Perbandingan Efektivitas Algoritma Brute Force dan Algoritma Backtracking dalam Persoalan N-Ratu

Muhammad Zaydan Athallah - 13521104
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13521104@std.stei.itb.ac.id

***Abstract*—Penyelesaian persoalan N-Ratu merupakan salah satu tantangan dalam bidang pemrograman dan kecerdasan buatan. Dalam penelitian ini, dilakukan perbandingan efektivitas antara algoritma Brute Force dan algoritma Backtracking dengan komputer modern dalam penyelesaian persoalan N-Ratu. Algoritma Brute Force mencoba semua kemungkinan solusi secara sistematis, sementara algoritma Backtracking menggunakan pendekatan rekursif dengan membatalkan langkah-langkah yang tidak valid. Pengujian dilakukan dengan memvariasikan ukuran papan catur dan jumlah ratu yang harus ditempatkan. Hasil pengujian menunjukkan bahwa algoritma Backtracking secara konsisten menghasilkan solusi dengan waktu yang lebih efisien dibandingkan dengan algoritma Brute Force. Selain itu, analisis juga dilakukan terhadap kompleksitas waktu dan ruang kedua algoritma. Penelitian ini memberikan wawasan yang berguna dalam pemilihan algoritma yang tepat untuk penyelesaian persoalan N-Ratu.**

Keywords—N-Ratu, algoritma Brute-Force, algoritma Backtracking, efektivitas, kompleksitas waktu, kompleksitas ruang, N-Queen.

I. Pendahuluan

Persoalan N-Ratu, atau yang dikenal juga dengan sebutan "The N-Queens Problem", adalah sebuah permasalahan yang terkenal dalam bidang matematika dan pemrograman. Persoalan ini melibatkan penempatan N buah ratu pada papan catur berukuran N x N sedemikian rupa sehingga tidak ada ratu yang saling serang dalam satu baris, satu kolom, maupun diagonal.

Persoalan N-Ratu memiliki signifikansi dalam pemrograman dan kecerdasan buatan karena membutuhkan pemikiran strategis dan pendekatan algoritma yang efektif untuk menemukan solusinya. Meskipun terlihat sederhana, persoalan ini memiliki kompleksitas yang meningkat seiring bertambahnya nilai N. Misalnya, pada papan catur 8x8 (N = 8), terdapat 92 solusi unik yang memenuhi aturan.

Penyelesaian persoalan N-Ratu menjadi penting karena memiliki aplikasi praktis dalam desain dan pengaturan sistem, seperti penjadwalan, tata letak papan sirkuit, dan optimisasi kombinatorial. Selain itu, persoalan ini juga menjadi subjek penelitian yang menarik dalam analisis algoritma dan pemecahan masalah.

Dalam upaya menyelesaikan persoalan N-Ratu, berbagai pendekatan algoritma telah dikembangkan. Dua pendekatan yang umum digunakan adalah algoritma Brute Force dan algoritma Backtracking. Algoritma Brute Force mencoba semua kemungkinan solusi secara sistematis, sedangkan algoritma Backtracking menggunakan pendekatan rekursif dengan membatalkan langkah-langkah yang tidak valid.

Dalam penelitian ini, akan dibandingkan efektivitas kedua algoritma tersebut dalam menyelesaikan persoalan N-Ratu. Tujuan dari penelitian ini adalah untuk mengevaluasi dan membandingkan kinerja keduanya dalam hal waktu eksekusi dan penggunaan sumber daya. Dengan pemahaman yang lebih baik tentang kelebihan dan kekurangan masing-masing algoritma, kita dapat memilih pendekatan yang paling efisien dalam menyelesaikan persoalan N-Ratu dan masalah serupa.

II. Landasan Teori

A. Algoritma Brute-Force

Algoritma *brute-force* adalah pendekatan yang lempang (straightforward) dalam memecahkan suatu persoalan, dimana pendekatan tersebut didasarkan pada pernyataan pada persoalan (problem statement) dan definisi konsep yang terlibat. Ketika menggunakan algoritma *brute-force*, langkah-langkah yang diperlukan untuk mencari solusi ditentukan dengan jelas dan tidak melibatkan strategi atau optimisasi yang kompleks.

Dalam algoritma *brute-force*, semua kemungkinan solusi yang memenuhi kriteria persoalan akan dipertimbangkan secara sistematis. Algoritma akan mencoba semua kemungkinan solusi secara berurutan atau melalui kombinasi tertentu untuk mencari solusi yang valid. Pendekatan ini sering kali melibatkan penggunaan perulangan dan pengujian untuk memeriksa apakah solusi yang diuji memenuhi kriteria yang ditetapkan dalam persoalan.

Kelebihan utama dari algoritma *brute-force* adalah kesederhanaannya. Karena pendekatannya yang lurus ke depan, algoritma ini mudah dipahami dan diimplementasikan. Selain itu, algoritma *brute-force* juga berguna dalam memverifikasi kebenaran solusi, terutama ketika solusi lain yang lebih efisien belum diketahui atau belum dapat ditemukan.

Namun, kelemahan utama dari algoritma *brute-force* adalah kompleksitas waktu yang tinggi. Karena mencoba semua kemungkinan solusi, algoritma ini dapat memerlukan waktu eksekusi yang lama, terutama saat menangani masalah dengan ukuran input yang besar. Selain itu, algoritma *brute-force* sering kali tidak efisien dalam menghadapi masalah dengan ruang pencarian yang besar atau kombinatorial yang kompleks.

Meskipun algoritma *brute-force* dapat menghasilkan solusi yang benar, terkadang tidak optimal. Dalam beberapa kasus, ada pendekatan yang lebih cerdas atau algoritma yang dapat mencapai solusi yang lebih efisien atau optimal dengan memanfaatkan sifat atau struktur khusus dari persoalan yang dihadapi.

Dalam penggunaannya, algoritma *brute-force* sering digunakan sebagai langkah awal dalam memecahkan persoalan yang kompleks. Pada tahap ini, algoritma *brute-force* dapat digunakan untuk menguji kebenaran solusi atau sebagai dasar untuk mengembangkan pendekatan yang lebih canggih.

Algoritma Brute-Force memiliki karakteristik sebagai berikut:

- Tidak cerdas dan tidak efisien: Algoritma Brute-Force memerlukan banyak komputasi dan waktu yang lama untuk menyelesaikannya. Pendekatan ini didasarkan pada upaya atau kekuatan secara langsung tanpa mempertimbangkan metode yang lebih cerdas atau efisien. Nama "force" pada algoritma ini menunjukkan pendekatan yang lebih mengandalkan usaha daripada kecerdasan.

- Algoritma naif: Algoritma Brute-Force sering disebut sebagai algoritma naif karena sederhana dan tidak memanfaatkan optimasi yang lebih kompleks. Pendekatan ini melibatkan pemeriksaan langsung terhadap semua kemungkinan solusi tanpa mempertimbangkan informasi konteks atau struktur masalah yang dapat mempercepat pencarian.

- Cocok untuk persoalan dengan ukuran kecil: Algoritma Brute-Force lebih cocok untuk persoalan dengan ukuran kecil karena sifatnya yang sederhana dan mudah diimplementasikan. Pada masalah dengan ruang pencarian yang kecil, algoritma ini dapat memberikan solusi yang akurat meskipun lambat.

- Digunakan sebagai dasar perbandingan: Algoritma Brute-Force sering digunakan sebagai dasar perbandingan dengan algoritma yang lebih canggih dan dioptimalkan. Dengan membandingkan hasil dari algoritma Brute-Force dengan solusi yang diberikan oleh algoritma lain, dapat dievaluasi keunggulan dari pendekatan yang lebih cerdas tersebut.

- Dapat digunakan untuk hampir semua persoalan: Algoritma Brute-Force dapat diterapkan pada hampir semua persoalan. Meskipun tidak efisien, algoritma ini dapat menyelesaikan masalah dengan cara mencoba semua kemungkinan solusi secara sistematis. Bahkan, ada beberapa persoalan yang hanya dapat diselesaikan dengan pendekatan brute-force, ketika tidak ada metode lain yang lebih efisien yang diketahui.

Secara keseluruhan, algoritma Brute-Force adalah pendekatan yang sederhana dan langsung untuk menyelesaikan masalah, meskipun dengan komputasi yang intensif. Meskipun kurang efisien, algoritma ini tetap menjadi pilihan ketika persoalan memiliki ukuran kecil atau tidak ada pendekatan yang lebih cerdas yang tersedia.

B. Algoritma Backtracking

Backtracking adalah sebuah metode pemecahan masalah yang kuat, terstruktur, dan sistematis, yang digunakan untuk memecahkan berbagai jenis persoalan, baik itu persoalan optimasi maupun non-optimasi. Metode ini melibatkan pencarian solusi secara incremental, dimulai dari titik awal dan secara bertahap membangun solusi secara rekursif dengan melakukan pemilihan langkah yang memungkinkan. Jika langkah yang dipilih tidak mengarah ke solusi, maka langkah tersebut akan dibatalkan (backtracked) dan dilakukan pemilihan langkah alternatif yang mungkin mengarah ke solusi.

Algoritma backtracking adalah sebuah perbaikan dari pendekatan exhaustive search yang memungkinkan eksplorasi solusi secara efisien. Pada pendekatan exhaustive search, semua kemungkinan solusi diperiksa dan dievaluasi satu per satu. Namun, dalam algoritma backtracking, hanya pilihan-pilihan yang mengarah ke solusi yang akan dieksplorasi, sedangkan pilihan-pilihan yang tidak mengarah ke solusi akan diabaikan. Dengan kata lain, algoritma backtracking mencoba setiap kemungkinan langkah ke depan, namun secara cerdas memotong atau memangkas (pruning) percabangan yang tidak relevan atau tidak memenuhi kriteria.

Algoritma backtracking sering digunakan dalam persoalan yang melibatkan pemilihan atau penempatan objek, pengaturan jadwal, pencarian lintasan atau rute, dan banyak lagi. Contohnya, dalam persoalan permainan seperti Sudoku atau N-Ratu, algoritma backtracking dapat digunakan untuk mencari solusi yang memenuhi aturan permainan dengan mempertimbangkan langkah-langkah yang memungkinkan dan mengabaikan langkah-langkah yang tidak valid.

Algoritma backtracking pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Kemudian, R.J. Walker, Golomb, dan Baumert memberikan penjelasan umum tentang algoritma backtracking. Metode ini terus dikembangkan dan diterapkan dalam berbagai bidang, termasuk kecerdasan buatan, optimasi kombinatorial, dan rekayasa perangkat lunak.

Properti umum dari algoritma runut-balik adalah sebagai berikut:

- Solusi persoalan: Solusi dalam algoritma runut-balik dinyatakan sebagai vektor dengan n-tuple, yang disimbolkan sebagai X = (x1, x2, ..., xn), di mana setiap elemen xi berasal dari himpunan Si. Umumnya, himpunan Si memiliki nilai yang sama untuk setiap komponen xi. Sebagai contoh, dalam persoalan 1/0 knapsack, himpunan Si adalah {0, 1}, dan setiap xi dapat bernilai 0 atau 1.

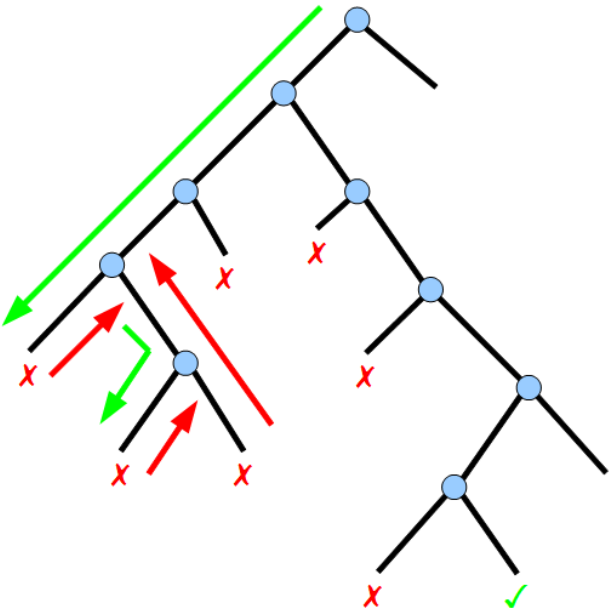
- Fungsi pembangkit nilai xk: Fungsi T(), yang dinyatakan sebagai predikat, digunakan untuk membangkitkan nilai untuk komponen xk dalam vektor solusi. Fungsi ini tergantung pada nilai-nilai x sebelumnya (x1, x2, ..., xk-1).

- Fungsi pembatas (bounding function): Fungsi B(x1, x2, ..., xk), yang juga dinyatakan sebagai predikat, digunakan untuk menentukan apakah kombinasi (x1, x2, ..., xk) memenuhi kendala (constraints). Fungsi ini menghasilkan nilai true jika kombinasi tersebut mengarah ke solusi, yaitu tidak melanggar batasan. Jika predikat bernilai true, pembangkitan nilai untuk xk+1 dilanjutkan. Namun, jika predikat bernilai false, kombinasi (x1, x2, ..., xk) dibuang.

Prinsip pencarian solusi menggunakan algoritma Runut-balik didasarkan pada pembangkitan simpul-simpul status secara bertahap, dengan tujuan untuk menciptakan jalur dari akar hingga daun. Aturan yang digunakan dalam pembangkitan simpul adalah mengikuti urutan depth-first (DFS), di mana simpul-simpul baru dibangkitkan dari simpul yang sedang diperluas.

Simpul-simpul yang telah dihasilkan dalam algoritma backtracking dikenal sebagai simpul hidup atau live node. Saat algoritma sedang melakukan ekspansi pada sebuah simpul hidup, simpul tersebut disebut simpul-E atau expand-node. Setiap kali simpul-E diperluas, jalur solusi yang sedang dibentuk menjadi lebih panjang. Namun, jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka simpul-E tersebut dianggap "mati" dan berubah menjadi simpul mati atau dead node.

Dalam algoritma backtracking, fungsi pembatas (bounding function) digunakan untuk mematikan simpul-E yang tidak mengarah ke solusi. Ketika sebuah simpul mati terdeteksi, yang berarti tidak mungkin ditemukan solusi dari simpul tersebut, simpul-simpul anaknya juga dipangkas (pruning) secara efisien. Jika proses pembentukan jalur mencapai simpul mati, algoritma akan melakukan backtracking ke simpul pada tingkat di atasnya, kemudian melanjutkan dengan menghasilkan simpul anak lainnya. Simpul anak baru ini kemudian menjadi simpul-E baru yang dieksplorasi. Proses pencarian akan berlanjut sampai mencapai simpul tujuan (goal node), di mana pencarian dihentikan.



Gambar 1. Sumber: http://www.w3.org/2011/Talks/01-14-steven-phenotype/

C. *Persoalan N-Ratu*

Persoalan N-ratu (N-Queens Problem) adalah sebuah persoalan yang terkait dengan penempatan ratu-ratu pada papan catur berukuran N×N sedemikian hingga tidak ada dua ratu yang saling menyerang satu sama lain. Dalam persoalan ini, ratu dapat menyerang secara horizontal, vertikal, dan diagonal. Tujuan dari persoalan N-ratu adalah menempatkan N ratu pada papan catur sehingga tidak ada dua ratu yang saling menyerang.

Persoalan N-ratu merupakan salah satu persoalan klasik dalam bidang teori graf dan pemrograman komputer. Meskipun tampak sederhana, persoalan ini memiliki tingkat kesulitan yang signifikan tergantung pada ukuran papan catur (N) yang diberikan. Ketika N semakin besar, jumlah kemungkinan penempatan ratu juga meningkat secara eksponensial, sehingga menyelesaikan persoalan ini menjadi lebih sulit.

Penyelesaian persoalan N-ratu melibatkan pencarian solusi yang memenuhi semua kriteria, yaitu penempatan N ratu pada papan catur dengan aturan bahwa tidak ada dua ratu yang saling menyerang. Salah satu pendekatan umum yang digunakan dalam menyelesaikan persoalan ini adalah dengan menggunakan algoritma backtracking. Dalam algoritma backtracking, solusi dihasilkan secara bertahap dengan mencoba semua kemungkinan penempatan ratu pada setiap baris papan catur, dan melakukan pemangkasan (pruning) ketika ditemukan bahwa penempatan tidak memenuhi kriteria.

Persoalan N-ratu memiliki aplikasi yang luas dalam bidang komputasi, seperti kecerdasan buatan, pemecahan masalah kombinatorial, dan optimisasi. Selain itu, persoalan ini juga digunakan sebagai tantangan dalam pengembangan algoritma dan pemrograman komputer, karena membutuhkan pemikiran strategis dan pemilihan langkah yang tepat untuk mencapai solusi yang valid.

Teka-teki ini pertama kali diajukan pada tahun 1848 oleh pemain catur bernama Max Bezzel, dan selama bertahun-tahun, banyak matematikawan, termasuk Gauss, telah bekerja pada teka-teki ini dan versinya yang lebih umum, yaitu persoalan N-ratu. Solusi pertama diberikan oleh Franz Nauck pada tahun 1850. Nauck juga memperluas teka-teki ini menjadi persoalan n-ratu (pada papan catur berukuran N×N - papan catur ukuran sembarang).

Untuk menyelesaikan persoalan ini, terdapat beberapa pendekatan yang dapat digunakan, yaitu:

1. Brute Force 1: Pendekatan ini melibatkan mencoba semua kemungkinan solusi untuk penempatan delapan buah ratu pada petak-petak papan catur. Penempatan ratu dilakukan secara acak. Namun, jumlah kemungkinan solusi yang perlu dievaluasi sangat besar, yaitu sebanyak C(64, 8) = 4.426.165.368.

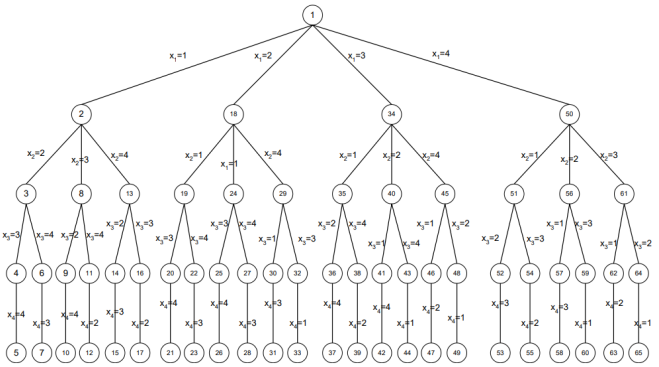
2. Brute Force 2: Pendekatan ini melibatkan meletakkan masing-masing ratu pada setiap baris yang berbeda. Untuk setiap baris, ratu dicoba ditempatkan mulai dari kolom 1, 2, ..., 8. Dengan menggunakan pendekatan ini, jumlah kemungkinan solusi yang perlu diperiksa berkurang menjadi 8^8 = 16.777.216.

3. Brute Force 3: Pendekatan ini menggambarkan solusi sebagai sebuah N-tuple, X = (x1, x2, ..., xn), di mana vektor solusi merupakan permutasi bilangan 1 hingga N. Pada kasus ini, dengan N=8, terdapat P(8,8) = 40.320 kemungkinan solusi yang perlu dievaluasi.

4. Backtracking: Algoritma dengan metode backtracking memperbaiki pendekatan Brute Force 3. Ruang solusi dalam pendekatan ini terdiri dari permutasi angka 1 hingga N. Setiap permutasi direpresentasikan sebagai lintasan dari akar ke daun dalam sebuah pohon. Sisi-sisi pada pohon ini diberi label nilai xi. Dengan menggunakan backtracking, algoritma dapat memotong pencarian yang tidak memenuhi kendala atau constraints, sehingga mengurangi jumlah solusi yang harus dievaluasi.

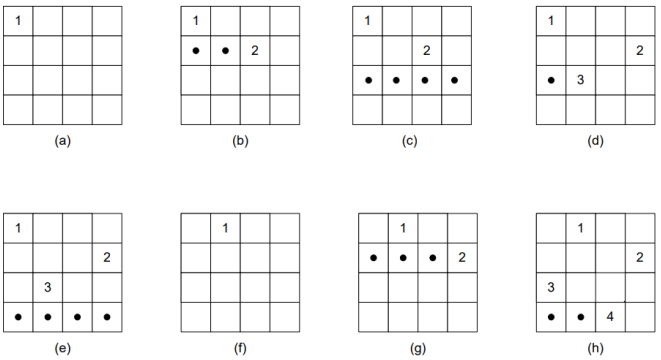
Dengan menggunakan salah satu dari pendekatan di atas, persoalan penempatan delapan ratu pada papan catur dapat diselesaikan. Namun, perlu diperhatikan bahwa pendekatan Brute Force 1 memiliki kompleksitas yang sangat tinggi dan tidak efisien untuk digunakan pada persoalan dengan skala yang lebih besar. Pendekatan Backtracking umumnya lebih efisien dalam menyelesaikan persoalan seperti ini.

Contoh Pohon ruang-status persoalan 4-Ratu :



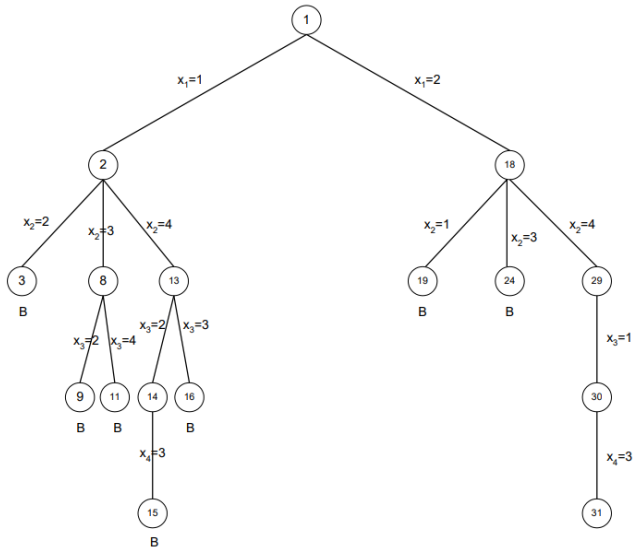
Gambar 2. Sumber : Algoritma Runut-balik (Backtracking)

Contoh solusi runut-balik persoalan 4-Ratu :



Gambar 3. Sumber : Algoritma Runut-balik (Backtracking)

Pohon ruang status persoalan 4-Ratu yang dibentuk selama pencarian :



Gambar 4. Sumber : Algoritma Runut-balik (Backtracking)

Simbol "B" di bawah simpul menunjukkan bahwa simpul tersebut telah dieliminasi (dibunuh) oleh fungsi pembatas (bounding function).

III. Implementasi

Dalam persoalan N-Ratu, kita harus menempatkan N buah ratu pada papan catur berukuran N×N sedemikian rupa sehingga tidak ada dua ratu atau lebih yang berada pada baris, kolom, atau diagonal yang sama. Untuk menyelesaikan persoalan ini, kita dapat menggunakan dua pendekatan yang berbeda, yaitu algoritma brute force dan algoritma backtracking.

D. Persoalan N-ratu Algoritma Brute-Force

Implementasi algoritma brute force mencoba semua kemungkinan penempatan ratu secara sistematis. Dalam hal ini, kita akan menghasilkan semua kemungkinan penempatan ratu pada setiap kolom secara berurutan. Setiap kali kita menempatkan seorang ratu, kita periksa apakah penempatan tersebut valid, yaitu tidak ada ratu lain yang saling menyerang. Jika penempatan tersebut valid, kita lanjutkan ke kolom berikutnya. Jika tidak valid, kita mencoba penempatan lain untuk ratu pada kolom saat ini. Algoritma brute force akan terus mencoba semua kemungkinan hingga menemukan solusi yang valid atau semua kemungkinan sudah dicoba.

Berikut adalah contoh implementasi algoritma brute force dalam Python :

```
def is_valid(board):
    # Check if the current board configuration is valid
    n = len(board)
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == j - i:
                return False
    return True

def solve_n_queens(n):
    solutions = []
    board = [0] * n  # Initialize the board

    def generate_configs(row):
        if row == n:
            if is_valid(board):
                solutions.append(board[:])
        else:
            for col in range(n):
                board[row] = col
                generate_configs(row + 1)

    generate_configs(0)
    return solutions
```

E. Persoalan N-ratu Algoritma Brute-Force

Implementasi algoritma backtracking menggunakan pendekatan rekursif untuk mencoba semua kemungkinan langkah dan memangkas percobaan yang tidak memenuhi aturan. Algoritma ini mencoba menempatkan seorang ratu pada setiap baris secara berurutan, dan setiap kali kita menempatkan seorang ratu, kita memeriksa apakah penempatan tersebut valid. Jika valid, kita melanjutkan untuk menempatkan ratu berikutnya pada baris selanjutnya. Jika tidak valid, kita melakukan backtracking atau membatalkan langkah sebelumnya dan mencoba penempatan yang berbeda.

Berikut adalah contoh implementasi algoritma backtracking dalam Python :

```
def is_valid(board, row, col):
    # Check if a queen can be placed at the given row and column
    for i in range(row):
        if board[i] == col or \
           board[i] == col - (row - i) or \
           board[i] == col + (row - i):
            return False
    return True

def solve_n_queens(n):
    solutions = []
    board = [-1] * n  # Initialize the board with -1 values

    def backtrack(row):
        if row == n:
```

```
            # Found a valid solution
            solutions.append(board[:])
        else:
            for col in range(n):
                if is_valid(board, row, col):
                    board[row] = col
                    backtrack(row + 1)
                    board[row] = -1
    backtrack(0)
    return solutions
```

Kedua algoritma, baik brute force maupun backtracking, dapat digunakan untuk menyelesaikan persoalan N-Ratu. Namun, algoritma backtracking memiliki keunggulan dalam melakukan pemotongan (pruning) percobaan yang tidak valid, sehingga dapat lebih efisien untuk ukuran papan catur yang besar. Algoritma brute force, di sisi lain, mencoba semua kemungkinan secara eksplisit tanpa melakukan pemotongan, sehingga dapat digunakan untuk ukuran papan catur yang lebih kecil.

IV. PENGUJIAN

Pengujian ini dilakukan dengan menggunakan berbagai ukuran papan catur (N) dan mencatat waktu yang dibutuhkan oleh masing-masing algoritma untuk menemukan solusi valid. Dengan demikian, dapat dievaluasi kinerja dan efisiensi kedua algoritma dalam menyelesaikan persoalan tersebut. Selain itu, pengujian ini juga akan membantu dalam memahami perbedaan kinerja antara kedua algoritma dan mempertimbangkan kelebihan serta kelemahan masing-masing dalam konteks yang lebih praktis.

F. Persoalan N-Ratu dengan (N=4)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
N                           : 4
Total number of solutions   : 2
Execution time              : 0.0 miliseconds
```

Gambar 5. Sumber : Arsip Pribadi

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                           : 4
Total number of solutions   : 2
Execution time              : 0.0 miliseconds
```

Gambar 6. Sumber : Arsip Pribadi

G. Persoalan N-Ratu dengan (N=6)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
N                           : 6
Total number of solutions   : 4
Execution time              : 22.99755859375 miliseconds
```

Gambar 7. Sumber : Arsip Pribadi

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                           : 6
Total number of solutions   : 4
Execution time              : 0.0 miliseconds
```

Gambar 8. Sumber : Arsip Pribadi

H. Persoalan N-Ratu dengan (N=8)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
N                           : 8
Total number of solutions   : 92
Execution time              : 9057.933837890625 miliseconds
```

Gambar 9. Sumber : Arsip Pribadi

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                           : 8
Total number of solutions   : 92
Execution time              : 5.509765625 miliseconds
```

Gambar 10. Sumber : Arsip Pribadi

I. Persoalan N-Ratu dengan (N=9)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
```

Gambar 11. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                           : 9
Total number of solutions   : 352
Execution time              : 26.997802734375 miliseconds
```

Gambar 12. Sumber : Arsip Pribadi

J. Persoalan N-Ratu dengan (N=10)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
```

Gambar 13. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                           : 10
Total number of solutions   : 724
Execution time              : 150.516845703125 miliseconds
```

Gambar 14. Sumber : Arsip Pribadi

K. Persoalan N-Ratu dengan (N=11)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
```

Gambar 15. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                           : 11
Total number of solutions   : 2680
Execution time              : 763.608154296875 miliseconds
```

Gambar 16. Sumber : Arsip Pribadi

L. Persoalan N-Ratu dengan (N=12)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
```

Gambar 17. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                               : 12
Total number of solutions       : 14200
Execution time                  : 4597.190185546875 miliseconds
```

Gambar 18. Sumber : Arsip Pribadi

M. Persoalan N-Ratu dengan (N=13)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
```

Gambar 19. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
N                               : 13
Total number of solutions       : 73712
Execution time                  : 28595.270263671875 miliseconds
```

Gambar 20. Sumber : Arsip Pribadi

N. Persoalan N-Ratu dengan (N=14)

- Algoritma *Brute-Force*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBruteForce.py"
```

Gambar 21. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

- Algoritma *Backtracking*

```
PS C:\Users\ASUS\MakalahSTIMA> python -u "c:\Users\ASUS\MakalahSTIMA\N-QueensBackTrack.py"
```

Gambar 22. Sumber : Arsip Pribadi

(Tidak dapat selesai < 2 menit)

Algoritma *brute-force* bekerja hanya sampai N=8 sedangkan algoritma *backtracking* bekerja hingga N=13. Algoritma *backtracking* sangatlah mangkus jika dibandingkan dengan algoritma *brute-force.*

Tabel Algoritma Brute-Force :

N	Execution Time (ms)
4	0.0
6	23.00
8	9657.93
9	-
10	-
11	-
12	-
13	-
14	-

Tabel 1. Waktu eksekusi algoritma *brute-force*

Tabel Algoritma Backtracking :

N	Execution Time (ms)
4	0.00
6	0.00
8	5.51
9	27.00
10	150.52
11	763.61
12	4597.19
13	28595.27
14	-

Tabel 2. Waktu eksekusi algoritma *backtracking*

V. KESIMPULAN

Berdasarkan hasil penelitian dan pengujian yang dilakukan, dapat disimpulkan bahwa algoritma Backtracking lebih efektif dalam menyelesaikan persoalan N-Ratu dibandingkan dengan algoritma Brute Force. Algoritma Backtracking menggunakan pendekatan rekursif dengan membatalkan langkah-langkah yang tidak valid, sehingga dapat mengurangi jumlah percobaan yang perlu dilakukan dan mempercepat pencarian solusi. Dalam pengujian dengan variasi ukuran papan catur dan jumlah ratu, algoritma Backtracking secara konsisten menghasilkan solusi dengan waktu yang lebih efisien.

Selain efektivitas dalam menyelesaikan persoalan N-Ratu, analisis kompleksitas waktu dan ruang juga dilakukan pada kedua algoritma. Algoritma Brute Force memiliki kompleksitas waktu eksponensial karena mencoba semua kemungkinan solusi secara sistematis. Sementara itu, algoritma Backtracking memiliki kompleksitas waktu yang lebih baik karena melakukan pemotongan (pruning) pada

langkah-langkah yang tidak valid. Namun, perlu diperhatikan bahwa kompleksitas waktu dan ruang kedua algoritma masih bergantung pada ukuran papan catur dan jumlah ratu yang harus ditempatkan.

Dalam konteks penggunaan pada komputer modern, algoritma Backtracking merupakan pilihan yang lebih optimal untuk penyelesaian persoalan N-Ratu. Dengan efisiensi waktu yang lebih tinggi, algoritma ini dapat mengurangi waktu komputasi yang diperlukan untuk menemukan solusi. Namun, perlu dipertimbangkan juga bahwa algoritma Backtracking memiliki batasan pada ukuran papan catur yang sangat besar.

Penelitian ini memberikan wawasan yang berharga dalam memilih algoritma yang tepat dalam menyelesaikan persoalan N-Ratu. Selain itu, hasil pengujian dan analisis kompleksitas dapat menjadi acuan bagi pengembangan algoritma yang lebih canggih dan efisien dalam menyelesaikan persoalan serupa.

Ucapan Terima Kasih

Penulis mengungkapkan rasa syukur kepada Tuhan Yang Maha Esa atas kesempatan untuk menyelesaikan makalah ini. Penulis juga ingin menyampaikan terima kasih yang tulus kepada dosen K2, Ibu Nur Ulfa Maulidevi, yang telah dengan penuh dedikasi membagikan pengetahuannya kepada penulis dan teman-teman sekelas di K2. Penulis juga ingin mengucapkan terima kasih kepada seluruh tim pengajar dan tim asisten IF2211 Strategi Algoritma yang telah memberikan bantuan dan dukungan. Semoga makalah ini dapat memberikan manfaat bagi para pembacanya.

Referensi

[1] Munir, R. 2023. Algoritma Brute Force. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Brute-Force-(2021)-Bag1.pdf. Diakses pada tanggal 22 Mei 2023.

[2] Munir, R. 2023. Algoritma runut-balik. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf. Diakses pada tanggal 22 Mei 2023.

[3] Geeks for Geeks. 2023. N Queen Problem | Backtracking-3. URL: https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/. Diakses pada tanggal 22 Mei 2023

[4] Coding Ninjas. 2023. N-Queen. URL: https://www.codingninjas.com/codestudio/library/n-queen. Diakses pada tanggal 22 Mei 2023

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, 22 Mei 2023

Muhammad Zaydan Athallah, 13521104.