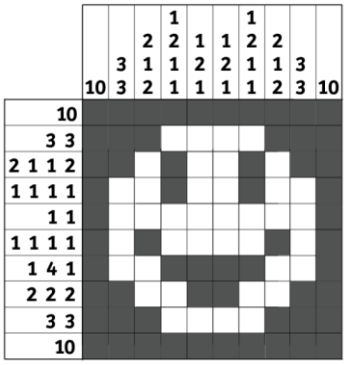
Perbandingan Algoritma *Brute Force* dan Runut-balik (*Backtracking*) dalam Penyelesaian Permainan Nonogram

Raffael Boymian Siahaan - 13522046
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13522046@std.stei.itb.ac.id

***Abstract*— Makalah ini mengeksplorasi Nonogram, teka-teki logika yang memadukan elemen gambar dan angka untuk menciptakan pola pada grid. Berasal dari Jepang pada tahun 1987, Nonogram telah berkembang dari kompetisi desain lampu pencakar langit menjadi fenomena global yang dipopulerkan melalui media cetak dan elektronik. Makalah ini berfokus pada penggunaan algoritma *Brute Force* dan *Backtracking* dalam penyelesaian Nonogram, menilai efisensi dan efektivitas keduanya dalam konteks dan kompleksitas waktu algoritma. Dengan membandingkan kedua metode tersebut, makalah ini mengidentifikasi pendekatan yang paling efektif untuk menyelesaikan teka-teki ini, memberikan wawasan bagi dan penggemar teka-teki dalam memilih strategi penyelesaian yang optimal.**

***Keywords—brute force*, runut-balik, *backtracking*, nonogram**

I. PENDAHULUAN



***Gambar 1.1** Permainan nonogram*

Diambil https://www.linkedin.com/pulse/multisize-nonogram-solver-1kyu-completed-took-me-three-shevel

Nonogram adalah teka-teki logika berbasis gambar di mana pemain mengisi atau membiarkan kosong sel-sel dalam sebuah grid berdasarkan angka-angka yang tercantum di sisi grid. Angka-angka tersebut merepresentasikan berapa banyak sel berturut-turut yang harus diisi dalam setiap baris atau kolom dengan setidaknya satu sel kosong di antara kelompok-kelompok angka yang berbeda. Misalnya, petunjuk "4 8 3" berarti ada empat, delapan, dan tiga kotak berturut-turut yang diisi, dengan setidaknya satu kotak kosong di antara masing-masing kelompok. Meskipun sering kali dimainkan dalam format hitam putih, nonogram juga dapat hadir dalam beberapa warna, di mana petunjuk angka berwarna menunjukkan warna sel yang harus diisi.

Konsep nonogram pertama kali muncul pada tahun 1987 ketika Non Ishida, seorang editor grafis dari Jepang, memenangkan kompetisi di Tokyo dengan desain gambar kotak menggunakan lampu pencakar langit yang dihidupkan dan dimatikan. Pada waktu yang hampir bersamaan, Tetsuya Nishio, seorang kusut profesional Jepang, menciptakan teka-teki serupa. Dari sinilah konsep "Paint by Numbers" dan teka-teki logika pembentuk gambar lahir. Teka-teki ini mulai muncul di majalah-majalah Jepang dan kemudian dipopulerkan oleh Nintendo melalui seri "Picross" untuk Game Boy dan Super Famicom.

Di Inggris, teka-teki ini mendapatkan nama "Nonograms" dari James Dalgety pada tahun 1990, dan sejak saat itu, teka-teki ini mulai diterbitkan secara rutin oleh The Sunday Telegraph. Buku-buku nonogram pertama diterbitkan di Jepang dan Inggris pada awal 1990-an, yang kemudian diikuti oleh publikasi di berbagai negara lain seperti Swedia, Amerika Serikat, dan Afrika Selatan. Seiring waktu, popularitas nonogram meningkat dan merambah ke media elektronik seperti mainan genggam dan game arcade.

Dalam penyelesaian permainan nonogram, algoritma memiliki peran penting dalam menentukan efisiensi dan efektivitas penyelesaian teka-teki. Algoritma yang dapat digunakan dalam penyelesaian nonogram adalah *Brute Force* dan Runut-balik (*Backtracking*).

Pembuatan makalah ini bertujuan untuk membandingkan ketiga algoritma tersebut dalam konteks penyelesaian permainan nonogram, mengevaluasi kinerja masing-masing berdasarkan waktu eksekusi, kompleksitas algoritma, dan efektivitas dalam menemukan solusi.

II. LANDASAN TEORI

A. *Algoritma Brute Force*

Algoritma *Brute Force* adalah pendekatan yang lempeng (*straightforward*) untuk memecahkan suatu persoalan. Algoritma ini didasarkan pada pernyataan pada persoalan (*problem statement*) dan Definisi/konsep yang dilibatkan. Algoritma ini dapat memecahkan persoalan dengan sangat sederhana, langsung, jelas caranya, dan seringkali disebut sebagai "sapu jagad" karena dapat menyelesaikan segala jenis persoalan tanpa mempedulikan efisiensi dari alur berpikir program yang dibuat.

Contoh dari algoritma *Brute Force* adalah sebagai berikut :

1. Mencari elemen terkecil atau terbesar pada suatu senarai (*array*)

2. Pencarian beruntun (*Sequential Search*)

3. Menghitung nilai eksponen, faktorial, perkalian matriks, uji bilangan prima, logika pengurutan (*sorting*), seperti *bubble sort* dan *selection sort*, pencocokan string (*String Matching/Pattern Matching*)

Karakteristik dari Algoritma *Brute Force* adalah :

1. Algoritma *Brute Force* umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan volume komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Kata "*Force*" mengindikasikan "tenaga" ketimbang "otak" Kadang-kadang algoritma *Brute Force* disebut juga algoritma naif (*naïve algorithm*).

2. Algoritma *Brute Force* lebih cocok untuk persoalan yang ukuran masukannya (n) kecil. Pertimbangannya adalah sederhana dan implementasinya mudah. Algoritma *Brute Force* sering digunakan sebagai basis pembanding dengan algoritma lain yang lebih mangkus.

3. Meskipun bukan metode *problem solving* yang mangkus, hampir semua persoalan dapat diselesaikan dengan algoritma *Brute Force*. Sangat sukar untuk menunjukkan persoalan yang tidak dapat diselesaikan dengan metode *Brute Force*. Bahkan, ada persoalan yang hanya dapat diselesaikan dengan *Brute Force*. Contoh: mencari elemen terbesar di dalam senarai.

Kekuatan dan kelemahan Algoritma *Brute Force* adalah sebagai berikut :

Kekuatan:

1. Algoritma *Brute Force* dapat diterapkan untuk memecahkan hampir sebagian besar masalah (*wide applicability*).

2. Algoritma *Brute Force* sederhana dan mudah dimengerti.

3. Algoritma *Brute Force* menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.

4. Algoritma *Brute Force* menghasilkan algoritma baku (*standard*) untuk tugas-tugas komputasi, seperti penjumlahan/perkalian n buah bilangan, menentukan elemen minimum atau maksimum di dalam senarai (larik)

Kelemahan:

1. Algoritma *Brute Force* jarang menghasilkan algoritma yang mangkus.

2. Algoritma *Brute Force* umumnya lambat untuk masukan berukuran besar sehingga tidak dapat diterima.

3. Tidak sekontruktif/sekreatif strategi pemecahan masalah lainnya.

Exhaustive search adalah teknik pencarian solusi secara solusi brute force untuk persoalan-persoalan kombinatorik, yaitu persoalan di antara objek-objek kombinatorik seperti permutasi, kombinasi, atau himpunan bagian dari sebuah himpunan. Langkah-langkah di dalam *exhaustive search*:

1. Enumerasi (*list*) setiap kemungkinan solusi dengan cara yang sistematis.

2. Evaluasi setiap kemungkinan solusi satu per satu, simpan solusi terbaik yang ditemukan sampai sejauh ini (*the best solution found so far*).

3. Bila pencarian berakhir, umumkan solusi terbaik (*the winner*).

Meskipun *exhaustive search* secara teoritis menghasilkan solusi, namun waktu atau sumber daya yang dibutuhkan dalam pencarian solusinya sangat besar. Contoh dari *exhaustive search* adalah *Travelling Salesperson Problem* (TSP) dan 1/0 *Knapsack Problem*.

B. *Algoritma Runut-Balik (Backtracking)*

Algoritma runut-balik (*backtracking*) merupakan algoritma yang merupakan metode pemecahan masalah yang mangkus, terstruktur, dan sistematis untuk persoalan optimasi maupun non-optimasi. Algoritma ini juga merupakan salah satu fase dalam algoritma traversal DFS, yang memungkinkan melakukan pemangkasan (*pruning*) apabila tidak ditemukan jalur saat pencarian mendalam.

Algoritma ini merupakan perbaikan dari exhaustive search, dimana pada exhaustive search, dilakukan tahap mencari semua kemungkinan solusi dan dievaluasi satu per satu, sedangkan pada algoritma backtracking, solusi yang dieksplorasi hanya pilihan yang mengarah ke solusi, sehingga simpul pilihan yang tidak mengarah ke solusi akan dipangkas (*pruning*).

Algoritma ini diperkenalkan oleh D. H. Lehmer pada 1950, kemudian R. Kemudian diuraikan lebih lanjut oleh, R.J Walker, Golomb, dan Baumert.

Properti umum algoritma runut-balik adalah sebagai berikut:

1. Solusi persoalan
    - Solusi dinyatakan sebagai vektor dengan n-tuple: $X = (x1 , x2 , \ldots, xn)$ dengan $xi \in Si$. Umumnya $S1 = S2 = \ldots = Sn$.
2. Fungsi pembangkit nilai Xk

- Dinyatakan sebagai predikat T() dengan T(x[1], x[2], …, x[k – 1]) membangkitkan nilai untuk xk , yang merupakan komponen vektor solusi.

3. Fungsi pembatas (*bounding function*)

   - Dinyatakan sebagai predikat B(x1, x2, …, xk) dengan B bernilai true jika (x1, x2, …, xk) mengarah ke solusi. Mengarah ke solusi artinya tidak melanggar kendala (constraints). Jika true, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika false, maka (x1 , x2 , …, xk) dibuang.

Semua kemungkinan solusi dari persoalan disebut ruang solusi (*solution space*). Ruang solusi direpresentasikan ke dalam struktur pohon berakar, dimana tiap simpul pada pohon menyatakan status (state) persoalan, sedangkan sisi (cabang) ditandai dengan nilai-nilai Xi. Lintasan dari akar ke daun menyatakan solusi yang mungkin dari permasalahan. Ruang solusi tersebut sering disebut sebagai pohon ruang status (*state space tree*).

Prinsip pencarian solusi dengan algoritma ini adalah sebagai berikut:

1. Solusi akan dicari dengan membangkitkan simpul-simpul status (secara DFS) sehingga menghasilkan sebuah lintasan dari akar ke daun.
2. Simpul-simpul yang sudah dibangkitkan dinamakan simpul hidup (*live node*), sedangkan simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand-node*).
3. Setiap kali simpul-E diperluas, lintasan yang terbentuk bertambah panjang.
4. Jika lintasan yang terbentuk tidak mengarah ke solusi, simpul-E tersebut akan dimatikan sehingga menjadi simpul mati (*dead node*). Ketika sebuah simpul mati, kita secara implisit telah memangkas (*pruning*) simpul-simpul anaknya dan akan melakukan *backtrack* ke simpul di atasnya, dan diteruskan dengan membangkitkan simpul anak lainnya (simpul-E yang baru).
5. Untuk mematikan simpul-E tersebut, terdapat sebuah fungsi yang diterapkan, yaitu fungsi pembatas (*bounding function*).
6. Pencarian akan dihentikan bila telah sampai pada *goal node*.

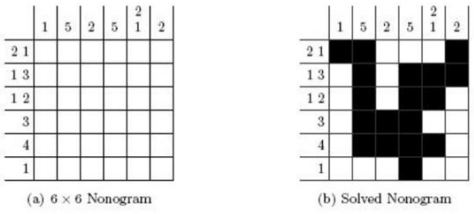
Algoritma runut balik sering digunakan pada persoalan *constraint satisfaction problems*, seperti *The N-Queens Problem*, *sum of subsets problem*, *graph colouring*, sirkuit hamilton, dsb.

C. *Nonogram*

Nonogram adalah teka-teki logika berbasis gambar di mana pemain mengisi atau membiarkan kosong sel-sel dalam sebuah grid berdasarkan angka-angka yang tercantum di sisi grid. Angka-angka tersebut merepresentasikan berapa banyak sel berturut-turut yang harus diisi dalam setiap baris atau kolom dengan setidaknya satu sel kosong di antara kelompok-kelompok angka yang berbeda.

Komponen utama yang terdapat pada permainan ini adalah:

1. *Grid*



Gambar 2.1 *Grid pada permainan nonogram*

Diambil dari https://medium.com/smith-hcv/solving-hard-instances-of-nonograms-35c68e4a26dfAbbreviations and Acronyms

Grid merupakan sebuah kotak yang terbagi menjadi sel-sel kecil yang akan diisi atau dibiarkan kosong. Dalam konteks nonogram, *grid* biasanya berukuran persegi n x n.

2. Angka Petunjuk

Gambar 2.2 *Angka petunjuk baris*

Diambil dari https://medium.com/smith-hcv/solving-hard-instances-of-nonograms-35c68e4a26dfAbbreviations and Acronyms

Angka-angka yang terletak di sisi atas dan kiri grid, menunjukkan jumlah sel yang harus diisi secara berurutan di setiap baris atau kolom.

3. Sel Kosong

Ruang antara kelompok-kelompok sel yang diisi, yang harus dibiarkan kosong. Biasa diberi tanda silang atau ‘X’.

Secara sederhana, cara memainkan nonogram adalah pemain harus menggunakan petunjuk angka di sisi atas dan kiri grid untuk menentukan sel mana yang harus diisi (biasanya dengan warna hitam) dan sel mana yang harus dibiarkan kosong. Petunjuk angka menunjukkan kelompok sel yang berurutan yang harus diisi dalam setiap baris atau kolom, dan harus ada setidaknya satu sel kosong antara setiap kelompok.

III. Analisis Persoalan dan Implementasi Algoritma

Algoritma yang akan dianalisis dan diimplementasikan dalam makalah ini adalah *brute force* dan runut-balik (*backtracking*).

A. *Algoritma Brute Force*

Tahapan penyelesaian permainan nonogram dengan menggunakan algoritma *brute force* secara garis besar dapat dideskripsikan sebagai berikut :

1. Menghasilkan semua kemungkinan kombinasi dari masing-masing baris dan kolom berdasarkan petunjuk yang diberikan mengunakan fungsi

‘`find_combinations()`’ Setiap kombinasi diwakili sebagai array yang berisi 'x' untuk blok yang diisi dan '-' untuk blok kosong.

2. Menghasilkan kombinasi kemungkinan cara mengisi grid pada Nonogram menggunakan fungsi ‘`cartesian_product()`’ untuk melakukan perkalian *cartesian* terhadap kombinasi baris dan kombinasi kolom.
3. Melakukan tahap evaluasi atau memeriksa setiap kombinasi baris dan kolom dari *cartesian product* untuk menemukan solusi yang memenuhi petunjuk untuk baris dan kolom menggunakan fungsi ‘`validate_combinations()`’ dan ‘`satisfies_clues()`’. Tahap evaluasi menggunakan konsep *transpose matrix*, dengan konsiderasi transpos dari matriks kombinasi kolom sama dengan matriks kombinasi baris apabila membentuk solusi.

Untuk mengilustrasikan tahapan di atas, Pseudocode untuk penyelesaian permainan nonogram dengan menggunakan algoritma *brute force* adalah sebagai berikut:

```
FUNCTION bruteforceSolver(row_hints, col_hints,
n)
    INITIALIZE row_combination and
col_combination to empty lists

    FOR each row in row_hints DO
        APPEND find_combinations(n, row) to
row_combination
    END FOR

    FOR each col in col_hints DO
        APPEND find_combinations(n, col) to
col_combination
    END FOR

    SET row_combinations to
cartesian_product(row_combination)
    SET col_combinations to
cartesian_product(col_combination)

    SET result to
validate_combinations(row_combinations,
col_combinations)

    IF result is not None THEN
        PRINT nonogram with row_hints, col_hints,
and result
    ELSE
        PRINT "Maaf, tidak ada solusi"
    END IF
END FUNCTION
```

Implementasi algoritma *brute force* lebih detail dapat dilihat melalui link GitHub berikut : https://github.com/slntkllr01/Nonogram-Solver

B. Algoritma runut-balik (backtracking)

Dekomposisi permasalahan nonogram ke dalam properti umum algoritma runut-balik adalah sebagai berikut :

1. Solusi persoalan

   Dinyatakan dalam matriks X = (X[1,1], X[1,2], …, X[n, n]) dengan xi ∈ {‘x’, ‘-‘}

2. Fungsi Pembangkit Nilai

   Proses pemilihan nilai untuk mengisi sel X[i, j] (pada indeks baris ke i, kolom ke j) dengan predikat T().

3. Fungsi Pembatas (*Bounding Function*)

   B((X[1,1], X[1,2], …, X[n, n]) akan bernilai true apabila memenuhi fungsi ‘`is_valid_partial_row()`’, yaitu sebagian dari baris yang sedang dibangun sesuai dengan petunjuk yang diberikan (jumlah ‘X’ <= petunjuk atau untuk length petunjuk > 1, berlaku setiap elemen petunjuk dalam list petunjuk tersebut tidak kontigu/bersebelahan).

Tahapan penyelesaian permainan nonogram dengan menggunakan algoritma *backtracking* secara garis besar dapat dideskripsikan sebagai berikut :

1. Menghasilkan semua kombinasi kolom yang mungkin berdasarkan petunjuk kolom saat ini menggunakan fungsi ‘`find_combinations()`’
2. Setiap kombinasi kolom yang mungkin berada di dalam grid akan ditempatkan sementara pada sebuah *temp*.
3. Fungsi ‘`is_valid_partial_row()`’ akan memeriksa setiap baris untuk memastikan bahwa penempatan kolom tidak melanggar petunjuk baris.
4. Jika memenuhi fungsi pembatas (*bounding function*), dilanjutkan pencarian ke kolom berikutnya. Jika tidak, lakukan *backtrack* ke pendahulunya. *Backtracking* dilakukan dengan menghapus kolom yang telah ditempatkan dari grid *temp*.
5. Jika semua kolom telah ditempatkan dan valid, solusi telah ditemukan dan me-*return* grid yang terisi.

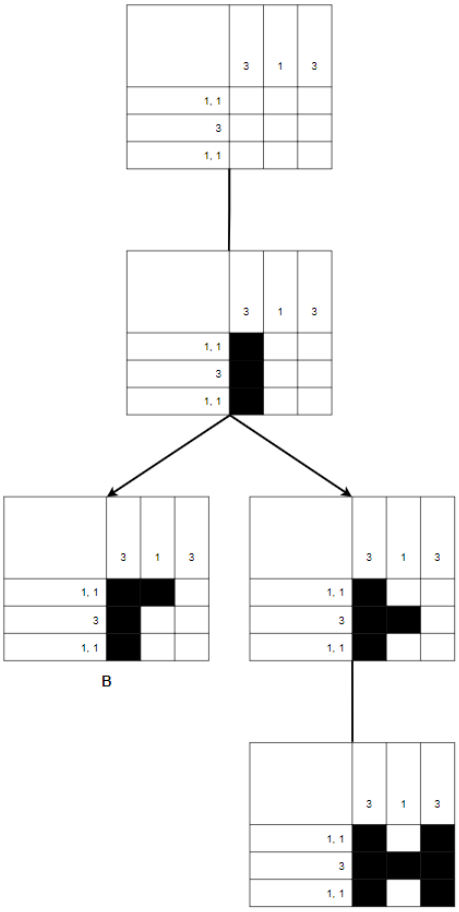
Dalam algoritma Nonogram dengan *backtracking*, fungsi pembatas digunakan untuk mengevaluasi apakah baris tertentu masih memiliki potensi untuk memenuhi petunjuk yang diberikan.

Sebagai contoh, misalkan sebuah nonogram berukuran 3 x 3 (n = 3) dengan suatu petunjuk row_hint[i] = [2] dan suatu baris row[i] = [‘x’, ‘-‘, ‘-‘] masih dianggap valid. Meskipun hanya terdapat satu ‘x’, baris ini memiliki kemungkinan (potensi) untuk memenuhi petunjuk dengan kondisi akhir [‘x’, ‘x’, ‘-‘], di mana dua ‘x’ berdekatan sesuai dengan petunjuk.

Sebaliknya, misalkan sebuah nonogram berukuran 5 x 5 (n = 5) dengan suatu petunjuk row_hint[i] = [1, 2] dan suatu baris row[i] = [‘x’, ‘x‘, ‘x‘, ‘-‘, ‘-‘] tidak memenuhi fungsi pembatas. Petunjuk [1, 2] menandakan bahwa harus ada satu ‘x’ yang terpisah dari dua ‘x’ berikutnya. Oleh

karena itu, tiga ‘x’ yang berurutan pada row[i] tidak sesuai dengan petunjuk. Sebuah kondisi yang memenuhi petunjuk tersebut adalah [‘x’, ‘-‘, ‘x‘, ‘x‘, ‘-‘], di mana satu ‘x’ diikuti oleh ruang kosong dan kemudian dua ‘x’ berdekatan.

Sekilas mengenai pencarian yang dilakukan dapat dilihat pada gambar di bawah.



Gambar 3.1 *Proses pencarian jalur menggunakan backtracking*

Diambil dari dokumentasi pribadi

Untuk mengilustrasikan tahapan di atas, Pseudocode untuk penyelesaian permainan nonogram dengan menggunakan algoritma runut-balik (*backtracking*) adalah sebagai berikut:

```
FUNCTION backtrackingSolver(row_hints, col_hints)
    DEFINE FUNCTION solve_recursive(grid,
row_hints, col_hints, col_idx)
        IF col_idx equals length of grid[0] THEN
            RETURN True
        END IF

        SET possible_cols to
find_combinations(length of col_hints,
col_hints[col_idx])

        FOR each col in possible_cols DO
            PLACE col temporarily in grid

            SET valid to True
            FOR row_idx from 0 to length of
row_hints DO
                IF NOT
is_valid_partial_row(grid[row_idx],
row_hints[row_idx]) THEN
                    SET valid to False
                    BREAK
                END IF
            END FOR

            IF valid THEN
                IF solve_recursive(grid,
row_hints, col_hints, col_idx + 1) THEN
                    RETURN True
                END IF
            END IF

            BACKTRACK by resetting grid
        END FOR

        RETURN False
    END FUNCTION

    INITIALIZE grid to list of '-' with size of
row_hints by col_hints
    IF solve_recursive(grid, row_hints,
col_hints, 0) THEN
        RETURN grid
    ELSE
        RETURN None
    END IF
END FUNCTION

FUNCTION backtrackingSolverMain(row_hints,
col_hints)
    SET solution to backtrackingSolver(row_hints,
col_hints)

    IF solution is not None THEN
        PRINT nonogram with row_hints, col_hints,
and solution
    ELSE
        PRINT "Maaf, tidak ada solusi"
    END IF
END FUNCTION
```

Dengan pseudocode fungsi pembatas (bounding function) sebagai berikut:

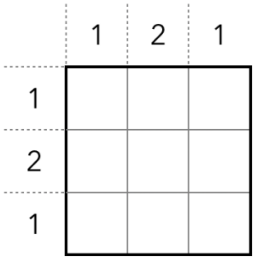
```
FUNCTION is_valid_partial_row(row, row_hint)
    INITIALIZE segments to empty list
    INITIALIZE current_segment to 0

    FOR each cell in row DO
        IF cell is 'x' THEN
            INCREMENT current_segment
        ELSE IF current_segment > 0 THEN
            APPEND current_segment to segments
            RESET current_segment to 0
        END IF
    END FOR

    IF current_segment > 0 THEN
        APPEND current_segment to segments
    END IF

    IF length of segments > length of row_hint THEN
        RETURN False
    END IF

    FOR i from 0 to length of segments DO
        IF segments[i] > row_hint[i] THEN
            RETURN False
        END IF
    END FOR

    RETURN True
END FUNCTION
```

Implementasi algoritma runut-balik (*backtracking*) lebih detail dapat dilihat melalui link GitHub berikut : https://github.com/slntkllr01/Nonogram-Solver

IV. Analisis dan Pengujian

Untuk melakukan pengujian, akan digunakan tiga percobaan, yaitu matriks nonogram berukuran 3 x 3, 5 x 5, dan 7 x 7. Hal yang akan diuji adalah hasil (output yang diberikan) dan waktu eksekusi program untuk masing-masing algoritma. Setelah dilakukan pengujian, akan dilakukan analisis kompleksitas waktu algoritma, serta membandingkan algoritma mana yang lebih efektif dalam penyelesaian permainan nanogram. Analisis ini penting untuk memahami efisiensi algoritma dalam berbagai skenario dan akan menjadi dasar untuk rekomendasi pengembangan algoritma selanjutnya. Kesimpulan yang ditarik dari analisis ini akan memberikan wawasan berharga tentang kepraktisan algoritma dalam penyelesaian Nonogram.

Kasus 1 : Matriks 3 x 3



Gambar 4.1 *Matriks nonogram 3 x 3*

Diambil dari google image

Dengan menggunakan algoritma *brute force*, didapat hasil berikut :

```
[1] Brute Force
[2] Runut-Balik (Backtracking)

Masukkan jenis algoritma yang ingin digunakan ! [1 / 2] 1


    1  2  1
  1 x ■ x
  2 x ■ ■
  1 ■ x x

Durasi: 10.448 ms
```

Gambar 4.2 *Hasil pencarian menggunakan algoritma brute force*

Diambil dari dokumentasi pribadi

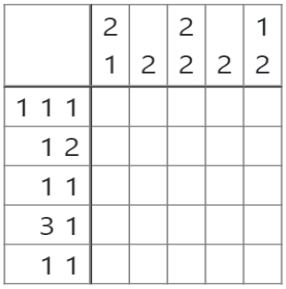
Dengan menggunakan algoritma *backtracking*, didapat hasil berikut :

```
[1] Brute Force
[2] Runut-Balik (Backtracking)

Masukkan jenis algoritma yang ingin digunakan ! [1 / 2] 2


    1  2  1
  1 ■ x x
  2 x ■ ■
  1 x ■ x

Durasi: 1.969 ms
```

Gambar 4.3 *Hasil pencarian menggunakan algoritma backtracking*

Diambil dari dokumentasi pribadi

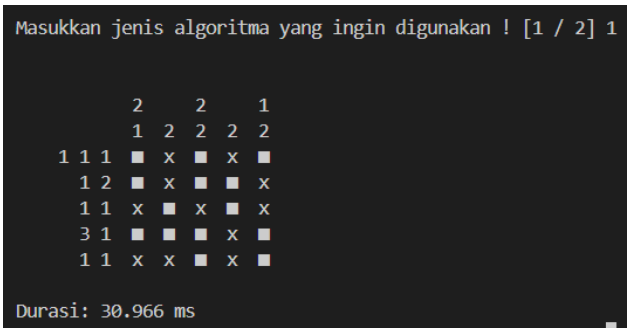
Kasus 1 : Matriks 5 x 5



Gambar 4.4 *Matriks nonogram 5 x 5*

Diambil dari google image

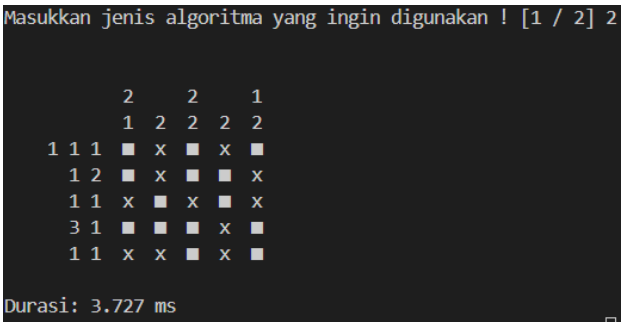
Dengan menggunakan algoritma *brute force*, didapat hasil berikut :



Gambar 4.5 *Hasil pencarian menggunakan algoritma brute force*

Diambil dari dokumentasi pribadi

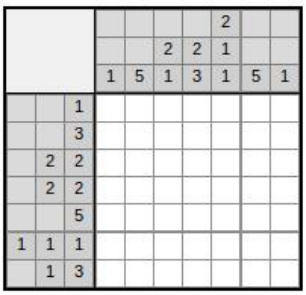
Dengan menggunakan algoritma *backtracking*, didapat hasil berikut :



Gambar 4.6 *Hasil pencarian menggunakan algoritma backtracking*

Diambil dari dokumentasi pribadi

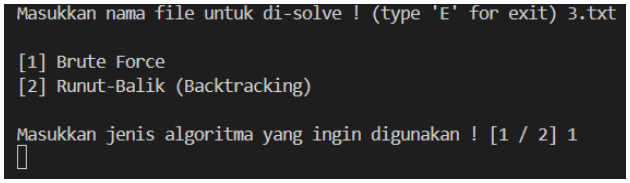
<u>Kasus 1 : Matriks 7 x 7</u>



Gambar 4.7 *Matriks nonogram 7 x 7*

Diambil dari google image

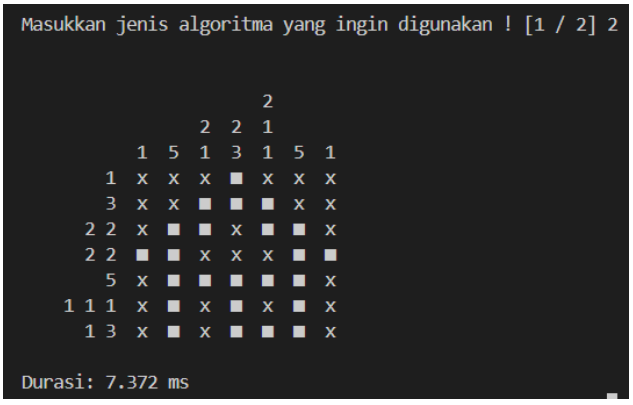
Dengan menggunakan algoritma *brute force*, didapat hasil berikut :



Gambar 4.8 *Hasil pencarian menggunakan algoritma brute force*

Diambil dari dokumentasi pribadi

Dengan menggunakan algoritma *backtracking*, didapat hasil berikut :



Gambar 4.9 *Hasil pencarian menggunakan algoritma backtracking*

Diambil dari dokumentasi pribadi

Berdasarkan hasil dari tiga percobaan yang telah dilakukan, data statistik waktu eksekusi yang diperoleh disajikan dalam tabel di bawah ini:

Ukuran Matriks	Brute Force (ms)	Backtracking (ms)
3 x 3	10,448	1,969
5 x 5	30,966	3,727

7 x 7	>300.000	7,372

Tabel 4.1 *Data Statistik waktu eksekusi kedua algoritma*

Diambil dari dokumentasi pribadi

Analisis kompleksitas waktu algoritma *brute force* (berdasarkan teori)

Implementasi algoritma *brute force* melakukan 1 kali pencarian kombinasi solusi (pemanggilan find_combinations()) untuk setiap petunjuk baris dan 1 kali untuk setiap petunjuk kolom. Jika ada r petunjuk baris dan c petunjuk kolom, maka fungsi ini dipanggil r + c kali. Kompleksitas waktu dari setiap pemanggilan fungsi adalah eksponensial terhadap panjang baris atau kolom n. Dalam kasus terburuk, kompleksitasnya adalah O(m^n), di mana m adalah jumlah maksimum kemungkinan kombinasi untuk satu petunjuk.

Algoritma *brute force* juga memanggil dua kali fungsi *cartesian product*, 1 kali untuk kombinasi baris dan 1 kali untuk kombinasi kolom. Kompleksitas waktu dari fungsi ini adalah O(k^n), di mana k adalah rata-rata ukuran dari setiap *array* dalam *list of arrays* dan n adalah jumlah *array* yang diberikan. Dalam konteks ini, k akan menjadi jumlah kombinasi yang dihasilkan oleh fungsi 'find_combinations()' untuk setiap petunjuk, dan n akan menjadi jumlah petunjuk baris atau kolom.

Selanjutnya, Fungsi untuk mengevaluasi setiap kemungkinan yang ada atau 'validate_combinations()' dipanggil sekali dengan dua set produk kartesian sebagai input. Jika kita asumsikan bahwa produk kartesian dari kombinasi baris menghasilkan R kombinasi dan produk kartesian dari kombinasi kolom menghasilkan C kombinasi, maka kompleksitas waktu dari 'validate_combinations()' adalah O(R * C * n^2).

Jadi, kompleksitas waktu keseluruhan dari algoritma *brute force* adalah O(m^(2n) * n^2) dengan asumsi bahwa R dan C tumbuh secara eksponensial dengan n.

Analisis kompleksitas waktu algoritma *backtracking* (Berdasarkan Teori)

Implementasi algoritma *backtracking* cukup memanggil 1 kali pencarian kombinasi solusi untuk setiap petunjuk kolom. jika ada c petunjuk kolom, maka fungsi ini akan dipanggil c kali. Setiap panggilan fungsi ini memiliki kompleksitas waktu eksponensial terhadap panjang kolom n, yang bisa dinyatakan sebagai O(m^n), di mana m adalah jumlah maksimum kemungkinan kombinasi untuk satu petunjuk.

Kemudian, fungsi 'solve_recursive()' atau fungsi untuk melakukan pencarian rekursif dalam eksplorasi solusi akan mencoba semua kombinasi kolom yang mungkin dan memeriksa validitas baris menggunakan *bounding function* 'is_valid_partial_row()'. Kompleksitas waktu dari fungsi ini sangat bergantung pada jumlah petunjuk dan bagaimana petunjuk tersebut membatasi ruang pencarian. Dalam kasus terburuk, kompleksitas waktu bisa mencapai O(n!) untuk grid n x n, karena setiap kolom bisa memiliki hingga n! kombinasi yang mungkin, dan backtracking mungkin perlu mengeksplorasi semua kombinasi ini.

Jadi, kompleksitas waktu keseluruhan dari algoritma *backtracking* adalah O(n!) dalam kasus terburuk, bahkan bisa lebih baik, atau lebih buruk untuk kasus n yang sangat besar.

Analisis kondisi hasil percobaan berdasarkan teori

Analisis yang didasarkan pada teori-teori yang telah dijelaskan diperkuat oleh data statistik yang disajikan dalam tabel. Dalam pemrosesan grid matriks berukuran 3 x 3, algoritma *brute force* memerlukan waktu sebesar 10,488 ms, yang tergolong tidak efisien untuk ukuran matriks yang relatif kecil. Sebagai perbandingan, algoritma *backtracking* hanya membutuhkan waktu 1,969 ms, dengan selisih waktu 8 – 9 ms. Pola yang sama juga terlihat pada grid matriks berukuran 5 x 5, di mana terdapat selisih waktu 27 ms.

Untuk grid matriks berukuran 7 x 7, algoritma *brute force* tidak menunjukkan kemajuan menuju solusi, dengan waktu yang cenderung tak terbatas atau tidak dapat diselesaikan. Sementara itu, algoritma backtracking menunjukkan perbedaan yang signifikan, hanya memerlukan waktu 7,372 ms untuk mencapai solusi.

V. KESIMPULAN

Dalam penyelesaian Nonogram, algoritma *Brute Force* dan *Backtracking* memiliki pendekatan yang berbeda. Meski solusi yang dihasilkan sudah pasti optimal, algoritma *Brute Force* dengan kompleksitas waktu O(m^(2n) * n^2) akan mencoba semua kemungkinan kombinasi tanpa mempertimbangkan informasi dari langkah-langkah sebelumnya, yang bisa sangat memakan waktu dan tidak efisien, terutama untuk grid Nonogram berukuran besar. Hal ini terjadi karena algoritma *Brute Force* tidak memiliki mekanisme untuk menghindari jalur yang jelas-jelas tidak akan mengarah pada solusi yang valid.

Di sisi lain, *Backtracking* adalah metode yang lebih 'cepat' dibandingkan *Brute Force*. Dengan kompleksitas waktu O(n!), Algoritma *Backtracking* akan membangun solusi langkah demi langkah dan melakukan pemangkasan apabila menemukan bahwa jalur saat ini tidak mungkin mengarah pada solusi yang valid. Hal ini membuat *Backtracking* jauh lebih cepat dan efisien karena hanya mengeksplorasi jalur yang potensial, menghemat waktu dan sumber daya komputasi. Oleh karena itu, Backtracking umumnya lebih efektif untuk Nonogram yang lebih kompleks.

VI. UCAPAN TERIMA KASIH

Dalam pemenuhan tugas makalah untuk mata kuliah Strategi Algoritma (IF2211), penulis ingin mengawali dengan ungkapan rasa syukur kepada Tuhan yang Maha Esa atas berkah, bimbingan, serta dukungan-Nya yang

memungkinkan penulis menyelesaikan makalah berjudul "Perbandingan Algoritma *brute force* dan runut-balik (*backtracking*) dalam Penyelesaian Permainan Nonogram".

Penulis ingin mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc., sebagai dosen mata kuliah Strategi Algoritma IF2211 Kelas K2 yang telah dengan penuh dedikasi memberikan pengajaran dan bimbingan kepada kami, para mahasiswa, selama satu semester ini.

Penulis juga ingin menyampaikan terima kasih kepada bapak Dr. Ir. Rinaldi, M.T. atas sumber pembelajaran Strategi Algoritma yang diberikan melalui website beliau, dan tentunya kepada semua pihak yang telah memberikan kontribusi, baik secara langsung maupun tidak langsung, dalam penulisan makalah ini.

Penulis juga ingin menyampaikan terima kasih sebesar-besarnya kepada keluarga, teman-teman, dan siapapun yang berkontribusi dalam pembuatan makalah ini dalam bentuk *support*/dukungan mental yang melancarkan pengerjaan tugas makalah ini.

Selain itu, penulis ingin menyampaikan permohonan maaf apabila terdapat kesalahan dalam penulisan makalah ini yang sekiranya dapat menyinggung pihak tertentu atau terdapat informasi yang menyesatkan.

VIDEO LINK AT YOUTUBE

https://youtu.be/9qW5UWXoJJ8

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian2.pdf

[5] Solving Hard Instances of Nonograms | by Abby Moss | Smith-HCV | Medium

[6] "K. Joost Batenburg dan Walter A. Kosters. "A Reasoning Framework for Solving Nonograms" 978-3-540-78275-9_33.pdf (springer.com)

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

Raffael Boymian Siahaan
13522046