Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

Implementasi Algoritma Branch and Bound Untuk Penyelesaian Permasalahan 15 Puzzle

Oleh:

13520121 – Nicholas Budiono

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2022

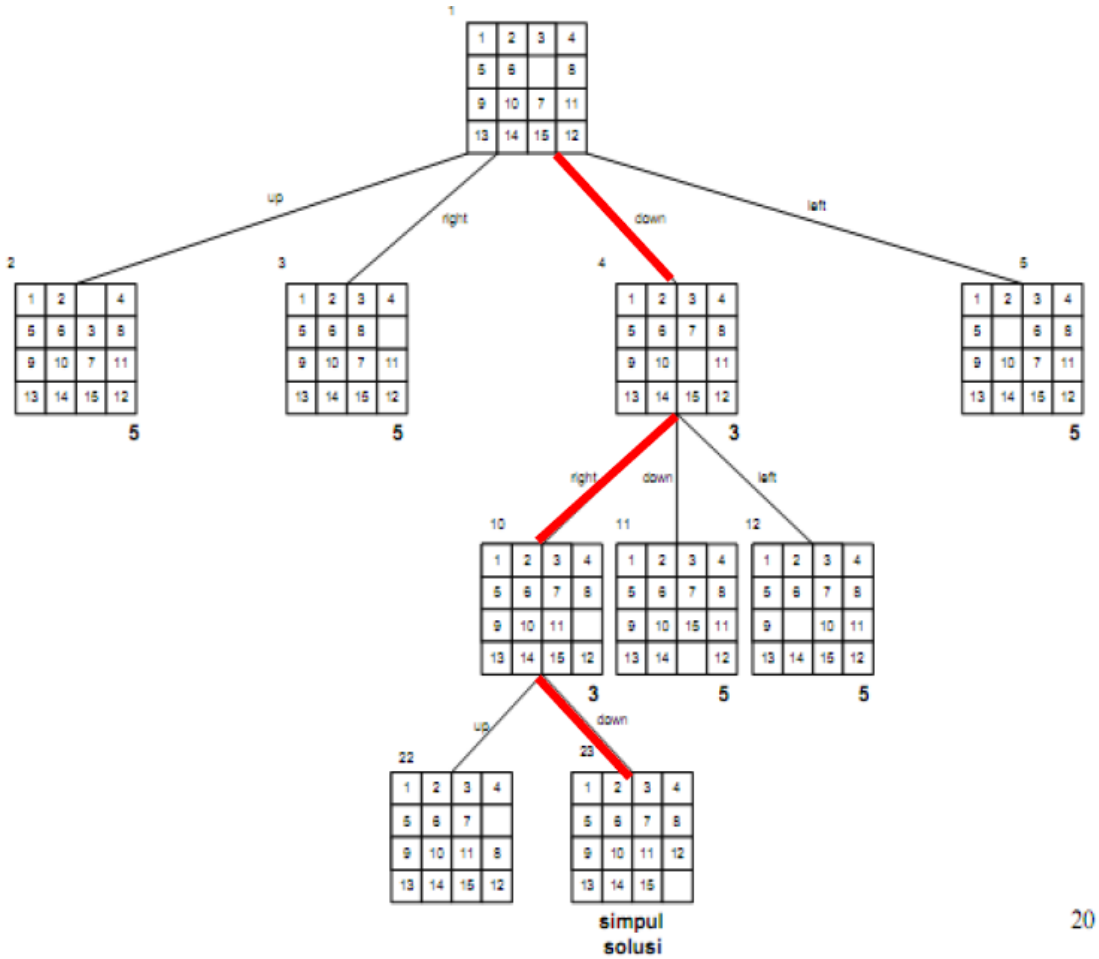
Penjelasan Branch and Bound dan 15 Puzzle:

Algoritma Branch and Bound:

Branch and Bound merupakan jenis algoritma yang bekerja dengan menggunakan BFS yang ditambah dengan *least cost search*. Dengan menggunakan nilai taksiran lintasan termurah ke simpul status tujuan yang memalui simpul status. Simpul berikutnya yang akan di-expand tidak berdasarkan pembangkitannya tetapi simpul yang memiliki cost yang paling kecil pada kasus minimasi. Biasanya, algoritma ini digunakan untuk persoalan optimasi.

15 Puzzle:

Puzzle satu ini bekerja dengan persoalan adalah terdapat 16 ubin yang masing-masing memiliki isinya, dengan pengecualian satu ubin yang kosong, hal ini dibuat agar ubin lain dapat berpindah. Dengan ketentuan demikian, kita diharuskan untuk membuat suatu pola dari angka pada ubin tersebut, contohnya adalah sebagai berikut.



Deskripsi Jalan Kode:

Pembacaan matriks masukan:

Masukan matriks pada kode ini terdapat dua cara, dengan cara masukan dalam bentuk file, dalam masukan secara manual, yaitu masukan satu-satu.

Pemrosessan data:

Matriks yang telah dimasukan diproses dengan melakukan Branch and Bound, pertama, dicari terlebih dahulu $\sum_{i=1}^{16} kurang(i) + X$, jika didapatkan ganjil, maka tidak bisa diselesaikan, jika genap, maka bisa diselesaikan. Setelah itu, jika genap, maka akan dilanjut dengan melakukan pergerakan dari 0 dan dimasukan ke dalam list. Juga dicari cost yang minimum. Matriks dengan cost yang minimum akan dicari lagi Gerakan yang bisa dari matriks tersebut (kecuali Gerakan yang diambil sebelumnya). Jika pada list tersebut terdapat matriks yang sama dengan matriks solusi, program akan keluar dari while-loop.

Pengeluaran Data Frame:

Dari proses diatas, akan dikeluarkan list dari matriks hidup, matriks-matriks yang dipakai sebagai Langkah penyelesaian, serta berapa banyak Langkah yang diperlukan untuk mencapai penyelesaian.

Deskripsi Kelas:

[1] __init__(Matriks, pos, P, C, W)
Inisialisasi kelas matriks dengan atribut pos sebagai matriks nya, P sebagai berapa Langkah dari awal, C sebagai cost, dan W sebagai Langkah awal yang dilakukan dari matriks tersebut.

[2] Turn_to_list(type, listIn)
Mengambil dari kelas matriks, dijadikan list dari data tersebut, type 0 sebagai pos, type 1 sebagai P, dan seterusnya.

[3] Print_matrix(matriks)
Print matriks dengan susunan yang rapi.

[4] GDown(matrix, last_move, p)
Melakukan branch dari satu matriks ke beberapa matriks yang memungkinkan untuk melakukan pergerakan.

[5] Print_kurangi_tabel(matrix)
Print i dan kurang(i) dari matriks yang ingin dilakukan pemecahan.

[6] Matrix_in(M, MPlist)
Mengecek apakah matriks M ada di dalam list matrix MPlist.

[7] Find0(Matrix)
Mencari lokasi dari ubin yang kosong dari matriks.

[8] Solvable(Matrix)
Mencari hasil dari $\sum_{i=1}^{16} kurang(i) + X$

[9] **Movable(Matrix, dir)**
Mengeluarkan Boolean jika 0 bisa dipindah dengan arah yang direpresentasikan dengan dir.

[10] **Move(Matrix, dir)**
Menukarkan posisi 0 dengan posisi ubin yang ada di sebalah dengan arah yang direpresentasikan dengan dir.

[11] **FindG(Matrix)**
Mencari aproksimasi g(i)

[12] **findC(Matrix, p)**
Mencari cost dari matriks

[13] **solve(matrix, solved_matrix)**
Melakukan penyelesaiian dengan menghasilkan semua simpul hidup, list dari matrks yang menjadi Langkah, serta berapa langkan yang dilakukan

Source Code:

Library.py:

```
import numpy as np


class MatrixPuzzle:

    # Bikin datatype baru
    def __init__(Matrix, pos, P, C, W):
        Matrix.pos = pos
        Matrix.P = P
        Matrix.C = C
        Matrix.W = W

    # mengubah list of datatype Matrix ke yg diinginkan
    def turn_to_list(type, listIn):
        listOut = []
        if(type == 0):
            for i in range(len(listIn)):
                listOut.append(listIn[i].pos)
        elif(type == 1):
            for i in range(len(listIn)):
                listOut.append(listIn[i].P)
        elif(type == 2):
            for i in range(len(listIn)):
                listOut.append(listIn[i].C)
```

```
        elif(type == 3):
            for i in range(len(listIn)):
                listOut.append(listIn[i].W)
        return listOut

    # Print matriks dengan susunan rapi
    def print_matrix(matrix):
        for i in range(4):
            for j in range(4):
                if(matrix[i][j] > 9):
                    print(f"| {matrix[i][j]} ", end="")
                else:
                    print(f"| {matrix[i][j]}  ", end="")
                if(j == 3):
                    print("|")

    # Melakukan branch dari satu matrix ke beberapa matrix yang
memungkinkan
    def GDown(matrix, last_move, p):
        LOut = []
        for i in range(4):
            if(MatrixPuzzle.movable(matrix, i) and i != last_move):
                temp = MatrixPuzzle.move(matrix, i)
                LOut.append(MatrixPuzzle(
                    temp, p + 1, MatrixPuzzle.findC(temp, p), i))
        return LOut

    # Print tabel dari i dan kurang(i)
    def print_kurangi_tabel(matrix):
        l = matrix.reshape(-1)
        tabel_kurangi = []
        for a in range(16):
            if(l[a] == 0):
                l[a] = 16
        for i in range(16):
            total = 0
            for j in range(i, 16):
                if(l[i] > l[j]):
                    total += 1
```

```
            tabel_kurangi.append([l[i], total])
        tabel_kurangi.sort()
        for b in range(16):
            if(l[b] == 16):
                l[b] = 0
        tabel_kurangi = [["i", "kurang(i)"]] + tabel_kurangi
        for k in range(17):
            for l in range(2):
                if(k > 9):
                    print(tabel_kurangi[k][l], end="   ")
                else:
                    print(tabel_kurangi[k][l], end="    ")
            print("")
        print("")

    # Mengecek apakah satu matrix ada di list matrix
    def matrix_in(M, MPlist):
        for i in range(len(MPlist)):
            if np.array_equal(M, np.array(MPlist[i].pos)):
                return True
        return False

    # Mencari lokasi dari 0 di matrix
    def find0(Matrix):
        for y in range(4):
            for x in range(4):
                if(Matrix[y][x] == 0):
                    return [x, y]

    # Mengeluarkan hasil dari kurang(i) + x
    def solvable(Matrix):
        total = 0
        if((MatrixPuzzle.find0(Matrix)[0] +
(MatrixPuzzle.find0(Matrix)[1]*4)) % 2 == 0):
            x = 1
        else:
            x = 0
        l = Matrix.reshape(-1)
        for a in range(16):
```

```
            if(l[a] == 0):
                l[a] = 16
        for i in range(16):
            for j in range(i, 16):
                if(l[i] > l[j]):
                    total += 1
        return total + x

    # Mengecek apakah gerakan tertentu itu valid atau tidak
    def movable(Matrix, dir):
        x = MatrixPuzzle.find0(Matrix)[0]
        y = MatrixPuzzle.find0(Matrix)[1]
        if (dir == 0):
            if(y - 1 > -1):
                return True
        elif(dir == 1):
            if(x + 1 < 4):
                return True
        elif (dir == 2):
            if(y + 1 < 4):
                return True
        elif(dir == 3):
            if(x - 1 > -1):
                return True
        return False

    # Melakukan pertukaran lokasi 0
    def move(Matrix, dir):
        Mout = np.copy(Matrix)
        x = MatrixPuzzle.find0(Matrix)[0]
        y = MatrixPuzzle.find0(Matrix)[1]
        if (dir == 0):
            Mout[y][x] = Matrix[y - 1][x]
            Mout[y - 1][x] = 0
        elif(dir == 1):
            Mout[y][x] = Matrix[y][x + 1]
            Mout[y][x + 1] = 0
        elif (dir == 2):
            Mout[y][x] = Matrix[y + 1][x]
```

```
            Mout[y + 1][x] = 0
        elif(dir == 3):
            Mout[y][x] = Matrix[y][x - 1]
            Mout[y][x - 1] = 0
        return Mout

    # Mencari g(i)
    def findG(Matrix):
        k = 1
        total = 0
        for i in range(4):
            for j in range(4):
                if(k > 15 and total == 0):
                    total = 0
                elif(Matrix[i][j] == k):
                    k += 1
                else:
                    total += 1
                    k += 1
        return total

    # Mencari c(i) dengan f(i) + g(i)
    def findC(Matrix, p):
        g = MatrixPuzzle.findG(Matrix)
        return g + p

    # Melakukan penyelesaiian, menghasilkan semua simpul hidup, list
dari matrix yang menjadi langkah, serta berapa langkah yang dilakukan
    def solve(matrix, solved_matrix):
        # inisialisasi
        list_of_matrix = []
        list_step_matrix = []
        p = 0
        last_move = -1

        # Melakukan penurunan dengan langkah-langkah yang ada
        list_of_matrix += MatrixPuzzle.GDown(matrix, last_move, p)
        p = 1
```

```
        last_move = MatrixPuzzle.turn_to_list(2,
list_of_matrix).index(
            min(MatrixPuzzle.turn_to_list(2, list_of_matrix)))

        # Diulang sampai ada solusi matrix yang ada di list_of_matrix
        while(not MatrixPuzzle.matrix_in(solved_matrix,
list_of_matrix)):
            temp_matrix = list_of_matrix[MatrixPuzzle.turn_to_list(2,
list_of_matrix).index(
                min(MatrixPuzzle.turn_to_list(2,
list_of_matrix)))].pos
            to_be_pop = MatrixPuzzle.turn_to_list(2,
list_of_matrix).index(
                min(MatrixPuzzle.turn_to_list(2, list_of_matrix)))

            list_step_matrix.append(list_of_matrix[to_be_pop])
            list_of_matrix += MatrixPuzzle.GDown(temp_matrix,
last_move, p)
            list_of_matrix.pop(to_be_pop)

            p += 1
            last_move = MatrixPuzzle.turn_to_list(2,
list_of_matrix).index(
                min(MatrixPuzzle.turn_to_list(2, list_of_matrix)))
        return list_of_matrix, list_step_matrix, p
```

Main.py:

```
import numpy as np
import library as lb
import time
from pathlib import Path
import os

# Baca input (via file/input langsung)
print("Baca dari file? (Y/N)")
choice = input()
if(choice == 'Y'):
    print("Masukkan nama file:")
    path_input = str(input())
    cur_path = os.path.dirname(__file__)
    new_path = os.path.relpath('TUCIL 3\\test\\' + path_input,
cur_path)
    with open(Path(new_path), 'r') as f:
        input_matrix = np.array(
            [[int(num) for num in line.split(' ')] for line in f])
else:
    input_matrix = np.zeros((4, 4))
    for y in range(4):
        for x in range(4):
            print("Masukkan Matrix[" + str(x + 1) + "][" + str(y + 1)
+ "] :")
            input_matrix[y][x] = int(input())

# Main program
solved_matrix = np.array(
    [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 0]])
start = time.time()
print()
lb.MatrixPuzzle.print_kurangi_tabel(input_matrix)
solvable = lb.MatrixPuzzle.solvable(np.array(input_matrix))
print(f'Hasil kurang(i) + X: {solvable}', end="\n\n")

# Mengecek apakah bisa di solve atau tidak
if((solvable) % 2 == 0):
```

```
    print("Matrix awal: ")
    lb.MatrixPuzzle.print_matrix(input_matrix)
    print("\nLangkah solusi: ")

    # Manggil fungsi untuk menyelesaikan,
    list_of_matrix, solved_list_matrix, p = lb.MatrixPuzzle.solve(
        input_matrix, solved_matrix)
    for i in range(len(solved_list_matrix)):
        lb.MatrixPuzzle.print_matrix((lb.MatrixPuzzle.turn_to_list(
            0, solved_list_matrix))[i])
    lb.MatrixPuzzle.print_matrix(solved_matrix)
    print(f"\nTotal simpul hidup: {len(list_of_matrix)}", end="\n\n")
    print(f'Waktu jalan: {time.time() - start} s')
else:
    print("Unsolvable!")
```

Screenshot:

Input:

Dari file:

```
Baca dari file? (Y/N)
Y
Masukkan nama file:
1.txt
```

Output:

```
i    kurang(i)
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
9    0
10   0
11   0
12   0
13   0
14   0
15   0
16   1

Hasil kurang(i) + X: 2

Matrix awal:
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 0  | 15 |

Langkah solusi:
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 0  |

Total simpul hidup: 3

Waktu jalan: 0.011002302169799805 s
```

Input:

Secara manual:

```
Baca dari file? (Y/N)
N
Masukkan Matrix[1][1] :
1
Masukkan Matrix[2][1] :
2
Masukkan Matrix[3][1] :
3
Masukkan Matrix[4][1] :
4
Masukkan Matrix[1][2] :
5
Masukkan Matrix[2][2] :
6
Masukkan Matrix[3][2] :
7
Masukkan Matrix[4][2] :
0
Masukkan Matrix[1][3] :
9
Masukkan Matrix[2][3] :
10
Masukkan Matrix[3][3] :
11
Masukkan Matrix[4][3] :
8
Masukkan Matrix[1][4] :
13
Masukkan Matrix[2][4] :
14
Masukkan Matrix[3][4] :
15
Masukkan Matrix[4][4] :
12
```

Output:

```
i    kurang(i)
1.0    0
2.0    0
3.0    0
4.0    0
5.0    0
6.0    0
7.0    0
8.0    0
9.0    1
10.0   1
11.0   1
12.0   0
13.0   1
14.0   1
15.0   1
16.0   8

Hasil kurang(i) + X: 14

Matrix awal:
| 1.0  | 2.0  | 3.0  | 4.0  |
| 5.0  | 6.0  | 7.0  | 0.0  |
| 9.0  | 10.0 | 11.0 | 8.0  |
| 13.0 | 14.0 | 15.0 | 12.0 |

Langkah solusi:
| 1.0  | 2.0  | 3.0  | 4.0  |
| 5.0  | 6.0  | 7.0  | 8.0  |
| 9.0  | 10.0 | 11.0 | 0.0  |
| 13.0 | 14.0 | 15.0 | 12.0 |

| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 0  |

Total simpul hidup: 5
```

Tabel:

Poin	Ya	Tidak
Program berhasil dikompilasi	✓	
Program berhasil *running*	✓	
Program dapat menerima input dan menuliskan output.	✓	
Luaran sudah benar untuk semua data uji	✓	
Bonus dibuat		✓

Source Code:

https://github.com/nicholass25/Stima_Tucil_3.git