Pemanfaatan Algoritma Backtracking dalam Penyelesaian Puzzle Hitori

Enrique Yanuar- 13522077
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): enriqueyanuar.7@gmail.com

Makalah ini menyajikan pemanfaatan algoritma backtracking dalam penyelesaian puzzle Hitori, sebuah permainan puzzle logika yang populer. Tujuan dari permainan Hitori adalah untuk menghilangkan beberapa sel dari sebuah grid sehingga tidak ada baris atau kolom yang mengandung angka yang sama, dan sel-sel yang tersisa membentuk satu kelompok yang terhubung tanpa bagian yang terisolasi. Kami mengeksplorasi pendekatan algoritmik untuk menyelesaikan puzzle Hitori menggunakan backtracking, yang secara sistematis mencari solusi dengan mengeksplorasi konfigurasi yang mungkin dan menarik kembali pilihan yang tidak valid. Detail implementasi, termasuk desain algoritma, pertimbangan efisiensi, dan teknik optimasi, dibahas dalam makalah ini. Hasil eksperimen menunjukkan efektivitas pendekatan backtracking dalam menemukan solusi untuk berbagai ukuran dan kompleksitas puzzle. Makalah ini diakhiri dengan wawasan tentang tantangan yang dihadapi, potensi perbaikan, dan arah penelitian masa depan dalam konteks penyelesaian puzzle algoritmik.

Keywords— Hitori puzzle, backtracking algorithm, logic puzzle, algorithmic solving, grid-based puzzle, connected cells, puzzle optimization, computational efficiency.

I. PENDAHULUAN

Puzzle Hitori adalah salah satu permainan logika yang menantang dan memerlukan pemikiran analitis yang mendalam. Permainan ini dimainkan di atas grid persegi yang diisi dengan angka-angka, dan tujuan utamanya adalah untuk menghilangkan beberapa sel sehingga setiap baris dan kolom tidak mengandung angka yang sama lebih dari sekali. Selain itu, sel-sel yang tersisa harus membentuk satu kelompok yang terhubung, memastikan tidak ada bagian yang terisolasi.

Hitori, yang berarti "sendiri" dalam bahasa Jepang, berasal dari negara Jepang dan telah menjadi populer di kalangan penggemar teka-teki di seluruh dunia. Puzzle ini pertama kali muncul di majalah teka-teki Jepang, dan sejak itu telah menarik minat banyak orang karena kesederhanaannya yang menipu dan tantangan logikanya yang kompleks.

3	2	5	4	5
2	3	4	3	5
4	3	2	4	4
1	3	3	5	5
5	4	1	2	3

Sumber: AI Project #1: Hitori

Cara bermain Hitori cukup sederhana dalam konsep namun menantang dalam pelaksanaan. Pemain harus mengidentifikasi dan menghapus sel-sel yang mengandung angka berulang di setiap baris dan kolom. Penghapusan dilakukan dengan menandai sel tersebut, biasanya dengan mewarnainya atau menandainya dengan tanda khusus. Aturannya, setelah penghapusan, tidak boleh ada angka yang sama dalam satu baris atau kolom lebih dari sekali. Sel-sel yang dihapus juga tidak boleh berdampingan secara horizontal atau vertikal. Selain itu, sel-sel yang tersisa harus membentuk satu kelompok yang terhubung secara langsung, tanpa ada sel yang terisolasi.

3	2	5	4	5
2	3	4	3	5
4	3	2	4	4
1	3	3	5	5
5	4	1	2	3

Sumber: AI Project #1: Hitori

Tantangan utama dalam permainan ini adalah menemukan kombinasi penghapusan yang memenuhi semua aturan yang ada, termasuk tidak menghilangkan lebih dari satu angka yang sama dari baris atau kolom yang sama, serta menjaga keterhubungan antar sel yang tersisa. Selain mengasah kemampuan berpikir logis, Hitori juga melatih pemain untuk bersabar dan teliti dalam menelusuri berbagai kemungkinan solusi, menjadikannya permainan yang sangat bermanfaat untuk melatih otak. Hitori telah menjadi salah satu favorit di kalangan penggemar teka-teki di seluruh dunia, karena tidak hanya menawarkan tantangan yang unik tetapi juga kepuasan tersendiri saat berhasil menyelesaikannya.

Algoritma backtracking merupakan salah satu pendekatan algoritmik yang efektif untuk menyelesaikan berbagai jenis puzzle, termasuk Hitori. Algoritma ini bekerja dengan cara mencoba setiap kemungkinan konfigurasi, dan jika suatu konfigurasi ditemukan tidak valid, algoritma ini akan kembali ke langkah sebelumnya (backtrack) dan mencoba konfigurasi lain. Pendekatan ini memungkinkan eksplorasi yang sistematis dan terstruktur dari ruang solusi, memastikan bahwa semua kemungkinan dipertimbangkan.

Makalah ini akan membahas secara mendetail tentang implementasi algoritma backtracking untuk menyelesaikan puzzle Hitori. Dimulai dengan penjelasan tentang dasar-dasar permainan Hitori dan aturan-aturannya, kemudian dilanjutkan dengan deskripsi algoritma backtracking serta bagaimana algoritma ini dapat diadaptasi untuk menyelesaikan puzzle Hitori. Selanjutnya, kami akan membahas berbagai teknik optimasi yang digunakan untuk meningkatkan efisiensi algoritma, serta hasil eksperimen yang menunjukkan kinerja algoritma pada berbagai ukuran dan kompleksitas puzzle.

Tujuan utama dari penelitian ini adalah untuk menunjukkan efektivitas algoritma backtracking dalam menyelesaikan puzzle Hitori dan memberikan wawasan tentang potensi perbaikan dan pengembangan lebih lanjut dalam bidang penyelesaian puzzle menggunakan pendekatan algoritmik. Dengan demikian, diharapkan penelitian ini dapat memberikan kontribusi yang berarti dalam pengembangan metode penyelesaian puzzle logika secara umum.

Melalui makalah ini, pembaca akan mendapatkan pemahaman yang komprehensif tentang bagaimana algoritma backtracking dapat diterapkan untuk menyelesaikan puzzle Hitori, serta tantangan dan peluang yang ada dalam implementasinya.

II. Dasar Teori

A. *Aturan Dasar permainan Hitori*

Hitori adalah teka-teki logika yang berasal dari Jepang, sering dimainkan pada grid persegi yang diisi dengan angka. Permainan ini melibatkan penghapusan angka tertentu berdasarkan aturan tertentu untuk mencapai solusi yang benar.

Berikut adalah prinsip utama dan metode bermain Hitori:

1. **Tujuan Permainan**

   Pemain harus menghilangkan beberapa sel sehingga tidak ada angka yang diulang dalam baris atau kolom yang sama.

2. **Aturan Hapus**

   Sel yang dihapus ditandai dan tidak boleh bersebelahan secara horizontal atau vertikal dengan sel lain yang dihapus.

3. **Koneksi Sel**

   Sel yang tidak dihapus harus tetap terhubung horizontal atau vertikal, membentuk satu kelompok besar tanpa ada bagian yang terisolasi.

4. **Strategi**

   Pemain sering menggunakan strategi eliminasi dan deduksi untuk menentukan sel mana yang harus dihapus, dengan mempertimbangkan implikasi setiap pilihan untuk struktur keseluruhan grid.

B. *Algoritma Backtracking*

Algoritma bakctracking, atau algoritma runut-balik, merupakan metode pemecahan masalah yang menggabungkan pendekatan pencarian sistematis dengan kemampuan untuk mundur dan mencoba jalur alternatif ketika menemui jalan buntu. Algoritma ini secara efektif memperbaiki pendekatan exhaustive search (pencarian menyeluruh) dengan hanya mengeksplorasi pilihan yang mengarah ke solusi, dan memangkas (pruning) simpul-simpul yang tidak mengarah ke solusi.

Algoritma backtracking pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950. Konsep ini kemudian diperluas dan diperinci oleh peneliti lainnya seperti R.J. Walker, Golomb, dan Baumert, yang menyajikan uraian umum tentang algoritma runut-balik. Algoritma ini sejak itu telah digunakan secara luas dalam berbagai bidang ilmu komputer dan matematika untuk memecahkan masalah kombinatorial dan optimasi.

Algoritma backtracking bekerja dengan cara menjelajahi ruang solusi secara sistematis, membangkitkan simpul-simpul status sehingga menghasilkan lintasan dari akar ke daun dalam struktur pohon ruang status (state space tree). Pada setiap simpul, algoritma ini akan:

1. Memilih sebuah opsi yang mungkin (nilai dari variabel).
2. Memeriksa apakah opsi tersebut mengarah ke solusi valid menggunakan fungsi pembatas (bounding function).
3. Jika opsi tersebut valid, algoritma melanjutkan ke langkah berikutnya (rekursi).
4. Jika opsi tersebut tidak valid, algoritma mundur (backtrack) ke langkah sebelumnya dan mencoba opsi lain.

```
procedure BacktrackAlgorithm(input k: integer)
{Mencari semua solusi masalah dengan metode backtracking; skema rekursif}
{
    Masukan: k, yaitu indeks komponen vektor solusi, x[k]. Diasumsikan x[1], x[2], ...,
    x[k-1] sudah ditentukan nilainya.
    Luaran: semua solusi x = (x[1], x[2], ..., x[n])
}

Algoritma:
for setiap x[k] ∈ T(x[1], x[2], ..., x[k-1]) do
    if B(x[1], x[2], ..., x[k]) = true then
        if (x[1], x[2], ..., x[k]) adalah lintasan dari akar ke simpul solusi then
            write(x[1], x[2], ..., x[k]) {cetak solusi}
        endif
        if k < n then
            BacktrackAlgorithm(k+1) {tentukan nilai untuk x[k+1]}
        endif
    endif
endfor
```

Sumber: Dokumentasi pribadi

III. IMPLEMENTASI

Dalam pengimplementasian algoritma backtrackting untuk menyelesaikan puzzle hitori ada beberapa hal yang harus dipersiapkan

A. Metodelogi

Untuk menemukan solusi pada puzzle dengan cara mencoba setiap kemungkinan konfigurasi satu per satu sampai solusi ditemukan atau semua kemungkinan telah dicoba. Backtracking efektif untuk teka-teki seperti Hitori karena ia memungkinkan pencarian mendalam pada ruang solusi dan membatalkan langkah yang salah (backtrack) ketika jalan buntu ditemui.

B. Struktu data

Struktur data (variable) yang akan digunakaan dalam implementasi algortima backtracking untuk menyelesaikan puzzle hitori adalah:

1. Grid

   Berupa array 2 dimensi dari integer untuk menyimpan angka angka dalam puzzle

2. Solution

   Berupa array 2 dimensi dari string untuk menyimpan status tiap sel sebagai sebagai 'B' (hitam), 'W' (putih), atau '.' (belum ditentukan).

C. Fungsi Utama

Fungsi utama yang digunakan dalam implementasi algoritma backtracking untuk menyelesaikan puzzle hitori adalah

1. Fungsi ‘is_valid_partial(grid, solution, n, row, col)’

   Fungsi ini bertujuan untuk memeriksa validitas parsial dari konfigurasi saat ini pada grid solusi. Validitas ini diperiksa berdasarkan dua aturan utama dari puzzle Hitori:

   - **Tidak ada Sel Hitam Berdekatan**: Fungsi ini memeriksa apakah ada sel hitam ('B') yang berdekatan secara horizontal atau vertikal. Jika ada, fungsi akan mengembalikan `False`.
   - **Uniknya Angka dalam Setiap Baris dan Kolom**: Fungsi ini juga memeriksa apakah setiap angka dalam baris dan kolom adalah unik, dengan hanya mempertimbangkan sel yang diberi label sebagai putih ('W'). Ini dilakukan dengan membangun set untuk baris dan kolom, dan memastikan bahwa tidak ada angka yang diulang.

2. Fungsi ‘is_connected(solution, n)’

   Fungsi ini memeriksa apakah semua sel putih dalam solusi masih terhubung secara horizontal atau vertikal, yang merupakan syarat dari solusi yang valid. Ini diimplementasikan dengan algoritma flood fill menggunakan stack:

   - **Pencarian Titik Awal:** Fungsi ini mencari sel pertama yang tidak hitam untuk memulai penelusuran.
   - **Penelusuran:** Menggunakan pendekatan depth-first search (DFS) dengan bantuan stack, fungsi ini menjelajahi semua sel putih yang terhubung.
   - **Pemeriksaan Konektivitas:** Setelah penelusuran selesai, fungsi ini memeriksa apakah masih ada sel putih yang belum dikunjungi, yang menandakan adanya kelompok putih yang terpisah.

3. Fungsi ‘solve_hitori(grid, n, solution, row=0, col=0)’

   Ini adalah fungsi rekursif yang menerapkan algoritma backtracking untuk mencari solusi puzzle:

   - **Basis Rekursi:** Jika baris melebihi grid (menunjukkan semua sel telah diproses), fungsi memanggil is_connected untuk memastikan semua sel putih terhubung.
   - **Iterasi Sel:** Fungsi mencoba dua pilihan untuk setiap sel—menjadikannya putih atau hitam—dan bergerak ke sel berikutnya secara rekursif.
   - **Backtracking:** Jika pilihan pada suatu sel mengarah pada kegagalan, maka pilihan tersebut dibatalkan (sel kembali ke status belum ditentukan '.'), dan algoritma mencoba pilihan lain.

4. Fungsi ‘print_solution(solution, grid)’

   Fungsi ini digunakan untuk mencetak solusi yang ditemukan. Sel yang diberi label sebagai hitam ('B') dicetak sebagai 'X', sementara sel putih ('W') menampilkan angka asli dari grid masukan.

D. Analisis Kompleksitas Waktu

1. Kompleksitas Waktu
   - Backtracking (`‘solve_hitori’` Fungsi)

Jumlah operasi untuk memverifikasi setiap pilihan (putih atau hitam) untuk setiap sel adalah sekitar O(n) untuk memeriksa keunikan baris dan kolom (karena melibatkan iterasi melalui setiap segmen baris dan kolom hingga posisi saat ini).
Karena ada n^2 sel, dan setiap sel memiliki faktor percabangan 2, kompleksitas waktu kasus terburuk dapat mencapai $O(2n^{n\wedge 2} . n)$. Hal ini karena setiap panggilan rekursif dapat mencapai kedalaman n^2 tingkat (satu untuk setiap sel), dan untuk setiap tingkat, mungkin beriterasi hingga n kali untuk validasi.

- Memeriksa Sel Putih yang Terhubung (`'is_connected'` Fungsi):

  Setelah mengisi grid, fungsi ini memeriksa apakah semua sel putih terhubung menggunakan pendekatan pencarian pertama kedalaman (DFS). DFS sendiri berjalan dalam $O(n^2)$ karena berpotensi mengunjungi setiap sel sekali untuk memastikan semua sel putih terhubung.

2. Kompleksitas Ruang

   - Panggilan Rekursif

     Kedalaman rekursi bisa mencapai n^2 dalam kasus terburuk (jika tumpukan rekursif mengeksplorasi setiap sel secara individual). Setiap panggilan rekursif menggunakan ruang untuk mempertahankan keadaan grid (`solution`) dan variabel lokal lainnya, tetapi karena `solution` dimodifikasi di tempat, tidak diperlukan ruang tambahan yang proporsional dengan n^2 per panggilan rekursif.

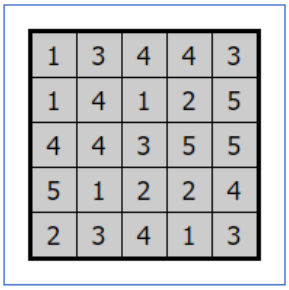
   - Penyimpanan `solution` dan `visited` Array

     Array `solution`, yang melacak keadaan grid, memerlukan ruang $O(n^2)$. Array `visited` dalam fungsi `is_connected` juga memerlukan ruang $O(n^2)$ untuk melacak sel mana yang telah dikunjungi selama DFS.

IV. Uji Coba dan Pembahasan

Berbagai kasus uji yang meliputi sampel dengan ukuran yang beragam telah digunakan untuk menguji program yang dikembangkan oleh penulis selama proses eksplorasi solusi dan penyusunan algoritma. Proses uji coba ini penting untuk memastikan bahwa solusi yang dibangun berfungsi dengan baik di berbagai skenario dan kondisi. Pengujian dilakukan secara berkesinambungan untuk mengevaluasi performa, keakuratan, dan keandalan program. Dalam laporan ini, akan dipresentasikan beberapa contoh kasus uji yang telah dijalankan beserta hasil yang diperoleh, memberikan wawasan tentang bagaimana program tersebut bertindak di bawah kondisi yang berbeda-beda.

1. Test Case 1 (5X5)

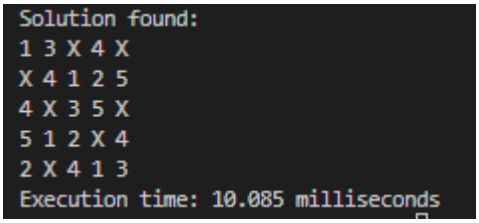


Test Case 1

Sumber: Dokumentasi pribadi

Hasil Uji Solusi Test Case 1

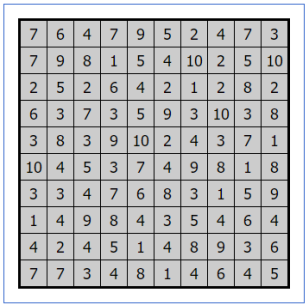
Sumber: Dokumentasi pribadi



Solusi Test Case 1 dari implementasi algoritma backtracking

Sumber: Dokumentasi pribadi

2. Test Case 2 (10X10)

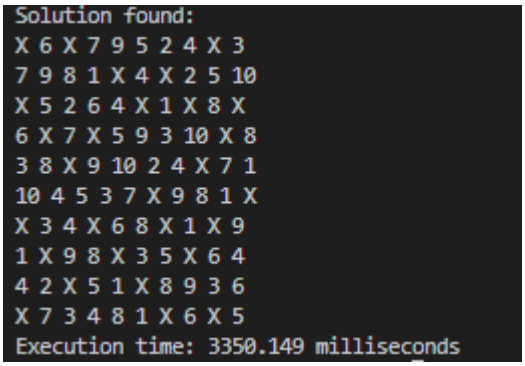


Test Case 2

Sumber: Dokumentasi pribadi

Hasil Uji Solusi Test Case 2

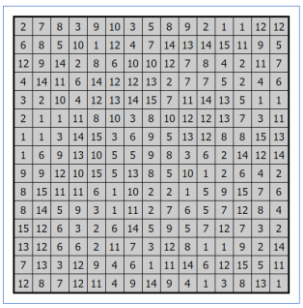
Sumber: Dokumentasi Pribadi



Solusi Test Case 2 dari implementasi algoritma backtracking

Sumber: Dokumentasi pribadi

3. Test Case 3 (15X15)

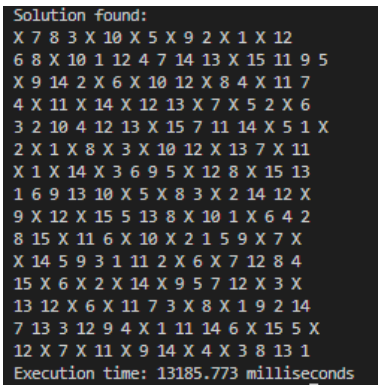


Test Case 3

Sumber: Dokumentasi Pribadi

Hasil Uji Solusi Test Case 3

Sumber: Dokumentasi Pribadi



Solusi Test Case 3 dari implementasi algoritma backtracking

Sumber: Dokumentasi pribadi

Hasil dari tiga tes yang dilakukan pada ukuran papan puzzle hitori yang berbeda menunjukkan bahwa penggunaan algoritma backtracking dalam menyelesaikan puzzle hitori memberikan hasil yang memuaskan dengan waktu yang proporsional terhadap ukuran puzzle. Semakin besar ukuran puzzle, waktu yang dibutuhkan untuk menyelesaikannya pun semakin lama. Oleh karena itu, pendekatan ini dapat dianggap cukup efektif, meskipun efektivitasnya menurun secara drastis pada puzzle berukuran di atas 15X15. Untuk puzzle ukuran 5X5, waktu eksekusi rata-rata adalah 15 milidetik. Sementara itu, pada ukuran 10X10, waktu eksekusi rata-rata adalah 5 detik, dan pada ukuran 15X15, waktu eksekusi rata-rata mencapai 15 menit.

V. KESIMPULAN

Algoritma backtracking terbukti efektif dalam menyelesaikan puzzle Hitori, memberikan solusi yang sistematis dan terstruktur di berbagai ukuran puzzle. Hasil uji coba menunjukkan bahwa waktu yang dibutuhkan untuk menyelesaikan puzzle meningkat seiring dengan bertambahnya ukuran papan, dengan puzzle berukuran kecil teratasi dengan cepat dan efisien, sementara puzzle berukuran besar memerlukan waktu yang jauh lebih lama. Efektivitas algoritma ini menurun pada puzzle berukuran di atas 15X15, yang menunjukkan perlunya optimasi lebih lanjut.

Untuk mengatasi keterbatasan ini, pengembangan lebih lanjut bisa mencakup penggunaan heuristik yang lebih canggih untuk memilih langkah yang paling menjanjikan, sehingga mengurangi jumlah backtracking yang diperlukan. Penggunaan paralelisme juga dapat menjadi strategi yang efektif, memungkinkan algoritma untuk memproses berbagai cabang secara bersamaan, sehingga mempercepat proses pencarian solusi. Selain itu, teknik pemotongan cabang (pruning) dapat diintegrasikan untuk memangkas pilihan yang tidak mengarah pada solusi, sehingga meningkatkan efisiensi.

Diharapkan dengan pengembangan dan optimisasi algoritma ini, akan semakin memperkuat kapasitas backtracking dalam menyelesaikan puzzle Hitori dan puzzle logika serupa lainnya, sehingga membuka peluang untuk

aplikasi yang lebih luas dalam bidang teka-teki dan pemecahan masalah komputasional.

PENUTUP

Dalam penutupan, penulis ingin mengungkapkan rasa syukur kepada Tuhan Yang Maha Esa, atas rahmat dan karunia-Nya yang telah memungkinkan penyelesaian makalah ini tentang Strategi Algoritma. Proses pembuatan makalah ini tidak hanya menantang, tetapi juga sangat memperkaya pengetahuan penulis dalam bidang yang sangat spesifik ini.

Penulis juga ingin mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Dr. Ir. Rinaldi Munir, M.T., dosen pembimbing kelas 01 Strategi Algoritma, yang telah memberikan penjelasan yang sangat jelas dan mendalam mengenai konsep-konsep Strategi Algoritma. Bapak Rinaldi Munir juga telah menyediakan berbagai sumber belajar yang menjadi acuan utama serta referensi pendukung yang sangat berguna dalam proses pembelajaran. Berkat bimbingan beliau, penulis dapat memahami materi dengan lebih luas dan aplikatif.

Selanjutnya, penghargaan juga ditujukan kepada semua pihak yang telah mendukung proses pembelajaran dan penulisan makalah ini, termasuk rekan-rekan sekelas yang telah memberikan umpan balik yang konstruktif dan sumber daya belajar yang telah digunakan sebagai referensi tambahan. Penulis berharap bahwa makalah ini tidak hanya menunjukkan pencapaian akademik, tetapi juga dapat memberikan kontribusi nyata bagi siapa saja yang tertarik dalam melakukan eksplorasi lebih lanjut mengenai penggunaan algoritma untuk menyelesaikan puzzle Hitori secara lebih efektif dan efisien.

LAMPIRAN

Tautan repositori:

https://github.com/mybajwk/Puzzle-Hitori-Solver

REFERENCES

[1] R. Munir. 2021. Algoritma Runut-balik (Backtracking) bagian 1, Informatika. Dilansir dari https://informatika.stei.itb.ac.id/~rinaldi.munir/. Diakses pada tanggal 11 Juni 2024.

[2] R. Munir. 2021. Algoritma Runut-balik (Backtracking) bagian 2, Informatika. Dilansir dari https://informatika.stei.itb.ac.id/~rinaldi.munir/. Diakses pada tanggal 11 Juni 2024.

[3] Z. J. B., "AI Project #1: Hitori," RIT Department of Computer Science. [Online]. Available: https://www.cs.rit.edu/~zjb/courses/ai/proj1.html. Accessed on: June 12, 2024.

[4] Nikoli Co., Ltd., "Hitori," Nikoli. [Online]. Available: https://www.nikoli.co.jp/en/puzzles/hitori/. Accessed on: June 12, 2024..

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

Enrique Yanuar (13522077)