

FINDING SHORTEST PATH

Tugas Individu 1 - Kecerdasan Buatan F



Hi!

THIS IS ME



Syomeron Ansell W

5025211250

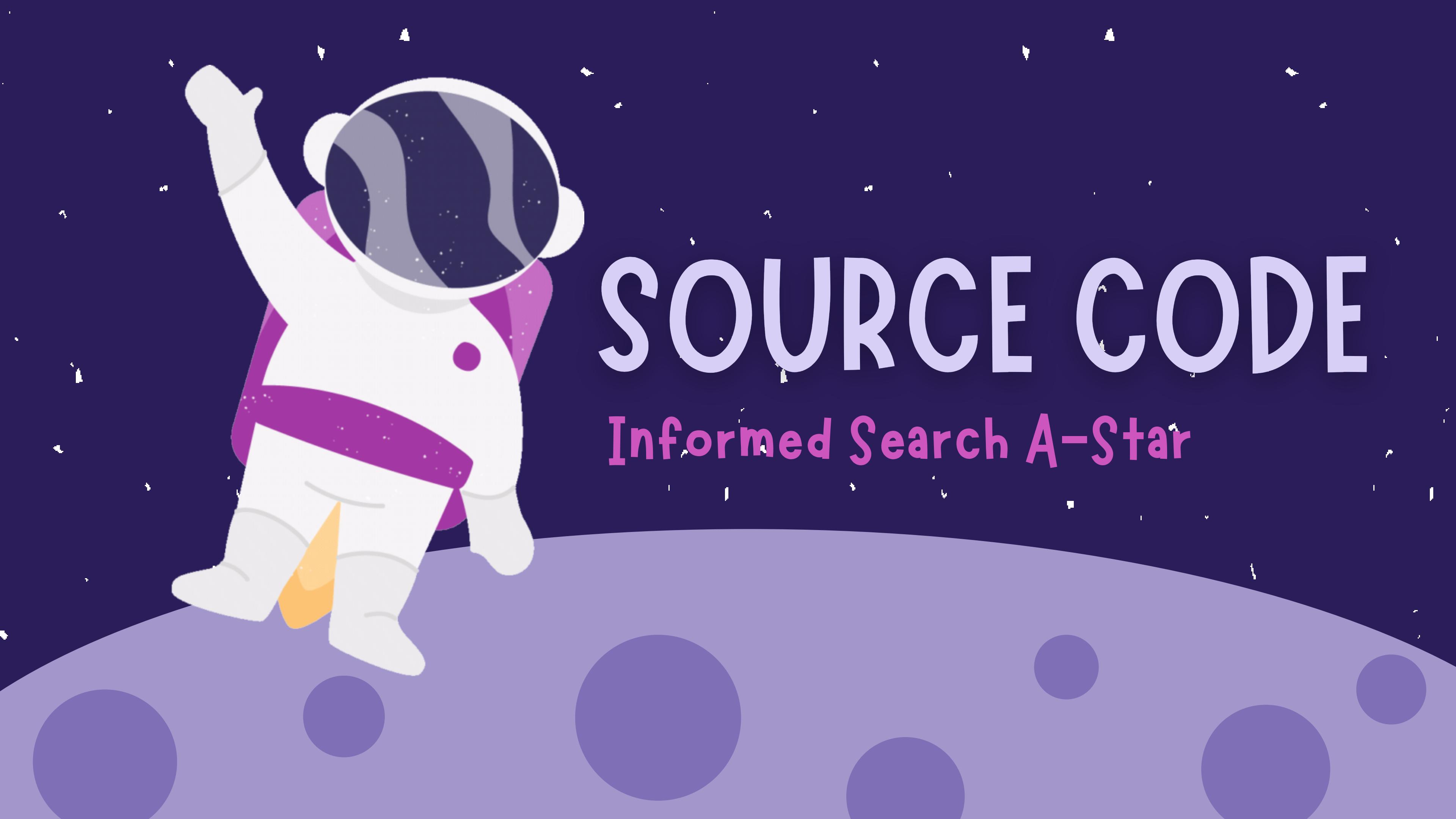
INFORMED SEARCH A*



Algoritma Informed Search A*

Algoritma Informed Search A* adalah algoritma pencarian jarak terpendek yang menggabungkan pendekatan Best-First Search dan Greedy Search. Ia menggunakan fungsi heuristik untuk memprediksi biaya tersisa dalam mencapai tujuan.

Algoritma A* mencari jalur optimal antara dua node/kota dengan mempertimbangkan biaya sejauh ini dan estimasi biaya tersisa. Ciri khasnya adalah pendekatan terinformasi, kombinasi BFS dan Greedy Search, serta keoptimalan jika fungsi heuristik memenuhi syarat tertentu.



SOURCE CODE

Informed Search A-Star



GRAPH_NODES

```
1 #Generate graph node/kota dan edge beserta cost
2 Graph_nodes = {
3     'A': [('B', 6), ('F', 3)],
4     'B': [('A', 6), ('C', 3), ('D', 2)],
5     'C': [('B', 3), ('D', 1), ('E', 5)],
6     'D': [('B', 2), ('C', 1), ('E', 8)],
7     'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
8     'F': [('A', 3), ('G', 1), ('H', 7)],
9     'G': [('F', 1), ('I', 3)],
10    'H': [('F', 7), ('I', 2)],
11    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
12 }
```

Graph_nodes

variabel Graph_nodes adalah representasi dari relasi antara node atau kota dalam suatu sistem. Graf tersebut terdiri dari beberapa node (misalnya, kota-kota) yang dihubungkan oleh edge (sisi) yang memiliki biaya atau jarak tertentu.



HEURISTIC(N)

Fungsi Heuristic(n) digunakan untuk menghasilkan estimasi biaya (heuristik) dari suatu node ke tujuan akhir (goal) dalam algoritma A*

Dalam implementasi ini, fungsi Heuristic(n) menggunakan pendekatan estimasi biaya Straight Line Distance (SLD) dari setiap node ke tujuan akhir. Dalam variabel SLD, setiap node diberi nilai estimasi biaya yang menggambarkan jarak atau biaya yang diperkirakan dari node tersebut ke tujuan akhir (node J). Nilai-nilai ini telah ditentukan sebelumnya secara manual berdasarkan estimasi jarak yang dianggap sesuai.

```
14 #Generate SLD setiap node ke goal
15 def Heuristic(n):
16     SLD = {
17         'A': 11,
18         'B': 6,
19         'C': 5,
20         'D': 7,
21         'E': 3,
22         'F': 6,
23         'G': 5,
24         'H': 3,
25         'I': 1,
26         'J': 0
27     }
28     return SLD[n]
```



ALGO A-STAR

AStar(start_node, stop_node)

Algoritma A* adalah algoritma pencarian jarak terpendek yang menggunakan pendekatan terinformasi. Algoritma ini mencari jalur optimal dari sebuah simpul awal ke simpul tujuan dengan mempertimbangkan biaya aktual dari simpul awal ke simpul saat ini serta estimasi biaya yang diperoleh dari fungsi heuristik.

```
30 #Algoritma A*
31 def AStar(start_node, stop_node):
32     open_set = set(start_node)
33     closed_set = set()
34     g = {}
35     parents = {}
36     g[start_node] = 0
37     parents[start_node] = start_node
38     while len(open_set) > 0:
39         n = None
40         for v in open_set:
41             if n == None or g[v] + Heuristic(v) < g[n] + Heuristic(n):
42                 n = v
43         if n == stop_node or Graph_nodes[n] == None:
44             pass
45         else:
46             for (m, weight) in get_neighbors(n):
47                 if m not in open_set and m not in closed_set:
48                     open_set.add(m)
49                     parents[m] = n
50                     g[m] = g[n] + weight
51                 else:
52                     if g[m] > g[n] + weight:
53                         g[m] = g[n] + weight
54                         parents[m] = n
55                         if m in closed_set:
56                             closed_set.remove(m)
57                             open_set.add(m)
58             if n == None:
59                 print('Path does not exist!')
60                 return None
61
62             if n == stop_node:
63                 path = []
64                 while parents[n] != n:
65                     path.append(n)
66                     n = parents[n]
67                 path.append(start_node)
68                 path.reverse()
69                 print('Path found: {}'.format(path))
70                 return path
71             open_set.remove(n)
72             closed_set.add(n)
73             print('Path does not exist!')
74             return None
```



ALGO A-STAR

AStar(start_node, stop_node)

Langkah-langkah dalam algoritma A* melibatkan:

- Inisialisasi himpunan `open_set`, `closed_set`, kamus `g`, dan kamus `parents`.
- Selama `open_set` tidak kosong, pilih simpul dengan nilai kombinasi terkecil antara biaya aktual dan estimasi biaya (heuristik) sebagai simpul saat ini.
- Periksa apakah simpul saat ini adalah simpul tujuan atau tidak memiliki tetangga. Jika ya, proses pencarian selesai.

```
30 #Algoritma A*
31 def AStar(start_node, stop_node):
32     open_set = set(start_node)
33     closed_set = set()
34     g = {}
35     parents = {}
36     g[start_node] = 0
37     parents[start_node] = start_node
38     while len(open_set) > 0:
39         n = None
40         for v in open_set:
41             if n == None or g[v] + Heuristic(v) < g[n] + Heuristic(n):
42                 n = v
43         if n == stop_node or Graph_nodes[n] == None:
44             pass
45         else:
46             for (m, weight) in get_neighbors(n):
47                 if m not in open_set and m not in closed_set:
48                     open_set.add(m)
49                     parents[m] = n
50                     g[m] = g[n] + weight
51                 else:
52                     if g[m] > g[n] + weight:
53                         g[m] = g[n] + weight
54                         parents[m] = n
55                         if m in closed_set:
56                             closed_set.remove(m)
57                             open_set.add(m)
58             if n == None:
59                 print('Path does not exist!')
60                 return None
61
62             if n == stop_node:
63                 path = []
64                 while parents[n] != n:
65                     path.append(n)
66                     n = parents[n]
67                 path.append(start_node)
68                 path.reverse()
69                 print('Path found: {}'.format(path))
70                 return path
71             open_set.remove(n)
72             closed_set.add(n)
73             print('Path does not exist!')
74             return None
```



ALGO A-STAR

AStar(start_node, stop_node)

Langkah-langkah dalam algoritma A* melibatkan:

- Untuk setiap tetangga dari simpul saat ini, perbarui biaya aktual jika jalur baru lebih efisien.
- Jika tidak ada simpul yang ditemukan atau jalur tidak ada, algoritma berakhir dan mencetak pesan yang sesuai.
- Jika jalur terpendek ditemukan, jalur tersebut dibangun dari simpul awal ke simpul tujuan menggunakan informasi dari kamus parents.

```
30 #Algoritma A*
31 def AStar(start_node, stop_node):
32     open_set = set(start_node)
33     closed_set = set()
34     g = {}
35     parents = {}
36     g[start_node] = 0
37     parents[start_node] = start_node
38     while len(open_set) > 0:
39         n = None
40         for v in open_set:
41             if n == None or g[v] + Heuristic(v) < g[n] + Heuristic(n):
42                 n = v
43         if n == stop_node or Graph_nodes[n] == None:
44             pass
45         else:
46             for (m, weight) in get_neighbors(n):
47                 if m not in open_set and m not in closed_set:
48                     open_set.add(m)
49                     parents[m] = n
50                     g[m] = g[n] + weight
51                 else:
52                     if g[m] > g[n] + weight:
53                         g[m] = g[n] + weight
54                         parents[m] = n
55                         if m in closed_set:
56                             closed_set.remove(m)
57                             open_set.add(m)
58             if n == None:
59                 print('Path does not exist!')
60                 return None
61
62             if n == stop_node:
63                 path = []
64                 while parents[n] != n:
65                     path.append(n)
66                     n = parents[n]
67                 path.append(start_node)
68                 path.reverse()
69                 print('Path found: {}'.format(path))
70                 return path
71             open_set.remove(n)
72             closed_set.add(n)
73             print('Path does not exist!')
74             return None
```



ALGO A-STAR

AStar(start_node, stop_node)

Algoritma A* memanfaatkan heuristik untuk memandu pencarian dan mengurangi jumlah simpul yang dieksplorasi, sehingga meningkatkan efisiensi dalam mencari jalur terpendek.

Pada implementasi ini, algoritma A* menggunakan graf yang didefinisikan dalam Graph_nodes dan fungsi heuristik Heuristic(n) untuk mengestimasi biaya tersisa dari setiap simpul ke simpul tujuan.

```
30 #Algoritma A*
31 def AStar(start_node, stop_node):
32     open_set = set(start_node)
33     closed_set = set()
34     g = {}
35     parents = {}
36     g[start_node] = 0
37     parents[start_node] = start_node
38     while len(open_set) > 0:
39         n = None
40         for v in open_set:
41             if n == None or g[v] + Heuristic(v) < g[n] + Heuristic(n):
42                 n = v
43         if n == stop_node or Graph_nodes[n] == None:
44             pass
45         else:
46             for (m, weight) in get_neighbors(n):
47                 if m not in open_set and m not in closed_set:
48                     open_set.add(m)
49                     parents[m] = n
50                     g[m] = g[n] + weight
51                 else:
52                     if g[m] > g[n] + weight:
53                         g[m] = g[n] + weight
54                         parents[m] = n
55                         if m in closed_set:
56                             closed_set.remove(m)
57                             open_set.add(m)
58             if n == None:
59                 print('Path does not exist!')
60                 return None
61
62             if n == stop_node:
63                 path = []
64                 while parents[n] != n:
65                     path.append(n)
66                     n = parents[n]
67                 path.append(start_node)
68                 path.reverse()
69                 print('Path found: {}'.format(path))
70                 return path
71             open_set.remove(n)
72             closed_set.add(n)
73             print('Path does not exist!')
74             return None
```



GET_NEIGHBORS

```
76     def get_neighbors(v):  
77         if v in Graph_nodes:  
78             return Graph_nodes[v]  
79         else:  
80             return None
```

get_neighbors(v)

Fungsi `get_neighbors(v)` digunakan untuk mendapatkan tetangga-tetangga dari suatu simpul atau node `v` dalam graf. Fungsi ini menerima parameter `v` yang merupakan simpul yang ingin dicari tetangganya.

PROGRAM

Input

```
82 #Tentukan start dan goal  
83 AStar('A', 'J')
```

Output

```
Path found: ['A', 'F', 'G', 'I', 'J']
```

A cartoon-style illustration of an astronaut in a white spacesuit with purple stripes, floating in a dark blue space filled with white stars. The astronaut's helmet has a circular window showing the stars. They are holding onto a white cylindrical object.

THANK YOU!

Do you have any questions?