Penerapan Algoritma A* dalam Sistem Pendeteksian Lokasi Terdekat untuk Evakuasi dan Pemadaman Kebakaran Hutan yang Efisien

Auralea Alvinia Syaikha - 13522148
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522148@std.stei.itb.ac.id

***Abstract*—Penelitian ini mengembangkan dan menerapkan algoritma *Greedy Best-First Search* (GBFS) dalam sistem pendeteksian lokasi terdekat untuk evakuasi dan pemadaman kebakaran. GBFS menggunakan pendekatan heuristik untuk menemukan rute tercepat ke tujuan, ideal untuk situasi darurat. Hasil simulasi dan pengujian kasus nyata menunjukkan bahwa GBFS secara signifikan mengurangi waktu pencarian dan meningkatkan efisiensi rute dibandingkan dengan algoritma tradisional seperti Depth-First Search (DFS) dan Breadth-First Search (BFS). Penelitian ini menyimpulkan bahwa GBFS efektif dan efisien untuk pendeteksian lokasi terdekat dalam evakuasi dan pemadaman kebakaran.**

Keywords—Graf, greedy best-first search, DFS, BFS, efisiensi rute

I. PENDAHULUAN

Sumber: https://www.forestdigest.com/

Kebakaran hutan merupakan bencana alam yang seringkali mengakibatkan kerusakan lingkungan yang signifikan serta mengancam keselamatan manusia dan properti. [1] Menurut data Badan Nasional Penanggulangan Bencana (BNPB), pada tahun 2023 saja terdapat sekitar 211 kasus kebakaran hutan dan lahan di Indonesia, menyebabkan kerugian ekonomi yang mencapai triliunan rupiah dan mengancam kehidupan masyarakat di berbagai wilayah. Penanganan kebakaran hutan yang efektif memerlukan strategi evakuasi dan pemadaman yang cepat dan efisien. Namun, realitas di lapangan menunjukkan bahwa proses ini sering kali menghadapi berbagai kendala, mulai dari akses yang sulit, terbatasnya sumber daya, hingga luasnya area yang terdampak. Ketidakefisienan dalam penentuan rute evakuasi dan lokasi pemadaman yang optimal dapat mengakibatkan peningkatan risiko bagi para korban dan petugas penyelamat, serta kerusakan yang lebih parah terhadap lingkungan.

Algoritma pencarian tradisional seperti Depth-First Search (DFS) dan Breadth-First Search (BFS), meskipun banyak digunakan, memiliki keterbatasan dalam hal efisiensi waktu dan ketepatan dalam situasi dinamis seperti kebakaran hutan. DFS cenderung melakukan pencarian yang mendalam tanpa mempertimbangkan jarak ke tujuan, sementara BFS melakukan pencarian menyeluruh namun tidak efisien dalam menemukan rute terpendek dengan cepat. Akibatnya, kedua algoritma ini kurang optimal untuk diterapkan dalam kondisi darurat yang membutuhkan respons cepat.

Sebaliknya, algoritma A* menawarkan solusi yang lebih efisien dengan pendekatan heuristik. Algoritma ini memprioritaskan penjelajahan berdasarkan jarak terdekat ke tujuan, memungkinkan identifikasi rute tercepat dan terdekat dengan cepat. Algoritma A* menggunakan fungsi heuristik yang memperkirakan jarak atau biaya perjalanan ke tujuan akhir, sehingga dapat meningkatkan efisiensi dalam proses evakuasi dan pemadaman kebakaran.

Makalah ini mengkaji penerapan dalam sistem pendeteksian lokasi terdekat untuk evakuasi dan pemadaman kebakaran hutan. Kami akan menganalisis bagaimana A* dapat mengatasi keterbatasan yang dihadapi oleh algoritma tradisional dan meningkatkan kecepatan serta ketepatan dalam penentuan rute. Selain itu, kami akan menyajikan hasil simulasi dan studi kasus nyata untuk mengilustrasikan efektivitas A* dalam berbagai skenario kebakaran hutan. Penelitian ini bertujuan memberikan kontribusi signifikan dalam meningkatkan respons terhadap kebakaran hutan, mengurangi dampak negatif, serta menyelamatkan lebih banyak nyawa dan properti.

II. TEORI DASAR

A. *Algoritma Route Planning*

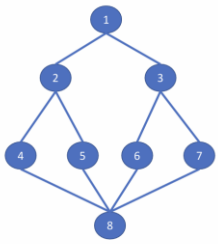
Algoritma perencanaan rute (*route planning*) adalah metode yang digunakan untuk menentukan jalur optimal dari satu titik ke titik lain dalam suatu jaringan, seperti peta jalan, jaringan transportasi, atau sistem komputer. Algoritma ini sangat penting dalam berbagai aplikasi, termasuk navigasi GPS, logistik, robotika, dan manajemen jaringan. Dalam konteks perencanaan rute, algoritma pencarian dapat dibagi menjadi dua kategori utama, yaitu:

1. Uninformed search
   Algoritma uninformed search, juga dikenal sebagai algoritma *blind search*, adalah metode pencarian yang tidak menggunakan informasi tambahan mengenai jarak atau biaya ke tujuan dalam proses pencarian. Algoritma ini hanya menggunakan informasi yang tersedia di tingkat lokal (simpul dan tetangganya) tanpa perkiraan jarak ke tujuan. Berikut adalah beberapa contoh algoritma yang tergolong *uninformed search*:
   a. BFS (*Breadth First Search*)
   b. DFS (*Depth First Search*)
   c. DLS (*Depth Limited Search*)
   d. IDS (*Iterative Deepening Search*)
   e. UCS (*Uniform Cost Search*)

2. Informed search
   Algoritma informed search, juga dikenal sebagai algoritma heuristic search, menggunakan informasi tambahan (heuristik) untuk memandu pencarian menuju tujuan dengan lebih efisien. Heuristik adalah fungsi yang memperkirakan biaya atau jarak dari simpul saat ini ke tujuan. Berikut adalah beberapa contoh algoritma yang tergolong *informed search*:
   a. *Greedy Best-first Search*
   b. A*

Algoritma *informed search* lebih efisien dalam menemukan rute optimal dibandingkan dengan algoritma *uninformed search* karena menggunakan informasi tambahan untuk mengarahkan pencarian secara lebih cerdas dan menghindari eksplorasi yang tidak perlu.

B. *Breadth-first Search (BFS)*



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

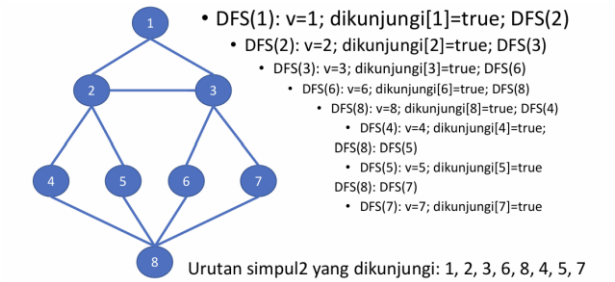
Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Sumber : BFS-DFS-2021-Bag1-2024.pdf (itb.ac.id)

[2] Algoritma Breadth-First Search (BFS) adalah metode pencarian yang mengeksplorasi graf atau pohon secara lapis demi lapis, dimulai dari simpul awal dan mengunjungi semua tetangganya sebelum melanjutkan ke level berikutnya. Algoritma ini efektif dalam menemukan rute terpendek pada graf yang tidak berbobot, dengan menggunakan antrian untuk menyimpan simpul yang akan dieksplorasi dan set atau array penanda untuk melacak simpul yang telah dikunjungi. BFS memiliki kompleksitas waktu O(V + E) dan kompleksitas ruang O(V), di mana V adalah jumlah simpul dan E adalah jumlah sisi dalam graf. Keunggulan BFS adalah kemampuannya untuk menemukan rute terpendek dan eksplorasi sistematis, meskipun memiliki keterbatasan dalam konsumsi memori yang besar dan ketidakefisienan dalam graf berbobot.

Meskipun BFS efektif dalam menemukan rute terpendek pada graf yang tidak berbobot, kekurangannya menjadi jelas ketika diterapkan pada graf yang berbobot atau besar. Karena BFS tidak mempertimbangkan bobot sisi, algoritma ini dapat menghabiskan banyak waktu dan sumber daya untuk mengeksplorasi simpul-simpul yang jaraknya jauh dari simpul awal, bahkan jika jalur alternatif yang lebih pendek tersedia. Dalam kasus graf yang sangat besar, memori yang dibutuhkan untuk menyimpan semua simpul yang telah dikunjungi juga dapat menjadi masalah serius. Akibatnya, ketika mencari rute terpendek pada graf yang berbobot atau besar, BFS seringkali kurang efisien dan tidak cukup. Dalam konteks evakuasi dan pemadaman kebakaran, di mana faktor waktu dan efisiensi sangat penting, penggunaan BFS mungkin tidak selalu menjadi pilihan yang optimal.

C. *Depth-first Search (DFS)*



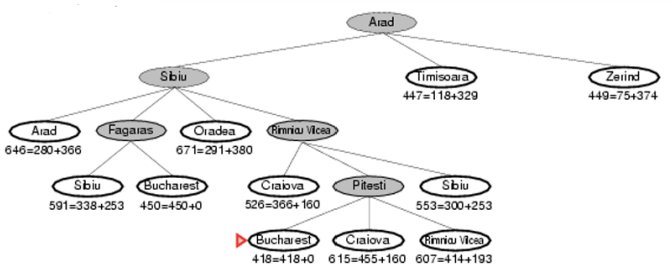
Sumber : BFS-DFS-2021-Bag1-2024.pdf (itb.ac.id)

[3] Algoritma Depth-First Search (DFS) adalah metode pencarian yang mengeksplorasi graf atau pohon dengan melakukan pencarian secara mendalam terlebih dahulu sebelum kembali dan mengeksplorasi cabang-cabang lain. DFS dimulai dari simpul awal, kemudian bergerak ke salah satu tetangganya dan terus bergerak ke simpul-simpul berikutnya hingga mencapai simpul tanpa tetangga yang belum dikunjungi atau simpul tujuan. Jika simpul tanpa tetangga tercapai, algoritma kembali ke simpul sebelumnya dan mengeksplorasi tetangga lainnya. Proses ini berlanjut hingga semua simpul telah dikunjungi atau simpul tujuan ditemukan. DFS menggunakan struktur data tumpukan (stack),

baik secara eksplisit dengan menggunakan tumpukan atau secara implisit melalui rekursi. DFS memiliki kompleksitas waktu O(V + E), di mana V adalah jumlah simpul dan E adalah jumlah sisi dalam graf, serta kompleksitas ruang O(V) dalam kasus terburuk, terutama karena penggunaan rekursi atau tumpukan untuk menyimpan simpul yang akan dieksplorasi.

DFS sangat efektif untuk masalah yang membutuhkan eksplorasi lengkap dari semua jalur, seperti dalam pemecahan masalah labirin atau pencarian solusi dalam permainan teka-teki. Namun, DFS memiliki keterbatasan dalam hal efisiensi untuk menemukan rute terpendek pada graf yang berbobot atau besar, karena cenderung melakukan pencarian yang mendalam tanpa mempertimbangkan jarak ke tujuan.

D. A (A Star)*



Sumber : BFS-DFS-2021-Bag1-2024.pdf (itb.ac.id)

Algoritma A* adalah algoritma pencarian yang digunakan untuk menemukan jalur terpendek antara simpul awal dan simpul tujuan dalam graf berbobot. Algoritma ini merupakan penggabungan dari algoritma pencarian heuristik dan pencarian garis lurus. A* menggunakan fungsi heuristik untuk memperkirakan biaya tambahan dari simpul saat ini ke simpul tujuan. Fungsi heuristik ini digunakan bersama dengan biaya sejauh ini untuk menentukan biaya total dari simpul saat ini ke simpul tujuan. [4] Berikut adalah rumus mencari bobot pada algoritma ini:

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

Keterangan:
g(n) : estimasi biaya dari simpul saat ini ke simpul tujuan.
h(n) : perkiraan biaya termurah jalur dari N ke tujuan.

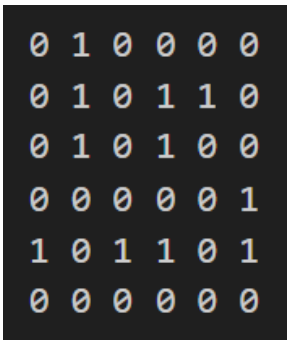
Algoritma A* bekerja dengan cara mengeksplorasi simpul-simpul dalam urutan yang paling menjanjikan, di mana prioritas diberikan pada simpul-simpul yang memiliki nilai f(n) yang paling rendah. Dengan pendekatan ini, A* memungkinkan untuk menemukan jalur terpendek dengan efisien, terutama jika heuristik yang digunakan cukup akurat. Algoritma ini sangat berguna dalam berbagai aplikasi, termasuk navigasi GPS, perencanaan rute, dan permainan komputer.

III. Implementasi Algoritma

Implementasi dilakukan menggunakan Bahasa pemrograman Python. Implementasi dilakukan dengan membuat algoritma pencarian rute evakuasi tercepat menggunakan A*.

A. Pemodelan Peta Lokasi

Langkah pertama adalah memodelkan peta lokasi kebakaran dalam bentuk matriks 2D, di mana setiap elemen dalam matriks mewakili posisi di peta. Nilai 0 menandakan jalur yang dapat dilewati, sedangkan nilai 1 menandakan jalur yang tidak dapat dilewati karena adanya kebakaran atau rintangan lainnya. Mengingat bahwa dataset lokasi aktual kebakaran hutan di Indonesia terbatas, maka berikut adalah contoh ilustrasinya.



Kemudian, matriks tersebut dapat dijadikan dataset untuk pencarian rute. Matriks tersebut disimpan dalam file txt untuk nantinya dibaca oleh program.

B. Penetapan titik awal dan tujuan

Setelah pemodelan peta, langkah berikutnya adalah menetapkan titik awal (lokasi petugas pemadam atau titik evakuasi terdekat) dan titik tujuan (lokasi yang terancam atau memerlukan bantuan). Sebagai contoh, ditetapkan titik (0,0) sebagai titik awalnya dan titik (4,4) sebagai titik tujuannya.

*C. Implementasi Algoritma A**

Algoritma A* memerlukan inisialisasi simpul awal dan tujuan, serta inisialisasi struktur data seperti himpunan terbuka dan himpunan tertutup untuk melacak simpul yang sudah diproses.

Setiap simpul yang dieksplorasi oleh algoritma A* memiliki nilai fungsi evaluasi yang diperkirakan, yang merupakan kombinasi antara biaya aktual untuk mencapai simpul tersebut dan estimasi biaya yang tersisa menuju tujuan. Estimasi biaya tersisa dapat dihitung menggunakan heuristik, seperti jarak Manhattan atau Euclidean.

Saat algoritma A* mencapai simpul tujuan, jalur terpendek dari titik awal ke titik tujuan direkonstruksi dari informasi yang disimpan dalam simpul-simpul yang dieksplorasi. Jalur ini kemudian dapat digunakan untuk menavigasi petugas pemadam atau tim evakuasi melalui peta lokasi dengan efisien.

Berikut adalah kode program yang digunakan untuk pencarian rute dalam kasus ini.

```
import heapq

# Load the map from the text fil
def load_map(filename):
    with open(filename, 'r') as f:
        map_data = [list(map(int, line.split())) for line in f]
    return map_data

# Define the heuristic function (Manhattan distanc
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# Get the neighbors of a nod
def get_neighbor (map_data, node):
    neighbors = []
    direction  = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    for direction in direction :
        neighbor = (node[0] + direction[0], node[1] + direction[1])
        if 0 <= neighbor[0] < len(map_data) and 0 <= neighbor[1] < len(map_data[0]):
            if map_data[neighbor[0]][neighbor[1]] == 0:
                neighbors.append(neighbor)
    return neighbors

# A* algorithm implementatio
def a_star_searc (map_data, start, goal):
    bpen_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)[1]
        print(f"Expanding node: {current}")

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        for neighbor in get_neighbor (map_data, current):
            print(f"Child node: {neighbor}")
            tentative_g_scor  = g_score[current] + 1
            e
            if neighbor not in g_score or tentative_g_scor  < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_scor
                f_score[neighbor] = tentative_g_scor  + heuristic(neighbor, goal)
                heapq.heappush(open_set, (f_score[neighbor], neighbor))

    return None
```

Berikut ini adalah fungsi utama dari program diatas.

```
# Main functio
if __name__ == "__main_  :
    map_data = load_map('map.txt')
    start = (0, 0)  # Example start position; modify as neede
    goal = (4, 4)  #dExample goal position; modify as neede
                   d
    path = a_star_searc (map_data, start, goal)
    if pathb
        print("Path foun   , path)
    else:     d:"
        print("No path foun   )
              d."
```

Pada program diatas, fungsi `load_map(filename)` bertanggung jawab untuk memuat peta dari file teks. Dengan menggunakan `with open(filename, 'r') as f`, kode membuka file dengan nama `filename` dalam mode baca ('r'), dan kemudian membaca setiap baris dari file tersebut. Setiap baris dipisahkan menggunakan `line.split()` dan diubah menjadi list of integers dengan `list(map(int, line.split()))`. Setelah semua baris dibaca, maka peta akan terbentuk dalam bentuk matriks yang merupakan list of lists, dan kemudian dikembalikan sebagai output dari fungsi ini.

Fungsi `heuristic(a, b)` menghitung estimasi biaya yang tersisa dari suatu titik `a` ke titik tujuan `b`. Dalam hal ini, digunakan metode jarak Manhattan, yang dihitung dengan mengambil perbedaan absolut koordinat x dan y antara kedua titik, lalu menjumlahkannya.

Fungsi `get_neighbors(map_data, node)` mendapatkan tetangga-tetangga dari suatu node pada peta. Dengan menggunakan `directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]`, kita mendefinisikan arah tetangga dari node saat ini (atas, kanan, bawah, kiri). Kemudian, untuk setiap arah, kita coba tentukan koordinat tetangga baru dengan menambahkan koordinat arah tersebut ke koordinat node saat ini. Jika tetangga tersebut berada dalam batas peta dan merupakan jalur yang dapat dilewati (nilai 0 dalam `map_data`), maka tetangga tersebut ditambahkan ke daftar tetangga.

Fungsi `a_star_search(map_data, start, goal)` adalah inti dari algoritma A*. Kita mulai dengan menginisialisasi himpunan terbuka `open_set`, serta `came_from`, `g_score`, dan `f_score`. Selama himpunan terbuka tidak kosong, kita mengeluarkan simpul dengan nilai f terkecil. Jika simpul tersebut adalah tujuan, maka kita rekonstruksi jalur dari simpul awal ke simpul tujuan dan mengembalikannya. Jika tidak, kita mengeksplorasi tetangga-tetangga dari simpul tersebut. Untuk setiap tetangga, kita mencetak informasi ekspansi, kemudian memperbarui nilai g dan f, serta menambahkan tetangga tersebut ke himpunan terbuka. Algoritma berhenti ketika mencapai simpul tujuan atau himpunan terbuka kosong, dan mengembalikan jalur terpendek dari titik awal ke titik tujuan, jika ada.

IV. Analisis

A. *Output Program*

Berdasarkan program yang tertera pada bab Implementasi Algoritma diatas, didapatkan output sebagai berikiut.

```
Expanding node: (0, 0)
Child node: (1, 0)
Expanding node: (1, 0)
Child node: (2, 0)
Child node: (0, 0)
Expanding node: (2, 0)
Child node: (3, 0)
Child node: (1, 0)
Expanding node: (3, 0)
Child node: (3, 1)
Child node: (2, 0)
Expanding node: (3, 1)
Child node: (3, 2)
Child node: (4, 1)
Child node: (3, 0)
```

```
Expanding node: (3, 2)
Child node: (3, 3)
Child node: (3, 1)
Child node: (2, 2)
Expanding node: (3, 3)
Child node: (3, 4)
Child node: (3, 2)
Expanding node: (3, 4)
Child node: (4, 4)
Child node: (3, 3)
Child node: (2, 4)
Expanding node: (4, 1)
Child node: (5, 1)
Child node: (3, 1)
Expanding node: (4, 4)
```

Pertama, algoritma mulai dengan mengekspansi node awal (0, 0) dan menemukan satu tetangga yang dapat dijangkau, yaitu node (1, 0). Selanjutnya, node (1, 0) diekspansi, menghasilkan dua tetangga yang dapat dijangkau, yaitu node (2, 0) dan node (0, 0).

Kemudian, proses berlanjut dengan pengeksplorasian node (2, 0), yang memiliki dua tetangga yang dapat dijangkau, yaitu node (3, 0) dan node (1, 0). Setelah itu, algoritma mengeksplorasi node (3, 0), yang memiliki dua tetangga yang dapat dijangkau, yaitu node (3, 1) dan node (2, 0).

Proses tersebut berlanjut hingga algoritma mencapai node tujuan (4, 4). Setelah mencapai node tujuan, algoritma membangun jalur kembali dari node tujuan ke node awal, dan output mencatatkan jalur yang ditemukan sebagai [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (4, 4)].

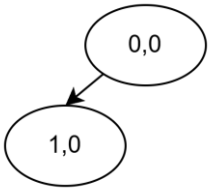
Dengan detail ekspansi simpul diatas, didapatkan jalur optimal untuk mencapai ke simpul tujuan sebagai berikut.

```
Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (4, 4)]
```

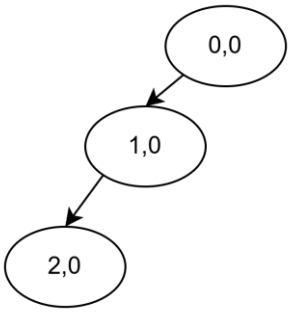
B. Penggambaran dalam Bentuk Graf

1. Iterasi 1

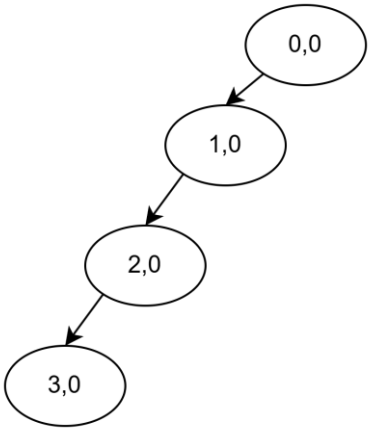
2. Iterasi 2



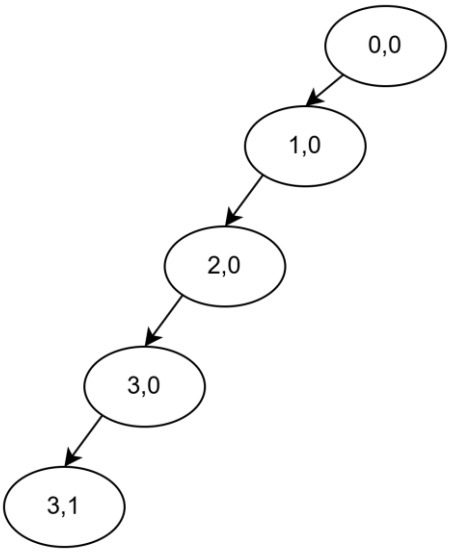
3. Iterasi 3



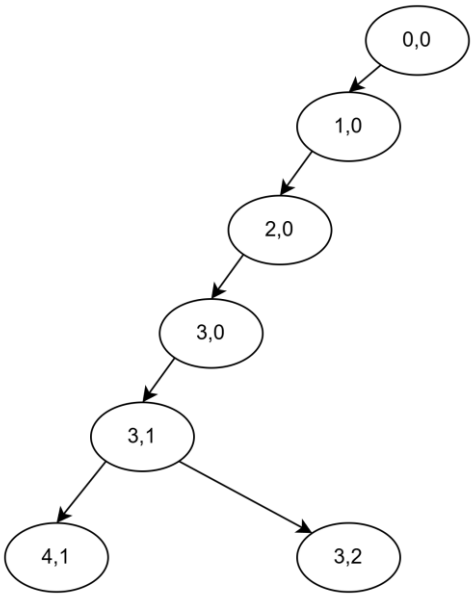
4. Iterasi 4



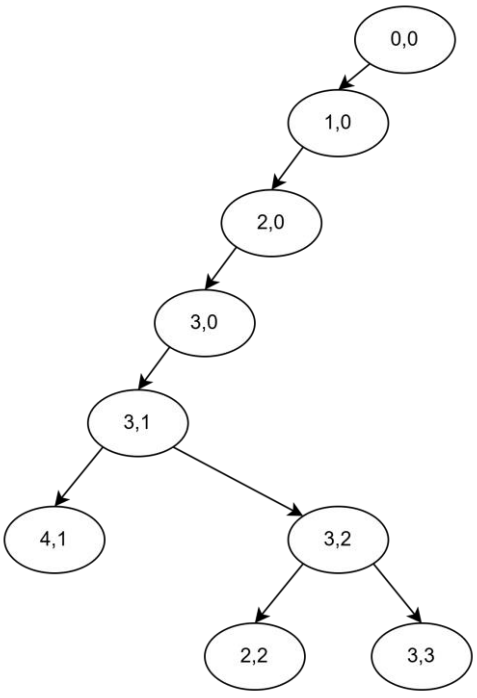
5. Iterasi 5



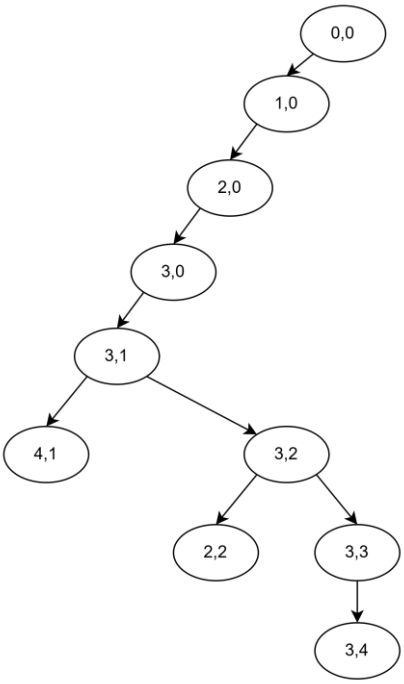
6. Iterasi 6



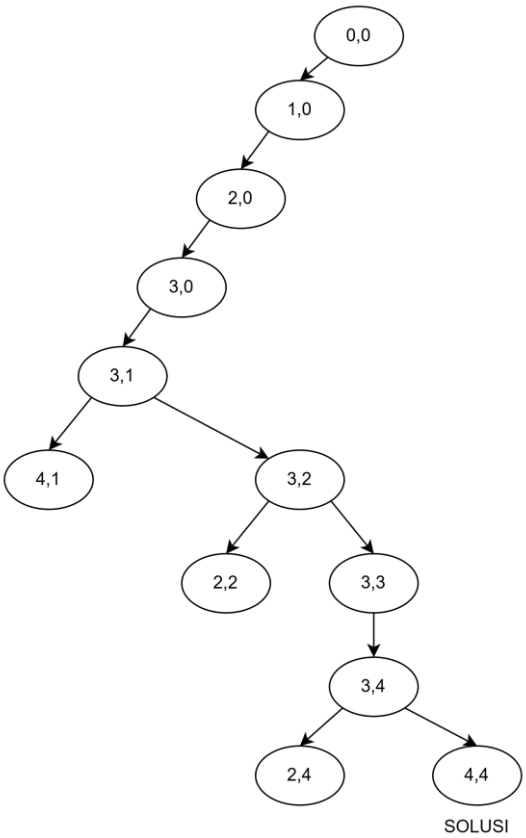
7. Iterasi 7



8. Iterasi 8



9. Iterasi 9



LINK VIDEO

Video Makalah : https://youtu.be/bf7g4D2SLQ0

UCAPAN TERIMA KASIH

Dalam kesempatan ini, penulis ingin menyampaikan ucapan terima kasih kepada semua pihak yang telah turut serta mendukung dan berkontribusi dalam penyelesaian makalah ini. Terima kasih kepada:

1. Tuhan Yang Maha Esa, berkat rahmat dan hidayah-Nya, makalah ini dapat terselesaikan dengan lancar
2. Keluarga dan teman yang telah memberikan dukungan kepada penulis selama pengerjaan makalah ini
3. Ir. Rila Mandala, M.Eng., Ph.D. dan Monterico Adrian, S.T., M.T., selaku dosen mata kuliah IF2211 Strategi Algoritma yang telah memberikan ilmu-ilmu dan materi dasar yang membantu dalam penulisan makalah ini

Ucapan terima kasih yang tulus kepada semua pihak yang telah turut serta dalam perjalanan penulisan makalah ini.

REFERENSI

[1] AntaraLampung. (2023). BNPB catat 211 kasus kebakaran hutan dan lahan hingga Juni 2023. *BNPB catat 211 kasus kebakaran hutan dan lahan hingga Juni 2023 - ANTARA News Lampung*. Diakses pada 10 Juni 2023, pukul 17.00 WIB.

[2] Munir, Rinaldi. (2021). Breadth Firs Search (BFS) dan Depth First Search (DFS) (Bagian 1). *https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BF-DFS-2021-Bag1.pdf* . *Diakses pada 10 Juni 2023, pukul 20.00 WIB.*

[3] Munir, Rinaldi. (2021). Breadth Firs Search (BFS) dan Depth First Search (DFS) (Bagian 2). *https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf* . *Diakses pada 10 Juni 2023, pukul 20.30 WIB.*

[4] Munir, Rinaldi. (2024). Penentuan rute (Route/Path Planning) – Bagian 2.https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf. *Diakses pada 10 Juni 2023, pukul 21.20 WIB.*

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Auralea Alvinia Syaikha
13522148