Penerapan Deretan, Sumasi, dan Rekursi dalam Penyelesaian Sudoku Menggunakan Algoritma *Backtracking*

Satria Octavianus Nababan - 13521168[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]satriaoctavianus28@gmail.com, 13521168@std.stei.itb.ac.id

***Abstract*— Program Sudoku Solver yang dibangun dalam makalah ini mengaplikasikan tiga konsep utama yaitu deretan, sumasi, dan rekursi untuk menyelesaikan tantangan permainan Sudoku. Penerapan deretan digunakan untuk merepresentasikan papan Sudoku dalam bentuk array dua dimensi, sedangkan sumasi diterapkan untuk menghitung total nilai elemen-elemen pada papan. Rekursi, khususnya pada algoritma *backtracking*, diterapkan untuk mencoba mengisi papan dengan angka yang sesuai dengan aturan Sudoku. Proses penyelesaian disertai dengan penjelasan langkah demi langkah melalui antarmuka grafis. Program diuji dengan dua iterasi untuk mengukur performa dan efisiensinya, yang menunjukkan variasi waktu penyelesaian berdasarkan kompleksitas papan Sudoku.**

Keywords*—Sudoku, deretan, sumasi, rekursi, *backtracking

I. PENDAHULUAN

Permainan Sudoku merupakan salah satu teka-teki angka yang populer di seluruh dunia, dikenal karena tantangannya yang melibatkan logika dan pola pengisian angka. Dalam Sudoku, pemain diharuskan untuk mengisi grid 9x9 dengan angka 1 hingga 9 sehingga tidak ada angka yang berulang di setiap baris, kolom, dan subgrid 3x3. Penyelesaian Sudoku memerlukan pendekatan sistematis yang dapat memanfaatkan konsep matematika, seperti deretan, sumasi, dan rekursi.

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

Gambar 1. Ilustrasi Permainan Sudoku
Sumber: sudoku-puzzles.net

Konsep deretan dan sumasi relevan dalam memastikan aturan dasar permainan, yakni bahwa setiap deretan angka dalam baris, kolom, dan subgrid memenuhi sifat unik dan jumlah yang teratur. Di sisi lain, rekursi menjadi alat yang sangat efektif dalam menerapkan algoritma *backtracking* untuk menemukan solusi. Algoritma *backtracking* memungkinkan eksplorasi setiap kemungkinan solusi dengan cara *trial-and-error* secara sistematis, hingga ditemukan konfigurasi angka yang memenuhi semua aturan Sudoku.

Penelitian dan pengembangan algoritma untuk menyelesaikan Sudoku tidak hanya menarik dari segi akademis, tetapi juga memiliki aplikasi praktis, seperti dalam bidang kecerdasan buatan dan optimasi. Makalah ini bertujuan untuk membahas penerapan konsep deretan, sumasi, dan rekursi dalam penyelesaian Sudoku menggunakan algoritma *backtracking*. Analisis akan mencakup penjelasan teoretis, implementasi algoritmik, dan evaluasi efisiensi metode yang digunakan.

II. LANDASAN TEORI

A. Deretan dan Sumasi

Deretan (*sequences*) atau barisan (*series*) adalah daftar terurut (*ordered list*) elemen-elemen diskrit. Elemen-elemen ini dapat berupa bilangan, simbol, atau elemen dari himpunan lainnya. Dalam matematika diskrit, deretan biasanya direpresentasikan sebagai fungsi dari *subset* bilangan bulat (N untuk bilangan alami atau P untuk bilangan bulat positif) ke sebuah himpunan S.

- Deretan bilangan genap positif 2,4,6,8, ...,dapat direpresentasikan sebagai fungsi $a(n) = 2n$ untuk $n \in N$.
- Deretan bilangan Fibonacci: 0,1,1,2,3,5, ..., direpresentasikan dengan relasi rekurens $F(n) = F(n-1) + F(n-2)$ untuk $n \geq 2$.

String adalah bentuk khusus dari deretan yang elemen-elemennya berupa karakter. Beberapa contoh string:

- "informatika" adalah string dengan panjang 11 karakter.
- "10100101" adalah string biner dengan panjang 8 bit.
- *String* kosong (λ) memiliki panjang 0.

Penggunaan *string* sebagai deretan banyak ditemukan dalam aplikasi pemrograman, seperti analisis teks, kompresi data, dan pemrosesan sinyal digital.

Penjumlahan elemen-elemen dalam deretan, yang disebut sumasi, didefinisikan sebagai jumlah elemen-elemen deretan dalam interval tertentu. Penjumlahan ini direpresentasikan menggunakan notasi sumasi (Σ). Jumlah deretan $am, am + 1, am + 2, \dots, an$ adalah $am + am + 1 + am + 2 + \dots + an$ atau dalam notasi sumasi: $\sum_{k=m}^{n} a_k$

Notasi sumasi digunakan secara luas untuk menganalisis efisiensi algoritma, terutama saat menghitung jumlah operasi dalam struktur kontrol seperti perulangan. Sumasi ganda adalah penjumlahan deretan yang dilakukan di dalam struktur bersarang (*nested loop*). Hal ini sering digunakan untuk menghitung total operasi dalam algoritma yang melibatkan *loop* bersarang. Contoh sumasi ganda $\sum_{i=1}^{4} \sum_{j=1}^{3} ij$

Deretan, *string*, dan sumasi memberikan dasar untuk memahami struktur dan efisiensi algoritma. Sumasi ganda, khususnya, memungkinkan kita untuk menghitung kompleksitas algoritma secara akurat, terutama dalam kasus algoritma dengan *loop* bersarang. Hal ini relevan dalam analisis algoritma pemecahan masalah seperti Sudoku, di mana setiap angka yang dimasukkan memerlukan validasi terhadap deretan angka di baris, kolom, dan *subgrid*.

B. Rekursi

Rekursi adalah konsep fundamental dalam matematika diskrit dan ilmu komputer, di mana suatu objek atau proses didefinisikan dalam terminologi dirinya sendiri. Pendekatan ini memungkinkan penyelesaian masalah yang kompleks dengan membaginya menjadi submasalah yang lebih kecil dan menyelesaikannya secara iteratif atau rekursif. Dalam algoritma penyelesaian Sudoku, rekursi menjadi metode utama untuk mengeksplorasi berbagai kemungkinan solusi melalui proses sistematis yang bersifat otomatis.

Metode rekursi mendefinisikan suatu objek atau fungsi dalam istilah dirinya sendiri. Contoh umum dari objek rekursif adalah fraktal, yang menunjukkan pola berulang pada berbagai skala, dan pohon biner, di mana setiap cabangnya dapat dianggap sebagai subpohon. Proses rekursi terdiri dari dua komponen utama:

1. Basis adalah bagian yang mendefinisikan kondisi awal atau nilai eksplisit, yang menghentikan proses rekursi.
2. Rekurens adalah bagian yang mendefinisikan hubungan antara nilai sekarang dengan nilai-nilai sebelumnya.

Kombinasi basis dan rekurens memastikan bahwa proses rekursi memiliki titik akhir dan menghasilkan nilai yang valid.

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri selama eksekusi. Struktur fungsi rekursif terdiri dari dua elemen utama:

- Basis, yaitu kondisi penghentian untuk mencegah *loop* tak berujung.
- Rekurens, yaitu proses iteratif yang menggunakan hasil dari pemanggilan sebelumnya untuk menghasilkan nilai baru.

Sebagai contoh, panjang string $L(S)$ dapat didefinisikan secara rekursif:

- Basis: L(λ)= 0, di mana λ adalah string kosong.
- Rekurens: $L(wx) = L(w) + 1$, di mana w adalah string dan x adalah karakter terakhirnya.

Dalam algoritma Sudoku Solver, rekursi digunakan untuk mencoba memasukkan angka ke dalam sel kosong berdasarkan aturan permainan. Setiap langkah bergantung pada hasil dari langkah sebelumnya untuk memastikan solusi valid.

Struktur data seperti pohon biner adalah contoh nyata dari penerapan rekursi. Pohon biner didefinisikan sebagai berikut:

- Basis: Pohon kosong adalah pohon biner.
- Rekurens: Jika $T1$ dan $T2$ adalah pohon biner, maka pohon dengan $T1$ dan $T2$ sebagai subpohon juga merupakan pohon biner.

Setiap simpul dalam pohon biner dapat dianggap sebagai akar dari sebuah subpohon, sehingga definisi pohon biner sepenuhnya bersifat rekursif. Dalam algoritma *backtracking* untuk Sudoku, setiap langkah mencoba solusi untuk satu sel, menciptakan percabangan baru dalam struktur pohon solusi.

Deretan rekursif adalah deretan bilangan yang didefinisikan dengan aturan rekursif, di mana setiap elemen bergantung pada elemen-elemen sebelumnya.

- Deretan pangkat dua
  - Basis: $a_0 = 1$
  - Rekurens: $An = 2An - 1$, untuk $n \geq 1$
  - Elemen-elemen deretan: 1,2,4,8,16, ...
- Deretan Fibonacci:
  - Basis: $F0 = 0, F1 = 1.$
  - Rekurens: $Fn = Fn - 1 + Fn - 2,$ untuk $n \geq 2n.$
  - Elemen-elemen deretan: 0,1,1,2,3,5,8, ...

Dalam algoritma Sudoku Solver, konsep deretan rekursif digunakan untuk mengeksplorasi kemungkinan angka yang dapat dimasukkan ke dalam grid berdasarkan validasi di baris, kolom, dan subgrid.

Rekursi memiliki beberapa keunggulan dalam penyelesaian Sudoku:

1. Rekursi memecah masalah besar menjadi submasalah kecil yang lebih mudah dikelola.
2. Algoritma dengan rekursi lebih mudah dipahami karena struktur logisnya yang menyerupai cara berpikir manusia.
3. Teknik ini dapat diperluas untuk grid dengan ukuran berbeda atau dengan aturan tambahan.

Rekursi memungkinkan algoritma Sudoku Solver untuk mengeksplorasi semua kemungkinan solusi secara sistematis, menjadikannya metode yang handal untuk menyelesaikan teka-teki ini. Dengan menggunakan pendekatan ini, algoritma dapat secara efisien menemukan solusi bahkan untuk konfigurasi Sudoku yang kompleks.

C. Relasi Rekurens

Relasi rekurens adalah persamaan yang mendefinisikan elemen-elemen dalam suatu deretan ($\{an\}$) berdasarkan satu atau lebih elemen sebelumnya dalam deretan yang sama. Relasi ini menyatakan An sebagai fungsi dari $An - 1, An - 2, \dots, An - k$, di mana k adalah jumlah elemen sebelumnya yang digunakan untuk menentukan nilai An.

Relasi rekurens sangat berguna dalam berbagai bidang seperti matematika, ilmu komputer, dan teknik, karena menyediakan cara sistematis untuk mendefinisikan dan menyelesaikan deretan bilangan yang memenuhi pola tertentu. Kondisi awal adalah nilai-nilai awal dalam deretan yang diperlukan untuk memulai proses perhitungan elemen-elemen selanjutnya. Sebagai langkah basis, kondisi awal memberikan titik awal yang unik untuk menghitung deretan.
Contoh:

1. Untuk relasi $An = 2An - 1 + 1$, dengan $A0 = 1$, elemen-elemen deretan dihitung sebagai $A1 = 3, A2 = 7, \ldots$
2. Barisan Fibonacci, yang didefinisikan sebagai $fn = fn - 1 + fn - 2$, memerlukan dua kondisi awal $f0 = 0$ dan $f1 = 1$. Kondisi awal ini secara unik menentukan elemen-elemen deretan Fibonacci, seperti $0,1,1,2,3,5,8, \ldots$

Solusi dari relasi rekurens adalah ekspresi eksplisit yang tidak melibatkan lagi elemen rekursif, tetapi langsung menyatakan An dalam bentuk fungsi eksplisit. Solusi ini diperoleh dengan menyelesaikan relasi rekurens menggunakan pendekatan iteratif atau metode sistematis. Misalkan relasi $An = 2An - 1 - An - 2$ dengan kondisi awal $A0 = 0$ dan $A1 = 3$. Solusi eksplisit dari relasi ini adalah $An = 3n$. Solusi ini diverifikasi dengan mengganti An ke dalam relasi rekurens dan memastikan bahwa persamaan tersebut terpenuhi.

Relasi rekurens dapat diselesaikan menggunakan dua pendekatan:

1. Iteratif yaitu lemen-elemen deretan dihitung satu per satu berdasarkan relasi dan kondisi awal.
   Contoh: Pada bunga majemuk, jumlah total pada periode n dihitung dengan relasi $An = (1 + r)An - 1$ secara iteratif.
2. Metode Sistematis yang mana pendekatan ini digunakan untuk relasi rekurens berbentuk homogen lanjar (*linear homogeneous recurrence relation*) seperti $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$. Solusi dari relasi ini melibatkan pencarian akar-akar karakteristik dari persamaan karakteristik $r^k - c_1 r^{k-1} - c_2 r^{k-2} - \ldots - c_{k-1} r - c_k = 0$

Relasi rekurens menyediakan pendekatan matematis yang efektif untuk mendefinisikan dan menganalisis deretan angka berdasarkan pola tertentu. Dengan memanfaatkan kondisi awal dan solusi eksplisit, relasi ini membantu dalam menyelesaikan berbagai masalah dalam matematika, ilmu komputer, dan aplikasi praktis lainnya. Penyelesaian relasi rekurens secara sistematis menggunakan metode persamaan karakteristik memungkinkan kita memahami struktur deretan dan mengoptimalkan perhitungan elemen-elemen berikutnya.

D. Algoritma *Backtracking*

Algoritma *backtracking* adalah metode sistematis untuk memecahkan masalah dengan mencoba berbagai kemungkinan solusi dan kembali ke langkah sebelumnya (*backtrack*) jika solusi yang sedang diuji tidak valid atau tidak mengarah ke penyelesaian masalah. *Backtracking* sering digunakan untuk masalah optimasi, pencarian, atau pengambilan keputusan yang memerlukan eksplorasi ruang solusi yang besar secara efisien. Contoh aplikasi umum dari algoritma ini adalah penyelesaian teka-teki Sudoku, pencarian Hamiltonian Path, masalah delapan ratu.

Backtracking menggunakan teknik pencarian berbasis *depth-first search* (DFS) yang mengeksplorasi semua kemungkinan solusi secara rekursif. Dalam setiap langkah, algoritma mencoba memilih satu solusi parsial dan memverifikasi apakah solusi tersebut valid. Jika valid, algoritma melanjutkan dengan solusi tersebut. Namun, jika solusi tidak valid atau tidak dapat berkembang menjadi solusi lengkap, algoritma "kembali" (*backtrack*) untuk mencoba solusi alternatif pada tingkat sebelumnya.

Cara kerja algoritma *backtracking* dapat dirangkum dalam langkah-langkah berikut:

1. Algoritma memulai dengan solusi kosong atau sebagian.
2. Pada setiap langkah, algoritma mencoba memilih satu opsi yang tersedia.
3. Setelah memilih opsi, algoritma memeriksa apakah opsi tersebut valid:
   - Jika valid, algoritma melanjutkan ke langkah berikutnya.
   - Jika tidak valid, algoritma membatalkan opsi tersebut (*backtrack*) dan mencoba opsi lain.
4. Jika solusi lengkap ditemukan, algoritma menghentikan proses dan mengembalikan solusi tersebut.
5. Jika semua opsi pada suatu tingkat telah dicoba dan tidak ada yang valid, algoritma kembali ke tingkat sebelumnya untuk mencoba opsi lain.

Proses ini terus berulang hingga algoritma menemukan solusi atau semua kemungkinan telah dieksplorasi. Beberapa keunggulan yang dimiliki oleh algoritma *backtracking* diantaranya

1. Efisiensi ruang solusi karena algoritma ini mengeksplorasi solusi secara sistematis, sehingga menghindari eksplorasi yang tidak perlu.
2. Pendekatan rekursif membuat algoritma mudah dipahami dan diimplementasikan. Algoritma juga dapat digunakan untuk berbagai jenis masalah kombinatorika.
3. Teknik ini dapat diperluas untuk menangani masalah dengan aturan tambahan atau ruang solusi yang lebih kompleks.

Namun algoritma ini juga memiliki beberapa kekurangan, yakni:

1. Kompleksitas algoritma *backtracking* untuk Sudoku bersifat eksponensial pada kasus terburuk, yaitu $O(9^m)$ di mana m adalah jumlah sel kosong. Namun, optimasi seperti memilih angka dengan probabilitas valid tertinggi dapat mengurangi eksplorasi yang tidak perlu.
2. Pada masalah dengan ruang solusi yang sangat besar, algoritma ini bisa menjadi tidak praktis tanpa optimasi tambahan seperti *pruning* atau heuristik.

Algoritma ini dapat diimplementasikan secara efektif untuk menyelesaikan berbagai jenis masalah logika dan kombinatorika, termasuk teka-teki Sudoku, yang memerlukan validasi angka pada baris, kolom, dan subgrid secara iteratif.

III. Implementasi Algoritma

Implementasi dari program ini dibuat menggunakan pustaka Tkinter dalam bahasa pemograman Python dan mencakup fitur seperti memecahkan Sudoku secara otomatis, menghasilkan papan Sudoku acak, menampilkan langkah-langkah solusi, serta menghitung waktu yang dibutuhkan untuk menyelesaikan teka-teki.

A. Pustaka yang digunakan

```
import tkinter as tk
from tkinter import messagebox, scrolledtext
import random
import time
```

Program ini memanfaatkan beberapa pustaka Python untuk mendukung fungsionalitas utamanya, terutama dalam membangun antarmuka pengguna, menampilkan pesan, dan melaksanakan operasi lainnya. Salah satu pustaka yang digunakan adalah tkinter, yang menjadi pustaka utama untuk membangun antarmuka grafis (GUI). Dengan pustaka ini, berbagai komponen seperti jendela utama (tk.Tk), tombol, kotak teks, dan label dibuat, memberikan pengalaman interaktif kepada pengguna. Penggunaan tkinter memungkinkan pengembang membangun aplikasi dengan tampilan yang *user-friendly*.

Selain itu, program ini memanfaatkan messagebox dari pustaka tkinter untuk menampilkan dialog pesan kepada pengguna. Dialog ini digunakan untuk memberikan peringatan atau notifikasi kesalahan ketika Sudoku tidak memiliki solusi. Dengan cara ini, pengguna mendapatkan umpan balik yang jelas mengenai status aplikasi atau langkah yang perlu diambil.

Komponen lain dari tkinter, yaitu scrolledtext, juga dimanfaatkan untuk menampilkan area teks yang dapat digulir. Area ini digunakan untuk menyajikan langkah-langkah solusi secara rinci, sehingga pengguna dapat dengan mudah mengikuti proses penyelesaian teka-teki tanpa dibatasi oleh ukuran tampilan yang tetap.

Untuk mendukung pengacakan dalam proses pembuatan teka-teki, program ini menggunakan pustaka random. Fungsi-fungsi dari pustaka ini digunakan untuk mengisi papan awal secara acak dan menghapus angka tertentu guna menciptakan teka-teki yang unik dan menantang. Dengan cara ini, setiap kali program dijalankan, pengguna mendapatkan pengalaman bermain yang berbeda.

Pustaka time juga memainkan peran penting dalam program ini, yaitu untuk mencatat dan menghitung durasi penyelesaian teka-teki. Waktu mulai dan selesai dicatat menggunakan fungsi dari pustaka ini, dan hasilnya ditampilkan pada antarmuka pengguna sebagai informasi tambahan. Fitur ini menambah elemen interaktif sekaligus memberikan pengguna wawasan tentang performa mereka dalam menyelesaikan Sudoku.

B. Struktur Utama

Kelas SudokuGame menjadi inti dari program, yang menangani logika permainan, algoritma penyelesaian, serta antarmuka pengguna.

```
N = 9

class SudokuGame:
    def __init__(self, root):
        self.root = root
        self.root.title("Sudoku Solver")
        self.board = [[0] * N for _ in range(N)]
        self.entries = [[None for _ in range(N)] for _ in range(N)]
        self.solver_cells = []  # Menyimpan posisi angka yang
        diisi oleh solver
        self.total_sum = 0  # Total sumasi angka di papan
        self.create_ui()
        self.generate_random_board()
```

Konstanta N diatur ke 9, mencerminkan ukuran papan Sudoku yang berisi 9x9 sel. Dalam metode __init__, berbagai atribut diinisialisasi untuk mengelola logika dan tampilan permainan. Atribut self.board adalah matriks 9x9 yang digunakan untuk menyimpan angka-angka pada papan Sudoku, sementara self.entries adalah matriks 9x9 yang terdiri dari *widget Entry*, memungkinkan pengguna untuk memasukkan angka secara langsung melalui antarmuka. Atribut self.solver_cells menyimpan posisi sel yang diisi oleh solver untuk pelacakan, dan self.total_sum mencatat total jumlah angka pada papan.

Metode create_ui() bertanggung jawab untuk membangun elemen-elemen antarmuka pengguna, termasuk label, tombol, dan area teks bergulir untuk penjelasan langkah demi langkah. Antarmuka ini memberikan pengalaman visual dan interaktif yang intuitif bagi pengguna. Metode generate_random_board() digunakan untuk menghasilkan papan Sudoku acak dengan angka-angka yang sudah diisi, di mana beberapa sel kemudian dikosongkan untuk memberikan tantangan permainan.

C. Antarmuka Program

```
    def create_ui(self):
        title = tk.Label(self.root, text="Sudoku Solver",
        font=("Arial", 16, "bold"))
        title.grid(row=0, column=0, columnspan=N, pady=10)

        for i in range(N):
            for j in range(N):
                entry = tk.Entry(self.root, width=2, font=("Arial",
                18), justify="center")
                entry.grid(row=i+1, column=j, padx=2, pady=2)
                self.entries[i][j] = entry

        solve_button = tk.Button(self.root, text="Solve",
        command=self.solve_sudoku_ui, bg="lightgreen")
        solve_button.grid(row=N+1, column=0, columnspan=3,
        pady=10)
```

```
    reset_button = tk.Button(self.root, text="Reset",
    command=self.reset_board, bg="lightcoral")
    reset_button.grid(row=N+1, column=6, columnspan=3,
    pady=10)

    explanation_label = tk.Label(self.root, text="Step-by-Step
    Explanation:", font=("Arial", 12, "bold"))
    explanation_label.grid(row=1, column=N+1, padx=10,
    pady=5, sticky='nw')

    self.explanation_area =
    scrolledtext.ScrolledText(self.root, width=40, height=20,
    font=("Arial", 10))
    self.explanation_area.grid(row=2, column=N+1,
    rowspan=8, padx=10, pady=5)
    self.explanation_area.config(state='disabled')

    self.time_label = tk.Label(self.root, text="Time Taken:
    0.0 seconds", font=("Arial", 12))
    self.time_label.grid(row=N+2, column=0, columnspan=N,
    pady=10)

    self.sum_label = tk.Label(self.root, text="Current Sum:
    0", font=("Arial", 12))
    self.sum_label.grid(row=N+3, column=0, columnspan=N,
    pady=5)
```

Antarmuka pengguna pada aplikasi ini dirancang untuk memberikan pengalaman interaktif dan informatif bagi pengguna saat bermain Sudoku. Elemen-elemen utama antarmuka mencakup beberapa *widget* penting. Judul berupa label menampilkan teks "Sudoku Solver" untuk memberi identitas aplikasi. Kotak input (*Entry Widgets*) tersusun dalam grid 9x9, memungkinkan pengguna memasukkan angka ke dalam papan Sudoku. Dua tombol utama memberikan fungsi penting, tombol Solve memicu algoritma penyelesaian otomatis, sedangkan tombol Reset mengatur ulang papan ke kondisi awal dengan teka-teki baru yang dihasilkan secara acak.

Selain itu, terdapat ScrolledText, area teks yang dirancang untuk menampilkan penjelasan langkah-langkah penyelesaian Sudoku secara rinci, sehingga pengguna dapat memahami proses di balik algoritma solver. Label waktu digunakan untuk menunjukkan durasi yang dibutuhkan untuk menyelesaikan teka-teki. Sementara itu, Label jumlah total (Sum) menunjukkan total angka yang saat ini ada di papan, menambah dimensi tambahan untuk pelacakan status permainan.

D. Pembuatan Papan Sudoku

```
def generate_random_board(self):
    self.board = [[0] * N for _ in range(N)]
    self.fill_grid(self.board)
    self.remove_numbers()
    self.solver_cells = []
    self.display_board()
    self.update_sum()

def fill_grid(self, board):
    numbers = list(range(1, N+1))
    for i in range(N):
        for j in range(N):
            if board[i][j] == 0:
                random.shuffle(numbers)
                for num in numbers:
                    if self.is_valid(board, i, j, num):
                        board[i][j] = num
                        if self.fill_grid(board):
                            return True
                        board[i][j] = 0
                return False
    return True

def remove_numbers(self):
    cells_to_remove = random.randint(40, 55)
    for _ in range(cells_to_remove):
        row, col = random.randint(0, N-1), random.randint(0,
        N-1)
        while self.board[row][col] == 0:
            row, col = random.randint(0, N-1), random.randint(0,
            N-1)
        self.board[row][col] = 0

def display_board(self):
    for i in range(N):
        for j in range(N):
            self.entries[i][j].config(state='normal')
            self.entries[i][j].delete(0, tk.END)
            if self.board[i][j] != 0:
                self.entries[i][j].insert(0, str(self.board[i][j]))
                self.entries[i][j].config(state='disabled',
                disabledforeground='black')

def get_board(self):
    for i in range(N):
        for j in range(N):
            if self.entries[i][j].get().isdigit():
                self.board[i][j] = int(self.entries[i][j].get())
            else:
                self.board[i][j] = 0

def is_valid(self, board, row, col, num):
    if num in board[row]:
        return False
```

```
    if num in [board[i][col] for i in range(N)]:
        return False
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(start_row, start_row + 3):
        for j in range(start_col, start_col + 3):
            if board[i][j] == num:
                return False
    return True
```

Logika pembuatan dan manipulasi papan Sudoku dalam aplikasi ini dirancang menggunakan metode acak dan algoritma *backtracking*. Metode generate_random_board() bertugas untuk membuat papan Sudoku awal yang valid secara aturan. Prosesnya dimulai dengan mengisi seluruh papan menggunakan metode rekursif fill_grid(), yang memanfaatkan *backtracking* untuk memastikan angka-angka yang dimasukkan tidak melanggar aturan Sudoku. Untuk setiap angka yang akan ditempatkan, metode is_valid() digunakan untuk memverifikasi bahwa angka tersebut tidak muncul lebih dari sekali di baris, kolom, atau subgrid 3x3 yang dicek. Setelah papan lengkap dihasilkan, metode remove_numbers() dipanggil untuk menghapus angka dari beberapa sel secara acak. Kombinasi logika ini memastikan teka-teki yang dihasilkan valid dan mengikuti aturan standar permainan Sudoku.

E. Penyelesaian Sudoku

```
def solve_sudoku(self, board, step=1):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 0:
                for num in range(1, 10):
                    if self.is_valid(board, i, j, num):
                        board[i][j] = num
                        self.solver_cells.append((i, j))
                        self.update_sum()  # Update sum setiap
                        angka ditempatkan
                        self.log_explanation(step, i, j, num)
                        if self.solve_sudoku(board, step+1):
                            return True
                        board[i][j] = 0
                        self.solver_cells.remove((i, j))
                        self.update_sum()
                        self.log_explanation(step, i, j, "Backtrack")
                return False
    return True

def solve_sudoku_ui(self):
    self.get_board()
    self.explanation_area.config(state='normal')
    self.explanation_area.delete(1.0, tk.END)
    self.explanation_area.insert(tk.END, "Starting Sudoku
    Solver...\n\n")
    start_time = time.time()
    if self.solve_sudoku(self.board):
        self.display_solver_board()
        self.explanation_area.insert(tk.END, "\nSudoku Solved
        Successfully!")
        messagebox.showinfo("Sudoku", "Sudoku Solved
        Successfully!")
    else:
        self.explanation_area.insert(tk.END, "\nNo solution
        exists for the given Sudoku.")
        messagebox.showerror("Sudoku", "No solution exists
        for the given Sudoku!")

    end_time = time.time()
    time_taken = end_time - start_time
    self.time_label.config(text=f"Time Taken:
    {time_taken:.2f} seconds")

    self.explanation_area.config(state='disabled')

def display_solver_board(self):
    for i in range(N):
        for j in range(N):
            self.entries[i][j].config(state='normal')
            self.entries[i][j].delete(0, tk.END)
            if self.board[i][j] != 0:
                self.entries[i][j].insert(0, str(self.board[i][j]))
                if (i, j) in self.solver_cells:
                    self.entries[i][j].config(disabledforeground='gra
                    y')
                else:
                    self.entries[i][j].config(disabledforeground='bla
                    ck')
            self.entries[i][j].config(state='disabled')
```

Metode solve_sudoku() dalam program ini mengimplementasikan algoritma *backtracking* untuk menyelesaikan teka-teki Sudoku. Proses dimulai dengan memeriksa setiap sel pada papan yang kosong (bernilai 0). Untuk setiap sel kosong, algoritma mencoba memasukkan angka dari 1 hingga 9 dan memeriksa apakah angka tersebut valid menggunakan metode is_valid(). Jika angka yang dipilih valid, angka tersebut ditempatkan di sel dan langkah tersebut dicatat dalam area penjelasan menggunakan metode log_explanation(). Jika angka yang dimasukkan tidak dapat diterima, algoritma melakukan *backtracking*, yaitu menghapus angka yang telah ditempatkan sebelumnya dan mencoba angka lain untuk sel tersebut. Proses ini berlanjut hingga teka-teki selesai atau tidak ada solusi lain yang ditemukan. Selain itu, metode ini juga memperbarui jumlah total angka pada papan setelah setiap langkah dengan memanggil metode update_sum(). Untuk integrasi dengan antarmuka pengguna, metode solve_sudoku_ui() menghubungkan solver dengan

elemen-elemen GUI. Metode ini memulai proses penyelesaian, menampilkan solusi yang ditemukan pada papan Sudoku, dan menampilkan pesan sukses atau error menggunakan kotak pesan (messagebox) untuk memberi umpan balik kepada pengguna.

F. Fitur Tambahan

```
def reset_board(self):
    self.generate_random_board()
    self.explanation_area.config(state='normal')
    self.explanation_area.delete(1.0, tk.END)
    self.explanation_area.config(state='disabled')

def log_explanation(self, step, row, col, action):
    self.explanation_area.config(state='normal')
    if action == "Backtrack":
        explanation = f"Step {step}: Backtracking at cell
        ({row+1}, {col+1}).\n"
    else:
        explanation = f"Step {step}: Placing {action} at cell
        ({row+1}, {col+1}).\n"
    self.explanation_area.insert(tk.END, explanation)
    self.explanation_area.see(tk.END)
    self.explanation_area.update_idletasks()
    self.root.update()

def update_sum(self):
    self.total_sum = sum(sum(row) for row in self.board)
    self.sum_label.config(text=f"Current Sum:
    {self.total_sum}")
```

Program juga dilengkapi dengan fitur tambahan untuk meningkatkan pengalaman pengguna, salah satunya adalah penghitungan waktu penyelesaian teka-teki. Dengan memanfaatkan modul time, durasi yang dibutuhkan untuk menyelesaikan papan Sudoku dihitung dan hasilnya ditampilkan pada label khusus di bagian bawah antarmuka pengguna. Selain itu, validasi angka yang dimasukkan juga diterapkan untuk memastikan setiap langkah mematuhi aturan Sudoku. Metode is_valid memverifikasi bahwa tidak ada angka yang sama dalam baris, kolom, atau subgrid 3x3. Jika ditemukan situasi di mana tidak ada solusi yang memungkinkan, aplikasi secara otomatis memberikan pesan kesalahan kepada pengguna untuk menunjukkan bahwa teka-teki tidak dapat diselesaikan.

IV. Penerapan Deretan, Sumasi, dan Rekursi pada Algoritma

Program ini memanfaatkan tiga konsep utama deretan, sumasi, dan rekursi untuk membangun solusi Sudoku interaktif.

A. Penerapan Deretan

Konsep deretan diterapkan untuk merepresentasikan papan Sudoku dalam bentuk array dua dimensi (*list of lists*). Dua struktur utama, yaitu self.board dan self.entries, digunakan untuk tujuan ini.

```
self.board = [[0] * N for _ in range(N)]
self.entries = [[None for _ in range(N)] for _ in range(N)]
```

Pada self.board disimpan angka-angka yang ada di papan Sudoku sebagai representasi internal, dan array ini dibentuk menggunakan *list*. Elemen-elemen dalam self.board diakses menggunakan indeks dua dimensi, seperti self.board[i][j]. Sementara itu, self.entries menyimpan referensi ke *widget* tk.Entry yang mewakili input pengguna di antarmuka grafis. Pada awalnya, self.entries diinisialisasi sebagai array dua dimensi yang berisi None, kemudian diisi dengan objek tk.Entry saat antarmuka dibuat.

Pada fungsi fill_grid, daftar angka acak disusun dengan mengacak deretan angka dari 1 hingga N menggunakan fungsi random.shuffle.

```
numbers = list(range(1, N+1))
random.shuffle(numbers)
```

Pertama, deretan angka dibuat dengan list(range(1, N+1)), yang menghasilkan daftar angka berturut-turut dari 1 hingga N. Daftar ini kemudian diacak menggunakan random.shuffle(numbers), sehingga urutan angka berubah menjadi acak. Deretan angka yang telah diacak ini digunakan untuk mencoba mengisi grid Sudoku secara acak, sambil tetap mematuhi aturan validitas Sudoku. Pendekatan ini membantu menciptakan variasi dalam pengisian angka pada grid selama proses pembangkitan papan Sudoku.

B. Penerapan Sumasi

Konsep sumasi dalam kode ini digunakan untuk menghitung total nilai elemen-elemen pada papan Sudoku.

```
self.total_sum = sum(sum(row) for row in self.board)
self.sum_label.config(text=f"Current Sum: {self.total_sum}")
```

Total ini disimpan dalam variabel self.total_sum, yang diperbarui setiap kali terjadi perubahan pada papan, misalnya ketika solver berhasil menempatkan angka di sebuah sel. Penghitungan dilakukan dengan fungsi sum() secara bertingkat, di mana sum(row) menghitung jumlah angka pada setiap baris, dan sum(sum(row) for row in self.board) menghitung total seluruh angka di papan. Nilai total ini kemudian ditampilkan kepada pengguna melalui label self.sum_label, memberikan informasi tambahan berupa jumlah angka yang ada di papan sebagai indikator perkembangan penyelesaian Sudoku.

C. Penerapan Rekursi

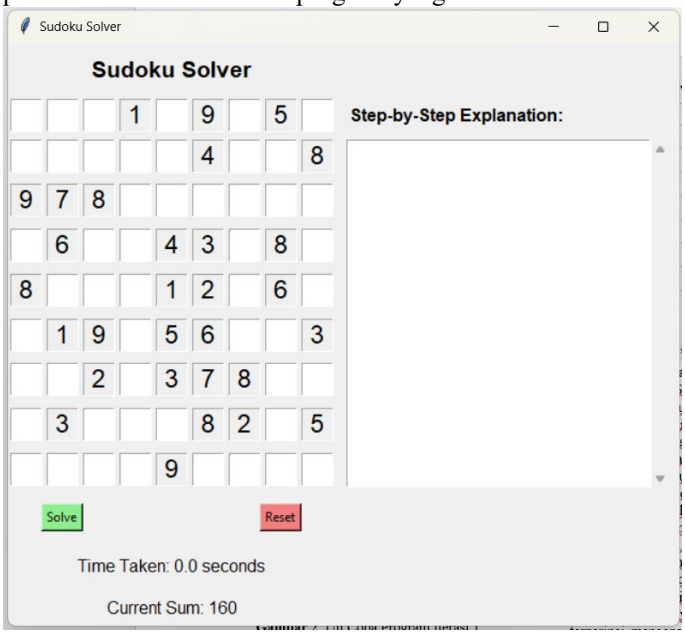
Konsep rekursi dalam kode ini digunakan untuk menyelesaikan Sudoku melalui algoritma *backtracking* yang diterapkan pada fungsi solve_sudoku. Fungsi ini memanggil dirinya sendiri untuk mencoba menempatkan angka pada sel kosong dengan mengikuti langkah-langkah tertentu. Pertama, fungsi mencari sel kosong pada papan. Kemudian, angka-angka dari 1 hingga 9 dicoba satu per satu. Jika angka tersebut valid sesuai aturan Sudoku, angka tersebut ditempatkan di sel, dan fungsi memanggil dirinya sendiri untuk melanjutkan ke langkah berikutnya. Jika solusi tidak ditemukan (*dead-end*), fungsi melakukan *backtracking*, yaitu mengembalikan kondisi sel menjadi kosong dan mencoba angka lain.

```
if self.solve_sudoku(board, step+1):
    return True
```

Proses *backtracking* ini memastikan bahwa algoritma dapat menemukan solusi yang sesuai. Pesan "*Backtracking*" ditampilkan di area penjelasan untuk memberikan gambaran kepada pengguna mengenai langkah-langkah algoritma. Dengan pendekatan rekursi ini, fungsi dapat menjelajahi semua kemungkinan hingga menemukan solusi yang tepat secara sistematis.

V. ANALISIS

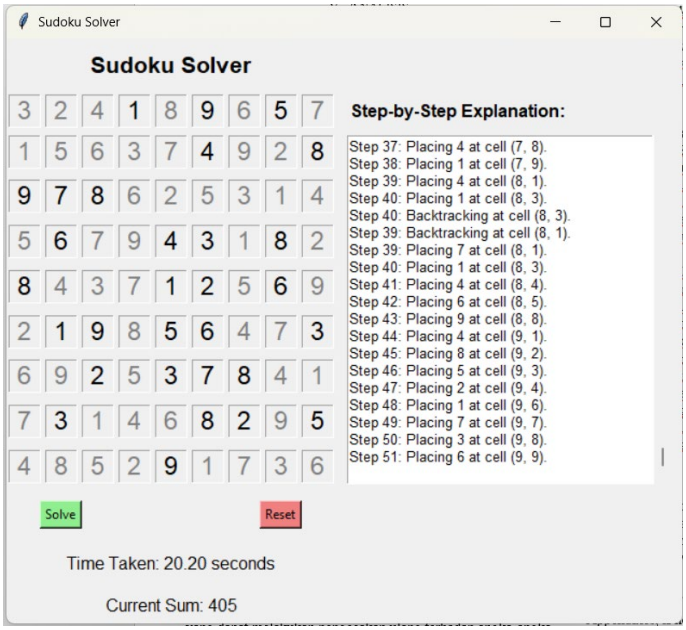
Program danalisis dengan melakukan pengujian sebanyak 2 kali iterasi untuk mendapatkan *insight* mendalam terkait performa dan efisiensi dari program yang telah dibuat.



Gambar 2. Tampilan Awal Program
(*Sumber*: Dokumen Pribadi)

Pada Gambar.2 merupakan tampilan awal Ketika memulai program. Program akan menampilkan papan Sudoku yang terdiri atas angka-angka acak untuk diselesaikan. Pada sebelah kanan papan permainan, program juga memberikan detail langkah demi langkah dalam area "Step-by-Step Explanation," yang memperlihatkan proses penempatan angka berdasarkan aturan validas. Terdapat tombol Solve yang dapat secara otomatis menyelesaikan program menggunakan algoritma yang telah diimplementasikan. Selain itu juga terdapat tombol Reset yang dapat melakukan pengecakan ulang terhadap angka-angka yang ada pada papan permainan Sudoku. Durasi waktu yang dibutuhkan setiap kali program telah menyelesaikan permainan akan ditampilkan pada bagian Time Taken. Hasil penjumlahan dari setiap angka yang ada pada papan permainan juga akan ditampilkan pada Current Sum yang akan secara otomatis bertambah setiap kali program mencoba menyelesaikan permainan.

Pengujian iterasi pertama dimulai dengan menekan tombol Solve dan program akan mencoba menyelesaikan tantangan dari permainan. Pada Gambar.3 dapat dilihat bahwa Program berhasil menyelesaikan tantangan Sudoku yang diberikan dengan mengisi setiap sel dengan angka yang valid sesuai aturan Sudoku, yaitu angka 1 hingga 9 tanpa pengulangan di setiap baris, kolom, dan subgrid 3x3.

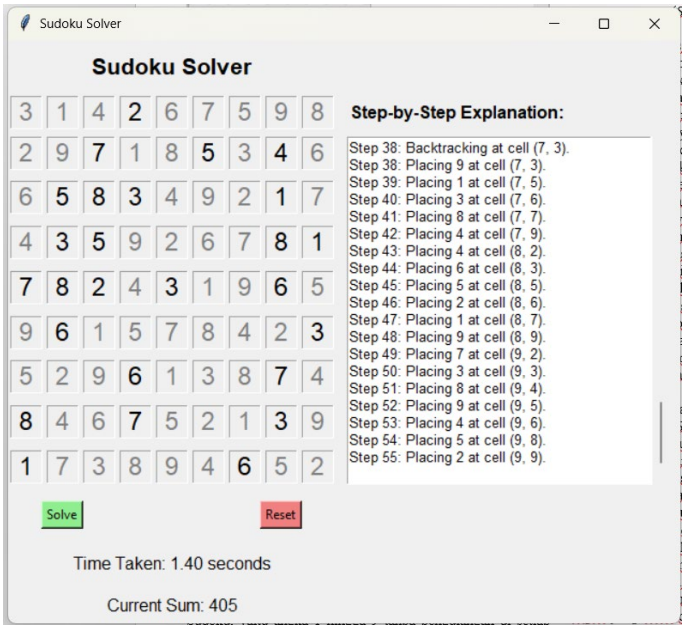


Gambar 3. Pengujian Program Iterasi 1
(*Sumber*: Dokumen Pribadi)

Proses penyelesaian ini dijelaskan secara mendetail dalam "Step-by-Step Explanation" yang menunjukkan bagaimana algoritma *backtracking* bekerja. Pada langkah tertentu, program mencoba menempatkan angka di sel kosong. Jika solusi tidak ditemukan, algoritma melakukan *backtracking* dengan menghapus angka yang telah ditempatkan dan mencoba angka lain. Misalnya, pada "Step 40: Backtracking at cell (8, 3)", program membatalkan angka di sel tersebut karena solusi gagal ditemukan, lalu melanjutkan dengan angka lain hingga papan berhasil diselesaikan.

Program menyelesaikan proses ini dalam 20,20 detik, mencakup seluruh proses rekursi, pengisian angka, dan *backtracking*. Durasi ini menunjukkan bahwa algoritma bekerja dengan cukup efisien, meskipun kompleksitas papan Sudoku memengaruhi waktu penyelesaian. Selain itu, program menghitung total sumasi angka di papan, yaitu 405, yang merupakan hasil penjumlahan angka di seluruh sel (dengan $45 \times 9 = 405$, di mana 45 adalah jumlah per baris untuk 9 baris). Informasi ini ditampilkan kepada pengguna sebagai "Current Sum: 405" untuk memberikan data tambahan yang relevan.

Pengujian iterasi kedua dilakukan dengan menekan tombol Reset sehingga mengacak ulang papan permainan yang memberikan tantangan permainan berbeda. Hasil pengujian iterasi kedua dapat dilihat pada Gambar.4 yang telah berhasil diselesaikan program sesuai dengan aturan Dibandingkan dengan pengujian pada iterasi pertama, perbedaan signifikan terlihat pada waktu penyelesaian. Program pada pengujian iterasi kedua hanya membutuhkan waktu 1,40 detik, jauh lebih cepat dibandingkan dengan 20,20 detik pada pengujian iterasi pertama. Hal ini menunjukkan bahwa kompleksitas papan Sudoku atau efisiensi proses *backtracking* dapat sangat memengaruhi durasi penyelesaian. Total sumasi angka, yaitu 405, tetap sama pada setiap penyelesaian permainan, menunjukkan bahwa papan Sudoku terdiri dari jumlah angka yang identik setelah selesai



Gambar 4. Uji Coba Program Iterasi 2
(*Sumber*: Dokumen Pribadi)

Secara keseluruhan, hasil ini memperlihatkan bagaimana variasi dalam pengaturan angka awal di papan memengaruhi waktu penyelesaian, meskipun algoritma yang digunakan tetap sama dan efisien.

VI. Kesimpulan

Program Sudoku Solver yang dibuat dengan menerapkan konsep deretan, sumasi dan rekursi telah berhasil menyelesaikan tantangan permainan menggunakan algoritma *backtracking*. Setiap langkah penyelesaian dijelaskan dengan rinci pada bagian "Step-by-Step Explanation", yang menunjukkan bagaimana algoritma bekerja melalui pengisian angka yang valid dan *backtracking* jika solusi tidak ditemukan. Program dapat menyelesaikan papan Sudoku dengan benar, memenuhi aturan bahwa angka 1 hingga 9 tidak boleh ada yang terulang pada setiap baris, kolom, dan subgrid 3x3 serta total sumasi angka di papan, yang tetap sama yaitu 405 pada setiap penyelesaian permainan

Waktu penyelesaian program bervariasi tergantung pada kompleksitas papan Sudoku yang diberikan. Untuk meningkatkan efisiensi program, algoritma *backtracking* dapat dioptimalkan dengan menggunakan teknik pencarian yang lebih cerdas, seperti *constraint propagation* dan heuristik, untuk memilih sel yang paling mungkin diisi terlebih dahulu, dengan menerapkan teknik *Minimum Remaining Values* (MRV) untuk memilih sel dengan jumlah kemungkinan nilai paling sedikit, sehingga mengurangi pencarian yang tidak perlu. Selain itu, menambahkan teknik *pruning* yang lebih efisien dapat mempercepat evaluasi dan menghindari pencarian pada cabang yang tidak mungkin menghasilkan solusi. Penggunaan teknik *caching* dan *memoization* juga dapat membantu menghindari perhitungan yang berulang, sehingga mempercepat penyelesaian masalah yang serupa di masa depan.

VII. Lampiran

[1] Repository github
https://github.com/satrianababan/Makalah-Matdis-13521168.git
[2] Video makalah https://youtu.be/JppywoEpbPU

References

[1] Munir, R. (2024) Homepage Rinaldi Munir. Sekolah Teknik Elektro dan Informatika (STEI) ITB. Deretan, Rekursi, dan Relasi Rekurens Bagian 1 dan 2. diakses pada 26 Desember 2024 Pukul 19.45, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis
[2] Name, Sudoku. (2024). Peraturan Sudoku. Diakses pada 26 Desember 2024 Pukul 20.15, dari https://www.sudoku.name/rules/id
[3] Melanie (2024). Backtracking: What is it? How do I use It?. Diakses pada 28 Desember 2024 Pukul 21.25, dari https://datascientest.com/
[4] Geeksforgeeks. (2024). Algorithm to Solve Sudoku | Sudoku Solver. Diakses pada 29 Desember 2024 pukul 20.3, dari https://www.geeksforgeeks.org/sudoku-backtracking-7/

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, 1 Januari 2025

Satria Octavianus Nababan
NIM. 13521168