          h = heuristic(neighbor, goal)
          neighbor_node = Node(neighbor, current, g, h)
          if not any(n.state == neighbor and n.f <= neighbor_node.f for n in open_list):
              heapq.heappush(open_list, neighbor_node)
  return None


def heuristic(state, goal):
  return abs(state[0] - goal[0]) + abs(state[1] - goal[1])


def neighbors(state):
  x, y = state
  return [((x + 1, y), 1), ((x - 1, y), 1), ((x, y + 1), 1), ((x, y - 1), 1)]


start = (0, 0)
goal = (3, 3)
path = a_star_search(start, goal, heuristic, neighbors)
print("Path from start to goal:", path)
```