Kelompok

Start

ANALISIS PERBANDINGAN GREEDY BEST FIRST SEARCH DAN A* SEARCH

Anggota Kelompok :

Helsa Sriprameswari Putri (5025221154)

Yasmin Putri Sujono (5025221273)

Nadya Saraswati Putri (5025221246)

Subject : KONSEP KECERDASAN ARTIFISIAL | INFORMED SEARCH

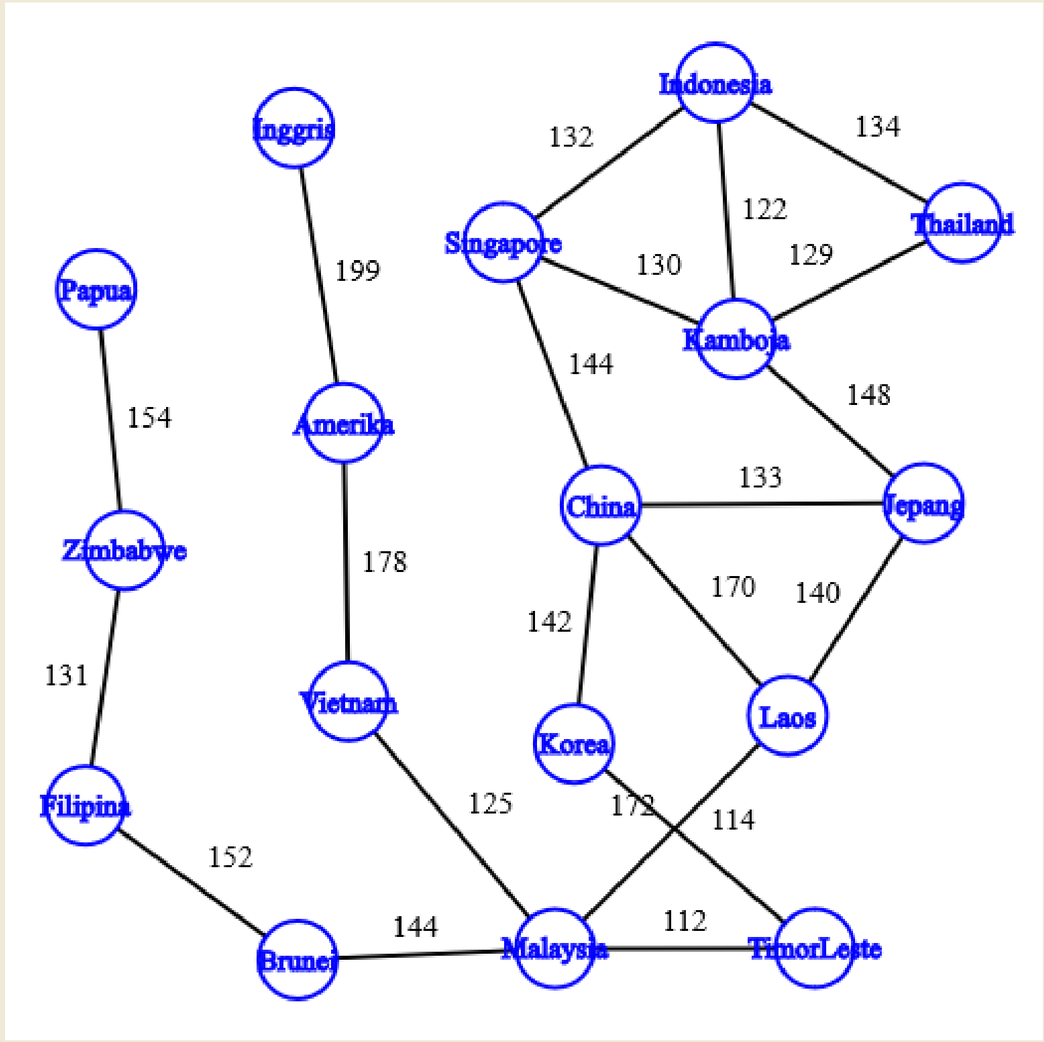
OVERVIEW

CASE DESCRIPTION

Dalam tugas kelompok ini, kami menerapkan dua Algoritma Informed Search, yaitu Algoritma A* dan Algoritma Greedy Best First Search, menggunakan bahasa pemrograman Python. Kami menciptakan kasus khusus kami sendiri yang kami sebut "World Distance Problem," yang merupakan pengembangan dari "Romania Problem." Dalam masalah ini, kami memiliki peta dunia dengan informasi jarak antara negara-negara yang berbeda. Tujuan dari kasus ini adalah untuk menemukan jalur terpendek dari satu negara ke negara lainnya.

Tujuan utama dari tugas ini adalah untuk mengevaluasi kinerja kedua metode pencarian informasi tersebut, yaitu A* dan Greedy Best First Search, serta membandingkan keduanya. Kami ingin menentukan metode yang lebih efisien dalam menyelesaikan masalah pencarian jalur terpendek dalam konteks World Distance Problem.

GRAPH MAP


Indonesia
Inggris
Singapore
Thailand
Papua
Kamboja
Amerika
China
Jepang
Zimbabwe
Vietnam
Laos
Korea
Filipina
Brunei
Malaysia
TimorLeste
132
134
122
199
130
129
144
148
154
133
178
170
140
142
131
125
172
114
152
144
112

TERMINOLOGI

- Informed Search : Pencarian dengan menggunakan knowledge yang spesifik kepada permasalahan yang dihadapi. Disebut juga sebagai Heuristic Search.
- Initial State : Kondisi awal atau titik awal dari masalah yang akan diselesaikan pada awal pencarian.
- Current State : Keadaan/kondisi pada saat pencarian dilakukan.
- Goal State : Kondisi yang ingin dicapai atau titik akhir dari pencarian.
- Heuristic Function / Fungsi Heuristik : Perkiraan atau estimasi tentang seberapa dekat atau seberapa baik suatu keadaan (state) dalam mencapai goal state atau solusi yang diinginkan.

A* SEARCH

Algoritma A* merupakan salah satu algoritma pencarian informasi yang menggunakan fungsi heuristik untuk mempercepat proses pencarian solusi terbaik. Keunggulan utama dari algoritma ini adalah konsepnya yang menggabungkan informasi tentang jarak yang sudah ditempuh dengan estimasi jarak yang masih perlu ditempuh untuk mencapai tujuan.

Cara kerja algoritma A* melibatkan dua nilai penting untuk setiap node dalam setiap langkah pencarian. Pertama, adalah cost aktual yang sudah ditempuh dari state awal ke state yang sedang dipertimbangkan, biasanya dinotasikan sebagai g(n). Kedua, adalah estimasi cost yang masih diperlukan untuk mencapai tujuan, biasanya dinotasikan sebagai h(n).

Algoritma A* selalu memilih node yang memiliki nilai **f(n) = g(n) + h(n)** paling kecil dalam setiap langkah pencarian. Di sini, f(n) adalah fungsi yang menghitung total cost dari state awal hingga mencapai goal state melalui state yang sedang dipertimbangkan.

A* SEARCH

Untuk mengimplementasikan algoritma A*, kita menggunakan struktur data Priority Queue untuk menyimpan node-node yang akan dievaluasi. Selain itu, kita juga memerlukan sebuah Heuristic Function yang dapat memberikan estimasi cost yang diperlukan untuk mencapai tujuan dari setiap state dalam graf. Dengan demikian, algoritma A* membantu dalam mencari solusi optimal dengan efisien.

Sehingga, formulasi untuk algoritma A* dapat dituliskan sebagai berikut,

$$f(n) = g(n) + h(n)$$

Keterangan :

- f(n) adalah hasil dari penjumlahan cost aktual yang telah ditempuh dan estimasi cost yang tersisa untuk mencapai tujuan melalui node tersebut.
- g(n) menggambarkan nilai cost aktual dari state awal hingga mencapai state akhir.
- h(n) adalah nilai estimasi cost yang tersisa untuk mencapai tujuan dari state yang sedang dipertimbangkan, seringkali disebut sebagai fungsi heuristik.

IMPLEMENTASI DENGAN A* SEARCH

LANGKAH-LANGKAH

1. Pembuatan graf yang merepresentasikan hubungan antar lokasi geografis dengan bobot edge yang menunjukkan jarak antara dua lokasi.
2. Fungsi heuristik yaitu memperkirakan jarak dari setiap node ke tujuan akhir.
3. Pencarian A* untuk mencari jalur terpendek dari node sumber ke node tujuan dengan mempertimbangkan biaya dan heuristik.
4. Iterasi tetangga untuk mengunjungi tetangga-tetangga dari node saat ini dan memperbarui nilai biaya sementara dan f(n).
5. Jalur terpendek yang ditemukan dengan mengikuti node-node yang disimpan dalam came_from.
6. Penghitungan total biaya jalur dengan menjumlahkan bobot edge dari setiap langkah dalam jalur.

```
import heapq
import networkx as nx

g = nx.Graph()

# Tambahkan nodes
g.add_nodes_from(["Indonesia", "Malaysia", "Singapore", "Thailand", "Kamboja", "China",
"Jepang", "Laos","Korea", "Timor Leste", "Vietnam", "Amerika", "Inggris", "Brunei",
"FIlipina", "Zimbabwe", "Papua New Guinea"])

# Tambahkan edges
g.add_edge("Indonesia", "Singapore", weight_=132)
g.add_edge("Indonesia", "Kamboja", weight_=122)
g.add_edge("Indonesia", "Thailand", weight_=134)
g.add_edge("Singapore", "Kamboja", weight_=130)
g.add_edge("Singapore", "China", weight_=144)
g.add_edge("Kamboja", "Thailand", weight_=129)
g.add_edge("Kamboja", "Jepang", weight_=148)
g.add_edge("China", "Jepang", weight_=133)
g.add_edge("China", "Laos", weight_=170)
g.add_edge("China", "Korea", weight_=142)
g.add_edge("Jepang", "Laos", weight_=140)
g.add_edge("Korea", "Timor Leste", weight_=114)
g.add_edge("Laos", "Malaysia", weight_=172)
g.add_edge("Timor Leste", "Malaysia", weight_=112)
g.add_edge("Malaysia", "Brunei", weight_=144)
g.add_edge("Malaysia", "Vietnam", weight_=125)
g.add_edge("Brunei", "FIlipina", weight_=152)
g.add_edge("Vietnam", "Amerika", weight_=178)
g.add_edge("Amerika", "Inggris", weight_=199)
g.add_edge("FIlipina", "Zimbabwe", weight_=131)
g.add_edge("Zimbabwe", "Papua New Guinea", weight_=154)
node_sumber = "Indonesia"
```

```
def a_star_search(graph, start, goal):
  # Fungsi heuristik (Manhattan distance) dari setiap node ke goal
  def heuristic(node, goal):
    # Koordinat geografis (latitude, longitude) node
    node_coords = {
      "Indonesia": (0, 0),
      "Malaysia": (1, 1),
      "Singapore": (1, 0),
      "Thailand": (1, 2),
      "Kamboja": (1, 3),
      "China": (2, 1),
      "Jepang": (2, 3),
      "Laos": (2, 2),
      "Korea": (2, 4),
      "Timor Leste": (0, 1),
      "Vietnam": (1, 4),
      "Amerika": (3, 5),
      "Inggris": (3, 6),
      "Brunei": (1, 5),
      "FIlipina": (1, 6),
      "Zimbabwe": (4, 7),
      "Papua New Guinea": (4, 8)
    }

    # Menghitung Manhattan distance antara node dan goal
    node_coord = node_coords.get(node, (0, 0))
    goal_coord = node_coords.get(goal, (0, 0))
    distance = abs(node_coord[0] - goal_coord[0]) + abs(node_coord[1] -
goal_coord[1])
    return distance
```

```
# Inisialisasi nilai g(n) dan f(n) dari setiap node dengan nilai tak terhingga
  g = {node: float('inf') for node in graph.nodes()}
  f = {node: float('inf') for node in graph.nodes()}

  # Inisialisasi heap prioritas dengan elemen awal (node sumber)
  open_list = [(0, start)]

  # Nilai g(n) dari node awal adalah 0
  g[start] = 0

  # Inisialisasi dictionary came_from
  came_from = {}

  while open_list:
    # Ambil node dengan f(n) terkecil dari heap prioritas
    f_current, current = heapq.heappop(open_list)

    # Jika sudah mencapai goal, maka selesai
    if current == goal:
      path = []
      while current != start:
        path.insert(0, current)
        current = came_from[current]
      path.insert(0, start)
      return path
```

```
  # Jika belum mencapai goal, Iterasi melalui tetangga-tetangga dari node saat ini
    for neighbor in graph[current]:
      # Hitung nilai g(n) sementara untuk tetangga ini
      tentative_g = g[current] + graph[current][neighbor]['weight_']

      # Jika nilai g(n) baru lebih baik daripada yang sebelumnya
      if tentative_g < g[neighbor]:
        # Perbarui nilai g(n) dan f(n) tetangga
        g[neighbor] = tentative_g
        f[neighbor] = tentative_g + heuristic(neighbor, goal)

        # Simpan node yang sebelumnya terbaik untuk tetangga ini
        came_from[neighbor] = current

        # Tambahkan tetangga ke heap prioritas
        heapq.heappush(open_list, (f[neighbor], neighbor))

  # Jika tidak ada jalur yang ditemukan
  return None

a_star_path = a_star_search(g, "Indonesia", "Malaysia")

#Hitung total cost
total_cost = 0
for i in range(len(a_star_path) - 1):
  node1 = a_star_path[i]
  node2 = a_star_path[i + 1]
  edge_weight = g[node1][node2]['weight_']
  total_cost += edge_weight

print("A* Path:", a_star_path)
print("Total Cost:", total_cost)
```

GBFS SEARCH

Greedy Best First Search adalah salah satu algoritma Pencarian Terinformasi yang memilih jalur pencarian yang paling menjanjikan atau terbaik berdasarkan estimasi jarak ke tujuan saat itu. Ini merupakan gabungan dari algoritma Depth-First Search dan Breadth-First Search yang membuatnya lebih efisien karena dapat memilih jalur yang paling menjanjikan berdasarkan situasi saat ini.

Algoritma Greedy BFS menggunakan Fungsi Heuristik untuk menghitung jarak perkiraan dari posisi saat ini ke tujuan dan selalu memilih simpul yang memiliki estimasi jarak terdekat pada setiap langkah pencarian hingga mencapai tujuan. Algoritma ini sering diimplementasikan dengan menggunakan struktur data Antrian Prioritas. Rumus yang digunakan untuk menghitung biaya terbaik dari simpul yang akan dijelajahi adalah sebagai berikut:

$$f(n) = h(n)$$

Keterangan:

- f(n) adalah biaya total yang diperkirakan dari simpul saat ini ke tujuan (goal state).
- h(n) adalah fungsi heuristik yang memberikan perkiraan jarak ke tujuan.

IMPLEMENTASI DENGAN GREEDY BEST FIRST SEARCH

LANGKAH-LANGKAH

1. Impor modul networkx sebagai nx. Kemudian buat objek graf, g, dengan menggunakan nx.Graph().
2. Tambahkan node-node ke dalam graf menggunakan metode add_nodes_from, yang diwakili sebagai string.
3. Tambahkan edge-edge ke dalam graf menggunakan metode add_edge, yang memiliki atribut weight untuk mengindikasikan jarak antara dua node.
4. Tentukan node sumber dan definisikan fungsi greedy_best_first_search untuk menjalankan algoritma Greedy Best First Search dengan menggunakan heuristik yang hanya mempertimbangkan node-node tetangga yang belum dieksplorasi tanpa memperhitungkan jarak sebenarnya.
5. Inisialisasi daftar terbuka open_list dengan node sumber dan kamus came_from yang akan digunakan untuk melacak jalur.

IMPLEMENTASI DENGAN GREEDY BEST FIRST SEARCH

LANGKAH-LANGKAH

6. Dalam loop while, iterasi selama open_list tidak kosong. Di dalam loop, ambil node yang memiliki nilai heuristik terendah dari open_list.
7. Jika node yang diambil adalah node tujuan, maka bangun jalur kembali dari node tujuan ke node sumber menggunakan kamus came_from.
8. Jika bukan node tujuan, iterasi melalui node-node tetangga yang belum dieksplorasi. Jika node tetangga belum ada dalam kamus came_from, tambahkan ke came_from dan open_list. open_list diurutkan berdasarkan heuristik sehingga node dengan nilai heuristik terendah akan diambil lebih dahulu.
9. Cetak jalur yang ditemukan oleh Greedy Best First Search dan total biaya jalur tersebut.

```
import heapq
import networkx as nx

g = nx.Graph()

# Tambahkan nodes
g.add_nodes_from(["Indonesia", "Malaysia", "Singapore", "Thailand", "Kamboja", "China",
"Jepang", "Laos","Korea", "Timor Leste", "Vietnam", "Amerika", "Inggris", "Brunei",
"Fllipina", "Zimbabwe", "Papua New Guinea"])

# Tambahkan edges
g.add_edge("Indonesia", "Singapore", weight_=132)
g.add_edge("Indonesia", "Kamboja", weight_=122)
g.add_edge("Indonesia", "Thailand", weight_=134)
g.add_edge("Singapore", "Kamboja", weight_=130)
g.add_edge("Singapore", "China", weight_=144)
g.add_edge("Kamboja", "Thailand", weight_=129)
g.add_edge("Kamboja", "Jepang", weight_=148)
g.add_edge("China", "Jepang", weight_=133)
g.add_edge("China", "Laos", weight_=170)
g.add_edge("China", "Korea", weight_=142)
g.add_edge("Jepang", "Laos", weight_=140)
g.add_edge("Korea", "Timor Leste", weight_=114)
g.add_edge("Laos", "Malaysia", weight_=172)
g.add_edge("Timor Leste", "Malaysia", weight_=112)
g.add_edge("Malaysia", "Brunei", weight_=144)
g.add_edge("Malaysia", "Vietnam", weight_=125)
g.add_edge("Brunei", "Fllipina", weight_=152)
g.add_edge("Vietnam", "Amerika", weight_=178)
g.add_edge("Amerika", "Inggris", weight_=199)
g.add_edge("Fllipina", "Zimbabwe", weight_=131)
g.add_edge("Zimbabwe", "Papua New Guinea", weight_=154)
node_sumber = "Indonesia"
```

```
def heuristic(node, goal):
  # Koordinat geografis (latitude, longitude) node
  node_coords = {
    "Indonesia": (0, 0),
    "Malaysia": (1, 1),
    "Singapore": (1, 0),
    "Thailand": (1, 2),
    "Kamboja": (1, 3),
    "China": (2, 1),
    "Jepang": (2, 3),
    "Laos": (2, 2),
    "Korea": (2, 4),
    "Timor Leste": (0, 1),
    "Vietnam": (1, 4),
    "Amerika": (3, 5),
    "Inggris": (3, 6),
    "Brunei": (1, 5),
    "Fllipina": (1, 6),
    "Zimbabwe": (4, 7),
    "Papua New Guinea": (4, 8)
  }

  # Menghitung Manhattan distance antara node dan goal
  node_coord = node_coords.get(node, (0, 0))
  goal_coord = node_coords.get(goal, (0, 0))
  distance = abs(node_coord[0] - goal_coord[0]) + abs(node_coord[1] - goal_coord[1])

  return distance
```

```
def greedy_best_first_search(graph, start, goal, heuristic):
  open_list = [(heuristic(start, goal), start)]

  # Inisialisasi dictionary came_from
  came_from = {}

  while open_list:
    _, current = open_list.pop(0)

# Jika sudah mencapai goal, maka selesai



    if current == goal:
      path = []
      while current != start:
        path.insert(0, current)
        current = came_from[current]
      path.insert(0, start)
      return path

#Jika belum mengapai goal, Iterasi melalui tetangga-tetangga dari node saat ini

    for neighbor in graph[current]:
      if neighbor not in came_from:
        came_from[neighbor] = current
        open_list.append((heuristic(neighbor, goal), neighbor))
        open_list.sort(key=lambda x: x[0])

 # Jika tidak ada jalur yang ditemukan
  return None
```

```
greedy_best_first_path = greedy_best_first_search(g, "Indonesia", "Malaysia",
heuristic)

# Hitung Total Cost
total_cost = 0
for node1, node2 in zip(greedy_best_first_path[:-1], greedy_best_first_path[1:]):
  edge_weight = g[node1][node2]['weight_']
  total_cost += edge_weight

print("Greedy Best First Path:", greedy_best_first_path)
print("Total Cost:", total_cost)
```

PERBEDAAM HASIL KEDUA ALGORITMA

```
A* Path: ['Indonesia', 'Kamboja', 'Jepang', 'Laos', 'Malaysia']
Total Cost: 582
```

```
Greedy Best First Path: ['Indonesia', 'Singapore', 'China', 'Laos', 'Malaysia']
Total Cost: 618
```

ANALISIS PERBANDINGAN KEDUA ALGORITMA

A* memiliki keunggulan utama dalam hal optimalitas, dapat menemukan **solusi optimal** dengan biaya terendah, sedangkan GBFS memiliki pencarian tak terbatas sehingga menghasilkan solusi yang suboptimal. Namun, GBFS memiliki potensi untuk lebih **cepat** dalam beberapa kasus. Karena algoritma ini hanya mempertimbangkan heuristik saat memilih node berikutnya, itu bisa membuat keputusan lebih cepat daripada A*, yang mempertimbangkan biaya sejauh ini dan heuristik. Namun, kecepatan GBFS bergantung pada heuristik yang digunakan.

PERBEDAAN KEDUA ALGORITMA

1. FUNGSI HEURISTIK

GBFS menggunakan fungsi heuristik (biasanya jarak Manhattan, Euclidean, atau heuristik lainnya) untuk mengestimasi biaya dari titik saat ini ke tujuan. Algoritma ini hanya mempertimbangkan heuristik untuk membuat keputusan tentang langkah berikutnya, tanpa mempertimbangkan biaya sejauh ini (g).

A* juga menggunakan fungsi heuristik, tetapi algoritma ini mempertimbangkan biaya sejauh ini (g) dan biaya perkiraan ke tujuan (h) untuk membuat keputusan tentang langkah berikutnya. Fungsi evaluasi A* adalah f(n) = g(n) + h(n), di mana g(n) adalah biaya sejauh ini dan h(n) adalah estimasi biaya ke tujuan.

2. OPTIMALITAS

GBFS tidak menjamin optimalitas. Artinya, GBFS dapat menemukan solusi yang lebih cepat tetapi tidak selalu optimal. Algoritma ini mungkin terjebak dalam siklus atau tidak dapat menemukan solusi optimal jika heuristiknya tidak baik.

A* dijamin optimal jika fungsi heuristiknya admissible (h(n) tidak melebihi biaya sebenarnya ke tujuan). Ini berarti bahwa A* akan selalu menemukan solusi dengan biaya terendah jika solusi ada.

PERBEDAAN KEDUA ALGORITMA

3. KOMPLEKSITAS WAKTU

GBFS biasanya memiliki kompleksitas waktu yang lebih rendah daripada A* karena hanya mempertimbangkan heuristik saat mengambil keputusan.

A* memiliki kompleksitas waktu yang lebih tinggi karena mempertimbangkan biaya sejauh ini dan heuristik pada setiap langkah.

4. PEMILIHAN NODE BERIKUTNYA

GBFS hanya mempertimbangkan heuristik untuk memilih node berikutnya. Itu berarti bahwa GBFS bisa saja memilih jalur yang jauh dari tujuan jika heuristiknya tidak akurat.

A* mempertimbangkan biaya sejauh ini dan heuristik untuk memilih node berikutnya, sehingga lebih mungkin untuk memilih jalur yang lebih baik ke tujuan.

CONCLUSION

Algoritma A* dan Greedy Search menghasilkan cost yang berbeda. A* cenderung memiliki cost lebih rendah karena menggabungkan informasi biaya aktual dengan heuristik saat memilih node berikutnya, sehingga dapat mengeksplorasi jalur yang lebih efisien. Meskipun begitu, dalam beberapa situasi, Greedy Search bisa memberikan solusi yang lebih baik, terutama jika tujuan akhir dekat dengan posisi awal dan heuristik yang digunakan sudah akurat.

Selain itu, jumlah node yang dieksplorasi oleh A* dan Greedy Search dapat bervariasi tergantung pada struktur grafik dan heuristik yang digunakan. Dalam beberapa kasus, A* membatasi jumlah node yang dieksplorasi, tetapi dalam kasus lain, A* Search mungkin harus memeriksa lebih banyak node dibandingkan dengan Greedy Search.

Untuk memilih algoritma pencarian yang tepat, penting untuk mempertimbangkan karakteristik khusus dari masalah yang dihadapi dan melakukan evaluasi kinerja dengan menguji berbagai kasus yang mencakup berbagai kondisi masalah yang mungkin terjadi.

REFERENCES

- https://www.geeksforgeeks.org/a-search-algorithm/
- https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/
- https://www.geeksforgeeks.org/difference-between-informed-and-uninformed-search-in-ai
- https://socs.binus.ac.id/2013/04/23/uninformed-search-dan-informed-search
- Hidayat, Lukman Rahmat., Pribadi, Muhammad Fachrul Risqi. 2019. Perbandingan Algoritma A* Dengan Algoritma Greedy Pada Penentuan Routing Jaringan. Vol.12 No.2.

THANK YOU