Penggunaan Algoritma Runut-Balik untuk Menyelesaikan Permainan Kakuro

Ariel Ansa Razumardi 13517040[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13517040@std.stei.itb.ac.id

***Abstract*—Pada era modern ini, permainan teka-teki atau *puzzle* mulai diminati banyak orang karena baik untuk mengisi waktu luang dan mengembangkan pola pikir untuk memecahkan masalah. Salah satu algoritma yang cocok dipakai untuk menyelesaikan permainan kakuro adalah menggunakan algoritma runut-balik atau *backtracking.* Algoritma runut-balik lebih baik daripada algoritma *bruteforce* karena kompleksitas waktunhya lebih baik.**

Kata Kunci—backtracking, kakuro, algoritma, permainan.

I. PENDAHULUAN

Pada era modern ini, teknologi informasi telah berkembang pesat dan peradaban serta kebiasaan manusia juga berubah seiring waktu mebgikuti perkembangan teknologi informasi tersebut. Salah satu kebiasaan manusia yang berubah dari era-era sebelumnya sampai era modern ini adalah dalam cara-cara manusia menghabiskan waktu luangnya. Salah satu dari cara tersebut adalah dengan bermain permainan.

Permainan telah menjadi hal yang sangat wajar pada era modern ini. Di rumah, di tempat bermain, di sekolah, dan tempat-tempat lainnya, permainan bisa kamu temukan. Salah satu jenis atau *genre* permainan yang ada adalah permainan *puzzle,* seperti contohnya adalah sudoku dan kakuro.

Permainan ber-*genre puzzle* atau teka-teki merupakan salah satu jenis permainan yang sangat bermanfaat bagi manusia. Selain melepas kelelahan dan keletihan yang di dapat dari pekerjaan dan hal-hal yang meyibukkan lainnya, Permainan *puzzle* ini juga bisa bermanfaat untuk melatih pola pikir dengan cara menghadapkan para pemainnya dengan masalah-masalah yang menarik dan menantang untuk dikerjakan.

Pada makalah ini, akan dibahas tentng salah satu permainan *puzzle* yang bernama kakuro. Kakuro adalah game di mana permain ditantang untuk mengisi beberapa kotak-kotak yang kosong dengan angka 1-9 dengan setiap kolom dan baris jika ditambahkan harus merupakan sebuah angka yang sudah diminta sebelumnya. Kakuro ini merupakan permainan yang cukup menarik, menantnag, dan bisa dibilang mirip dengan permainan lainnya yang bernama sudoku, walaupun kedua permainan tersebut memeiliki peraturan yang jelas berbeda, tapi kakuro dan sudoku bisa diselesaikan dengan cara yang relatif mirip.. Cara-cara tersebut bisa bermacam-macam, tapi yang akan dibahas pada makalah ini adalah tentang algoritma runtut-balik atau *backtracking*.

Penjelasan mengenai detil permainan kakuro dan algoritma *backtracking* yang akan digunakan pada makalah ini akan menyusul pada bagian-bagian selanjutnya dari makalah ini

II. DASAR TEORI

A. Pengenalan Kakuro

Kakuro, atau yang sering disebut dengan nama lain Cross Sums di Amerika Serikat dan Kakro di Jepang adalah sebuah permainan yang sangat menyenangkan dan membuat orang ketagihan yang merupakan ekivalen matematis dari teka-teki silang. Di Jepang, popularitas Kakuro atau Kakro seperti disebut di sana sangatlah besar, hanya dinomorduakan oleh Sudoku.

B. Sejarah Kakuro

Teka-teki Kakuro pertama di mana disebut Cross Sums dan diterbitkan pada tahun 1966 oleh Dell Magazines (mereka memperkenalkan teka-teki Sudoku ke dunia satu dekade kemudian). Sejak itu Kakuro alias Cross-Sum telah menjadi fitur reguler dalam publikasi Math & Logic di AS. Cross-Sums juga muncul di majalah Game Kappa.

Teka-teki Kakuro alias Cross-Sum berhasil sampai ke Jepang hanya pada tahun 1980, tempat itu diimpor oleh Maki Kaji, presiden teka-teki Nikoli, yang menamainya Kasan Kurosu - kombinasi kata Jepang untuk "penjumlahan" dan pelafalan bahasa Jepang bahasa Inggris kata "silang."

6 tahun setelah itu dirilis Nikoli berganti nama menjadi Kasan Kurosu sebagai Kakro, dan merilis buklet Kakro / Kakuro pertama. Sejak itu 22 buku Kakuro / Kakro mengikuti dan menunjukkan statis bahwa lebih dari satu juta buku Kakuro telah terjual.

Teka-teki Kakro / Kakuro muncul di 100 majalah dan surat kabar Jepang hari ini, dan kegilaan teka-teki Kakuro ada di posisi kedua, tepat setelah Sudoku. Menurut Nikoli, Kakuro / Kakro adalah teka-teki peringkat teratas hingga 1992 dan baru saat itulah Kakuro Puzzle diambil alih oleh Sudoku.

Seperti halnya Sudoku, kegemaran pasar dimulai hanya setelah puzzle diekspor kembali ke Barat dari Jepang. Kegilaan Kakuro Puzzles dimulai, ketika The Guardian dan The Daily Mail, sebagai tanggapan atas kegilaan Sudoku, memperkenalkan teka-teki Kakro setiap hari sebagai Kakuro Puzzles di Inggris.

Sama seperti Sudoku, teka-teki Kakuro menyebar. Teka-teki Kakuro sekarang dapat ditemukan di semua aliran buku utama dan majalah penerbit. Buku puzzle Kakuro juga mulai tersebar luas, karena semakin banyak judul Kakuro yang dirilis. Penerbit berasumsi bahwa dengan kegemaran Sudoku masih kuat, akan ada lebih banyak minat pada Kakuro / Cross-Sums.

C. Cara Bermain Kakuro

Teka-teki Kakuro bisa memiliki panjang dan lebar yang berbeda, meskipun demikian, kebanyakan teka-teki Kakuro yang ada memiliki panjang dan lebar yang sama atau persegi.

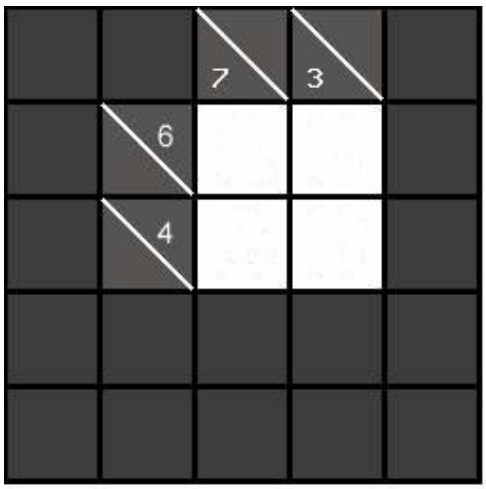
Kakuro memiliki 3 jenis kotak:

1. Kotak petunjuk : Kotak yang memiliki bilangan petunjuk di bagian bawah dan/atau di bagian kanan.
2. Kotak putih : Kotak tempat bilangan jawaban antara 1 sampai 9 diisi, kalau belum diisi dengan bilangan dari soalnya.
3. Kotak hitam : Kotak yang tidak berfungsi sebagai apapun.

Aturan dari Kakuro adalah untuk menempatkan digit bilangan dari 1 sampai 9 di dalam kotak-kotak putih yang masih kosong sehingga jumlah dari digit-digit bilangan dari suatu himpunan kotak-kotak putih yang berurutan memanjang atau melebar, sama dengan bilangann yang terdapat pada kotak petunjuk yang berada di ujung kiri atau ujung atas dari kotak-kotak tersebut.

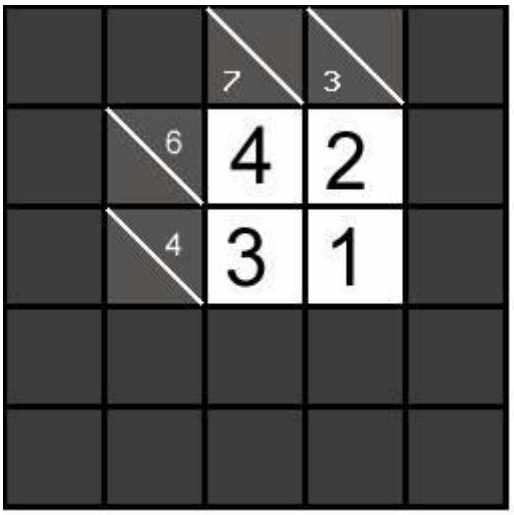
Permainan selesai atau teka-teki dianggap sudah dipecahkan ketika semua kotak putih sudah terisi dan tidak ada aturan yang dilanggar.

Kakuro juga memiliki relasi yang signifikan antara banyak kotak dan jumlah dari digit bilangan kotak-kotak tersebut. Contohnya adalah jika ada dua kotak yang memiliki jumlah bilangan kotak tersebut 3, maka salah satu kotak pasti bernilai 1 dan kotak lainnya pasti bernilai 2.



Gambar 1 Ilustrasi Kakuro (1)

(*https://www.daily-sudoku.com/kakurorules/*)



Gambar 2 Ilustrasi Kakuro (2)

(*https://www.daily-sudoku.com/kakurorules/*)

D. Pengenalan Algoritma Runut-balik (Backtracking)

Ada 2 cara untuk melihat Algoritma runut-balik:

1. Melihat sebagai sebuah fase di dalam algoritma traversal DFS
2. Melihat sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis.

Algoritma runut-balik merupakan perbaikan dari *exhaustive search*. Kalau dalam *exhaustive search*, semua kemungkinan solusi dieksplorasi, pada algoritma runut-balk, hanya pilihan yang mengarah ke solusi yang akan diproses, pilihan yang tidak akan mengarah ke proses tentu tidak akan di proses. Hal ini dilakukan dengan cara memangkas (*pruning*) simpul-simpul yang sudah pasti tidak akan mengarah ke solusi.

E. Properti Umum Algoritma Runut-balik (Backtracking)

Ada 3 properti umum yang dimiliki oleh algoritma runut-balik:

1. Solusi Persoalan

Solusi dinyatakan dengan vektor yang memiliki *n-tuple:* $X = (x_1, x_2, x_3, ..., x_n)$ yang melambangkan langkah-langkah yang dapat diambil, $x_i \in S_i$. Contoh : Solusi $S_i = \{0,1\}$, $x_i = 0$ atau 1

2. Fungsi Pembangkit

Fungsi pembangkit nilai x_k dinyatakan sebagai predikat . $T(k)$ $T(k)$ ini akan membangkitkan nilai untuk x_k , yang merupakan komponen vektor solusi.

3. Fungsi Pembatas

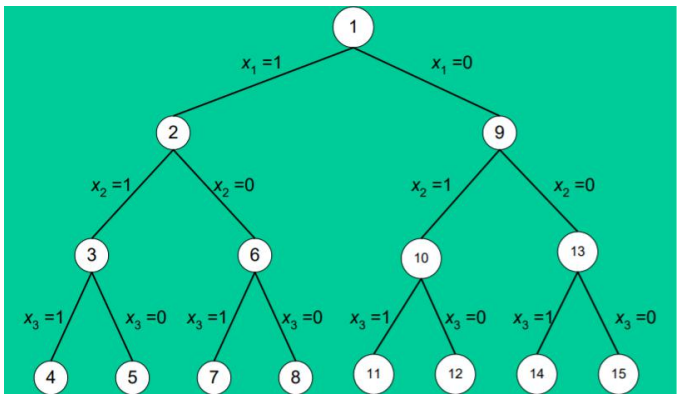
Fungsi pembatas dinyatakan seabagai predikat $B(x_1, x_2, x_3, ..., x_k)$, B akan bernilai *true* jika parameternya mengarah ke solusi. Jika hasil dari B bernilai *true*, maka pembangkitkan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false,* maka kombinasi langkah-langkah $(x_1, x_2, x_3, ..., x_k)$ akan dibuang dan tidak diproses lebih lanjut.

F. Pengorganisasian Solusi Algoritma Runut-balik (Backtracking)

Semua kemungkinan solusi dari suatu persoalan disebut sebagai ruang solusi atau *solution space* dari persoalan tersebut. Contohnya adalah, pada persoalan *knapsack* 0/1 untuk jumlah barang n = 3, solusi dapat dinyatakan sebagai (x_1, x_2, x_3) dengan $x_i \in \{0,1\}$, lalu ruang solusinya adalah sebagai berikut:

{(0, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1) }

Ruang solusi dapat diorganisasikan ke dalam struktur pohon, di mana tiap simpul dalam struktur pohon tersebut menyatakan status (*state*) dari persoalan dan sisi (cabang) dilabeli dengan nilai-nilai x_i dari $X = (x_1, x_2, x_3, ..., x_n)$. Lintasan dari akar ke daun menyatakan solusi yang mungkin dan seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai pohon ruang status (*state space tree*)



Gambar 3 Pohon Ruang Status *Knapsack* 0/1

(http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Runut-balik-(2018).pdf)

G. Prinsip Pencarian Solusi Algoritma Runut-balik (Backtracking)

Prinsip pencarian solusi algoritma runut-balik adalah sebagai berikut:

1. Solusi akan dicari dengan membentuk lintasan dari akar ke daun dengan aturan pembentukan yang dipakai adalah mengikuti aturan DFS (*depth first search*)
2. Simpul-simpul yang sudah diciptakan dinamakan simpul hidup atau *live node*
3. Simpul hidup yang sedang diperluas dinamakan simpul ekspan atau *expand node*
4. Setiap kali simpul ekspan diperluas, lintasan yang dibangun olehnya bertambah panjang.
5. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul exspan tersebut dibuang sehingga menjadi simpul mati atau *dead node*. Untuk menentukan apakah lintasan yang sedang dibentuk akan mengarah ke solusi atau tidak, digunakan fungsi pembatas B. Sebuah *dead node* sudah tidak akan pernah diperluas lagi.
6. Jika pembentukan lintasan berakhir di *dead node,* maka proses pencarian akan melakukan runut-balik atau *backtrack* ke simpul orang tuanya (simpul yang menciptakan *dead node* tersebut) dan dilanjutkan dengan membangkitkan simpul anak lainnya..
7. Pencarian dihentikan bila simpul tujuan atau *goal node* telah dicapai

III. Penggunaan Algoritma Runut-Balik untuk Menyelesaikan Permainan Kakuro

Pada bagian ini , akan dijelaskan cara untuk menyelesaikan permainan Kakuro dengan menggunakan algoritma runut-balik. Algoritma ini dipilih karena jika dibandingkan dengan *bruteforce,* algoritma runut-balik secara konsep akan lebih baik kompleksitas waktunya dibandingkan dengan *bruteforce*. Itulah alasan dipilih algoritma runut-balik untuk permasalahan ini dan bukan *bruteforce*.

Untuk menggunakan algoritma *backtracking* atau runut-balik, hal pertama yang harus dilakukan adalah menedefinisikan properti umum algoritma runut-balik untuk permainan Kakuro, yaitu seperti yang ada pada bagian dasar teori, solusi permasalahan, fungsi pembangkit, dan fungsi pembatas.

Solusi permasalahan untuk permasalahan ini adalah berupa *n-tuple* $X = (x_{ij}, x_{ij}, x_{ij}, ..., x_{ij})$ dengan i dan j adalah indeks baris dan kolom dari kotak pada matriks yang berwarna putih dan awalnya kosong dengan $x_{ij} \in \{1,2,3,4,5,6,7,8,9\}$. Jadi

solusi permasalahan tersebut menggambarkan kotak-kotak putih kosong diisi dengan bilangan-bilangan yang terdapat pada solusi permasalahan.

Properti umum selanjutnya adalah fungsi pembangkit. Pada permasalahan permainan Kakuro ini, fungsi pembangkit akan melihat kondisi dari matriks Kakuro dan mencari kotak putih yang masih kosong, lalu akan dicoba diisi dengan angka 1 sampai 9.

Properti umum yang terakhir adalah fungsi pembatas. Pada permasalahan permainan Kakuro ini, ada beberapa hal yang harus dibatasi, yaitu:

1. Bilangan yang diisi ke kotak, tidak boleh membuat jumlah bilangan-bilangan dari satu himpunan kotak-kotak yang berurutan memanjang atau melebar melebihi bilangan di kotak petunjuknya yang berada di ujung atas untuk yang melebar dan ujung kiri untuk yang memanjang.
2. Bilangan yang diisi ke kotak tidak boleh sama dengan bilangan lain yang berada di satu himpunan kotak-kotak yang berurutan memanjang atau melebar dengan kotak yang akan diisi.

Karena telah didefinisikan ketiga properti umum algoritma runut-balik untuk permasalahan permainan Kakuro, sekarang algoritma tersebut bisa langsung diimplementasikan dengan program. Berikut adalah cuplikan kode program untuk menyelesaikan permainan Kakuro dalam bahasa python.

```
def backtrack(self, row, col):
        if       self.map[row][col].type       ==       "v"       and
self.map[row][col].value[0] == 0:
            for i in range(1,10):
                if (not self.finished) and self.possibleValue(row, col,
i):
                    self.map[row][col].value[0] = i
                    print(str(row) + "," + str(col) + ": ")
                    print(self.map[row][col].value[0])
                    if col == len(self.map[row]) - 1:
                        if row < len(self.map) - 1:
                            self.backtrack(row + 1, 0)
                        else:
                            self.finished = True
                    else:
                        self.backtrack(row, col + 1)
                    if not self.finished:
                        self.map[row][col].value[0] = 0
        else:
            if col == len(self.map[row]) - 1:
                if       (not       self.isFinalRowNonV(row))       or
self.possibleValue(row, col, self.map[row][col].value[0]):
                    if row < len(self.map) - 1:
                        self.backtrack(row + 1, 0)
                    else:
                        self.finished = True
            else:
                if     (not     self.isFinalColNonV(row,     col))     or
self.possibleValue(row, col, self.map[row][col].value[0]):
                    self.backtrack(row, col + 1)
        return


    def possibleValue(self, row, col, value):
        possible = True
        i = row-1
        rowValue = value
        while self.map[i][col].type == "v":
            rowValue += self.map[i][col].value[0]
            if self.map[i][col].value[0] == value:
                possible = False
            i -= 1
        if self.map[i][col].type == "b":
            maxRowValue = self.map[i][col].value[0]
        elif self.map[i][col].type == "d":
            maxRowValue = self.map[i][col].value[0]
        if rowValue > maxRowValue or (rowValue != maxRowValue and
self.isFinalRowV(row, col)):
            possible = False


        j = col-1
        colValue = value
        while self.map[row][j].type == "v":
            colValue += self.map[row][j].value[0]
            if self.map[row][j].value[0] == value:
                possible = False
            j -= 1
        if self.map[row][j].type == "b":
            maxColValue = self.map[row][j].value[1]
        elif self.map[row][j].type == "r":
            maxColValue = self.map[row][j].value[0]
        if colValue > maxColValue or (colValue != maxColValue and
self.isFinalColV(row, col)):
            possible = False


        return possible
```

Pada cuplikan kode di atas, terdapat dua fungsi, yaitu backtrack() dan possibleValue(). Fungsi backtrack() merupakan implementasi dari fungsi pembakit yang berfungsi untuk mencari kotak putih yang kosong dan mencoba mengisi kotak putih kosong tersebut dengan bilangan 1 sampai dengan 9 sampai dia mencapai *goal node* yaitu seluruh kotak putih telah terisi oleh bilangan, baik dari awal sudah terisi oleh soal, maupun diisi oleh algoritma *backtracking*.

Sedangkan, fungsi possibleValue() merupakan implementasi dari fungsi pembatas yang berfungsi untuk membatasi kedua batasan yang telah disebutkan sebelumnya. Cara fungsi tersebut bekerja membuang lintasan-lintasan yang sudah pasti tidak berujung ke solusi adalah dengan sebelum melanjutkan pembentukan lintasan, cek dulu apakah dia masih

di dalam batasan yang sudah ditetapkan atau dia sudah keluar batas, kalau sudah keluar batas, maka pembentukan lintasan untuk simpul ekspan tersebut tidak akan lanjut ke langkah selanjutnya, sehingga tidak akan langkah-langkah redundan yang dilakukan menelusuri ke daun yang sudah pasti bukan solusi.

Cara kerja program secara keseluruhannya adalah pertama dia akan menerima input dari file eksternal berupa matriks dalam format tertentu yang menggambarkan permainan Kakuro. Lalu, program akan mulai memproses semua kotak yang berada pada matriks mulai dari yang paling atas kiri. Jika kotak tersebut merupakan kotak hitam, kotak putih yang sudah terisi atau kotak petunjuk, maka program akan langsung memproses kotak yang berada di sebelah kanannya kalau ada, kalau tidak ada, maka program akan memproses kotak pertama dari baris selanjutnya. Kalau kotak tersebut merupakan kotak putih yang kosong, maka kotak tersebut akan coba diisi dengan bilangan 1 sampai 9, lalu saat mencoba mengsi akan dipakai fungsi pembatas agar hanya yang dalam batasan yang dapat melanjutkan prosesnya. Jika batasan nya dilampaui, maka program akan melakukan *backtracking* ke kotak sebelumnya dan program akan berlanjut. Program akan terus berlanjut sampai sebuah solusi ditemukan atau semua kemungkinan solusi telah menjadi *dead node*.

IV. Perbandingan Algoritma Runut-Balik dan Bruteforce

Pada bagian ini , akan dijelaskan perbandingan antara algoritma runut-balik dengan algoritma bruteforce dalam menyelesaikan permainan Kakuro. Seperti yang sudah disebutkan pada bagian sebelumnya, secara teori, algoritma runut-balik akan lebih cepat daripada algoritma *bruteforce* untuk permainan Kakuro karena algoritma runut-balik tidak akan memproses lebih lanjut sebuah lintasan jika lintasan tersebut duah pasti tidak akan menghasilkan solusi berdasarkan fungsi pembatas, sehingga jumlah kemungkinan solusi yang di ceknya akan lebih sedikit.

Berikut ini akan diberikan bukti menggunakan program perbandingan antara algoritma *bruteforce* dan algoritma runut balik dalam menyelesaikan permainan Kakuro:

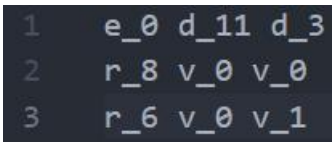
A. Penjelasan Format Input

Berikut ini adalah format penamaan yang dipakai di file eksternal:

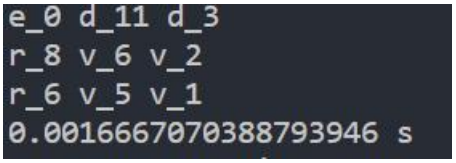
1. Setiap string yang dipisahi spasi merupakan satu kotak.
2. Pada setiap string, sebelum simbol '_' adalah tipe kotak.
    A. e : kotak hitam
    B. v : kotak hitam
    C. d : kotak petunjuk ke bawah
    D. r : kotak petunjuk ke kanan
    E. b : kotak petunjuk ke bawah dan ke kanan
3. Pada setiap string, setelah simbol '_' adalah value dari kotak.

B. Data Uji

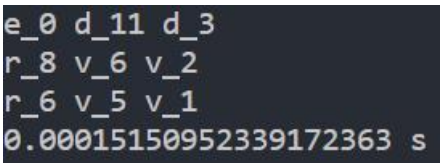
Waktu yang ditampilkan merupakan waktu rata-rata setelah 1000 kali percobaan.



Gambar 4 Input
(Data Pribadi)



Gambar 5 Output Bruteforce
(Data Pribadi)



Gambar 6 Output Runut-balik
(Data Pribadi)

C. Analisis data uji

Berdasarkan data uji di bagian sebelumnya, terbukti bahwa algoritma Runut-Balik jauh lebih baik daripada algoritma *bruteforce* karena seperti dilihat di data uji, dengan testing 1000 kali, algoritma Runut-Balik bekerja 10 kali lebih cepat daripada algoritma *bruteforce.*

Dengan itu, terbukti dengan program bahwa untuk kasus permainan Kakuro, algoritma Runut-Balik jauh lebih baik daripada algoritma *bruteforce* secara kompleksitas waktu.

V. Kesimpulan

Berdasarkan hal-hal yang telah saya sampaikan di atas, terbukti bahwa algoritma runut-balik cukup cocok untuk permainan Kakuro, apalagi kalau dibandingkan dengan algoritma *bruteforce.*

Namun, bukan berarti algoritma runut balik adalah yang terbaik, bisa jadi ada algoritma-algoritma lain yang mungkin lebih baik tapi belum dicoba saja kebagusannya dengan permainan Kakuro.

VI. Ucapan Terima Kasih

Penulis mengucapkan terima kasih ke semua pihak yang telah membantu dalam kelancaran dan selesainya pembuatan makalah ini. Pertama-tama penulis berterima kasih kepada Tuhan Yang Maha Esa atas berkat rahmat-Nya lah penulis

dapat membuat makalah yang berjudul "Penggunaan Algoritma Runut-Balik untuk Menyelesaikan Permainan Kakuro" ini. Lalu, penulis juga ingin berterimakasih kepada semua pihak lainnya yang telah .

DAFTAR PUSTAKA

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Runut-balik-(2018).pdf
diakses pada 26 April 2019

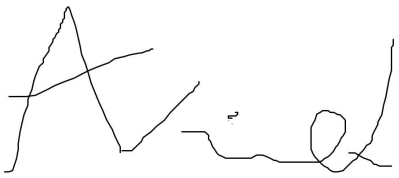
[2] https://www.daily-sudoku.com/kakurorules/
diakses pada 26 April 2019

[3] https://www.kakurolive.com/about-kakuro.php
diakses pada 26 April 2019

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, 26 April 2019



Ariel Ansa Razumardi 13517040