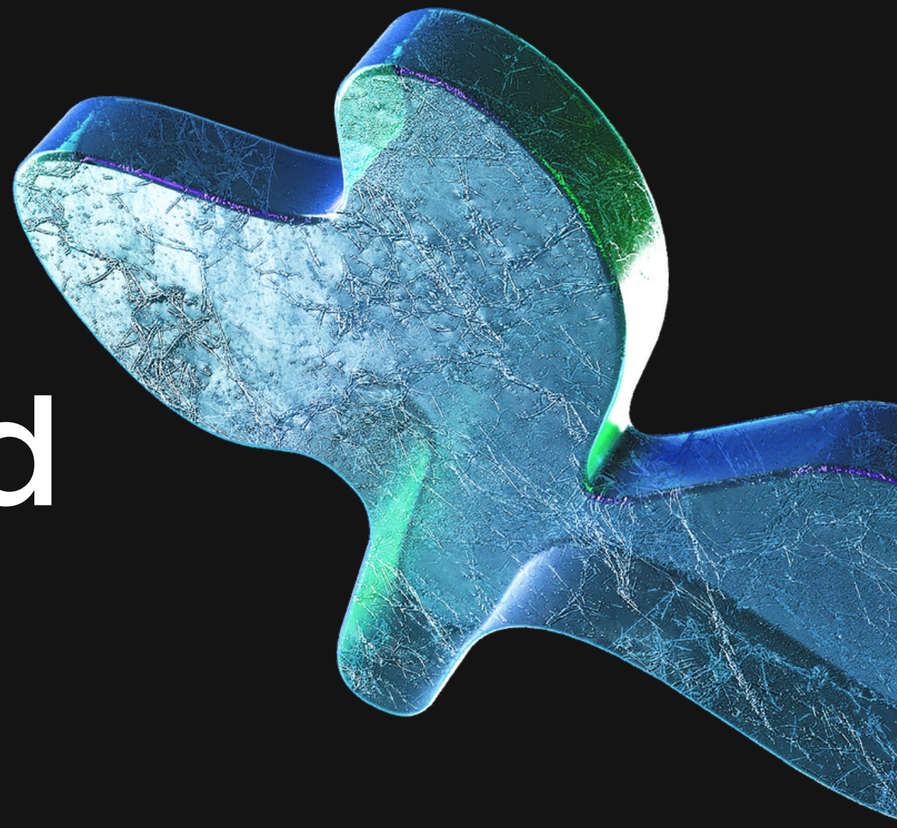
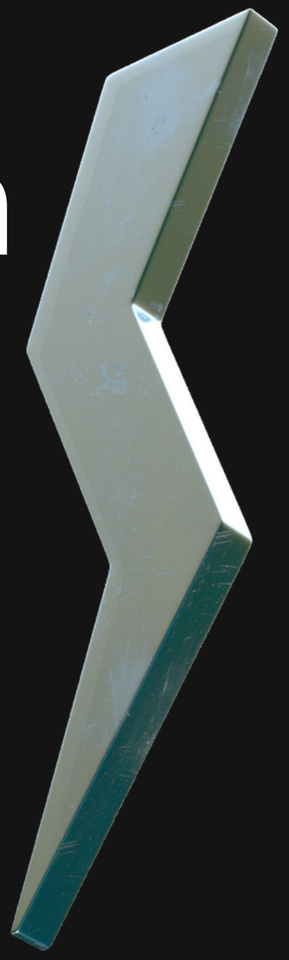
Kecerdasan Buatan F

Finding Shortest Path using Informed Search A*



Najma Ulya Agustina-5025211239

Penjelasan Metode A*



ADALAH SEBUAH ALGORITMA PENCARIAN YANG DIGUNAKAN UNTUK MENEMUKAN JALUR TERPENDEK ATAU SOLUSI OPTIMAL DRI SUATU MASALAH YANG TERDIRI DARI SEJUMLAH NODE ATAU TITIK DALAM SEBUAH GRAF.

Cara Kerja A* dalam Mencari Rute Terpendek dari 2 Kota

Step 1

Membuat graf yang merepresentasikan jaringan jalan antara kota-kota.

Step 2

Menentukan titik awal dan titik akhir di graf. Titik awal adalah kota asal dan titik akhir adalah kota tujuan.

Step 3

Memberikan bobot atau jarak antara setiap pasangan kota yang terhubung dengan jalan. Bobot ini dapat diukur dalam satuan jarak seperti kilometer atau mil.

Step 4

Menghitung nilai heuristik atau perkiraan jarak terpendek dari setiap kota ke titik akhir menggunakan fungsi heuristik seperti jarak Euclidean atau Manhattan.

Step 5

Memasukkan titik awal ke dalam himpunan "open" yang berisi node yang belum dieksplorasi.

Cara Kerja A* dalam Mencari Rute Terpendek dari 2 Kota

Step 6

Algoritma A* mencari node dengan biaya total terkecil, yaitu biaya langkah sejauh ini dari titik awal ditambah dengan perkiraan biaya yang tersisa ke titik akhir, dari himpunan "open".

Step 7

Algoritma A* mengeluarkan node yang memiliki biaya total terkecil dari himpunan "open" dan menambahkannya ke himpunan "closed" yang berisi node yang telah dieksplorasi.

Step 8

Algoritma A* memeriksa setiap kota yang terhubung dengan kota yang dipilih, dan menghitung jarak baru dari titik awal ke kota yang terhubung tersebut.

Step 9

Langkah 6 sampai 8 diulang sampai kota tujuan ditemukan atau tidak ada kota lagi yang tersisa di himpunan "open".

Step 10

Jika kota tujuan ditemukan, algoritma A* membangun jalur terpendek dari titik awal ke kota tujuan dengan melacak kembali kota yang disimpan di himpunan "closed".

Step 11

ika tidak ada jalur yang tersedia dari titik awal ke kota tujuan, algoritma A* mengeluarkan nilai kosong atau menunjukkan bahwa jalur tidak ditemukan.

Implementasi

astar2.py > ...

```python
import heapq

class Graph:
    def __init__(self):
        self.nodes = set()
        self.edges = {}
        self.heuristics = {}

    def add_node(self, node):
        self.nodes.add(node)

    def add_edge(self, from_node, to_node, distance):
        if from_node not in self.edges:
            self.edges[from_node] = {}
        self.edges[from_node][to_node] = distance

    def add_heuristic(self, node, distance):
        self.heuristics[node] = distance
```

Implementasi

```
20      def A_star_algorithm(self, start, goal):
21          open_set = [(0, start)]
22          came_from = {}
23          gscore = {node: float("inf") for node in self.nodes}
24          gscore[start] = 0
25          fscore = {node: float("inf") for node in self.nodes}
26          fscore[start] = self.heuristics[start]
27
28          while open_set:
29              current = he   (variable) current: Any
30              if current =
31                  path = [current]
32                  while current in came_from:
33                      current = came_from[current]
34                      path.append(current)
35                  path.reverse()
36                  return path
37              for neighbor in self.edges[current]:
38                  tentative_gscore = gscore[current] + self.edges[current][neighbor]
39                  if tentative_gscore < gscore[neighbor]:
40                      came_from[neighbor] = current
41                      gscore[neighbor] = tentative_gscore
42                      fscore[neighbor] = tentative_gscore + self.heuristics[neighbor]
43                      heapq.heappush(open_set, (fscore[neighbor], neighbor))
44          return None
```

Implementasi

```
45
46  # Generate graph
47  graph = Graph()
48
49  # Add nodes
50  graph.add_node("A")
51  graph.add_node("B")
52  graph.add_node("C")
53  graph.add_node("D")
54  graph.add_node("E")
55  graph.add_node("F")
56  graph.add_node("G")
57
58  # Add edges with cost
59  graph.add_edge("A", "B", 2)
60  graph.add_edge("A", "C", 3)
61  graph.add_edge("B", "D", 4)
62  graph.add_edge("B", "E", 1)
63  graph.add_edge("C", "F", 5)
64  graph.add_edge("D", "G", 3)
65  graph.add_edge("E", "G", 2)
66  graph.add_edge("F", "G", 4)
```

```
68  # Set heuristics
69  graph.add_heuristic("A", 10)
70  graph.add_heuristic("B", 8)
71  graph.add_heuristic("C", 6)
72  graph.add_heuristic("D", 4)
73  graph.add_heuristic("E", 6)
74  graph.add_heuristic("F", 4)
75  graph.add_heuristic("G", 0)
76
77  # Define start and goal
78  start = "A"
79  goal = "G"
80
81  # Run A* algorithm
82  path = graph.A_star_algorithm(start, goal)
83
84  # Print the result
85  if path:
86      print(" -> ".join(path))
87  else:
88      print("Path not found")
89
```

Terimakasih!

KBF