# O algorytmie

Algorytm A* od wierzchołka początkowego tworzy ścieżkę, za każdym razem wybierając wierzchołek x z dostępnych w danym kroku niezbadanych wierzchołków tak, by minimalizować funkcję f ( x ) f(x) zdefiniowaną:

```
f ( x ) = g ( x ) + h ( x ) ,
```

gdzie:

```
g ( x ) - droga pomiędzy wierzchołkiem początkowym a x.
h ( x ) - przewidywana przez heurystykę droga od x do wierzchołka docelowego.
```

W każdym kroku algorytm dołącza do ścieżki wierzchołek o najniższym współczynniku f. Kończy w momencie natrafienia na wierzchołek będący wierzchołkiem docelowym.

**Inicjalizacja parametrów**

```
async def tractor():
    directions = ['N', 'W', 'S', 'E']
    current_rotation = 'N'
    last_rotation = None
    start = (0,0)
    current_position = start
    goal = (10,6)

    frontier = PriorityQueue()
    frontier.put(start, 0)
    cost_so_far = {}
    cost_so_far[start] = 0
    start_flag = True

    while not frontier.empty():
        local_graph = []
        current = frontier.get()
        last_position = current
        if not start_flag:
            if current[0] == last_position[0] and current[1] > last_position[1]:
                current_rotation = 'E'
            elif current[0] == last_position[0] and current[1] < last_position[1]:
                current_rotation = 'W'
            elif current[0] > last_position[0]:
                current_rotation = 'S'
            elif current[0] < last_position[0]:
                current_rotation = 'N'
        last_rotation = current_rotation
```

**budawanie "lokalnego drzewa"**

```
        for i, direction in enumerate(directions):
            reader, writer = await asyncio.open_connection('127.0.0.1', 8888)
            writer.write(("rotate " + direction + "\n").encode())
            data = await reader.readline()
            time.sleep(0.7)
            writer.write("move\n".encode())
            data = await reader.readline()
            result = data.decode().split()
            time.sleep(0.7)
            if result[0] == "OK":
                if direction == 'N':
                    local_graph.append((current[0] - 1, current[1]))
                elif direction == 'S':
                    local_graph.append((current[0] + 1, current[1]))
```

```
                elif direction == 'W':
                    local_graph.append((current[0], current[1] - 1 ))
                else:
                    local_graph.append((current[0], current[1] + 1 ))

                writer.write(("rotate " + directions[((i + 2) % 4)] + "\n").encode())
                data = await reader.readline()
                time.sleep(0.7)
                writer.write("move\n".encode())
                data = await reader.readline()
                time.sleep(0.7)
```

**właściwa część algorytmu**

```
        writer.write(("rotate " + current_rotation + "\n").encode())
        data = await reader.readline()
        time.sleep(0.7)
        writer.write("move\n".encode())
        data = await reader.readline()
        time.sleep(0.7)

        if current == goal:
            break

        for next in local_graph:
            new_cost = cost_so_far[current] + 1
            if next not in cost_so_far or new_cost < cost_so_far[next]:
                cost_so_far[next] = new_cost
                priority = new_cost + heuristic(goal, next)
                frontier.put(next, priority)

        start_flag = False
        writer.close()
```

**heurestyka**

```
def heuristic(a, b):
    (x1, y1) = a
    (x2, y2) = b
    return abs(x1 - x2) + abs(y1 - y2)
```