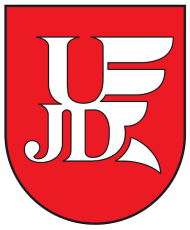


Podstawy sztucznej inteligencji

Instrukcja do laboratorium nr 12:

Algorytm A^*

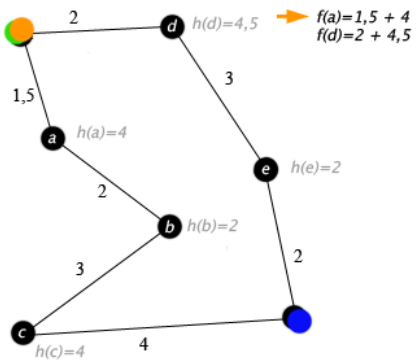
1 Cel laboratorium

Celem laboratorium jest zapoznanie się z algorytmem A^* i zaimplementowanie tego algorytmu do rozwiązania określonego problemu.

2 Wprowadzenie

A* (wymawiane „A-star”) to algorytm przechodzenia przez graf i przeszukiwania ścieżki , który jest używany w wielu dziedzinach informatyki ze względu na jego kompletność, optymalność i optymalną wydajność. W porównaniu z algorytmem Dijkstry, algorytm A* znajduje tylko najkrótszą ścieżkę od określonego źródła do określonego celu, a nie drzewo najkrótszej ścieżki od określonego źródła do wszystkich możliwych celów. Jest to niezbędny kompromis w przypadku stosowania heurystyki ukierunkowanej na określony cel.

Algorytm wyszukiwania A* jest prostym i wydajnym algorytmem wyszukiwania, który może być użyty do znalezienia optymalnej ścieżki pomiędzy dwoma węzłami w grafie. Zostanie on wykorzystany do wyszukiwania najkrótszej ścieżki. Jest on rozszerzeniem algorytmu najkrótszej ścieżki Dijkstry (Dijkstra's Algorithm). Rozszerzenie polega tu na tym, że zamiast używać kolejki priorytetowej do przechowywania wszystkich elementów, używamy stert (drzew binarnych) do ich przechowywania. Algorytm wyszukiwania A* wykorzystuje również funkcję heurystyczną, która dostarcza dodatkowych informacji dotyczących tego, jak daleko od węzła docelowego się znajdujemy. Funkcja ta jest używana w połączeniu ze strukturą danych f-heap w celu zwiększenia efektywności wyszukiwania.



Poniżej znajduje się przykład zadania z zastosowaniem algorytmu A^*

Zaimplementuj algorytm A*, aby znaleźć najkrótszą ścieżkę z punktu początkowego (0, 0) do punktu końcowego (4, 4) na planszy 5x5. Plansza

zawiera przeszkody, które uniemożliwiają ruch przez niektóre pola. Plansza 5x5 reprezentowana przez listę list, gdzie 0 oznacza wolne pole, a 1 oznacza przeszkodę.

```
grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]
```

Rozwiązanie

```
import heapq

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_set = []
    heapq.heappush(open_set, (0, start))

    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            return reconstruct_path(came_from, current)

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (current[0] + dx,
```

```
                    current[1] + dy)

                if 0 <= neighbor[0] < rows and 0
                    <= neighbor[1] < cols:
                    if grid[neighbor[0]][
                        neighbor[1]] == 1:
                        continue

                    tentative_g_score = g_score[
                        current] + 1

                    if neighbor not in g_score
                        or tentative_g_score <
                        g_score[neighbor]:
                        came_from[neighbor] =
                            current
                        g_score[neighbor] =
                            tentative_g_score
                        f_score[neighbor] =
                            tentative_g_score +
                            heuristic(neighbor,
                            goal)
                        heapq.heappush(open_set,
                            (f_score[neighbor],
                            neighbor))

        return None

    def reconstruct_path(came_from, current):
        path = []
        while current in came_from:
            path.append(current)
            current = came_from[current]
        path.append(current)
        path.reverse()
        return path

    grid = [
        [0, 1, 0, 0, 0],
        [0, 1, 0, 1, 0],
        [0, 0, 0, 1, 0],
```

```
51              [0, 1, 1, 1, 0],
52              [0, 0, 0, 0, 0]
53          ]
54
55          start = (0, 0)
56          goal = (4, 4)
57
58          path = astar(grid, start, goal)
59
60
61          print("Shortest path:", path)
```

Przenalizujmy składowe tego algorytmu.

Funkcja heuristic(a, b) oblicza odległość Manhattan między dwoma punktami a i b. Jest to suma różnic między współrzędnymi x i y tych punktów.

$$h(a, b) = |a_x - b_x| + |a_y - b_y|$$

Gdzie:

- a_x, a_y to współrzędne punktu a,
- b_x, b_y to współrzędne punktu b.

Koszt dotarcia do węzła n (oznaczany jako $g(n)$) to suma kosztów przejścia przez kolejne węzły na ścieżce od punktu początkowego do n. Dla siatki, w której każdy ruch ma stały koszt 1, koszt ten można obliczyć jako:

$$g(n) = g(\text{current}) + 1$$

Gdzie:

- $g(\text{current})$ to koszt dotarcia do bieżącego węzła.

$$f(n) = g(n) + h(n, \text{goal})$$

Gdzie:

- $g(n)$ to rzeczywisty koszt dotarcia do węzła n,
- $h(n, \text{goal})$ to estymowany koszt dotarcia z węzła n do węzła docelowego (heurystyka).

Ścieżka jest odtwarzana, podążając wstecz od węzła końcowego goal do punktu początkowego start za pomocą zbioru came_from, który przechowuje, skąd przyszliśmy do każdego węzła:

path = [current]

while current in came_from:

current = came_from[current]

path.append(current)

path.reverse()

3 Zadania

Proszę o realizację następujących zadań.

1. Zaimplementuj algorytm A*, aby znaleźć najkrótszą ścieżkę z punktu początkowego (0, 0) do punktu końcowego (3, 3) na planszy 4x4. Plansza zawiera przeszkody, które uniemożliwiają ruch przez niektóre pola. Plansza 4x4 reprezentowana przez listę list, gdzie 0 oznacza wolne pole, a 1 oznacza przeszkodę.

```
grid = [
    [0, 1, 0, 0],
    [0, 1, 0, 1],
    [0, 0, 0, 1],
    [0, 1, 0, 0]
]
```

2. Zaimplementuj algorytm A*, aby znaleźć najkrótszą ścieżkę z punktu początkowego (0, 0) do punktu końcowego (5, 5) na planszy 6x6. Plansza zawiera kilka przeszkód. Plansza 6x6 reprezentowana przez listę list, gdzie 0 oznacza wolne pole, a 1 oznacza przeszkodę.

```
grid = [
    [0, 0, 1, 0, 0, 0],
    [0, 1, 1, 1, 1, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 1, 1, 1, 0, 1],
```

```
        [0, 0, 0, 1, 0, 1],
        [0, 1, 0, 0, 0, 0]
    ]
```

3. Zaimplementuj algorytm A*, aby znaleźć najkrótszą ścieżkę z punktu początkowego (0, 0) do punktu końcowego (4, 4) na planszy 5x5. Plansza zawiera bardziej skomplikowane przeszkody. Plansza 5x5 reprezentowana przez listę list, gdzie 0 oznacza wolne pole, a 1 oznacza przeszkodę.

```
    grid = [
        [0, 1, 1, 1, 0],
        [0, 0, 0, 1, 0],
        [1, 1, 0, 1, 0],
        [0, 0, 0, 0, 1],
        [1, 1, 1, 0, 0]
    ]
```

4. Zaimplementuj algorytm A*, aby znaleźć najkrótszą ścieżkę na planszy 5x5. Punkt początkowy i końcowy mogą być różne niż (0, 0) i (4, 4). Plansza 5x5 reprezentowana przez listę list, gdzie 0 oznacza wolne pole, a 1 oznacza przeszkodę. Punkt początkowy i końcowy przekazane jako parametry.

```
    grid = [
        [0, 0, 1, 0, 0],
        [1, 0, 1, 0, 1],
        [0, 0, 0, 0, 0],
        [1, 1, 0, 1, 0],
        [0, 0, 1, 0, 0]
    ]
```

5. Zaimplementuj algorytm A*, aby znaleźć najkrótszą ścieżkę z punktu początkowego (0, 0) do punktu końcowego (6, 6) na planszy 7x7. Plansza zawiera wąskie przejścia. Plansza 7x7 reprezentowana przez listę list, gdzie 0 oznacza wolne pole, a 1 oznacza przeszkodę.

```
    grid = [
```

```
            [0, 0, 0, 1, 0, 0, 0],
            [1, 1, 0, 1, 1, 1, 0],
            [0, 0, 0, 0, 0, 1, 0],
            [0, 1, 1, 1, 0, 1, 0],
            [0, 1, 0, 0, 0, 1, 0],
            [0, 1, 0, 1, 1, 1, 0],
            [0, 0, 0, 1, 0, 0, 0]
        ]
```

4 Sprawozdanie

W sprawozdaniu przedstaw wyniki swoich eksperymentów:

- Imię i nazwisko
- Nazwę ćwiczenia
- Opis algorytmu
- Zestawienie wyników (wykresy, tabele z komentarzem).
- Wnioski końcowe Skrypt programu (załącz jako osobny plik z rozszerzeniem .py w kursie Podstawy sztucznej inteligencji na portalu e-nauka).

Skrypt programu (załącz jako osobny plik z rozszerzeniem .py w kursie Podstawy sztucznej inteligencji na portalu e-nauka).