Implementasi Solusi Permainan Sudoku dengan *Crook's Pencil-and-Paper Algorithm*

Aplikasi Algoritma *Greedy* dan Algoritma *Backtracking*

Rayhan Kinan Muhannad - 13520065
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): rayhankinan@gmail.com

***Abstract*—Terdapat banyak strategi dan algoritma program komputer untuk menyelesaikan suatu *puzzle* Sudoku, seperti *brute force*, *backtracking*, *stochastic search*, *constraint programming*, dan lain – lain. Salah satu algoritma yang menarik untuk diimplementasikan adalah dengan menggunakan *Crook's Pencil-and-Paper Algorithm* untuk menemukan solusi *puzzle* Sudoku. *Crook's Pencil-and-Paper Algorithm* merupakan suatu algoritma *greedy* yang menggunakan konsep *preemptive sets* untuk mengeliminasi kemungkinan angka yang dapat diisikan pada *grid* Sudoku. Jika terdapat lebih dari satu solusi Sudoku pada persoalan, dapat digunakan algoritma *backtracking* terhadap ruang solusi yang telah dieliminasi sebelumnya oleh *Crook's Pencil-and-Paper Algorithm*. Implementasi algoritma tersebut pada makalah ini direalisasikan dengan menggunakan program berbahasa Java.**

Keywords*—*Puzzle*, *Sudoku*, *Greedy*, *Backtracking*, *Crook's Pencil-and-Paper Algorithm

I. PENDAHULUAN

Sudoku merupakan suatu *puzzle* berbasis logika serta menggunakan konsep kombinatorial untuk mengisi *slot* angka-angka kosong yang terdapat di dalam *grid*. Pada umumnya, ukuran *grid* yang digunakan oleh permainan Sudoku adalah berukuran 9 × 9 *grid*. Objektif dari permainan Sudoku tersebut adalah mengisi seluruh *grid* kosong dengan suatu angka tertentu, dimana setiap kolom, baris, dan *subgrid* berukuran 3 × 3 *grid* harus mengandung seluruh angka 1 – 9. Awalnya, terdapat beberapa *grid* yang sudah terisi sebelumnya agar solusi yang terbentuk pada Sudoku tersebut unik.

Konsep dari permainan Sudoku sudah hadir pada akhir abad ke-19. Pada saat itu, beberapa surat kabar di negara Prancis menerbitkan beberapa *puzzle* yang memiliki konsep mirip dengan Sudoku. Objektif dan peraturan dari *puzzle* tersebut sedikit berbeda dengan Sudoku modern, dimana ukuran *grid* yang digunakan sama-sama memiliki ukuran 9 × 9, tetapi setiap kolom, baris, dan diagonal (dimana Sudoku menggunakan *subgrid* 3 × 3) harus mengandung seluruh angka 1 – 9.

Permainan Sudoku modern pertama kali dicetuskan oleh Howard Garns yang dipublikasikan pada *Dell Magazine* pada tahun 1979. Akan tetapi, Sudoku baru mulai dikenal oleh masyarakat awam dikarenakan Maki Kaji (鍛治 真起), presiden dari perusahaan *puzzle* Jepang yang bernama "Nikoli". Maki Kaji, disebut-sebut sebagai "*the father of Sudoku*", mengenalkan warga Jepang dengan permainan yang beliau sebut sebagai *Sūji wa dokushin ni kagiru* (数字は独身に限る), yang memiliki arti "*the digits must be single*" atau "*the digits are limited to one occurrence*". Kemudian, nama tersebut diringkas menjadi Sudoku (数独) seperti yang dikenal saat ini.

		1	9					8
6				8	5		3	
		7		6		1		
	3	4		9				
			5		4			
				1		4	2	
		5		7		9		
	1		8	4				7
7					9	2		

Gambar 1.1. Contoh *Puzzle* Sudoku

Sumber: ams.org

Terdapat beberapa strategi dan algoritma yang dapat digunakan untuk menyelesaikan Sudoku. Strategi pertama yang dapat digunakan Sudoku adalah dengan menggunakan gabungan dari *brute force algorithm* serta *backtracking*. Strategi ini merupakan strategi paling mendasar serta membutuhkan waktu komputasi yang paling lama. Hal ini dikarenakan untuk menemukan solusi, strategi tersebut memerlukan pengecekkan setiap kemungkinan solusi pada Sudoku. Untuk setiap *grid* yang belum terisi, paling maksimum terdapat 9 angka yang dapat diisikan pada *grid* tersebut. Jika tidak terdapat angka yang mungkin diisikan pada suatu *grid* kosong, maka dilakukan *backtracking*. Oleh karena itu, kompleksitas dari strategi

penyelesaikan ini adalah $O(9^{n \cdot n})$ dengan n merupakan panjang atau lebar dari Sudoku (dalam kasus ini nilai n adalah 9).

Strategi selanjutnya yang dapat diimplementasikan sebagai solusi dari Sudoku adalah *stochastic search* atau pencarian secara stokastik. Strategi ini menggunakan *random based algorithms* serta statistika untuk mencari solusi dari *puzzle* Sudoku. Beberapa contoh dari algoritma yang digunakan oleh strategi ini adalah *simulated annealing*, *genetic algorithm*, *tabu search*, dan *Boltzmann machine*. Seluruh algoritma tersebut secara garis besar bekerja dengan langkah-langkah sebagai berikut. Pertama – tama, tetapkan angka secara "acak" pada setiap *grid* kosong. Kemudian, hitung berapa jumlah *error* yang terjadi pada urutan "acak" tersebut. Terakhir, "acak" kembali seluruh angka *error* pada Sudoku hingga tidak ada *error* yang terjadi. Algoritma stokastik yang telah dipaparkan sebelumnya mengatur cara "pengacakan" yang terjadi pada *grid*. Sama seperti strategi gabungan dari *brute force* dan *backtracking*, kompleksitas algoritma dari strategi penyelesaian ini adalah $O(9^{n \cdot n})$. Tetapi, secara heuristik strategi ini memiliki waktu penyelesaian lebih cepat dikarenakan adanya optimisasi cara "pengacakan" *grid* oleh algoritma stokastik tersebut.

Terakhir, strategi yang dapat diimplementasikan sebagai solusi dari Sudoku adalah dengan menggunakan gabungan dari *Crook's pencil-and-paper algorithm* serta *backtracking*. *Crook's pencil-and-paper algorithm* merupakan suatu algoritma *greedy* yang bertujuan untuk melakukan *pruning* ruang solusi yang akan dicek oleh algoritma *backtracking*. *Crook's pencil-and-paper algorithm* merupakan algoritma yang diformulasikan oleh J. F. Crook pada *paper* yang beliau publikasikan dengan judul "*A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles*". *Crook's pencil-and-paper algorithm* dinamakan seperti itu dikarenakan algoritma tersebut relatif mudah untuk dikerjakan secara langsung, tanpa bantuan komputer. Tetapi, algoritma yang beliau formulasikan menggunakan beberapa konsep matematika, seperti *occupancy theorem* dan *preemptive sets*, membutuhkan nalar manusia untuk menggunakannya secara efektif. Pada kasus terburuknya, kompleksitas algoritma dari strategi penyelesaian ini adalah $O(9^{n \cdot n})$. Tetapi, sama seperti algoritma stokastik, secara heuristik strategi ini memiliki waktu penyelesaian yang sangat cepat jika dibandingkan dengan strategi *brute force*.

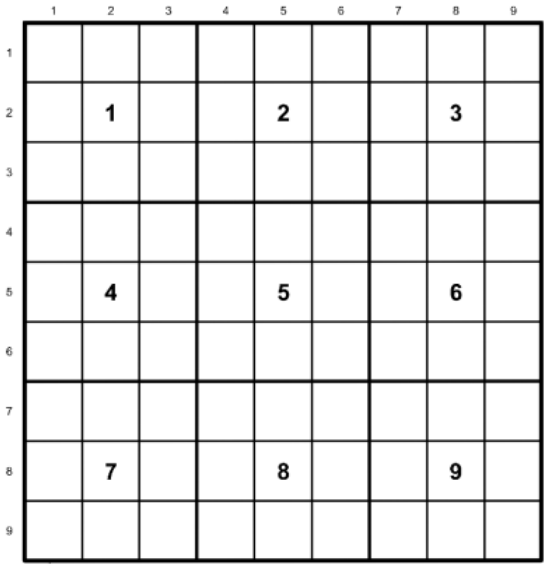
Jika diperhatikan secara seksama, seluruh strategi dan algoritma yang telah dipaparkan oleh penulis tidak mempunyai kompleksitas algoritma dengna bentuk polinomial. Hal ini dikarenakan *puzzle* Sudoku merupakan salah satu dari permasalahan *NP-Complete*, dimana solusi dari Sudoku dapat dikonfirmasi kebenarannya secara cepat (dengan kompleksitas waktu polinomial $O(n^2)$), tetapi untuk menemukan solusi secara tepat, dibutuhkan algoritma pencarian secara *brute force*.

Secara heuristik, strategi gabungan dari *Crook's pencil-and-paper algorithm* serta *backtracking* merupakan strategi dengan waktu penyelesaian yang paling singkat diantara ketiga strategi tersebut. Oleh karena itu, penulis mengangkat strategi tersebut sebagai pokok bahasan utama pada Tugas Makalah IF2211 Strategi Algoritma. Alasan lain penulis menangkat strategi tersebut sebagai pokok bahasa utama adalah *Crook's pencil-and-paper algorithm* merupakan algoritma yang mudah jika dikerjakan dengan bantuan nalar manusia, tetapi agak sulit jika hendak diimplementasikan di dalam program komputer.

II. Landasan Teori

A. *Definisi dari Sudoku*

Sudoku dimainkan pada papan dengan ukuran 9 × 9 *grid*. Terdapat total 81 *grid* yang dapat diisikan oleh angka pada papan, yang mana akan dikelompokkan kembali menjadi sembilan *subboard* dengan ukuran 3 × 3 *grid* dan seluruh *subboard* tersebut tidak berpotongan. *Subboard* tersebut juga dapat dikatakan dengan *box*, dengan indeks 1 hingga 9 jika dihitung dari kiri atas hingga kanan bawah.



Gambar 2.1. *Box* pada Sudoku

Sumber: ams.org

Notasi yang digunakan untuk mereferensikan suatu *grid* pada papan Sudoku adalah dengan menspesifikasikan nomor baris, kemudian baru menspesifikasikan nomor kolomnya. Sebagai contoh, *grid* pada baris ke-6 dan kolom ke-7 akan direferensikan dengan notasi $c(6, 7)$.

Objektif atau tujuan akhir yang hendak dicapai oleh *puzzle* Sudoku tersebut dapat dikonkretkan menjadi pernyataan definisi sebagai berikut:

Definisi 1 (*Sudoku Solution*): *The solution of a Sudoku puzzle requires that every row, column, and box contain all the numbers in the set* $[1, 2, 3, \ldots, 9]$ *and that every grid be occupied by one and only one number.*

Definisi tersebut mengimplikasikan bahwa tidak terdapat baris, kolom, maupun *box* yang memiliki *grid* dengan angka berulang. Oleh karena itu, salah satu langkah yang dilakukan pada *Crook's pencil-and-paper algorithm* adalah dengan menuliskan *list* seluruh kemungkinan angka yang mungkin menempati suatu *grid* pada setiap *grid* di

papan Sudoku. *List* dari setiap kemungkinan angka tersebut juga dapat dikatakan dengan *markup* dari *grid*.

Hal selanjutnya yang harus didefinisikan adalah *preemptive sets*. *Preemptive sets* merupakan salah satu struktur data utama yang digunakan pada *Crook's pencil-and-paper algorithm*. *Preemptive sets* itu sendiri digunakan untuk melakukan *pruning* pada ruang solusi Sudoku, hingga *grid* tersebut hanya dapat diisi oleh satu angka saja, atau *markup* yang terbentuk pada papan Sudoku sudah paling sederhana. Definisi formal dari *preemptive sets* adalah sebagai berikut:

Definisi 2 (*Preemptive Sets*): *A preemptive set is composed of numbers from the set* $[1, 2, 3, \dots, 9]$ *and is a set of size* $m, 2 \leq m \leq 9$, *whose numbers are potential occupants of* m *grids exclusively, where exclusively means that no other numbers in the set* $[1, 2, 3, \dots, 9]$ *other than the members of the preemptive set are potential occupants of those* m *grids. A preemptive set is denoted by:*

$$\{[n_1, n_2, n_3, \dots, n_m], [c(i_1, j_1), c(i_2, j_2), c(i_3, j_3), \dots, c(i_m, j_m)]\}$$

Where $[n_1, n_2, n_3, \dots, n_m], 1 \leq n_i \leq 9 \; for \; i = 1, 2, 3, \dots, m$, *denotes the set of numbers in the preemptive set and* $[c(i_1, j_1), c(i_2, j_2), c(i_3, j_3), \dots, c(i_m, j_m)]$ *denotes the set of* m *grids in which the set* $[n_1, n_2, n_3, \dots, n_m]$, *and subsets thereof, exclusively occur.*

Pada *Crook's pencil-and-paper algorithm*, *preemptive sets* tidak bisa diaplikasikan secara pada kumpulan *grid* secara sembatang. Terdapat *range – range* tertentu dimana suatu *preemptive sets* dapat diformulasikan dan digunakan. Definisi formal dari *range* tersebut adalah sebagai berikut:

Definisi 3 (*Range of Preemptive Sets*): *The range of a preemptive set is a* row, $column$, *or* box *in which all of the grids of the preemptive set are located. When* $m = 2$ *or* $m = 3$, *the range can be one of the sets* $[row, box]$ *or* $[column, box]$.

Ketika suatu *preemptive set* terbentuk, sebenarnya secara tidak langsung hal tersebut mengimplikasikan bahwa pada *range* tersebut, himpunan m nilai yang terkandung dialokasikan secara eksklusif pada setiap *grid* yang ada pada *preemptive set*. Hal ini akan dibahas lebih detil pada bagian pembahasan teorema-teorema yang berlaku pada Sudoku.

B. Teorema Sudoku

Beberapa teorema yang digunakan oleh *Crook's pencil-and-paper algorithm* adalah *occupancy theorem* serta *preemptive sets*. Kedua teorema tersebut berfungsi sebagai *rule* dan/atau *guide* yang digunakan dalam mencari solusi Sudoku. Pertama – tama, akan dibahas terlebih dahulu mengenai *occupancy theorem*. Definisi formal beserta bukti dari *occupancy theorem* adalah sebagai berikut:

Teorema 1 (*Occupancy Theorem*): *Let X be a preemptive set in a Sudoku puzzle markup. Then every number in X that appears in the markup of grids not in X over the range of X cannot be a part of the puzzle solution.*

Bukti. *If any number in X is chosen as the entry for a grid not in X, then the number of numbers to be distributed over the* m *grids in X will be reduced to* $m - 1$, *which means that one of the* m *grids in X will be unoccupied, which in turn violates the Sudoku solution definition. Hence, to continue a partial solution, all numbers in X must be eliminated wherever they occur in grids not in X over the range of X*.

Teorema tersebut digunakan untuk melakukan *pruning* pada *markup* pada *range* dari *preemptive sets*. Hal tersebut sangatlah berguna dikarenakan jumlah kemungkinan ruang solusi yang terbentuk pada Sudoku menjadi berkurang. Tidak hanya itu, teorema ini dapat digunakan secara berulang – ulang pada setiap baris, kolom, maupun *box* dengan nilai m yang berbeda – beda pada setiap iterasinya.

Teorema yang dibahas selanjutnya adalah teorema *preemptive sets*. Definisi formal dari teorema *preemptive sets* adalah sebagai berikut:

Teorema 2 (*Preemptive Sets*): *There is always a preemptive set that can be invoked to unhide a hidden set, which then changes the hidden set into a preemptive set except in the case of singleton.*

Teorema tersebut menyatakan bahwa terdapat suatu *hidden set* pada baris, kolom, maupun *box* yang dapat dibentuk menjadi suatu *preemptive sets* dengan *range* masing-masing. Teorema ini merupakan bagian yang menyebabkan *Crook's pencil-and-paper algorithm* relatif mudah digunakan oleh manusia, tetapi tidak oleh komputer. Hal ini dikarenakan untuk mencari suatu *hidden set*, dibutuhkan pencarian secara *brute force* oleh komputer, dimana manusia dapat melihatnya secara jelas. Tetapi, hal tersebut tidak menyebabkan *Crook's pencil-and-paper* tidak bisa diimplementasikan oleh komputer, hanya saja cara yang digunakannya cukup kompleks dan memakan waktu.

C. Algoritma Pensil-dan-Kertas Crook

James R. Crook, seorang profesor ilmu komputer pada Universitas Winthrop mempublikasikan suatu *paper* yang berjudul "*A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles*", dimana beliau membuat suatu algoritma untuk menyelesaikan *puzzle* Sudoku dengan hanya menggunakan pensil dan kertas. Secara garis besar, algoritma yang digunakan oleh Prof. Crook tersebut cukup sederhana dan mudah diimplementasikan oleh manusia. Algoritma ini dibangun dari kumpulan berbagai teorema dan metode matematis yang digunakan untuk menyelesaikan Sudoku. Akan tetapi, metode yang beliau gunakan terhitung cukup mudah dihafal, tidak seperti metode-metode lainnya yang membutuhkan pengguna untuk menghafalkan berbagi jenis pola pada papan Sudoku.

Berikut ini merupakan 5 langkah utama yang menjadi dasar dari *Crook's pencil-and-paper algorithm*. Setiap

langkah yang dijabarkan tersebut harus dikembangka lebih lanjut dengan algoritma dan struktur data lebih spesifik jika ingin diimplementasikan menggunakan program komputer, tetapi jika hanya dengan menggunakan kertas dan pensil, hal tersebut tidak diperlukan.

- Langkah 1: Tambahkan *markup* pada seluruh *grid* di papan Sudoku. *Markup* merupakan *list* dari semua kemungkinan angka yang dapat diisi pada *grid*.
- Langkah 2: Cari *singleton* (jumlah elemen pada *list markup* adalah 1) pada seluruh *grid* di papan Sudoku. Hal ini berguna untuk mengeliminasi *grid* yang pasti terisi oleh suatu angka tertentu.
- Langkah 3: Cari seluruh *preemptive sets* pada seluruh baris, kolom, *box*, kombinasi baris-*box*, dan kombinasi kolom-*box* di papan Sudoku. Menurut teorema *preemptive sets*, akan selalu terdapat suatu *preemptive sets* dalam suatu *hidden sets*. Langkah ini merupakan salah satu langkah yang *tricky* dikarenakan dibutuhkannya ketelitian untuk mencari seluruh *preemptive sets* dari seluruh *grid* pada Sudoku. Perlu diperhatikan pula, pengguna harus mengecek seluruh kemungkinan *range* yang terdapat di Sudoku, dimana *range* tersebut berupa baris, kolom, *box*, baris-*box*, serta kolom-*box*.
- Langkah 4: Hilangkan seluruh *markup* diluar *range* dari *preemptive sets* yang telah ditemukan. Agar memaksimalkan hasil, diusahakan menghilangkan *markup* yang berasal dari *preemptive sets* dengan *m* yang lebih besar terlebih dahulu. Hal tersebut akan dibahas lebih lanjut pada pembahasan *Crook's pencil-and-paper algorithm* sebagai algoritma *greedy*.
- Langkah 5: Jika seluruh *grid* telah terisi, maka akhiri algoritma. Jika terdapat suatu *grid* yang kosong pada papan Sudoku, maka pengguna harus melakukan pencocokan solusi yang mungkin terbentuk. Pada program komputer, hal tersebut dapat dicapai dengan menggunakan algoritma *backtracking* dengan menggunakan struktur data *search tree* untuk melakukan pencarian pada solusi pada ruang solusi yang telah dilakukan *pruning*. Pada kertas, pengguna dapat melakukan pencocokan setiap kemungkinan angka yang mungkin dengan melakukan penebakkan atas angka pada *grid* tersebut.

Crook's pencil-and-paper algorithm dikatakan sebagai algoritma *greedy* dikarenakan pada langkah ke-4, terdapat eliminasi *markup* pada *range* dari *preemptive sets* yang bersesuaian dengan tujuan memaksimalkan jumlah *markup* yang dihilangkan pada papan Sudoku. Hal tersebut dicapai dengan menggunakan *preemptive sets* secara berurut dari yang terbesar hingga terkecil. Urutan tersebut merupakan urutan yang menyebabkan maksimum lokal, dimana pada *algoritma greedy* setiap langkah yang menghasilkan optimum lokal diharapkan langkah – langkah sisanya akan mengarah pada optimum global. Berikut ini merupakan elemen – elemen algoritma *greedy* pada *Crook's pencil-and-paper algorithm*:

- Himpunan kandidat: himpunan *preemptive sets* yang dapat diambil dari suatu *range* tertentu pada papan Sudoku.
- Himpunan solusi: himpunan *preemptive sets* yang telah terpilih.
- Fungsi solusi: memeriksa apakah seluruh *hidden sets* pada *range* tertentu pada papan Sudoku sudah habis.
- Fungsi seleksi: memilih kandidat *preemptive sets* dengan nilai *m* paling besar.
- Fungsi kelayakan: memeriksa apakah *preemptive sets* yang terbentuk sudah sesuai dengan definisi dari *preemptive sets* tersebut (*range* bersesuaian serta jumlah angka sama dengan jumlah *grid*).
- Fungsi objektif: memaksimalkan jumlah *markup* yang tereliminasi oleh *preemptive sets*.

D. Algoritma Backtracking pada Permasalahan Sudoku

Tidak semua *puzzle* Sudoku memiliki solusi unik atas seluruh *grid* pada papannya. Oleh karena itu, *Crook's pencil-and-paper algorithm* juga dapat menghasilkan papan Sudoku dengan seluruh solusi yang mungkin. Jika suatu papan Sudoku memiliki berbagai solusi, maka tidak setiap *grid* pada papan Sudoku yang merupakan hasil perhitungan dari *Crook's pencil-and-paper algorithm* akan terisi dengan angka. Sebagai gantinya, untuk menjabarkan seluruh kemungkinan solusi papan Sudoku, *grid* tersebut terisi dengan *markup* tertentu yang jika dikombinasikan dengan *markup* pada *grid* kosong lainnya, akan menghasilkan beberapa solusi dari papan Sudoku. Untuk melakukan *traversing* ruang solusi yang terbentuk, dibutuhkan algoritma *backtracking* dengan menggunakan *search tree* untuk mencari seluruh kemungkinan solusi tersebut. Berikut ini merupakan definisi formal dari konsep pencarian solusi pada papan Sudoku adalah sebagai berikut:

Kondisi 1 (*Random Choice*): *When no preemptive set in any row, column, or box can be broken into smaller preemptive sets, then an empty grid must be chosen and a number randomly chosen from the grid's markup in order to continue solving the puzzle.*

Kondisi tersebut merupakan ide basis dari fungsi pembangkit yang akan digunakan pada algoritma *backtracking*. Dengan memilih angka *random* dari suatu *markup* pada *grid*, program dapat melakukan *traversing* seluruh ruang solusi yang terdefinisi. Oleh karena itu, untuk memudahkan *traversing*, digunakan struktur data *search tree* agar mempermudah pencatatan setiap *node* yang telah dikunjungi.

Terakhir, agar algoritma *backtracking* dapat dijalankan, program perlu mendefinisikan suatu konsep yang dinamakan fungsi pembatas. Fungsi pembatas yang didefinisikan pada kasus ini adalah papan Sudoku yang terbentuk pada *traversing* ruang solusi bukanlah suatu solusi papan Sudoku yang tepat (lihat Definisi 1). Definisi formal dari solusi papan Sudoku yang tidak tepat adalah sebagai berikut:

Definisi 4 (*Sudoku Violation*): *A violation in Sudoku occurs when the same number occurs two or more times in the same row, column, or box.*

Oleh karena itu, dilakukan pengecekkan pada setiap *grid* yang terbentuk pada *traversing* ruang solusi. Agar mempersingkat waktu eksekusi pada program, ketika dibentuk suatu solusi papan Sudoku yang baru, selain ditambahkan angka pada *grid* kosong tersebut juga mengurangi kemunculan angka tersebut pada *markup* dengan *grid* yang memiliki baris, kolom, atau *box* dengan *grid* yang baru diisi tersebut.

E. Penentuan Jumlah Solusi Sudoku

Dalam konsep matematis, suatu *puzzle* Sudoku dapat dimodelkan sebagai *vertex coloring problem* pada teorema graf. Hal ini dapat dilakukan dengan mengganti angka – angka dari 1 hingga 9 yang terdapat di dalam *grid* Sudoku dengan sembilan warna graf yang berbeda – beda, mengganti papan Sudoku menjadi graf, serta mengganti setiap *grid* Sudoku menjadi *vertex*. Objektif yang harus dicapai untuk menyelesaikan Sudoku berubah menjadi mewarnai graf tersebut sedemikian rupa sehingga masing-masing sembilan warna terdapat pada setiap baris, kolom, dan *box*. Teorema pewarnaan graf dengan cara tersebut dinamakan *proper graph coloring*. Jumlah warna minimum yang dapat mewarnai setiap *vertex* tanpa terdapat dua *vertex* berhubungan memiliki warna yang sama pada suatu graf (misalkan graf G) disebut bilangan kromatik (disimbolkan dengan $\lambda(G)$). Definisi formal dari teorema pewarnaan graf G adalah sebagai berikut:

Teorema 3 (*Completion Chromatic Polynomial*): *Let G be a finite graph with v vertices. Let C be a partial coloring of t vertices of G using d_0 colors. Let $p_{G,C}(\lambda)$ be the number of ways of completing the coloring using λ colors to obtain a proper coloring of G. Then $p_{G,C}(\lambda)$ is a monic polynomial (in λ) with integer coefficients of degree $v - t$ for $\lambda \geq d_0$.*

Jumlah solusi dari suatu persoalan *puzzle* Sudoku merupakan bilangan polinomial kromatik dari sembilan warna ($p_{G,C}(9)$) dari representasi papan Sudoku sebagai graf G, dimana setiap *grid* berhubungan dua arah dengan seluruh *grid* yang berada pada baris, kolom serta *box* dari *grid* itu sendiri.

III. IMPLEMENTASI

A. Struktur Program

Struktur *directory tree* dari *source code* implementasi program adalah sebagai berikut:

```
src
|   App.java
|
├──algorithms
|       SearchTree.java
|
├──datastructures
|       Cell.java
|       Column.java
|       Grid.java
|       Markup.java
|       PreemptiveSet.java
|       Row.java
|       Sudoku.java
|       SudokuException.java
|
└──utils
        CSVReader.java
```

Gambar 3.1. Struktur *directory tree* program

Sumber: dokumen penulis

Dapat dilihat pada gambar diatas, *directory* program terdiri atas tiga *package* serta beberapa file *source code* yang ditulis dengan menggunakan bahasa Java. Ketiga *package* tersebut adalah *algorithms*, *datastructures*, serta *utils*. Satu – satunya file *source code* yang terletak diluar ketiga *package* tersebut adalah App.java, yang merupakan *driver* dari keseluruhan program ini. Setiap *package* merupakan folder yang mengandung file *source code* yang memiliki fitur dan fungsionalitas yang mirip dan sering berinteraksi satu sama lain. Untuk mengakses *source code* selengkapnya, dapat diakses menggunakan pranala ini.

B. Package algorithms

Package ini berisi algoritma yang digunakan untuk melakukan pencarian solusi Sudoku. Dengan menggunakan kelas *SearchTree*, program memasukkan suatu objek Sudoku yang telah dibaca dari file dengan format csv. Setelah itu, program akan memanggil metode *solve* yang terdapat pada objek Sudoku itu sendiri agar dilakukan *pruning* menggunakan *Crook's pencil-and-paper algorithm*. Jika semua *grid* terisi oleh algoritma tersebut, maka program berhenti. Jika terdapat *grid* yang belum terisi oleh angka, maka akan dilakukan *backtracking* yang diimplementasikan dengan *while loop* serta struktur data *stack*.

C. Package datastructures

Package ini berisi struktur data utama yang digunakan untuk merepresentasikan papan Sudoku pada program. Kelas pertama yang dibahas adalah *Cell*. Kelas *Cell* merupakan representasi *grid* dari papan Sudoku pada program. Kemudian, kelas yang berikutnya akan dibahas adalah *Markup*. Kelas *Markup* merupakan representasi *list markup* yang terdapat pada setiap *grid* di papan Sudoku pada program. Kelas *Markup* itu sendiri merupakan kelas *collection* terhadap nilai-nilai yang dicatat pada kelas *Cell*. Selanjutnya, kelas yang berikutnya akan dibahas adalah *PreemptiveSets*. Kelas *PreemptiveSets* merupakan representasi dari *preemptive sets* yang telah didefinisikan sebelumnya. Selain itu, kelas *PreemptiveSets* merupakan

kelas *collection* terhadap kelas *Cell* yang terdapat pada papan Sudoku. Selanjutnya, kelas yang akan dibahas adalah *Row*, *Column*, dan *Grid*. Ketiga kelas tersebut memiliki fungsionalitas yang hampir mirip, yaitu sebagai *wrapper class* terhadap *buffer* yang terdiri atas kumpulan kelas *Cell*. Terakhir, kelas yang akan dibahas adalah *Sudoku* dan *SudokuException*. Kelas *Sudoku* merupakan representasi papan Sudoku pada program komputer, sedangkan kelas *SudokuException* merupakan kelas turunan dari *Exception* yang berfungsi untuk melakukan *exception handling* untuk hal – hal yang berhubungan dengan kelas *Sudoku*. Kelas *Sudoku* juga mengandung pengimplementasian dari *Crook's pencil-and-paper algorithm* pada metode *solve*.

D. *Package utils*

Package ini berisi *tools* atau *utilities* yang digunakan untuk pembacaan file dengan format csv (*comma separated values*) dan mengubahnya menjadi objek *Sudoku*. Dengan menggunakan kelas *CSVReader*, program dapat menerima masukan *path* file dari pengguna, kemudian mencarinya pada *filesystem*, dan terakhir membacanya menjadi objek *Sudoku*.

E. *Pengujian*

Berikut ini merupakan salah satu contoh kasus penyelesaian Sudoku menggunakan program yang telah diimplementasikan. Penulis menggunakan *puzzle* yang terdapat pada buku "*Solving Sudoku*" oleh Michael Mepham (2005). Penulis mengambil contoh kasus dengan tingkat kesulitan *diabolical*, dimana diperlukan algoritma *backtracking* untuk mencari solusi dari Sudoku tersebut.

	9		7			8	6	
	3	1			5		2	
8		6						
		7		5				6
			3		7			
5				1		7		
						1		9
	2		6			3	5	
	5	4			8		7	

Gambar 3.2. Contoh persoalan Sudoku dengan tingkat kesulitan *diabolical*

Sumber: Solving Sudoku

```
2 9 5 | 7 4 3 | 8 6 1
4 3 1 | 8 6 5 | 9 2 7
8 7 6 | 1 9 2 | 5 4 3
------+-------+------
3 8 7 | 4 5 9 | 2 1 6
6 1 2 | 3 8 7 | 4 9 5
5 4 9 | 2 1 6 | 7 3 8
------+-------+------
7 6 3 | 5 2 4 | 1 8 9
9 2 8 | 6 7 1 | 3 5 4
1 5 4 | 9 3 8 | 6 7 2

Waktu eksekusi: 0.142 s
```

Gambar 3.3. Solusi dari persoalan Sudoku dengan tingkat kesulitan *diabolical*

Sumber: dokumen penulis

Pada *repository source code*, penulis telah menyiapkan beberapa tes kasus lainnya dengan derajat kesulitan dan tingkat *backtracking* yang berbeda – beda. Seluruh tes kasus tersebut ditulis dengan menggunakan format csv sebagai format masukkan file utama program.

IV. KESIMPULAN

Sudoku merupakan suatu *puzzle* yang membutuhkan kemampuan berlogika serta kombinatorial angka untuk menyelesaikannya. Oleh karena itu, terdapat banyak strategi pemecahan Sudoku, diantaranya adalah *brute force*, *backtracking*, *stochastic search*, dan *Crook's pencil-and-paper algorithm*. Pada makalah ini, penulis mengambil strategi *Crook's pencil-and-paper algorithm* yang merupakan algoritma *greedy* untuk melakukan *pruning* terhadap ruang solusi Sudoku yang terbentuk. *Crook's pencil-and-paper algorithm* itu sendiri bekerja dengan menggunakan konsep matematis *preemptive sets* untuk melakukan partisi terhadap *grid* dan eliminasi *markup*. *Preemptive sets* itu sendiri dapat ditarik dari *hidden set* yang terdapat pada setiap baris, kolom, *box* (*grid* dengan ukuran 3 × 3), baris-*box*, serta kolom-*box*. Jika tidak terdapat *hidden set* pada Sudoku, maka *Crook's pencil-and-paper algorithm* sudah mengeliminasi ruang solusi hingga bentuk paling sedikit. Jika ternyata masih terdapat *grid* kosong, maka dilakukan algoritma *backtracking* dengan menggunakan *search tree* dengan kemungkinan angka yang terdapat pada masing – masing *markup* dari *grid* kosong tersebut hingga ditemukan solusi yang diinginkan.

VIDEO LINK AT YOUTUBE

https://youtu.be/dWvNIYwUlEg

ACKNOWLEDGMENT

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan kasih karunia-Nya yang telah memberikan kesehatan dan kesempatan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih sebesar – besarnya kepada seluruh pihak yang telah membantu penulis dalam menyelesaikan makalah ini, diantaranya adalah:

1. Dr. Nur Ulfa Maulidevi, S.T, M.Sc selaku dosen pengampu kelas K02 mata kuliah IF2211 Strategi Algoritma.
2. Seluruh pihak yang telah membuat materi secara *open-source* yang penulis kutip pada makalah ini.

REFERENCES

[1] Spittel, Ali. 2018. *Coming Back to Old Problems: How I Finally Wrote a Sudoku Solving Algorithm*, https://dev.to/aspittel/how-i-finally-wrote-a-sudoku-solver-177g, diakses pada 18 Mei 2022.

[2] GeeksforGeeks. 2022. *Sudoku Backtracking-7*, https://www.geeksforgeeks.org/sudoku-backtracking-7/, diakses pada 18 Mei 2022.

[3] Christine, Sharon. 2018. *Sudoku Solving Algorithms*, https://www.tutorialspoint.com/Sudoku-Solving-algorithms, diakses pada 18 Mei 2022.

[4] Fok, Wy. 2020. *Solve Sudoku more elegantly with Crook's algorithm in Python*, https://towardsdatascience.com/solve-sudoku-more-elegantly-with-crooks-algorithm-in-python-5f819d371813, diakses pada 19 Mei 2022.

[5] Bartel, Grant. 2017. *How To Win Sudoku*, https://towardsdatascience.com/how-to-win-sudoku-3a82d05a57d, diakses pada 19 Mei 2022.

[6] J. F. Crook. 2009. *A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles*, https://www.ams.org/notices/200904/tx090400460p.pdf, diakses pada 19 Mei 2022.

[7] Mepham, Michael. 2005. *Solving Sudoku*, http://www.sudoku.org.uk/PDF/Solving_Sudoku.pdf, diakses pada 19 Mei 2022.

[8] Berggren, Patrik dan David Nilsson. *A study of Sudoku solving algorithms*, https://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group6Alexander/final/Patrik_Berggren_David_Nilsson.report.pdf, diakses pada 19 Mei 2022.

[9] Kaya, Ali. 2020. *The Mathematics Behind Sudoku: Solving Strategy*, https://abakcus.com/article/the-mathematics-behind-sudoku-solving-strategy/, diakses pada 19 Mei 2022.

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 2022

Rayhan Kinan Muhannad 13520065