Penerapan Algoritma *Backtracking* dalam Menyelesaikan *Sudoku with 4 Given Digits*

Putri Nadia Salsabila (13518094)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518094@std.stei.itb.ac.id

***Abstrak*—Sudoku adalah puzzle penempatan angka yang berbasis logika dan kombinasi. Sudoku dimainkan dalam sebuah tabel berbentuk persegi berukuran 9x9, yang didalam tabel tersebut berisi 9 blok, sehingga setiap kolom, setiap baris, dan setiap blok mengandung semua digit dari 1 hingga 9. Algoritma mempunyai peranan yang penting dalam pemngembangan penyelesaian suatu teka-taki. Salah satu algoritma yang banyak digunakan dalam penyelesaian masalah teka-teki adalah algoritma Backtracking. Algoritma Backtracking berperan dalam menentukan jawaban dari permainan Sudoku. Pada makalah ini, akan digunakan algoritma Backtracking dalam menyelesaikan sudoku yang hanya diberikan 4 digit sebagai petunjuk.**

Keywords—algoritma, backtracking, sudoku, 4 digit.

I. PENDAHULUAN

Algoritma Backtracking merupakan sebuah alat yang penting untuk dapat menyelesaikan permasalahan pemenuhan terbatas, seperti teka – teki silang, aritmatika verbal, sudoku dan berbagai macam puzzle sejenisnya. Dalam penggunaannya backtracking bergantung pada perintah yang diberikan oleh penggunanya (prosedur kotak hitam) yang mana menentukan permasalahan untuk diselesaikan dengan cara kandidat parsial dan bagaimana mereka dikembangkan untuk menjadi kandidat penyelesaian maslaah secara sepenuhnya.

Sudoku merupakan permainan yang sangat terkenal dan digemari karena dapat melatih logika dan otak. Sudoku memilik peraturan tidak boleh ada angka yang sama dalam setiap baris, setiap kolom, dan setiap blok. Sudoku terus mengalami perkembangan. Salah satu masalah sudoku yang dikembangkan adalah sudoku dengan 4 *given digits* atau yang dikenal dengan nama *Magic Square Sudoku*.

II. LANDASAN TEORI

A. Sudoku

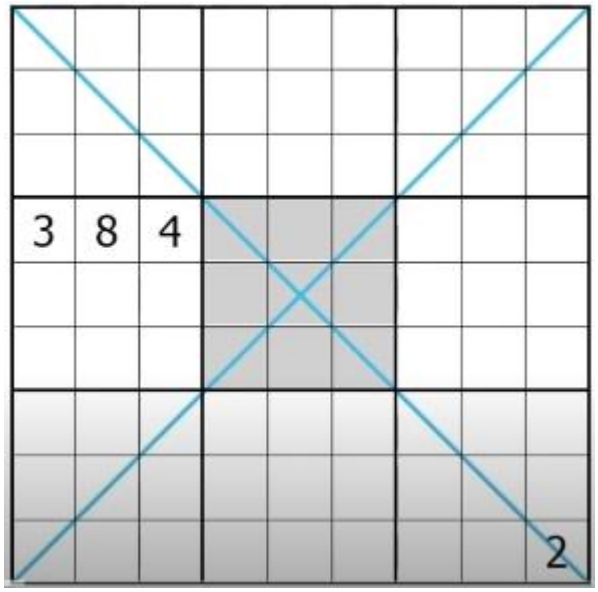
Sudoku (数独 sūdoku), yang juga dikenal Number Place atau Nanpure, adalah sejenis teka-teki logika. Tujuannya adalah untuk mengisikan angka-angka dari 1 sampai 9 ke dalam jaring-jaring 9×9 yang terdiri dari 9 kotak 3×3 tanpa ada angka yang berulang di satu baris, kolom atau kotak.

Sudoku pertama kali diterbitkan di sebuah surat kabar Prancis pada 1895. Permainan sudoku dipengaruhi oleh matematikawan Swiss Leonhard Euler, yang membuat terkenal Latin square.

Nama "Sudoku" adalah singkatan bahasa Jepang dari "Suuji wa dokushin ni kagiru" (数字は独身に限る), artinya "angka-angkanya harus tetap tunggal".

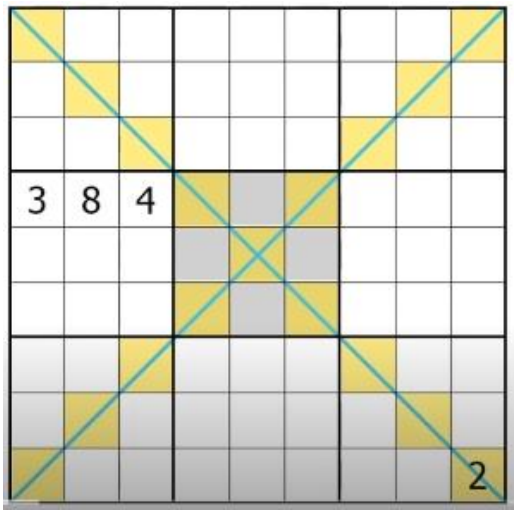
B. Sudoku dengan 4 Given Digits

Sudoku dengan 4 *given digits* ini diciptakan oleh Aad van de Wetering. Sudoku ini juga mempunyai nama Magic Square Sudoku. Sudoku ini memiliki peraturan yang sama dengan sudoku yang lainnya yaitu angka hanya dapat muncul sekali dalam setiap baris, kolom, dan blok.

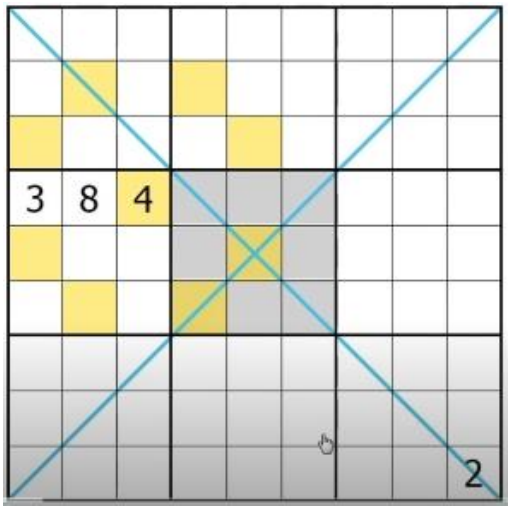


Pada Sudoku ini terdapat konstrain tambahan, yaitu:

1. Dua diagonal utama pada sudoku, mengandung angka 1-9 pada setiap diagonalnya



2. Sel yang mempunyai *knight move* (seperti di catur), tidak boleh mengandung angka yang sama

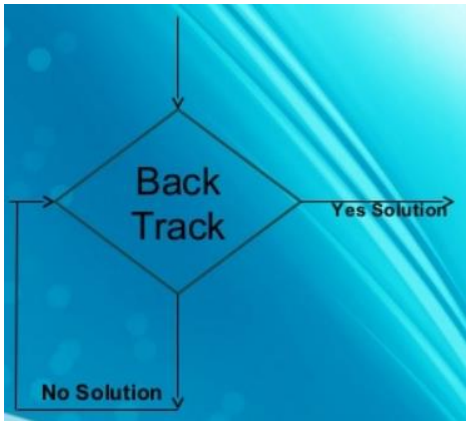


3. Blok yang berada di tengah sudoku, harus membentuk *magic square*, yaitu setiap kolom, baris, dan diagonal, ketika ditambah akan menghasilkan angka yang sama.

C. Algoritma Backtracking

Algoritma backtracking pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Dalam perkembangannya R. J Walker merupakan orang yang pertama kali memberikan deskripsi algoritmik backtracking pada tahun 1960. Kemudian banyak pula tokoh-tokoh seperti S. Golamb dan L. Baumer yang mengembangkan algoritma backtracking ini.

Algoritma runut-balik atau backtracking adalah algoritma perbaikan dari algoritma brute-force yang secara sistematis mencoba seluruh cara untuk mencari solusi dari sebuah masalah. Algoritma runut-balik berbasis pada algoritma Deep-First Search (DFS), Pada algoritma DFS pencarian solusi dilakukan dengan cara menelusuri node-node dari sebuah tree secara pre-order.



Gambar 1. Ilustrasi penggambaran algoritma backtracking (sumber: http://www.w3.org/2011/Talks/01-14-steven-phenotype/)

Backtracking adalah teknik algoritmik untuk menyelesaikan masalah secara rekursif dengan mencoba membangun solusi secara bertahap, satu per satu, menghilangkan solusi yang gagal memenuhi kendala masalah di setiap titik waktu (saat ini, disebut waktu berlalu hingga mencapai level pohon pencarian mana saja).

Nama backtrack didapatkan dari sifat algoritma ini yang memanfaat karakteristik himpunan solusinya yang sudah disusun menjadi suatu pohon solusi.

Ada tiga tipe masalah dalam algoritma Backtracking, yaitu:

1. *Decision Problem* - Mencari solusi yang layak.
2. Masalah Optimasi - Mencari solusi terbaik.
3. *Enumerarion Problem* - Menemukan semua solusi yang layak.

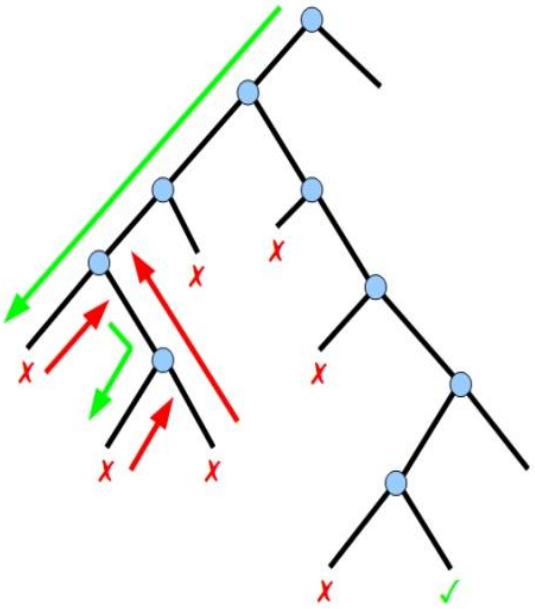
Langkah-langkah pencarian solusi dengan algoritma Backtracking adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Simpul yang sudah dilahirkan dinamakan simpul hidup dan simpul yang sedang diperluas dinamakan simpul E atau simpul *expand*. Aturan penomoran simpul dilakukan dari atas ke bawah, dilakukan sesuai dengan urutan pembentukan.

2. Jika lintasan yang sedang dibentuk dari perluasan simpul *expand* tidak mengarah ke solusi, maka simpul *expand* tersebut dibunuh dan menjadi simpul mati. Dimana simpul mati merupakan simpul yang tidak akan diperluas lagi. Bounding merupakan fungsi yang digunakan untuk membunuh simpul *expand.*

3. Jika posisi terakhir ada di simpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lain dan jika tidak ada simpul anak yang lain maka dilakukan backtracking ke simpul orang tuanya.

4. Pencarian dihentikan jika telah menemukan solusi atau tidak ada lagi simpul hidup yang dapat diperluas.



Gambar 2. Ilustrasi penyelesaian dengan algoritma backtracking (sumber: http://www.w3.org/2011/Talks/01-14-steven-phenotype/)

D. Properti Umum Metode Backtracking

Algoritma Backtracking mempunyai properti:

1. Ruang solusi

   Ruang solusi adalah ruang yang menyatakan kumpulan solusi suatu masalah. Ruang solusi dinyatakan sebagia vektor dengan n-tuple: $X = (x_1, x_2, x_3, ..., x_n)$, $x_i \in S_i$. Dengan adanya kemungkinan $S_1 = S_2 = ... = S_n$

   Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1

2. Fungsi pembangkit

   Fungsi pembangkit dinyatakan sebagai predikat T(k) . T(k) membangkitkan nilai untuk X_k atau solusi potensial, yang merupakan komponen vektor solusi.

3. Fungsi Pembatas

   Fungsi pembatas dinyatakan sebagai predikat $B(x_i, x_2, x_3, ..., x_i)$ untuk setiap solusi potensial yang dibangkitkan oleh T(k). B bernilai true jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika true, maka pembangkitan nilai untuk x_k+1 dilanjutkan, tetapi jika false, maka $(x_1, x_2, \ldots, x_k)$ dibuang.

E. Kelebihan dan Kekurangan Algoritma Backtracking

Kelebihan dari algoritma Backtracking:

1. Dibandingkan dengan Algoritma Pemrograman Dinamis, algoritma backtracking dalam beberapa kasus lebih efektif.
2. Algoritma Backtracking merupakan pilihan terbaik dalam menyelesaikan *tactical problem.*
3. Algoritma Backtracking efektif dalam menyelesaikan *constraint satisfication problem.*
4. Pada algoritma Greedy, mendapatkan solusi global yang optimal membutuhkan prosedur yang lama dan bergantung pada *user statements*, tapi pada algoritma Backtracking solusi global yang optimal merupakan hal yang mudah didapat.
5. Algoritma Backtracking merupakan algoritma yang sederhana untuk diimplementasi dan mudah untuk diprogram.
6. Akurasi dari algoritma Backtracking sudah pasti.
7. States yang berbeda disimpan dalam stack, sehingga data dapat digunakan kapanpun.

Kekurangan algoritma Backtracking:

1. Pendekatan algoritma Backtracking merupakan pendekatan yang tidak efisien terhadap penyelesaian *strategic problem.*
2. Keseluruhan *runtime* dari algoritma Backtracking pada dasarnya lambat.
3. Untuk menyelesaikan permbasalahan yang besar, terkadang algoritma Backtracking membutuhkan bantuan dari algoritma yang lain, seperti algoritma Branch and Bound.
4. Algoritma Backtracking membutuhkan *memory space* yang besar untuk menampung fungsi state yang berbeda dalam stack untuk permasalahan yang besar.
5. *Thrashing* merupakan salah satu masalah utama dari algoritma Backtracking

F. Skema Umum Algoritma Backtracking

Dalam algoritma Backtracking, pencarian dapat dilakukan dengan iteratif maupun rekursif. Akan tetapi jika pencarian dilakukan secara iteratif, dibutukan pula *incrementing* dan *decrementing*secara manual beserta pengontrolnya.

Pencarian menggunakan skema rekrsif dianggap lebih tepat dan efisien dikarenakan pencarian solusi dalam pembangkitan simpul lebih sesuai dengan algoritma backtracking. Berikut ini skema iteratif dan skema rekursif dari algoritma Backtracking:

1.Skema iteratif

```
procedure RunutBalikI(input n:integer) { versi iteratif }
{Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif.
 Masukan: n, yaitu panjang vektor solusi
 Keluaran: solusi x = (x[1], x[2], ..., x[n])}

Delarasi:
    k : integer

Algoritma:
    k ← 1
    while k > 0 do
        if (x[k] belum dicoba sedemikian sehingga x[k] ← T(k)) and
          (B(x[1], x[2], ... ,x[k])= true)
        then
          if (x[1],x[2],...,x[k]) adalah lintasan dari akar ke daun
          then
            CetakSolusi(x)
          endif
          k ← k+1 {indeks anggota tupple berikutnya}
        else {x[1], x[2],..., x[k] tidak mengarah ke simpul solusi }
          xk ← k-1 {runut-balik ke anggota tupple sebelumnya}
    endif
    endwhile
    { k = 0 }
```

Gambar 3. Pseudocode iteratif backtracking (sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Runut-balik-(2020).pdf)

2.Skema rekursif

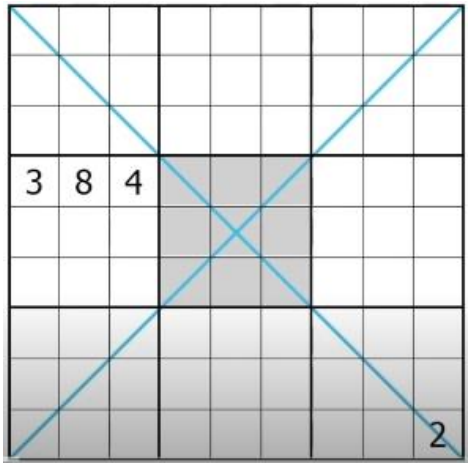
```
void findSolutions(n, other params) :
    if (found a solution) :
        solutionsFound = solutionsFound + 1;
        displaySolution();
        if (solutionsFound >= solutionTarget) :
            System.exit(0);
        return

    for (val = first to last) :
        if (isValid(val, n)) :
            applyValue(val, n);
            findSolutions(n+1, other params);
            removeValue(val, n);
```

Gambar 4. Pseudocode iteratif backtracking (sumber: https://www.geeksforgeeks.org/backtracking-introduction/)

III. IMPLEMENTASI ALGORITMA BACKTRACKING TERHADAP PENYELESAIAN SUDOKU DENGAN 4 *GIVEN DIGITS*

Implementasi alagoritma Backtracking pada permainan sudoku dengan 4 *given digits* akan dibahas pada bagian ini.

Sudoku dengan 4 *given digits* ini berbentuk:



Langkah Penyelesaian:

1. Menentukan angka di blok tengah sudoku, yang harus menghasilkan *magic square*

   Pada *magic square*, angka 5 selalu menempati posisi di tengah. Serta pada setiap baris dan kolom pada *magic square* di sudoku ini berjumlah 15.

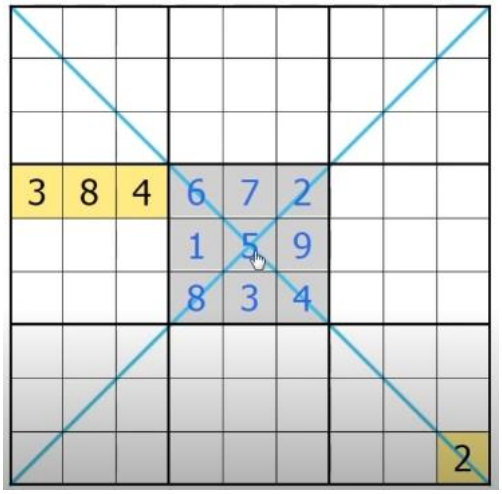
```
#initialise empty 3 by 3 grid
grid = []
grid.append([0,0,0])
grid.append([0,5,0])
grid.append([0,0,0])

SUM=15 #Each Row, Column and Diagonal will add up to 15
```

   Kemudian dilakukan bactracking untuk menemukan semua kemungkinan dari *magic square.* Dilakukan pula fungsi pengecekan, untuk mengecek apakah kondisi dari *magic square* terpenuhi.

```
#A backtracking/recursive function to check all possible combinations
def solveGrid(grid):
  #Find next empty cell
  for i in range(0,9):
    row=i//3
    col=i%3
    if grid[row][col]==0:
      for value in range (1,10):
        #Can only use numbers that have not been used yet
        if not(value in grid[0] or value in grid[1] or value in grid[2]):
          grid[row][col]=value
          #sleep(0.0001)
          myPen.clear()
          drawGrid(grid)
          myPen.getscreen().update()
          if checkGrid(grid):
            print("Grid Complete and Checked")
            return True
          else:
            if solveGrid(grid):
              return True
      break
  print("Backtrack")
  grid[row][col]=0
```

```
#A function to check if the grid is a magic square
def checkGrid(grid):
  global SUM
  for row in range(0,3):
      for col in range(0,3):
        if grid[row][col]==0:
          return False
  for row in range(0,3):
    if (grid[row][0]+grid[row][1]+grid[row][2])!=SUM:
      return False
  for col in range(0,3):
    if (grid[0][col]+grid[1][col]+grid[2][col])!=SUM:
     return False
  if (grid[0][0]+grid[1][1]+grid[2][2])!=SUM:
   return False
  if (grid[0][2]+grid[1][1]+grid[2][0])!=SUM:
   return False

  #We have a magic square!
  return True
```

2. Dengan algoritma bactracking ditemukan jawaban untuk mengisi blok tengah dari sudoku 4 *given digits* ini.



3. Dari sudoku yang diisi, ditentukan angka yang lain menggunakan algoritma bactracking

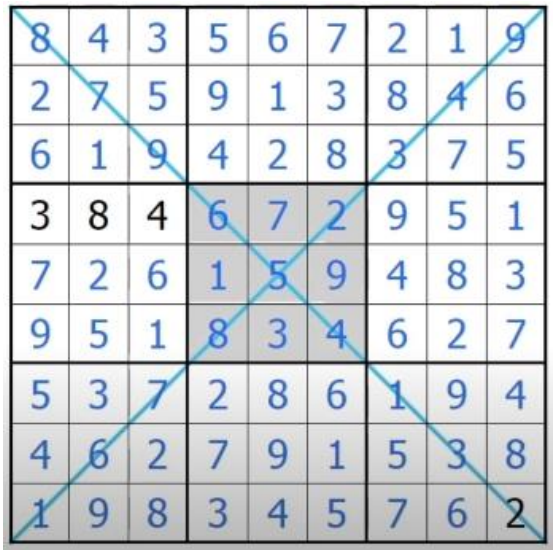
```
#A backtracking/recursive function to check all possible combinations
def solveGrid(grid):
  #Find next empty cell
  for i in range(0,81):
    row=i//9
    col=i%9
    if grid[row][col]==0:
      for value in range (1,10):
        #Check that this value has not already be used on this row
        if not(value in grid[row]):
          #Check that this value has not already be used on this column
          if not value in (grid[0][col],grid[1][col],grid[2][col],
          grid[3][col],grid[4][col],grid[5][col],grid[6][col],
          grid[7][col],grid[8][col]):
```

```
          #Identify which of the 9 squares we are working on
          square=[]
          if row<3:
            if col<3:
              square=[grid[i][0:3] for i in range(0,3)]
            elif col<6:
              square=[grid[i][3:6] for i in range(0,3)]
            else:
              square=[grid[i][6:9] for i in range(0,3)]
          elif row<6:
            if col<3:
              square=[grid[i][0:3] for i in range(3,6)]
            elif col<6:
              square=[grid[i][3:6] for i in range(3,6)]
            else:
              square=[grid[i][6:9] for i in range(3,6)]
          else:
            if col<3:
              square=[grid[i][0:3] for i in range(6,9)]
            elif col<6:
              square=[grid[i][3:6] for i in range(6,9)]
            else:
              square=[grid[i][6:9] for i in range(6,9)]
```

4. Kemudian dari sudah diisi tadi dicek apakah sudah full dalam blok nya

```
#A function to check if the grid is full
def checkGrid(grid):
  for row in range(0,9):
      for col in range(0,9):
        if grid[row][col]==0:
          return False

  #We have a complete grid!
  return True
```

5. Sudoku dengan 4 *given digits* sudah terisi semua



IV. KESIMPULAN

Sudoku dengan 4 *given* digits merupakan permasalahan yang menggabungkan permasalahan sudoku tradisional dengan permasalahana *magic square*. Algoritma Backtracking dapat dimanfaatkan dalam memecahkan gabungan kedua masalah ini, karena baik *magic square* maupun sudoku tradisional dapat dipecahkan melalui algoritma Backtracking.

VIDEO LINK AT YOUTUBE

https://youtu.be/XxksAPbmtOM

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur yang sebesar-besarnya kepada rahmat Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga makalah berjudul “Penerapan Algoritma Backtracking dalam Menyelesaikan Sudoku with 4 Given Digits” dapat diselesaikan.

Penulis juga mengucapkan terima kasih kepada dosen pengajar IF2211 Strategi Algoritma yang telah memberikan bimbingan dan ilmu terkait materi Strategi Algoritma ini.

Penulis juga berterima kasih kepada keluarga penulis, terutama kepada orang tua penulis, yang selalu mensupport penulis dalam menjalani perkuliahan ini dalam kondisi apapun.

DAFTAR PUSTAKA

[1] Cormen, T. H. dkk. 2009. Introduction to Algorithms (Edisi Ketiga). The MIT Press.

[2] Munir, Rinaldi. 2020. Slide Kuliah Algoritma Runut Balik IF2211 Strategi Algoritma. Bandung: Institut Teknologi Bandung. Diakses pada 2 Mei 2020, 10.00 WIB.

[3] Yang, Wilbur dan Mary Wootters. 2017. CS161 Lecture 13. Stanford. Diakses pada 2 Mei 2020, 10.00 WIB.

[4] Geeksforgeeks. Backtracking Introduction. https://www.geeksforgeeks.org/backtracking-introduction/. Diakses pada 2 Mei 2020, 10.00 WIB.

[5] Geeksforgeeks. Magic Square. https://www.geeksforgeeks.org/magic-square/. Diakses pada 2 Mei 2020, 10.00 WIB.

[6] Crackingthecryptic. https://cracking-the-cryptic.web.app/sudoku/2QM8JHJ4HB. Diakses pada 2 Mei 2020, 10.00 WIB.

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.

Bogor, 2 Mei 2020

Putri Nadia Salsabila - 13518094