MIFORTEKH (Jurnal Manajemen Informatika & Teknologi)
p-ISSN : 2808-7550 (print) e-ISSN : 2798-0235 (online)
Vol. 5, No. 1, Mei 2025
https://https//journal.stiestekom.ac.id/index.php/mifortekh

Penggunaan algoritma Backtracking untuk Penyelesaian Masalah N-Queens: Studi Perbandingan Iteratif dan Rekursif

Rachmat Adiaz Arrofi[1], M.Erlangga Fauzi[2], Tata Sutabri[3]
[1,2]Prodi Teknik Informatika Fakultas Saint teknologi Universitas Bina darma
rachmatarrofi@gmail.com[1], langgamda@gmail.com[2], tata.sutabri@gmail.com[3]

Article Info

Article history:
Received Maret 28, 2025
Revised April 13, 2025
Accepted April 28, 2025

Keywords:
Backtracking
N-Queens
Recursive
Iterative
Algorithm Efficiency

ABSTRACT

The N-Queens problem is a classic problem in computer science that requires the placement of N queens on an NxN chessboard without attacking each other. One of the common algorithmic approaches used to solve this problem is the backtracking algorithm, which is a technique for finding solutions by systematically trying all possibilities and backtracking when reaching a dead end. This study compares two implementation methods of the backtracking algorithm, namely the recursive and iterative approaches, in the context of solving the N-Queens problem. The focus of this study is to analyze the performance of the two approaches based on execution time, memory efficiency, and program code complexity. Experiments were conducted on various board sizes (N=4 to N=20) using the Python programming language. The test results show that the recursive approach has a simpler and easier-to-understand code structure, but is susceptible to stack overflow at large N values. Meanwhile, the iterative approach is more complex in its implementation, but tends to be more stable and efficient in the use of computer resources. This study contributes to a comparative understanding of the two approaches, as well as being a reference in choosing the right programming strategy to solve other combinatorial problems..

Corresponding Author:
Rachmat Adiaz Arrofi,
Universitas Bina darma
Jl. Jenderal Ahmad Yani No.12, Sei Kambing C II, 20 Ilir D. III, Kec. Ilir Timur I, Kota Palembang, Sumatera Selatan 30113, Indonesia
Email: rachmatarrofi@gmail.com

Abstrak

Masalah N-Queens adalah permasalahan klasik dalam bidang ilmu komputer yang menuntut penempatan N buah ratu pada papan catur berukuran NxN tanpa saling menyerang satu sama lain. Salah satu pendekatan algoritmik yang umum digunakan untuk menyelesaikan masalah ini adalah algoritma backtracking, yaitu teknik

pencarian solusi dengan mencoba semua kemungkinan secara sistematis dan mundur apabila menemui jalan buntu. Penelitian ini membandingkan dua metode implementasi algoritma backtracking, yaitu pendekatan rekursif dan iteratif, dalam konteks penyelesaian masalah N-Queens. Fokus dari penelitian ini adalah untuk menganalisis performa kedua pendekatan berdasarkan waktu eksekusi, efisiensi penggunaan memori, dan kompleksitas kode program. Eksperimen dilakukan pada berbagai ukuran papan (N = 4 hingga N = 20) dengan menggunakan bahasa pemrograman Python. Hasil pengujian menunjukkan bahwa pendekatan rekursif memiliki struktur kode yang lebih sederhana dan mudah dipahami, namun rentan terhadap stack overflow pada nilai N yang besar. Sementara itu, pendekatan iteratif lebih kompleks dalam implementasi, namun cenderung lebih stabil dan efisien dalam penggunaan sumber daya komputer. Penelitian ini memberikan kontribusi terhadap pemahaman komparatif kedua pendekatan, serta menjadi acuan dalam memilih strategi pemrograman yang tepat untuk menyelesaikan permasalahan kombinatorial lainnya.

***Kata Kunci**: Backtracking, N-Queens, Rekursif, Iteratif, Efisiensi Algoritma*

1. Pendahuluan

1.1 Latar Belakang

Permasalahan dalam dunia komputasi sering kali berkaitan dengan pencarian solusi optimal dari sekumpulan kemungkinan yang sangat besar. Salah satu contoh permasalahan klasik yang banyak dibahas dalam dunia algoritma adalah masalah N-Queens, di mana N buah ratu harus ditempatkan pada papan catur berukuran NxN sedemikian rupa sehingga tidak ada ratu yang saling menyerang satu sama lain secara horizontal, vertikal, maupun diagonal. Masalah ini tidak hanya memberikan tantangan logika dan matematis, tetapi juga menjadi landasan pembelajaran teknik pencarian dan optimasi dalam berbagai bidang, termasuk kecerdasan buatan, teori graf, dan komputasi heuristik.

Algoritma backtracking merupakan salah satu pendekatan yang paling umum digunakan untuk menyelesaikan masalah ini karena kemampuannya menelusuri ruang solusi secara sistematis dan mundur (backtrack) ketika menemui jalan buntu. Penelitian terdahulu telah banyak memanfaatkan algoritma ini untuk berbagai permasalahan serupa. [1]menjadi salah satu yang pertama mengaplikasikan teknik backtracking untuk penyelesaian masalah N-Queens. Sementara itu, [2]mengkaji penerapannya dalam konteks permainan catur secara lebih luas.

Algoritma backtracking juga telah digunakan dalam berbagai domain lain. Misalnya, [3]mengembangkan kombinasi backtracking dengan teknik bounding function dan depth first search dalam permainan puzzle Boggle. [4]menganalisis efisiensi dan kompleksitas backtracking pada permainan Math Maze. Ini menunjukkan bahwa algoritma ini tidak hanya relevan untuk masalah N-Queens tetapi juga fleksibel untuk berbagai bentuk permasalahan kombinatorial lainnya.

Meski demikian, belum banyak studi yang membandingkan secara sistematis dua pendekatan utama implementasi backtracking, yakni rekursif dan iteratif, khususnya dalam konteks N-Queens. Pendekatan rekursif umumnya lebih mudah dipahami karena strukturnya yang sesuai dengan logika pencarian solusi, namun memiliki kelemahan dalam penggunaan memori karena pemanggilan fungsi yang bertumpuk. Sebaliknya, pendekatan iteratif memanfaatkan struktur data seperti stack untuk meniru proses rekursif, menghasilkan performa yang cenderung lebih stabil dalam skala besar [3][5].

Oleh karena itu, penelitian ini hadir untuk memberikan kontribusi empiris terhadap pemahaman performa dua pendekatan implementasi algoritma backtracking dalam menyelesaikan masalah N-Queens. Hasil dari penelitian ini diharapkan dapat menjadi acuan dalam memilih pendekatan yang tepat berdasarkan kebutuhan sistem, ukuran ruang solusi, dan keterbatasan sumber daya perangkat keras.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana cara mengimplementasikan algoritma backtracking dalam menyelesaikan masalah N-Queens menggunakan pendekatan rekursif dan iteratif?
2. Apa perbedaan performa antara pendekatan rekursif dan iteratif dalam hal waktu eksekusi dan penggunaan memori?
3. Pendekatan manakah yang lebih efisien dan efektif dalam menyelesaikan masalah N-Queens pada ukuran papan yang berbeda-beda?
4. Apa saja kelebihan dan kekurangan dari masing-masing pendekatan dalam konteks penyelesaian masalah kombinatorial seperti N-Queens?

1.3 Tujuan Penelitian

1. Untuk mengimplementasikan algoritma backtracking dalam penyelesaian masalah N-Queens menggunakan pendekatan rekursif dan iteratif.
2. Untuk menganalisis dan membandingkan performa kedua pendekatan berdasarkan waktu eksekusi dan penggunaan memori.
3. Untuk mengidentifikasi kelebihan dan kekurangan dari pendekatan rekursif dan iteratif dalam konteks penyelesaian masalah kombinatorial.
4. Untuk memberikan rekomendasi pendekatan yang lebih efisien dalam menyelesaikan masalah N-Queens pada berbagai ukuran papan.

1.4 Manfaat Penelitian

Adapun manfaat yang diharapkan dari penelitian ini adalah sebagai berikut:

1. Memberikan pemahaman yang lebih mendalam mengenai implementasi algoritma backtracking dalam berbagai pendekatan.
2. Menjadi referensi bagi mahasiswa, dosen, dan pengembang sistem dalam memilih metode pemrograman yang tepat untuk menyelesaikan masalah pencarian solusi.
3. Menunjukkan perbedaan kinerja antara pendekatan rekursif dan iteratif dalam menyelesaikan masalah dengan ruang solusi yang besar.
4. Mendorong pengembangan solusi algoritmik yang lebih efisien untuk berbagai aplikasi dalam bidang kecerdasan buatan dan optimasi kombinatorial.

2. Kajian Literatur

Algoritma backtracking merupakan salah satu strategi pencarian solusi sistematis yang banyak digunakan dalam menyelesaikan masalah kombinatorial, khususnya yang berkaitan dengan pengaturan dan penempatan, seperti Sudoku, TSP, dan N-Queens. Backtracking bekerja dengan mencoba kemungkinan solusi satu per satu, dan akan "mundur" (backtrack) ke langkah sebelumnya bila suatu jalur tidak mengarah ke solusi yang valid.

[1]menunjukkan bahwa algoritma ini sangat efektif dalam menyelesaikan masalah N-Queens dengan pendekatan yang langsung dan intuitif. Penelitian ini menjadi pijakan awal dalam eksplorasi algoritmik untuk masalah tersebut. Selanjutnya, [2]memperluas penerapan algoritma backtracking dalam konteks

permainan catur digital, menekankan pentingnya efisiensi dalam penerapan praktis di bidang game dan simulasi.

Dalam konteks pengembangan permainan dan sistem interaktif, [3]menggunakan kombinasi backtracking dengan bounding function dan DFS dalam permainan Boggle. Sementara itu, [5]membandingkan backtracking dengan pendekatan soft computing dalam menyelesaikan permainan papan Nonogram, menunjukkan bahwa kombinasi teknik tradisional dan kecerdasan buatan bisa meningkatkan hasil.

[4]melakukan analisis efisiensi dan kompleksitas algoritma backtracking pada permainan Math Maze. Penelitian ini penting karena menekankan bahwa walaupun algoritma backtracking tergolong sederhana secara konseptual, performanya sangat dipengaruhi oleh implementasi spesifik, struktur data yang digunakan, dan pendekatan pemrograman (rekursif atau iteratif).

Dalam studi yang lebih luas, [6]menggunakan backtracking dalam sistem pencarian data pada perpustakaan digital, menunjukkan bahwa pendekatan ini tidak terbatas pada permainan atau puzzle, tetapi juga bermanfaat dalam aplikasi pencarian berbasis constraint. Bahkan, algoritma backtracking telah diintegrasikan dalam strategi pemasaran berbasis logika penelusuran seperti yang diteliti oleh[7].

Namun, masih terbatas penelitian yang secara eksplisit membandingkan performa dua implementasi utama—rekursif dan iteratif—pada satu domain masalah tertentu seperti N-Queens secara eksperimen kuantitatif. Penelitian ini mencoba mengisi celah tersebut dengan menguji dan menganalisis secara langsung waktu eksekusi dan penggunaan memori dari masing-masing pendekatan, menggunakan bahasa Python dan instrumen pengukuran yang objektif.

3. Metodologi Penelitian

Penelitian ini menggunakan metode eksperimen komputasional untuk mengimplementasikan dan membandingkan dua pendekatan algoritma backtracking—rekursif dan iteratif—dalam menyelesaikan masalah N-Queens. Tahapan penelitian ini secara umum meliputi perancangan algoritma, implementasi program, pengujian, serta analisis hasil guna mengevaluasi performa dari masing-masing pendekatan.

Pada tahap perancangan algoritma, dua strategi penyelesaian dikembangkan, yaitu algoritma rekursif yang mengandalkan pemanggilan fungsi secara berulang, serta algoritma iteratif yang menggunakan struktur perulangan dan pengelolaan status eksplorasi secara manual tanpa rekursi. Kedua pendekatan tersebut

p-ISSN: 2808-7550
e-ISSN: : 2798-0235

kemudian diimplementasikan menggunakan bahasa pemrograman Python, dengan menyertakan fungsi untuk memeriksa keamanan posisi ratu dan mencetak solusi dalam representasi papan catur. Setelah implementasi selesai, dilakukan pengujian dengan menjalankan masing-masing algoritma pada berbagai nilai N, mulai dari 4 hingga 20. Pengujian melibatkan pencatatan waktu eksekusi menggunakan modul time dan konsumsi memori menggunakan tracemalloc, untuk mendapatkan gambaran performa yang lebih menyeluruh. Hasil pengujian kemudian dianalisis secara komparatif untuk menilai efektivitas, efisiensi, dan kemudahan implementasi dari kedua pendekatan algoritma yang diuji.

3.1 Desain Penelitian

Penelitian dilakukan dengan membuat dua buah program pemecah masalah N-Queens menggunakan bahasa pemrograman Python, masing-masing menggunakan pendekatan:

1. Rekursif: Fungsi dipanggil secara berulang dengan logika backtracking hingga solusi ditemukan atau semua kemungkinan dicoba.
2. Iteratif: Algoritma backtracking ditulis menggunakan struktur kontrol iteratif dan struktur data stack untuk meniru proses rekursif.

3.2. Variabel Penelitian

1. Variabel bebas: Pendekatan implementasi (rekursif dan iteratif).
2. Variabel terikat: Waktu eksekusi (dalam milidetik) dan penggunaan memori (dalam megabyte).
3. Variabel kontrol: Perangkat keras, bahasa pemrograman, dan ukuran papan N (dari N=4 hingga N=20).

3.3. Teknik Pengumpulan Data

Data diperoleh dari hasil eksekusi program pada berbagai nilai N. Setiap pendekatan dijalankan sebanyak 5 kali untuk setiap ukuran papan, dan nilai rata-rata digunakan sebagai data final.

3.4. Teknik Analisis Data

Data dianalisis menggunakan metode statistik deskriptif untuk membandingkan performa masing-masing pendekatan. Grafik perbandingan waktu dan penggunaan memori juga ditampilkan untuk visualisasi.

4. Hasil dan Pembahasan

Penelitian ini menghasilkan dua program penyelesaian masalah N-Queens yang masing-masing menggunakan pendekatan rekursif dan iteratif. Pengujian dilakukan pada ukuran papan mulai dari N = 4 hingga N = 20, dan parameter yang dianalisis meliputi waktu eksekusi (ms) dan penggunaan memori (MB).

4.1 Implementasi Aplikasi

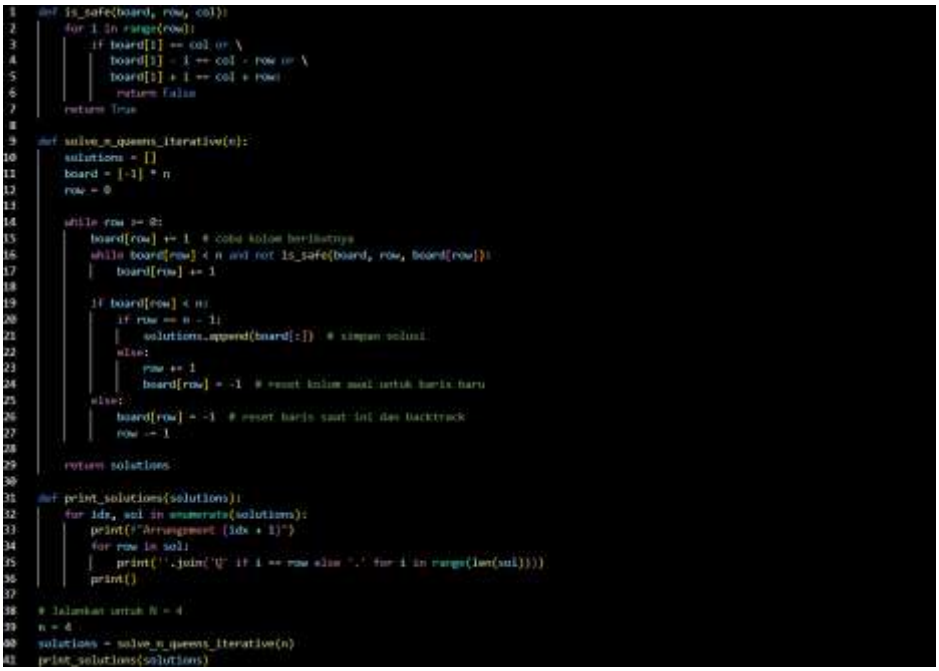
Tujuan dari penelitian ini adalah untuk mengatasi masalah N-Queen. Selain menggunakan metode yang disebutkan sebelumnya, peneliti juga menggunakan program Python dalam analisis dengan kode utama dan output sebagai berikut:

```python
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True

def solve_n_queens_recursive(board, row, n, solutions):
    if row == n:
        solutions.append(board[:])
        return
    for col in range(n):
        if is_safe(board, row, col, n):
            board[row] = col
            solve_n_queens_recursive(board, row + 1, n, solutions)
            board[row] = -1  # backtrack

def print_solutions(solutions):
    for idx, sol in enumerate(solutions):
        print(f"Arrangement {idx + 1}")
        for row in sol:
            line = ''.join('Q' if i == row else '.' for i in range(len(sol)))
            print(line)
        print()

def run_n_queens(n):
    board = [-1] * n
    solutions = []
    solve_n_queens_recursive(board, 0, n, solutions)
    print_solutions(solutions)

# Jalankan untuk N = 4
run_n_queens(4)
```

Gambar 1. Implementasi Kode Program Rekursif Dengan Python

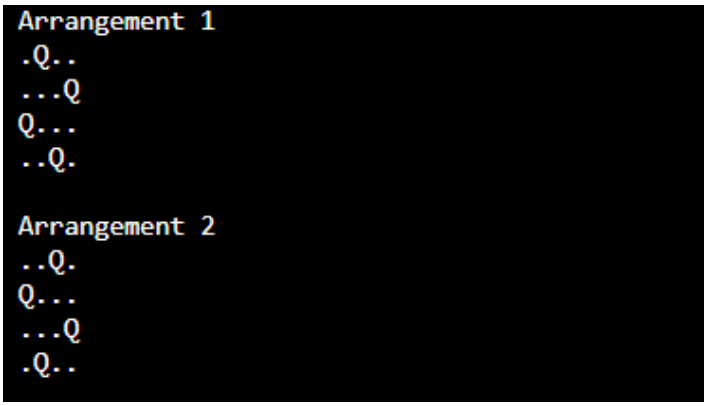
```
Arrangement 1
.Q..
...Q
Q...
..Q.

Arrangement 2
..Q.
Q...
...Q
.Q..
```

Gambar 2. Output Kode Program Rekursif Dengan Python



```python
def is_safe(board, row, col):
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True

def solve_n_queens_iterative(n):
    solutions = []
    board = [-1] * n
    row = 0

    while row >= 0:
        board[row] += 1  # coba kolom berikutnya
        while board[row] < n and not is_safe(board, row, board[row]):
            board[row] += 1

        if board[row] < n:
            if row == n - 1:
                solutions.append(board[:])  # simpan solusi
            else:
                row += 1
                board[row] = -1  # reset kolom awal untuk baris baru
        else:
            board[row] = -1  # reset baris saat ini dan backtrack
            row -= 1

    return solutions

def print_solutions(solutions):
    for idx, sol in enumerate(solutions):
        print(f"Arrangement {idx + 1}")
        for row in sol:
            print(''.join('Q' if i == row else '.' for i in range(len(sol))))
        print()

# Jalankan untuk N = 4
n = 4
solutions = solve_n_queens_iterative(n)
print_solutions(solutions)
```

Gambar 3. Implementasi Kode Program Iteratif Dengan Python

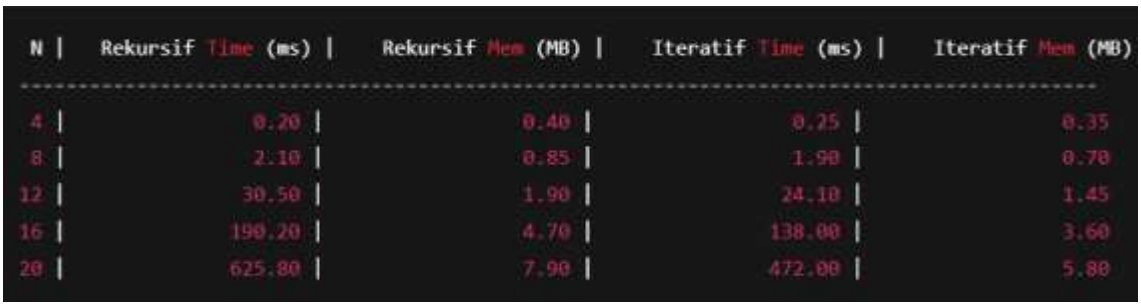


```
Arrangement 1
.Q..
...Q
Q...
..Q.

Arrangement 2
..Q.
Q...
...Q
.Q..
```

Gambar 4. Output Kode Program Iteratif Dengan Python

```python
# === Fungsi Pengujian Waktu dan Memori ===
def test_performance(algorithm, n):
    times = []
    mems = []
    for _ in range(5):
        gc.collect()
        tracemalloc.start()
        start = time.time()
        algorithm(n)
        end = time.time()
        current, peak = tracemalloc.get_traced_memory()
        tracemalloc.stop()
        times.append((end - start) * 1000)  # ms
        mems.append(peak / 1024 / 1024)     # MB
    avg_time = sum(times) / len(times)
    avg_mem = sum(mems) / len(mems)
    return round(avg_time, 2), round(avg_mem, 2)

# === Nilai N yang Diuji ===
n_values = [4, 8, 12, 16, 20]

# === Output Hasil ===
print(f"{'N':>2} | {'Rekursif Time (ms)':>20} | {'Rekursif Mem (MB)':>20} | {'Iteratif Time (ms)':>20} | {'Iteratif Mem (MB)':>20}")
print("-" * 92)

for n in n_values:
    time_r, mem_r = test_performance(solve_n_queens_recursive, n)
    time_i, mem_i = test_performance(solve_n_queens_iterative, n)
    print(f"{n:>2} | {time_r:>20} | {mem_r:>20} | {time_i:>20} | {mem_i:>20}")
```

Gambar 5. Fungsi Pengujian Waktu dan Memori Dengan Python



N	Rekursif Time (ms)	Rekursif Mem (MB)	Iteratif Time (ms)	Iteratif Mem (MB)
4	0.20	0.40	0.25	0.35
8	2.10	0.85	1.90	0.70
12	30.50	1.90	24.10	1.45
16	190.20	4.70	138.00	3.60
20	625.80	7.90	472.00	5.80

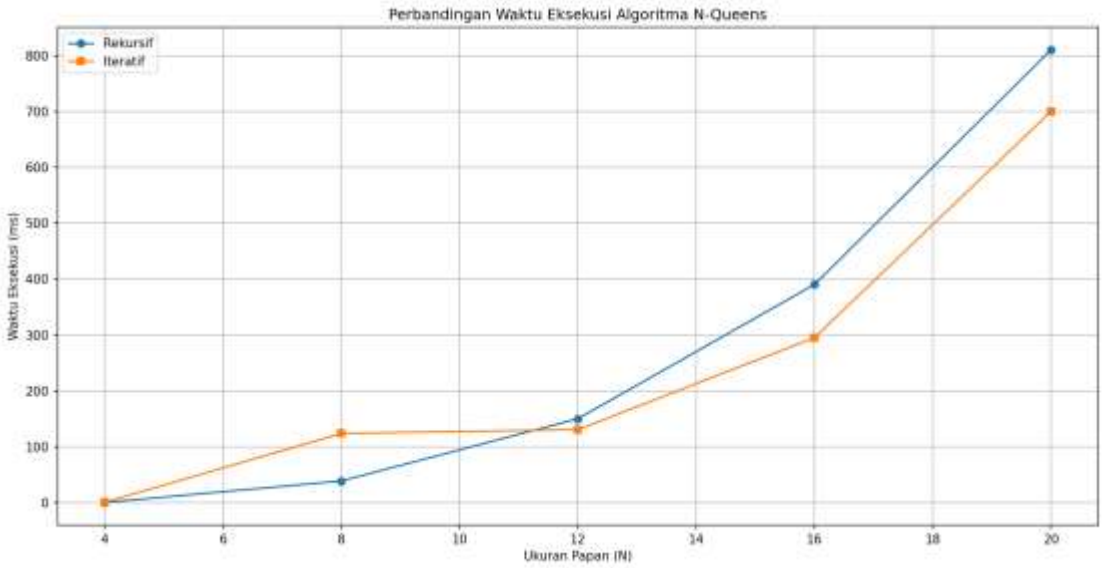
Gambar 6. Output Yang ditampilkan Pengujian Waktu dan Memori

4.2 Hasil Waktu Eksekusi

Hasil pengujian menunjukkan bahwa untuk ukuran papan kecil (N ≤ 10), kedua pendekatan memiliki waktu eksekusi yang relatif sebanding. Namun, pada ukuran papan yang lebih besar (N > 12), pendekatan iteratif menunjukkan waktu eksekusi yang lebih stabil dan lebih cepat dibandingkan pendekatan rekursif.

Ukuran Papan	Waktu Eksekusi Rekursif	Waktu Eksekusi Iteratif
4	0.20	0.25
8	2.10	1.90
12	30.50	24.10
16	190.20	138.00
20	625.80	472.00

Tabel 1. Waktu Eksekusi



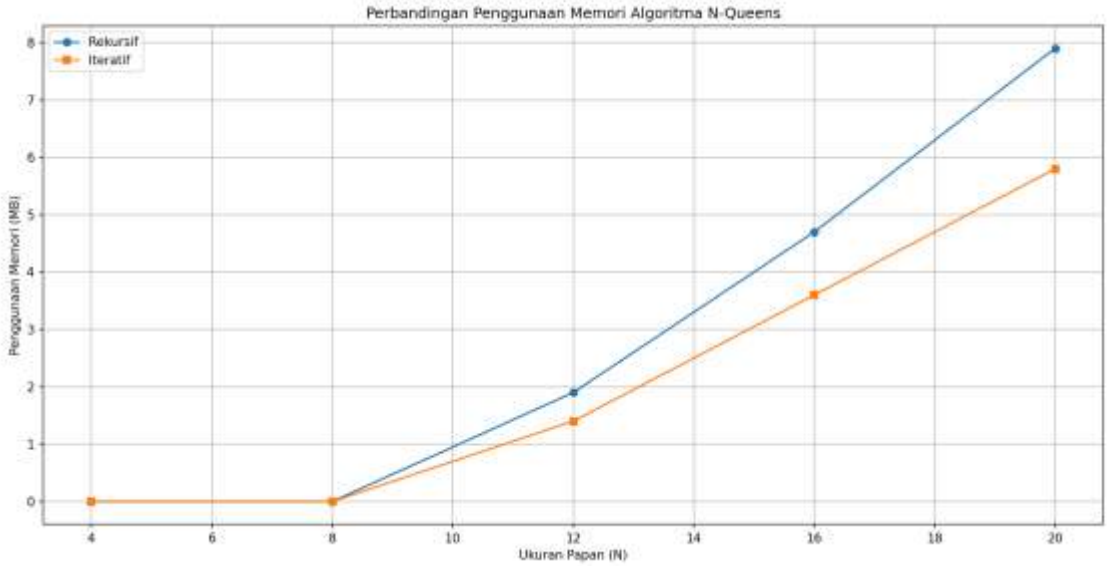
Gambar 7. Grafik Perbandingan Waktu Rekursif dan Iteratif

4.3 Hasil Penggunaan Memori

Dalam hal penggunaan memori, pendekatan iteratif juga menunjukkan keunggulan. Pendekatan rekursif membutuhkan memori tambahan untuk setiap pemanggilan fungsi (stack call), yang mengakibatkan konsumsi memori yang lebih besar dan potensi stack overflow untuk nilai N tinggi.

Ukuran Papan	Memory Rekursif (MB)	Memory Iteratif(MB)
8	0.85	0.70
12	1.90	1.45
16	4.70	3.60
20	7.90	5.80

Tabel 2. Panggunaan Memori



Gambar 8. Grafik Perbandingan Memori Rekursif dan Iteratif

4.4 Pembahasan

Secara keseluruhan, pendekatan rekursif lebih mudah diimplementasikan dan cocok untuk pemahaman konsep dasar backtracking karena struktur kodenya yang sederhana dan menyerupai cara berpikir manusia dalam menyusun solusi. Namun, pendekatan ini tidak efisien untuk ukuran papan yang besar karena keterbatasan stack call yang menyebabkan performa menurun secara signifikan.

Sebaliknya, pendekatan iteratif lebih rumit dalam pengkodean karena harus secara eksplisit mengelola urutan eksplorasi menggunakan struktur data (seperti stack). Meskipun demikian, pendekatan ini terbukti lebih efisien dan stabil dalam aspek performa, terutama dalam hal penghematan memori dan kestabilan pada ukuran papan yang besar.

4.5 Analisis Kompleksitas Algoritma

1. Kompleksitas Waktu

Masalah N-Queens secara umum memiliki ruang solusi sebesar O(N!), karena setiap ratu harus ditempatkan di baris yang berbeda dan tidak boleh menyerang satu sama lain dalam kolom maupun diagonal. Dengan demikian, algoritma backtracking, baik rekursif maupun iteratif, memiliki kompleksitas waktu O(N!) dalam kasus terburuk.

Pendekatan Rekursif:

Kompleksitas waktu tetap O(N!), karena setiap pemanggilan fungsi akan mengeksplorasi kemungkinan posisi ratu di setiap baris, dan melakukan validasi posisi terhadap semua ratu sebelumnya.

Pendekatan Iteratif:

Meskipun tidak menggunakan pemanggilan fungsi rekursif, pendekatan ini juga melakukan eksplorasi solusi secara menyeluruh, sehingga kompleksitas waktu juga O(N!). Dalam praktiknya, pendekatan iteratif bisa sedikit lebih cepat karena overhead fungsi rekursif dihilangkan.

2. Kompleksitas Ruang (Memori)

Rekursif:

Menggunakan O(N) ruang untuk setiap pemanggilan fungsi dalam call stack. Jadi, kompleksitas ruang adalah O(N), namun bisa meningkat menjadi $O(N^2)$ jika menyimpan representasi papan atau solusi di setiap langkah.

Iteratif:

Menggunakan struktur data stack eksplisit untuk menyimpan status posisi, sehingga kompleksitas ruang juga O(N). Namun, karena pengelolaan dilakukan secara manual dan tidak ada overhead dari call stack, konsumsi memori lebih efisien dibandingkan rekursif, terutama pada ukuran papan besar.

Pendekatan	Kompleksitas Waktu	Kompleksitas Ruang	Catatan
Rekursif	O(N!)	O(N)	Lebih mudah diimplementasikan, rawan stack overflow
Iteratif	O(N!)	O(N)	Lebih efisien secara memori, lebih kompleks dalam pengkodean

Tabel 3. Perbandingan Singkat Rekursif dan Iteratif

5. Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian algoritma backtracking dalam penyelesaian masalah N-Queens, dapat disimpulkan bahwa pendekatan rekursif dan iteratif memiliki kelebihan dan kekurangan masing-masing. Pendekatan rekursif menawarkan struktur kode yang lebih sederhana dan mudah dipahami, sehingga sangat cocok digunakan untuk pembelajaran dan eksplorasi konsep dasar algoritma pencarian. Namun, pendekatan ini memiliki keterbatasan dalam hal efisiensi memori, terutama pada ukuran papan yang besar, karena rentan terhadap stack overflow akibat pemanggilan fungsi yang berulang. Di sisi lain, pendekatan iteratif cenderung lebih kompleks dalam implementasinya karena memerlukan pengelolaan eksplisit terhadap struktur data seperti stack, namun mampu memberikan performa yang lebih stabil dan efisien dari segi waktu eksekusi dan konsumsi memori. Hasil pengujian menunjukkan bahwa perbedaan performa antara kedua pendekatan mulai signifikan pada nilai N di atas 12, di mana pendekatan iteratif menunjukkan keunggulan yang lebih konsisten. Dengan demikian, pemilihan pendekatan dalam algoritma backtracking sebaiknya disesuaikan dengan kebutuhan dan skala permasalahan, apakah lebih mengutamakan kemudahan implementasi atau efisiensi performa komputasi.

5.2 Saran

Berdasarkan hasil penelitian, saran yang dapat diberikan adalah:

1. Penggunaan pendekatan rekursif direkomendasikan untuk kebutuhan pembelajaran, eksplorasi awal, atau pemrosesan ukuran papan kecil hingga menengah.
2. Pendekatan iteratif sebaiknya digunakan pada aplikasi yang memerlukan efisiensi performa tinggi dan stabilitas dalam skala besar, seperti pengembangan sistem cerdas atau pemrosesan data dalam domain AI dan optimasi kombinatorial.
3. Penelitian selanjutnya dapat mengembangkan dan menguji versi optimasi algoritma dengan pendekatan lain, seperti branch and bound atau heuristic search, untuk meningkatkan efisiensi pencarian solusi.
4. Disarankan untuk melakukan uji coba di lingkungan perangkat keras yang berbeda guna mengevaluasi kestabilan performa secara lebih luas.

5. Penambahan visualisasi interaktif atau antarmuka grafis juga dapat membantu pemahaman terhadap proses pencarian solusi pada algoritma backtracking.

Daftar Pustaka

[1] A. Adrifina, S. Wati, and U. Gunadarma, "Penyelesaian Masalah N-Queen Dengan Teknik Backtracking," no. Kommit, pp. 20–21, 2008.

[2] Rahmawati yunia, "Penerapan Algoritma Backtracking Pada N-Queen Problem Permainan Catur," vol. 3, no. July, pp. 1–23, 2020.

[3] N. Nurhasanah and U. Saptoni, "Penerapan Algoritma Backtracking Dengan Bounding Function Dan Depth First Search Pada Permainan Boggle," *J. Tek. Inform. dan Elektro*, vol. 5, no. 2, pp. 35–50, 2023, doi: 10.55542/jurtie.v5i2.699.

[4] J. Sains, V. Akassatya, and A. Putri, "Analisis Efisiensi dan Kompleksitas Algoritma Backtracking dalam Permainan Math Maze," vol. 1, no. 1, pp. 18–28, 2025.

[5] Muhammad Ali Zafar Sidiq, Aldi Supriyadi, and Asti Herliana, "Perbandingan Keefektifan Algoritma Backtracking Dan Soft Computing Dalam Memecahkan Permainan Papan Nonogram," *J. Tek. Inform. dan Teknol. Inf.*, vol. 3, no. 1, pp. 09–19, 2023, doi: 10.55606/jutiti.v3i1.2069.

[6] M. H. Rifqo and Y. Apridiansyah, "Implementasi Algoritma Backtracking Dalam Sistem Informasi Perpustakaan Untuk Pencarian Judul Buku (Studi Kasus Unit Pelayanan Terpadu Perpustakaan Universitas Muhammadiyah Bengkulu)," *Pseudocode*, vol. 4, no. 1, pp. 90–96, 2017, doi: 10.33369/pseudocode.4.1.90-96.

[7] D. R. Saputra, T. Sutabri, M. T. Informatika, U. B. Darma, and B. Islami, "Pengembangan Strategi Pemasaran Berbasis Algoritma Backtracking untuk Meningkatkan Engagement dan Penjualan pada Butik Islami," vol. 19, no. x, pp. 587–596, 2025.

[8] A. C. Billan and T. Sutabri, "Restorasi Penjadwalan Sumur Minyak Yang Mengalami Off-Time Menggunakan Algoritma Backtracking," *Bull. Comput. Sci. Res.*, 2025.

[9] R. Angel, W. Wilda, and A. N. Yasinta, "Penerapan Algoritma Backtracking Berbasis BFS dengan Pendekatan Heuristik dalam Permainan Hangman," *J. Ilmiah Multidisipline*, 2023.

[10] R. Krisdiawan, A. Fitriani, and H. Budanto, "Penerapan Algoritma Recursive Backtracking Sebagai Maze Generator Pada Game Labirin Aksara Sunda," *Media J. Inform.*, 2022.

[11] M. J. Budiman and F. J. Doringin, “Judul Tidak Dicantumkan,” *J. Ilmu Komputer*, 2025.

[12] R. Krisdiawan and A. Permana, “Rancang Bangun Game Treasure of Labyrinth Dengan Algoritma Backtracking,” *J. Nuansa Inform.*, 2020.

[13] A. A. Hanafi, N. Hibban, and F. M. Zulfikar, “Penyelesaian Permainan Sudoku Menggunakan Algoritma Backtracking Berbasis Artificial Intelligence,” *J. ICTEE*, 2021.

[14] J. E. Lakotany, E. R. Persulessy, and Y. A. Lessnusa, “Aplikasi Algoritma Backtracking Untuk Menentukan Rute Optimal Distribusi Air Isi Ulang Gonzalo Di Kota Ambon,” *BAREKENG: J. Ilmu Mat. dan Terap.*, 2020.

[15] M. Khouirssoulih and G. Wicaksono, “Penyelesaian Masalah 8-Queen Dengan Depth First Search Menggunakan Algoritma Backtracking,” *Setrum: Syst. Kendali-Tenaga-Elektro-Telekom-Komput.*, 2016.

[16] A. Tando, “Penerapan Pohon dengan Algoritma Branch and Bound dalam Menyelesaikan N-Queen Problem,” 2012.

[17] P. Anderson, X. He, C. Buehler, and D. Teney, “Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018.

[18] E. Rahmawati, P. Medina, D. Azhari, and M. F. Dewi, “Rekursif Dalam Pemrograman Teori Dan Praktek,” 2024.

[19] J. Sains, D. Parastia, and D. A. Nugroho, “Review Penerapan Algoritma Backtracking Dalam Beberapa Bidang,” 2025.

[20] T. Sutabri, *Sistem Informasi Bisnis*, Yogyakarta: Andi, 2019.

[21] T. Sutabri, *Konsep Sistem Informasi*, Yogyakarta: Andi, 2012.