

Penerapan Algoritma Runut-Balik pada Permainan Minesweeper

Ditra Rizqa Amadia - 13521019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13521019@std.stei.itb.ac.id

Abstract—Minesweeper adalah sebuah permainan teka-teki yang populer selama beberapa dekade. Pemain harus menggunakan deduksi logika dalam memecahkan teka-teki pada permainan Minesweeper. Dalam makalah ini, penulis menyajikan pendekatan sistematis untuk mencari solusi pada teka-teki Minesweeper menggunakan strategi algoritma runut-balik.

Keywords—Minesweeper, pemecahan teka-teki, algoritma runu-balik, kepuasan batasan, kecerdasan buatan.

I. PENDAHULUAN

Minesweeper adalah sebuah permainan teka-teki klasik yang menantang pemain untuk menemukan ranjau tersembunyi tanpa meledakannya pada papan permainan berkotak. Pemain dapat mencari letak ranjau dengan bantuan angka pada masing-masing kotak yang menunjukkan jumlah ranjau pada kotak yang bertentangan. Dengan aturan yang sederhana dan permainan yang adiktif, Minesweeper telah memikat banyak orang selama beberapa dekade. Permainan ini menawarkan kombinasi yang membutuhkan keberuntungan dan strategi, mengharuskan pemain untuk membuat keputusan yang bijaksana dan mengandalkan kecerdasan logika untuk memecahkan teka-teki.

Namun, seiring dengan meningkatnya ukuran papan dan jumlah ranjau, memecahkan teka-teki Minesweeper menjadi semakin rumit dan memakan waktu. Ketika pemain dihadapkan pada papan yang besar dengan jumlah ranjau yang banyak, tugas untuk menemukan ranjau dengan aman tanpa melakukan kesalahan meledakkan ranjau menjadi semakin sulit. Dalam beberapa kasus, pemain mungkin terjebak dalam situasi yang di mana keputusan sulit harus dibuat dengan risiko tinggi.

Dalam makalah ini, penulis menyajikan sebuah pendekatan analitis dalam memecahkan teka-teki Minesweeper menggunakan algoritma runut-balik. Algoritma runut-balik adalah teknik algoritma yang dapat mencari semua solusi yang mungkin dengan membentuk kombinasi solusi yang mungkin, memeriksa ketidaksesuaian pada masing-masing solusi, dan membatalkan solusi yang salah jika diperlukan. Algoritma ini memungkinkan pemecahan teka-teki Minesweeper dengan pendekatan yang terarah, dengan mencoba semua kemungkinan langkah dan membuat keputusan yang cerdas berdasarkan informasi yang diperoleh. Dengan menerapkan strategi

algoritma ini, penulis bertujuan untuk menyediakan metode yang efisien dan dapat diandalkan untuk memecahkan teka-teki Minesweeper dengan tingkat kesulitan yang bervariasi.

Tujuan dari makalah ini adalah mengembangkan algoritma yang dapat secara efektif memecahkan teka-teki Minesweeper dengan mempertimbangkan aturan dan batasan permainan. Pendekatan ini memanfaatkan algoritma runut-balik yang dikenal luas karena kemampuannya dalam menangani masalah yang memiliki batasan dengan efisien. Dengan cara sistematis mengevaluasi kemungkinan penempatan ranjau dan menduga pembukaan kotak yang aman, algoritma yang penulis tunjukkan bertujuan menemukan solusi yang benar untuk setiap teka-teki Minesweeper yang diberikan.

Dengan memperkenalkan pendekatan sistematis dan analitis dalam pemecahan permainan Minesweeper menggunakan algoritma runut-balik, makalah ini bertujuan untuk berkontribusi pada bidang yang lebih luas dalam algoritma pemecahan teka-teki. Wawasan yang diperoleh dari makalah ini dapat diaplikasikan dalam kecerdasan buatan dan masalah optimisasi yang melibatkan batasan.

II. LANDASAN TEORI

A. Minesweeper

Minesweeper adalah permainan teka-teki komputer populer di mana pemain diberikan papan berkotak yang awalnya ditutup. Tujuan dari permainan ini adalah untuk mengungkap semua kotak aman di papan sambil menghindari kotak beranjau. Permainan ini membutuhkan kombinasi deduksi logis dan pengambilan keputusan strategis.

Dalam Minesweeper, permainan direpresentasikan sebagai papan berbentuk persegi yang terdiri dari beberapa kotak. Setiap kotak dapat berada dalam salah satu dari tiga keadaan: tertutup, terbuka atau bertanda bendera. Kotak-kotak yang tertutup awalnya disembunyikan dari pemain, dan isinya dapat berupa ranjau. Kotak yang terungkap bertuliskan angka yang menunjukkan jumlah ranjau pada kotak bertentangan. Kotak-kotak yang diberi tanda bendera ditandai oleh pemain untuk menunjukkan lokasi dugaan ranjau.

Tugas pemain adalah untuk mengungkap semua kotak aman tanpa meledakkan ranjau. Dengan menggunakan petunjuk angka yang terungkap saat membuka kotak, pemain harus menduga keberadaan atau ketiadaan ranjau di kotak-

kotak tetangga. Tujuannya adalah untuk secara strategis mengidentifikasi dan memberi tanda bendera pada lokasi ranjau sambil secara bertahap mengungkap kotak-kotak aman. Permainan dimenangkan ketika semua kotak aman telah terungkap, dan permainan berakhir dengan kekalahan jika sebuah ranjau meledak.

Aturan dalam Minesweeper adalah sebagai berikut:

- 1) Papan permainan terdiri dari kotak-kotak, beberapa di antaranya berisi ranjau yang tersembunyi.
- 2) Pemain memulai permainan dengan mengklik salah satu kotak tertutup yang kemudian terungkap.
- 3) Setiap kotak yang terungkap mengungkapkan angka yang menunjukkan jumlah ranjau yang berdekatan mulai dari 0 hingga 8.
- 4) Angka-angka pada kotak-kotak yang terungkap berfungsi sebagai petunjuk untuk menentukan keberadaan ranjau di kotak-kotak tetangga.
- 5) Pemain dapat memberikan bendera pada suatu kotak yang dicurigai mengandung ranjau.
- 6) Kotak-kotak yang terungkap tanpa ranjau yang berdekatan secara otomatis akan mengungkap sel-sel tetangganya hingga menemui ranjau yang berdekatan.
- 7) Jika sebuah ranjau terungkap, permainan berakhir, dan pemain kalah.
- 8) Jika semua kotak aman terungkap tanpa meledakkan ranjau apa pun, permainan berakhir, dan pemain menang.

B. Constraint Satisfaction Problem (CSP)

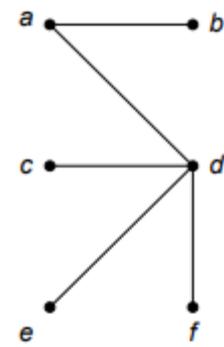
Masalah kepuasan batasan atau *Constraint Satisfaction Problem (CSP)* adalah masalah komputasi di mana tujuannya adalah mencari solusi yang memenuhi sekumpulan batasan [1]. Masalah ini melibatkan pencarian nilai untuk sekumpulan variabel, dengan mempertimbangkan batasan atau kondisi tertentu untuk memenuhi semua batasan yang ditentukan secara simultan.

Dalam masalah kepuasan batasan, variabel-variabel menggambarkan nilai-nilai yang belum diketahui atau variabel keputusan, sementara batasan-batasan menentukan hubungan dan pembatasan antara variabel-variabel tersebut. Tujuannya adalah menetapkan nilai-nilai pada variabel-variabel tersebut sedemikian rupa sehingga semua batasan terpenuhi. Solusi untuk CSP merupakan kombinasi dari penugasan nilai pada variabel yang memenuhi semua batasan yang diberikan.

CSP muncul dalam berbagai domain dan memiliki berbagai aplikasi termasuk kecerdasan buatan, penjadwalan, perencanaan, dan pemecahan teka-teki. CSP menyediakan kerangka formal untuk pemodelan dan pemecahan masalah yang melibatkan batasan dan ketergantungan logis.

C. Pohon

Pohon adalah graf tak berarah terhubung yang tidak mengandung sirkuit [2].

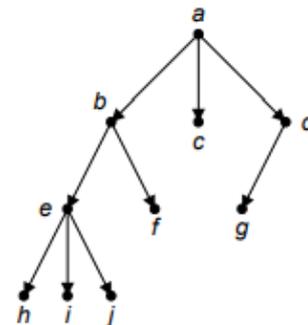


Gambar 2.1 Pohon, sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf>

Misalkan $G = (V, E)$ adalah graf tak berarah sederhana dan jumlah simpulnya n . Maka, semua pernyataan di bawah ini adalah ekuivalen:

- G adalah pohon
- Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal
- G terhubung dan memiliki $m = n - 1$ buah sisi
- G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi
- G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit
- G terhubung dan semua sisinya adalah jembatan

Pohon berakar (*rooted tree*) adalah pohon yang satu buah simpulnya diperlukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah [3].



Gambar 2.2 Pohon berakar, sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf>

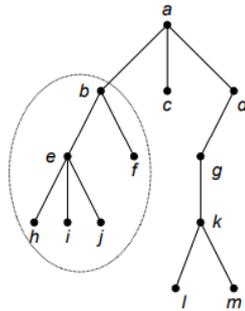
Anak (*child*) adalah anak simpul dan orangtua (*parent*) adalah orangtua dari simpul. Pada 2.2 simpul b adalah orangtua dari simpul e dan f . Simpul e dan f adalah anak dari simpul b , namun simpul b bukan orangtua dari simpul g .

Lintasan (*path*) adalah rute dari suatu simpul ke simpul lain melewati satu atau beberapa sisi. Pada 2.2 lintasan a ke j adalah a, b, e, j . Panjang lintasan dari a ke j yaitu 3.

Saudara kandung (*sibling*) adalah relasi simpul dengan simpul orangtua yang sama. Pada gambar pohon berakar simpul f adalah saudara kandung simpul e , tetapi simpul g

bukan saudara kandung simpul e karena simpul orangtua mereka berbeda.

Upapohon (*subtree*) adalah bagian pohon yang terdapat pada pohon utama. Pohon dapat dibentuk dari beberapa upapohon secara rekursif.



Gambar 2.3 Upapohon sumber:

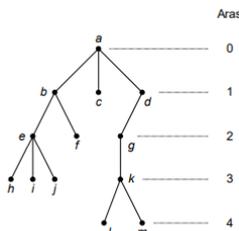
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf>

Derajat (*degree*) adalah banyaknya upapohon (atau jumlah anak) pada simpul tersebut. Pada 2.2 derajat a adalah 3, derajat b adalah 2, derajat d adalah 1, dan derajat c adalah 0. Derajat yang dimaksud pada pohon adalah derajat keluar. Derajat maksimum dari semua simpul merupakan derajat pohon itu sendiri. Pada gambar pohon berakar derajat maksimum pohon yaitu berderajat 3.

Daun (*leaf*) adalah simpul yang derajatnya 0 (atau tidak mempunyai anak). Pada 2.2 simpul h, i, j, f, c, dan g adalah daun.

Simpul dalam (*internal nodes*) adalah simpul yang mempunyai anak. Pada 2.2 simpul a, b, d, dan e adalah simpul dalam.

Aras (*level*) adalah tingkat suatu simpul pada pohon relatif terhadap akarnya.

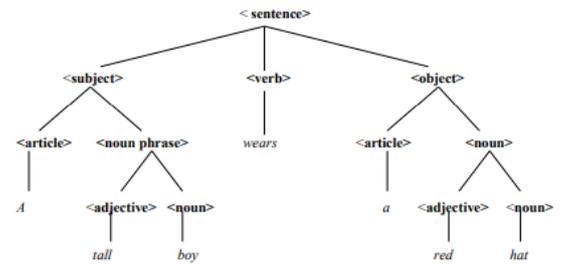


Gambar 2.4 Aras pohon, sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf>

Tinggi (*height*) atau kedalaman (*depth*) adalah aras maksimum dari suatu pohon. Pada 2.4 kedalamannya adalah 4.

Pohon n-ary adalah pohon berakar yang setiap simpul cabangnya mempunyai paling banyak n buah anak.



Gambar 2.5 Pohon n-ary, sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf>

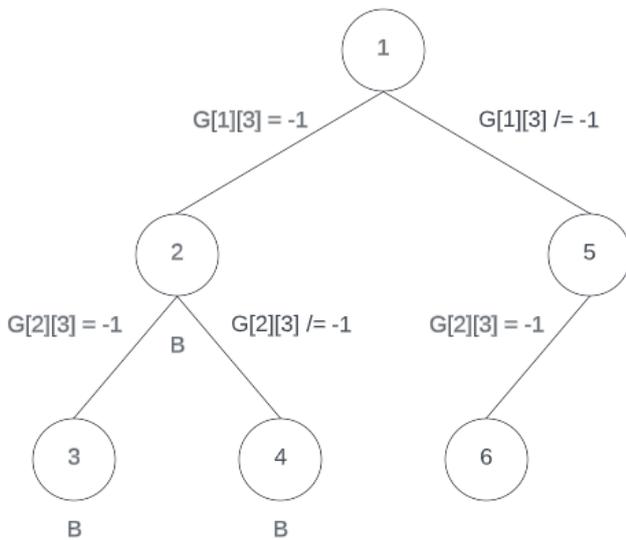
Pohon n-ary dikatakan teratur atau penuh jika setiap simpul cabangnya mempunyai tepat n anak.

D. Backtracking Algorithm

Runut-balik atau *backtracking* adalah sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis, baik untuk persoalan optimasi maupun non-optimasi[4]. Runut-balik adalah variasi pendekatan yang lebih cerdas. Ide utamanya adalah membangun solusi satu komponen pada satu waktu dan mengevaluasi kandidat yang sebagian terkonstruksi sebagai berikut. Jika solusi yang sebagian terkonstruksi dapat dikembangkan lebih lanjut tanpa melanggar batasan masalah, itu dilakukan dengan mengambil opsi yang sah pertama untuk komponen berikutnya yang tersisa. Jika tidak ada opsi yang sah untuk komponen berikutnya, tidak perlu mempertimbangkan alternatif untuk komponen yang tersisa. Dalam kasus ini, algoritma melakukan backtrack untuk menggantikan komponen terakhir solusi yang sebagian terkonstruksi dengan opsi berikutnya[6].

Algoritma runut-balik adalah teknik pemecahan masalah yang berbasis rekursi, di mana solusi dicari secara sistematis dengan mencoba semua kemungkinan, dan mundur (*backtrack*) ketika solusi tidak memenuhi batasan atau keadaan yang tidak diinginkan. Algoritma ini sering digunakan untuk mencari solusi dalam ruang pencarian yang besar dan kompleks, di mana pendekatan *brute force* tidak efisien.

Algoritma runut-balik memanfaatkan sifat rekursif untuk menggambarkan langkah-langkah berulang dalam ruang pencarian. Pada setiap langkah, algoritma mencoba satu pilihan, dan jika pilihan tersebut tidak menghasilkan solusi yang sah, algoritma akan rekursif melakukan *backtrack* untuk mencoba pilihan lain. Proses ini berlanjut sampai solusi ditemukan atau seluruh ruang pencarian telah dieksplorasi.



Gambar 3.3 Pohon pencarian teka-teki A

Algoritma pertama-tama memeriksa kotak $G[1][2]$. Kotak memiliki $\text{numOfMines}(G[1][2]) = 1$ sehingga terdapat 1 ranjau pada salah satu kotak tetangganya. $G[1][2]$ memiliki dua kotak tetangga yang belum terbuka yaitu $G[1][3]$ dan $G[2][3]$.

Pada gambar 3.3 algoritma membuat pohon kemungkinan solusi dan mengeksplorasi semua kemungkinan tersebut. Simpul 1 menunjukkan kondisi awal di saat isi dari $G[1][3]$ dan $G[2][3]$ masih belum diketahui. Algoritma kemudian mengevaluasi kemungkinan solusi bahwa $G[1][3]$ berisi ranjau sehingga pohon membentuk simpul baru yaitu simpul 2. Simpul ini memenuhi batasan yaitu

$$\begin{aligned} \text{numOfMines}(G[1][2]) - \text{actualNumOfMines}(G[1][2]) &= 0 \\ 1 - 1 &= 0 \\ \text{dan} \\ \text{numOfMines}(G[2][2]) - \text{actualNumOfMines}(G[2][2]) &= 0 \\ 1 - 1 &= 0. \end{aligned}$$

Kemudian algoritma mengevaluasi kemungkinan solusi baru bahwa $G[2][3]$ berisi ranjau sehingga pohon membentuk simpul baru yaitu simpul 3. Namun asumsi tersebut melanggar batasan

$$\begin{aligned} \text{numOfMines}(G[1][2]) - \text{actualNumOfMines}(G[1][2]) &= 0 \\ 1 - 2 &\neq 0 \\ \text{dan} \\ \text{numOfMines}(G[2][2]) - \text{actualNumOfMines}(G[2][2]) &= 0 \\ 1 - 2 &\neq 0, \end{aligned}$$

sehingga algoritma melakukan *backtrack* kembali ke simpul 2.

Dari simpul 2, algoritma mengevaluasi kemungkinan solusi bahwa $G[2][3]$ tidak berisi ranjau sehingga pohon membentuk simpul baru yaitu simpul 4. Namun solusi ini juga salah karena tidak memenuhi

$$\begin{aligned} \text{numOfMines}(G[3][2]) - \text{actualNumOfMines}(G[3][2]) &= 0 \\ 1 - 0 &\neq 0, \end{aligned}$$

sehingga algoritma melakukan *backtrack* kembali ke simpul 2. Semua kemungkinan solusi pada simpul 2 sudah dieksplorasi

dan tidak ada yang memenuhi sehingga algoritma melakukan *backtrack* lagi ke simpul 1.

Dari simpul 1, algoritma mengevaluasi kemungkinan solusi bahwa $G[1][3]$ tidak berisi ranjau. Walaupun

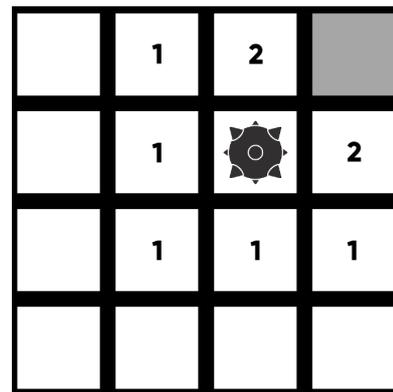
$$\text{numOfMines}(G[1][2]) - \text{actualNumOfMines}(G[1][2]) = 0$$

belum terpenuhi, namun masih terdapat kotak tetangga yang kosong dan banyak kotak tetangga yang kosong lebih besar sama dengan jumlah ranjau yang dibutuhkan kotak $G[1][2]$ sehingga evaluasi belum melanggar batasan.

Kemudian algoritma mengevaluasi kemungkinan solusi baru bahwa $G[2][3]$ berisi ranjau dan evaluasi tersebut memenuhi semua batasan yang ada yaitu

$$\begin{aligned} \text{numOfMines}(G[1][2]) - \text{actualNumOfMines}(G[1][2]) &= 0 \\ 1 - 1 &= 0, \\ \text{numOfMines}(G[2][2]) - \text{actualNumOfMines}(G[2][2]) &= 0 \\ 1 - 1 &= 0, \\ \text{numOfMines}(G[3][2]) - \text{actualNumOfMines}(G[3][2]) &= 0 \\ 1 - 1 &= 0, \\ \text{numOfMines}(G[3][3]) - \text{actualNumOfMines}(G[3][3]) &= 0 \\ 1 - 1 &= 0, \\ \text{numOfMines}(G[3][4]) - \text{actualNumOfMines}(G[3][4]) &= 0 \\ 1 - 1 &= 0, \end{aligned}$$

sehingga solusi pada simpul 6 adalah solusi yang benar. Gambar 3.4 menunjukkan hasil akhir dari solusi simpul 6.



Gambar 3.4 Solusi Teka-teki A

IV. IMPLEMENTASI

Implementasi dimulai dengan membuat representasi papan awal dengan sebuah matriks. Angka dalam matriks menunjukkan beberapa kondisi yaitu -1 menunjukkan sel yang belum diungkap, 0 untuk sel kosong, dan angka positif menunjukkan jumlah ranjau yang ada pada kotak-kotak tetangganya.

```

board = [
  [1, -1, 1, 1],
  [1, 2, -1, 1],
  [0, 1, 1, 1],
  [0, 0, 0, 0]
]

```

Gambar 4.1 Representasi papan sebagai matriks

Kedua, program melakukan iterasi melalui setiap kotak pada papan.

Ketiga, evaluasi untuk setiap kotak, apakah kotak tersebut tertutup atau terbuka. Apabila kotak terbuka dan berisi angka positif, program memeriksa jumlah kota yang belum terungkap yang bertetangga dengan kotak yang sedang dievaluasi. Jika jumlah kotak yang belum terungkap sama dengan angka pada kotak yang sedang dievaluasi, maka seluruh kotak tetangga yang belum terungkap merupakan ranjau. Apabila jumlah ranjau yang sudah terungkap pada kotak bertetangga sama dengan angka pada kotak yang sedang dievaluasi, maka semua kotak tetangga yang belum terungkap merupakan kotak yang aman.

```
def backtrack(row, col):
    # basis rekursif
    if row < 0 or row >= rows or col < 0 or col >= cols or board[row][col] != -1:
        return

    board[row][col] = 0

    # Periksa semua kotak tetangga
    for evaluatingRow in [-1, 0, 1]:
        for evaluatingColumn in [-1, 0, 1]:
            if evaluatingRow == 0 and evaluatingColumn == 0:
                continue
            new_row = row + evaluatingRow
            new_col = col + evaluatingColumn
            if new_row >= 0 and new_row < rows and new_col >= 0 and new_col < cols:
                if board[new_row][new_col] == -1: # Kotak belum terungkap
                    board[new_row][new_col] = 0 # Kotak diungkap
                    if is_mine(board, new_row, new_col):
                        board[new_row][new_col] = -1
                        continue
                # Eksplorasi kotak tetangga lainnya secara rekursif
                backtrack(new_row, new_col)
```

Gambar 4.2 Algoritma runut-balik

Keempat, apabila salah satu syarat pada langkah ketiga tidak terpenuhi, maka program akan mencari semua kemungkinan solusi menggunakan algoritma runut-balik. Apabila sel yang terungkap adalah ranjau, program melakukan runut-balik dan menjelajahi solusi lain. Apabila kotak yang diungkap aman, maka program mengulangi lagi dari langkah 3.

Program mengulangi langkah 3 sampai 4 hingga semua kotak aman telah diungkap.

V. KESIMPULAN

Berdasarkan hasil implementasi di atas, teori algoritma runut-balik dapat diaplikasikan ke dalam pencarian solusi permainan Minesweeper. Solusi ditemukan dengan cara mencoba seluruh kemungkinan solusi pada ruang pencarian menggunakan algoritma runut-balik. Pencarian solusi menggunakan algoritma ini lebih cepat dibandingkan dengan pencarian menggunakan algoritma *brute force*.

VI. PENUTUP

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa yang telah mengaruniakan kelancaran dan

kemudahan untuk penulis dalam penyusunan makalah ini. Penulis ucapkan terima kasih untuk kedua orangtua penulis atas dukungannya selama proses pendidikan penulis. Tidak lupa juga penulis ucapkan terima kasih kepada Ir. Rila Mandala, M.Eng., Ph.D. sebagai dosen pengampu mata kuliah strategi algoritma (IF2211) untuk kelas K3 yang telah menjadi pembimbing penulis dalam menjalani mata kuliah ini hingga memberi kesempatan kepada penulis untuk menulis makalah ini.

REFERENSI

- [1] A. A. Bulatov, V. Guruswami, A. Krokhin, and D. Marx. The Constraint Satisfaction Problem: Complexity and Approximability, Dagstuhl: 2015, pp. 23-25
- [2] Munir, Rinaldi (2003). Pohon (Bag. 1) Bahan Kuliah IF2120 Matematika Diskrit, URL: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf> Diakses: 22 Mei 2023
- [3] Munir, Rinaldi (2003). Pohon (Bag. 2) Bahan Kuliah IF2120 Matematika Diskrit, URL: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf> Diakses: 22 Mei 2023
- [4] Munir, Rinaldi (2003). Algoritma Runut-balik (Bag. 1) Bahan Kuliah IF2120 Matematika Diskrit, URL: https://cdn-edunex.itb.ac.id/38015-Algorithm-Strategies-Parallel-Class/88439-Algorithm-Runut-balik-Backtracking/1647232773709_Algoritma-backtracking-2021-Bagian1.pdf Diakses: 22 Mei 2023
- [5] Munir, Rinaldi (2003). Algoritma Runut-balik (Bag. 2) Bahan Kuliah IF2120 Matematika Diskrit, URL: https://cdn-edunex.itb.ac.id/38015-Algorithm-Strategies-Parallel-Class/88439-Algorithm-Runut-balik-Backtracking/1647403518191_Algoritma-backtracking-2021-Bagian2.pdf Diakses: 22 Mei 2023
- [6] Leivitin, Anany. Introduction to The Design & Analysis of Algorithms 3rd ed, Pearson: 2012. pp. 424

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023


Ditra Rizqa Amadia
13521019