

LAPORAN
IMPLEMENTASI ALGORITMA Pencarian dalam penentuan rute
TERPENDEK
UTS KECERDASAN BUATAN D SEMESTER GENAP



Disusun Oleh:

Krisna Bimantoro

(202210370311254)

FAKULTAS TEKNIK
PROGRAM STUDI INFORMATIKA
UNIVERSITAS MUHAMMADIYAH MALANG

2024

Latar Belakang:

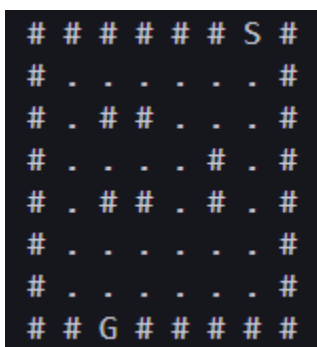
Andi berkenalan dengan Tika melalui aplikasi Tinder. Setelah 5 hari chattingan, mereka berencana untuk meet up yang pertama kali di sebuah taman di dalam perumahannya Tika. Mereka setuju bertemu dan “janjian” pada pukul 15.30. Andi tiba di depan perumahan Tika pukul 15.25. Mengetahui bahwa Tika benci dengan laki-laki yang tidak tepat waktu, Andi pun panik untuk segera sampai di taman. Saat ini dia hanya bermodalkan peta lokasi dia sekarang dan lokasi taman berada. Bantulah Andi menemukan rute terpendek agar dapat segera sampai di lokasi taman tersebut untuk menghindari kemarahan dari si Tika dengan mengimplementasikan algoritma Searching.

Deskripsi Proyek:

Anda diminta untuk membuat sebuah program yang dapat menentukan rute terpendek antara dua titik di dalam sebuah peta menggunakan algoritma pencarian. Program ini akan menemukan rute terpendek berdasarkan jarak yang diukur menggunakan koordinat x dan y.

Spesifikasi Aplikasi:

1. Buatlah peta yang berisi titik-titik yang dapat dilalui dan titik-titik yang tidak dapat dilalui.



Keterangan Peta:

S = Start (Titik Awal)

G = Goal (Titik Tujuan)


= Dinding/Rumah (Tidak Dapat Dilalui)

. = Jalan (Dapat Dilalui)

2. Implementasikan minimal 2 algoritma pencarian untuk menentukan rute terpendek antara dua titik pada peta, kemudian bandingkan dan analisa hasilnya

- **Greedy best first search (GBFS)**

- Mendefinisikan Peta



```
1  peta = [  
2      ['#', '#', '#', '#', '#', '#', 'S', '#'],  
3      ['#', '.', '.', '.', '.', '.', '.', '#'],  
4      ['#', '.', '#', '#', '.', '.', '.', '#'],  
5      ['#', '.', '.', '.', '.', '#', '.', '#'],  
6      ['#', '.', '#', '#', '.', '#', '.', '#'],  
7      ['#', '.', '.', '.', '.', '.', '.', '#'],  
8      ['#', '.', '.', '.', '.', '.', '.', '#'],  
9      ['#', '#', 'G', '#', '#', '#', '#', '#']  
10 ]
```

Pada peta ini terdapat S=Titik awal, G=Titik Akhir, #=tidak dapat dilalui, dan .=dapat dilalui. Dalam bentuk array.

- Function find(peta):



```
1  def find(peta):  
2      start = goal = None  
3      for i in range(len(peta)):  
4          for j in range(len(peta[i])):  
5              if peta[i][j] == 'S':  
6                  start = (i, j)  
7              elif peta[i][j] == 'G':  
8                  goal = (i, j)  
9      return start, goal  
10
```

Function yang digunakan untuk menentukan posisi titik awal dan titik akhir pada peta, menggunakan dua looping dan variabel i dan j yang digunakan untuk menempatkan posisi x dan y (x,y)

- Function heuristic(a, b):

```
1 def heuristic(a, b):
2     return abs(a[0] - b[0]) + abs(a[1] - b[1])
3
```

Fungsi yang digunakan untuk menentukan jarak (Manhattan Distance) dengan rumus $h(n) = |x_1 - x_2| + |y_1 - y_2|$. Dengan menggunakan parameter a dan b, a sebagai x dan b sebagai y.

- Function gbfs(peta)

```
1 def gbfs(peta):
2     start, goal = find(peta)
3     if not start or not goal:
4         return None
5
6     rows, cols = len(peta), len(peta[0])
7     visited = set()
8     pq = [(0, start)] # priority queue dengan elemen (heuristic, position)
9     came_from = {start: None}
10
11     while pq:
12         _, current = heapq.heappop(pq)
13         if current in visited:
14             continue
15         visited.add(current)
16
17         if current == goal:
18             break
19
20         # Tetangga (atas, bawah, kiri, kanan)
21         for direction in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
22             neighbor = (current[0] + direction[0], current[1] + direction[1])
23             if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols and neighbor not in visited:
24                 if peta[neighbor[0]][neighbor[1]] != '#':
25                     heapq.heappush(pq, (heuristic(neighbor, goal), neighbor))
26                     if neighbor not in came_from:
27                         came_from[neighbor] = current
28
29     # Rekonstruksi jalur dari goal ke start
30     path = []
31     if goal in came_from:
32         current = goal
33         while current:
34             path.append(current)
35             current = came_from[current]
36         path.reverse()
37
38     return path
```

Fungsi yang digunakan untuk membuat algoritma gbfs dengan memasukkan parameter **peta** dengan dimulai dari menentukan titik awal dan titik akhir dan pengecekan jalur yang dilalui kemudian menggunakan fungsi heuristik untuk memberikan nilai pada setiap node yang dapat dilalui.

- Function visualisasi (peta,path)

```
1 def visualisasi(peta, path):
2     peta_visual = np.zeros((len(peta), len(peta[0])))
3     for i in range(len(peta)):
4         for j in range(len(peta[i])):
5             if peta[i][j] == '#':
6                 peta_visual[i][j] = 1 # Dinding
7
8     for (x, y) in path:
9         if peta[x][y] not in ['S', 'G']:
10             peta_visual[x][y] = 0.5 # Jalur
11
12     start, goal = find(peta)
13     peta_visual[start[0]][start[1]] = 0.3 # Titik awal
14     peta_visual[goal[0]][goal[1]] = 0.7 # Titik tujuan
15
16     plt.imshow(peta_visual, cmap='Blues')
17     plt.show()
18
```

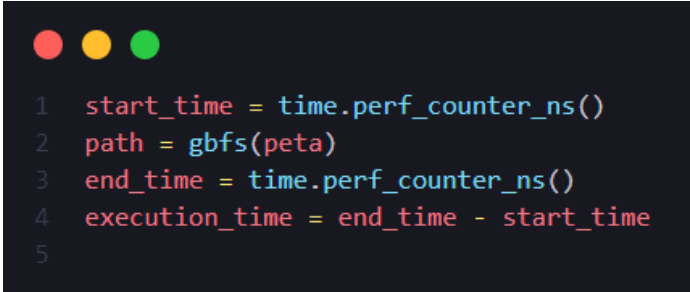
Fungsi yang digunakan untuk membuat GUI peta yang dilalui oleh algoritma GBFS dari titik awal ke titik akhir, dan memberikan dinding dan jalan yang dapat dilalui.

- Function rute_terpendek (peta,path)

```
1 def rute_terpendek(peta, path):
2     peta_copy = [row[:] for row in peta] # Membuat salinan dari peta asli
3     for (x, y) in path:
4         if peta_copy[x][y] not in ['S', 'G']:
5             peta_copy[x][y] = '-' # Tandai rute terpendek dengan '-'
6     for row in peta_copy:
7         print(' '.join(row))
8
```

Fungsi yang digunakan untuk membuat output peta jalan terpendek dengan menggunakan -=.

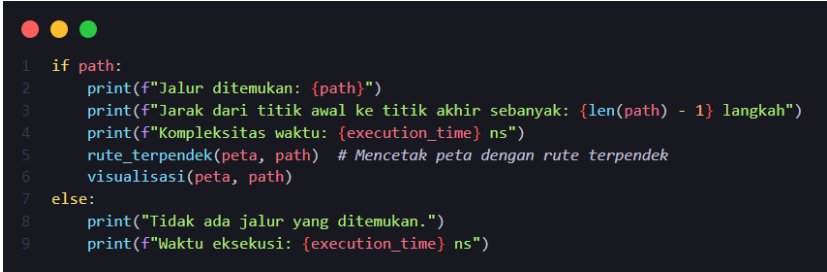
- Kompleksitas Waktu



```
1 start_time = time.perf_counter_ns()
2 path = gbfs(peta)
3 end_time = time.perf_counter_ns()
4 execution_time = end_time - start_time
5
```

Untuk memberikan kompleksitas waktu dalam satuan nanosecond yang digunakan untuk melakukan perbandingan kecepatan algoritma dalam mencari titik akhir

- Menjalankan dan menampilkan output



```
1 if path:
2     print(f"Jalur ditemukan: {path}")
3     print(f"Jarak dari titik awal ke titik akhir sebanyak: {len(path) - 1} langkah")
4     print(f"Kompleksitas waktu: {execution_time} ns")
5     rute_terpendek(peta, path) # Mencetak peta dengan rute terpendek
6     visualisasi(peta, path)
7 else:
8     print("Tidak ada jalur yang ditemukan.")
9     print(f"Waktu eksekusi: {execution_time} ns")
```

Menampilkan apakah jalur ditemukan apa tidak, dan memberikan output setiap langkah titik awal ke titik akhir dalam (x,y). Memberikan langkah jarak dari titik awal ke titik akhir. Menampilkan kompleksitas waktu dalam ns, mencetak peta dengan rute terpendek. Kemudian menampilkan GUI visualisasi dari penggunaan algoritma GBFS dalam menentukan titik awal ke titik akhir.

- A*

- Mendefinisikan Peta

```

1  peta = [
2      ['#', '#', '#', '#', '#', '#', 'S', '#'],
3      ['#', '.', '.', '.', '.', '.', '.', '#'],
4      ['#', '.', '#', '#', '.', '.', '.', '#'],
5      ['#', '.', '.', '.', '.', '#', '.', '#'],
6      ['#', '.', '#', '#', '.', '#', '.', '#'],
7      ['#', '.', '.', '.', '.', '.', '.', '#'],
8      ['#', '.', '.', '.', '.', '.', '.', '#'],
9      ['#', '#', 'G', '#', '#', '#', '#', '#']
10 ]

```

Pada peta ini terdapat S=Titik awal, G=Titik Akhir, #=tidak dapat dilalui, dan .=dapat dilalui. Dalam bentuk array.

- Function find(peta):

```

1  def find(peta):
2      start = goal = None
3      for i in range(len(peta)):
4          for j in range(len(peta[i])):
5              if peta[i][j] == 'S':
6                  start = (i, j)
7              elif peta[i][j] == 'G':
8                  goal = (i, j)
9      return start, goal
10

```

Function yang digunakan untuk menentukan posisi titik awal dan titik akhir pada peta, menggunakan dua looping dan variabel i dan j yang digunakan untuk menempatkan posisi x dan y (x.y)

- Function heuristic(a, b):

```

1  def heuristic(a, b):
2      return abs(a[0] - b[0]) + abs(a[1] - b[1])
3

```

Fungsi yang digunakan untuk menentukan jarak (Manhattan Distance) dengan rumus $h(n) = |x_1 - x_2| + |y_1 - y_2|$. Dengan menggunakan parameter a dan b, a sebagai x dan b sebagai y.

- Function a_star(peta)

```
1 def a_star(peta):
2     start, goal = find(peta)
3     if not start or not goal:
4         return None
5
6     rows, cols = len(peta), len(peta[0])
7     open_set = [(0, start)] # priority queue dengan elemen (f_score, position)
8     came_from = {}
9     g_score = {start: 0}
10    f_score = {start: heuristic(start, goal)}
11
12    while open_set:
13        _, current = heapq.heappop(open_set)
14
15        if current == goal:
16            path = []
17            while current in came_from:
18                path.append(current)
19                current = came_from[current]
20            path.append(start)
21            path.reverse()
22            return path
23
24        neighbors = [(current[0] + dx, current[1] + dy)
25                     for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]]
26        for neighbor in neighbors:
27            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols:
28                if peta[neighbor[0]][neighbor[1]] == '#':
29                    continue
30                tentative_g_score = g_score[current] + 1
31                if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
32                    came_from[neighbor] = current
33                    g_score[neighbor] = tentative_g_score
34                    f_score[neighbor] = tentative_g_score + \
35                        heuristic(neighbor, goal)
36                    heapq.heappush(open_set, (f_score[neighbor], neighbor))
37
38    return None
```

Fungsi yang digunakan untuk membuat algoritma A* dengan memasukkan parameter **peta** dengan dimulai dari menentukan titik awal dan titik akhir dan pengecekan jalur yang dilalui menggunakan fungsi heuristik untuk memberikan nilai pada setiap node yang dapat dilalui kemudian menambahkan setiap node dengan urutan dari titik akhir ke titik awal untuk menentukan jalur terpendek menggunakan A*

- Function visualisasi (peta,path)


```

1 def viisualisasi(peta, path):
2     peta_visual = np.zeros((len(peta), len(peta[0])))
3     for i in range(len(peta)):
4         for j in range(len(peta[i])):
5             if peta[i][j] == '#':
6                 peta_visual[i][j] = 1 # Dinding
7
8     for (x, y) in path:
9         if peta[x][y] not in ['S', 'G']:
10             peta_visual[x][y] = 0.5 # Jalur
11
12     start, goal = find(peta)
13     peta_visual[start[0]][start[1]] = 0.3 # Titik awal
14     peta_visual[goal[0]][goal[1]] = 0.7 # Titik tujuan
15
16     plt.imshow(peta_visual, cmap='Blues')
17     plt.show()
18

```

Fungsi yang digunakan untuk membuat GUI peta yang dilalui oleh algoritma GBFS dari titik awal ke titik akhir, dan memberikan dinding dan jalan yang dapat dilalui.

- Function rute_terpendek (peta,path)

```

1 def rute_terpendek(peta, path):
2     peta_copy = [row[:] for row in peta] # Membuat salinan dari peta asli
3     for (x, y) in path:
4         if peta_copy[x][y] not in ['S', 'G']:
5             peta_copy[x][y] = '-' # Tandai rute terpendek dengan '-'
6     for row in peta_copy:
7         print(' '.join(row))

```

Fungsi yang digunakan untuk membuat output peta jalan terpendek dengan menggunakan `-`.

- Kompleksitas Waktu

```

1 start_time = time.perf_counter_ns()
2 path = a_star(peta)
3 end_time = time.perf_counter_ns()
4 execution_time = end_time - start_time

```

Untuk memberikan kompleksitas waktu dalam satuan nanosecond yang digunakan untuk melakukan perbandingan kecepatan algoritma dalam mencari titik akhir

- Menjalankan dan menampilkan output

```
1 if path:
2     print(f"Jalur ditemukan: {path}")
3     print(f"Jarak dari titik awal ke titik akhir sebanyak: {len(path) - 1} langkah")
4     print(f"Waktu eksekusi: {execution_time} ns")
5     rute_terpendek(peta, path) # Mencetak peta dengan rute terpendek
6     visualisasi(peta, path)
7 else:
8     print("Tidak ada jalur yang ditemukan.")
9     print(f"Waktu eksekusi: {execution_time} ns")
10
```

Menampilkan apakah jalur ditemukan apa tidak, dan memberikan output setiap langkah titik awal ke titik akhir dalam (x,y). Memberikan langkah jarak dari titik awal ke titik akhir. Menampilkan kompleksitas waktu dalam ns, mencetak peta dengan rute terpendek. Kemudian menampilkan GUI visualisasi dari penggunaan algoritma GBFS dalam menentukan titik awal ke titik akhir.

- **Analisa Perbandingan kedua algoritma**

Pada tugas kali ini saya membandingkan 2 algoritma heuristic yaitu algoritma melakukan pencarian yang menggunakan pengetahuan spesifik masalah untuk menemukan solusi dengan lebih efisien. Saya memilih kedua algoritma ini dikarenakan ingin membandingkan kedua algoritma yaitu **Greedy best first search (GBFS)** dan **A*** apakah memiliki kompleksitas waktu yang sama dan mana algoritma terbaik yang lebih tepat digunakan untuk melakukan pencarian rute terpendek dari titik awal ke titik akhir.

- A. Greedy best first search (GBFS)**

Algoritma pencarian yang memperluas node yang paling dekat dengan tujuan, seperti yang diestimasi oleh fungsi heuristik $h(n)$. Memiliki keunggulan pada kecepatan searching dan mudah untuk diimplementasikan namun memiliki kekurangan yang cukup fatal dikarenakan tidak optimal hanya menjamin solusi yang ditemukan adalah solusi yang memiliki biaya

terendah dan memungkinkan algoritma ini tidak menemukan solusi jika ruang pencarian besar dan kompleks

B. A*

Algoritma pencarian yang memperluas node dengan nilai terendah dari $g(n) + h(n)$

$g(n)$ = biaya untuk mencapai node

$h(n)$ = estimasi biaya ke tujuan

Memiliki keunggulan dalam menemukan solusi yang optimal atau jalur terpendek jika dibandingkan dengan GBFS. Serta penggunaannya yang fleksibel dan menjamin mendapatkan penyelesaian atau menemukan solusi tetapi terdapat kekurangan yaitu kecepataannya yang lebih rendah dikarenakan lebih lama serta penggunaan memori yang lebih tinggi.

C. Analisa dalam penerapan algoritma GBFS dan A* dalam peta yang digunakan untuk menemukan rute

a. Kompleksitas waktu

Pada pengujian kompleksitas waktu kali ini dilakukan sebanyak 5 kali untuk pengtesan data dan didapatkan data berikut:

Kompleksitas Waktu nanosecond(ns)		
	Greedy best first search (GBFS)	A*
	65900	74900
	42300	59700
	37200	63700
	55100	69500
	58800	78400
Rata-rata(avg)	51860	69240

Berdasarkan data yang ditunjukkan algoritma A* memiliki kompleksitas waktu pencarian lebih lama dibandingkan algoritma GBFS. Hal ini dikarenakan A* tidak hanya mempertimbangkan estimasi biaya ke tujuan (heuristic) namun juga menambahkan

biaya untuk mencapai node. Sehingga mengakibatkan waktu searching lebih lama.

b. Jarak tempuh

- **Greedy best first search (GBFS)**

```
Jarak dari titik awal ke titik akhir sebanyak: 13 langkah
Kompleksitas waktu: 41200 ns
# # # # # S #
# - - - - - #
# - # # . . . #
# - . . . # . #
# - # # . # . #
# - - . . . . #
# . - . . . . #
# # G # # # # #
```

- **A***

```
Jarak dari titik awal ke titik akhir sebanyak: 11 langkah
Waktu eksekusi: 65600 ns
# # # # # S #
# . . . - - - #
# . # # - . . #
# . . . - # . #
# . # # - # . #
# . - - - . . #
# . - . . . . #
# # G # # # # #
```

A* memiliki jarak tempuh yang lebih singkat dibandingkan GBFS dengan hanya 11 langkah dibandingkan GBFS yang 13 langkah, hal ini dikarenakan A* mempertimbangkan biaya untuk mencapai setiap node yang dilaluinya ke titik awal ke titik akhir dengan menggunakan heuristic + biaya untuk mencapai node. Kemudian menghasilkan searching yang optimal (jalur terpendek) dibandingkan GBFS.

c. Memori

Dikarenakan A* memiliki kompleksitas yang lebih besar dibandingkan GBFS hal ini mengakibatkan penggunaan memori pada algoritma A* lebih besar karena mempertimbangkan biaya perjalanan dan heuristic. Apabila semakin kompleks atau semakin

besar ruang pencariannya hal ini juga mengakibatkan penggunaan memori yang lebih besar juga.

d. Ketepatan solusi

Tidak selamanya algoritma GBFS menghasilkan pencarian yang tidak tepat ada kalanya algoritma GBFS dan A* memiliki hasil rute yang sama dengan kecepatan pencarian GBFS yang jauh lebih cepat tetapi hal ini hanya terjadi pada pencarian yang dalam kompleks kecil atau tidak memiliki penghalang dalam pencariannya.

3. Tampilkan rute terpendek beserta jaraknya.

- Greedy best first search (GBFS)

```
Jarak dari titik awal ke titik akhir sebanyak: 13 langkah
Kompleksitas waktu: 41200 ns
# # # # # S #
# - - - - - #
# - # # . . . #
# - . . . # . #
# - # # . # . #
# - - . . . . #
# . - . . . . #
# # G # # # # #
```

Keterangan Peta:

S = Start (Titik Awal)

G = Goal (Titik Tujuan)

= Dinding/Rumah (Tidak Dapat Dilalui)

. = Jalan (Dapat Dilalui)

- = Rute Terpendek

- A*

```

Jarak dari titik awal ke titik akhir sebanyak: 11 langkah
Waktu eksekusi: 65600 ns
# # # # # S #
# . . . - - - #
# . # # - . . #
# . . . - # . #
# . # # - # . #
# . - - - . . #
# . - . . . . #
# # G # # # # #

```

Keterangan Peta:

S = Start (Titik Awal)

G = Goal (Titik Tujuan)

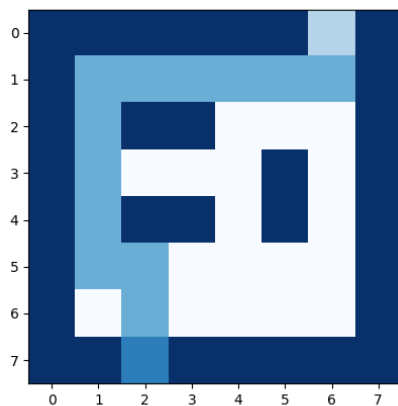
= Dinding/Rumah (Tidak Dapat Dilalui)

. = Jalan (Dapat Dilalui)

- = Rute Terpendek

4. Bonus sebagai nilai tambahan : Buatlah GUI visualisasi untuk menampilkan hasilnya

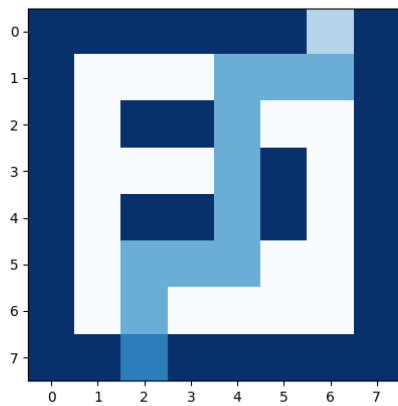
- Greedy best first search (GBFS)



Titik awal berada di koordinat (0,6) dan titik akhir berada di koordinat (7,2)

Warna biru muda mendefinisikan titik awal, dan warna biru tua muda mendefinisikan titik akhir. Biru tua sebagai dinding (tidak dapat dilalui) dan putih sebagai jalan yang dapat dilalui.

- A*



Titik awal berada di koordinat (0,6) dan titik akhir berada di koordinat (7,2)

Warna biru muda mendefinisikan titik awal, dan warna biru tua muda mendefinisikan titik akhir. Biru tua sebagai dinding (tidak dapat dilalui) dan putih sebagai jalan yang dapat dilalui.