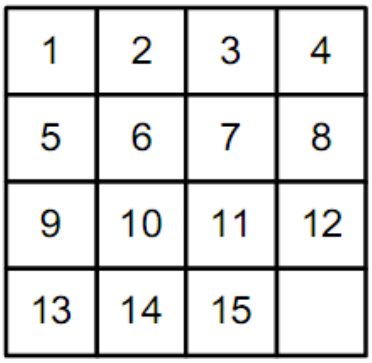
LAPORAN
TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA
Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*

Disusun oleh:
13518117 – Muhammad Firas

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2020

BAB I
DESKRIPSI TUGAS

Buatlah program dalam Python untuk menyelesaikan persoalan 15-Puzzle dengan menggunakan Algoritma *Branch and Bound* seperti pada materi kuliah. Nilai *bound* tiap simpul adalah penjumlahan cost yang diperlukan untuk sampai suatu simpul x dari akar, dengan taksiran *cost* simpul x untuk sampai ke *goal*. Taksiran cost yang digunakan adalah jumlah ubin tidak kosong yang tidak berada pada tempat sesuai susunan akhir *(goal state)*. Untuk semua instansiasi persoalan 15-puzzle, susunan akhir yang diinginkan sesuai dengan Gambar 1.



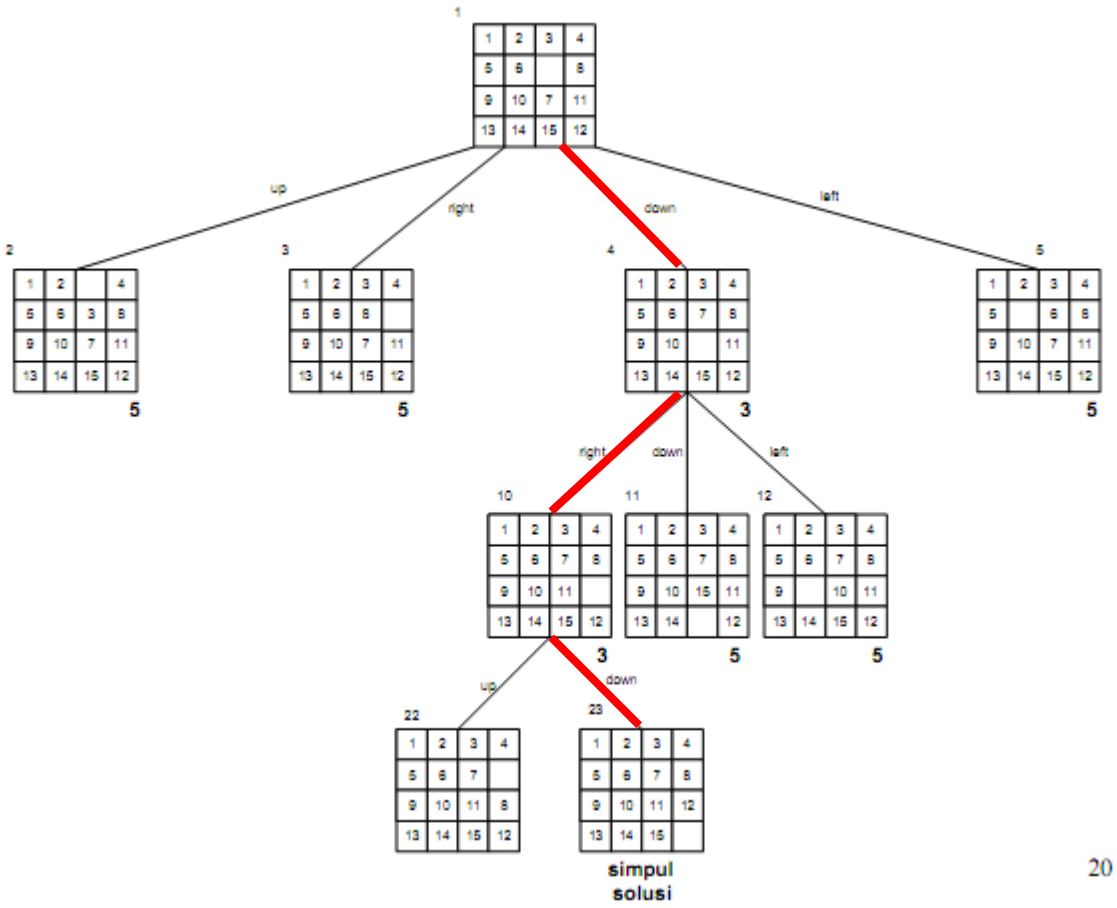
Gambar 1. Susunan Akhir persoalan 15-puzzle

Masukan: matriks yang merepresentasikan posisi awal suatu instansiasi persoalan 15-puzzle. Matriks dibaca dari berkas teks.

Program harus dapat menentukan apakah posisi awal suatu masukan dapat diselesaikan hingga mencapai susunan akhir, dengan mengimplementasikan fungsi Kurang(i) dan posisi ubin kosong di kondisi awal (X), seperti pada materi kuliah. Jika posisi awal tidak bisa mencapai susunan akhir, program akan menampilkan pesan tidak bisa diselesaikan,. Jika dapat diselesaikan, program dapat menampilkan urutan matriks rute (*path*) aksi yang dilakukan dari posisi awal ke susunan akhir. Sebagai contoh pada Gambar 2, matriks yang ditampilkan ke layar adalah matriks pada simpul 1, simpul 4, simpul 10 dan simpul 23.

Keluaran:

1. Matriks posisi awal.
2. Nilai dari fungsi Kurang(i) untuk setiap ubin tidak kosong pada posisi awal (nilai ini tetap dikeluarkan, baik persoalan bisa diselesaikan atau tidak bisa diselesaikan).
3. Nilai dari $\sum_{i=1}^{16} KURANG(i) + X$
4. Jika persoalan tidak dapat diselesaikan (berdasarkan hasil butir 2) keluarkan pesan.
5. Jika persoalan dapat diselesaikan (berdasarkan hasil butir 2), menampilkan urutan matriks seperti pada penjelasan sebelumnya.
6. Waktu eksekusi.
7. Jumlah simpul yang dibangkitkan dalam pohon ruang status.



Gambar 2. Contoh Pohon Ruang Status Persoalan 15-puzzle

Data Uji:
Buatlah 5 buah instansiasi persoalan 15-puzzle, dengan 2 kasus tidak dapat diselesaikan dan 3 kasus yang dapat diselesaikan. Instansiasi persoalan juga disertakan dalam pengumpulan Tucil 3.

BAB II
ANALISIS PERSOALAN

Untuk pemecahan masalah pada persoalan ini, digunakan algoritma *Branch and Bound*. Algoritma ini mirip dengan algoritma BFS tetapi pada simpul dalam queue diekspansi dengan cost yang paling kecil terlebih dahulu. Simpul ini akan berhenti diekspansi saat state simpul sama dengan goal.

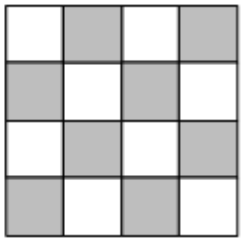
Untuk mengetahui apakah suatu state pada akar dapat mencapai goal atau tidak, digunakan rumus sebagai berikut:

$$\sum_{i=1}^{16} KURANG(i) + X$$

KURANG(i) = banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin

X = 1 jika ubin kosong berada pada sel yang diarsir, X = 0 jika ubin kosong berada pada sel yang tidak diarsir.

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

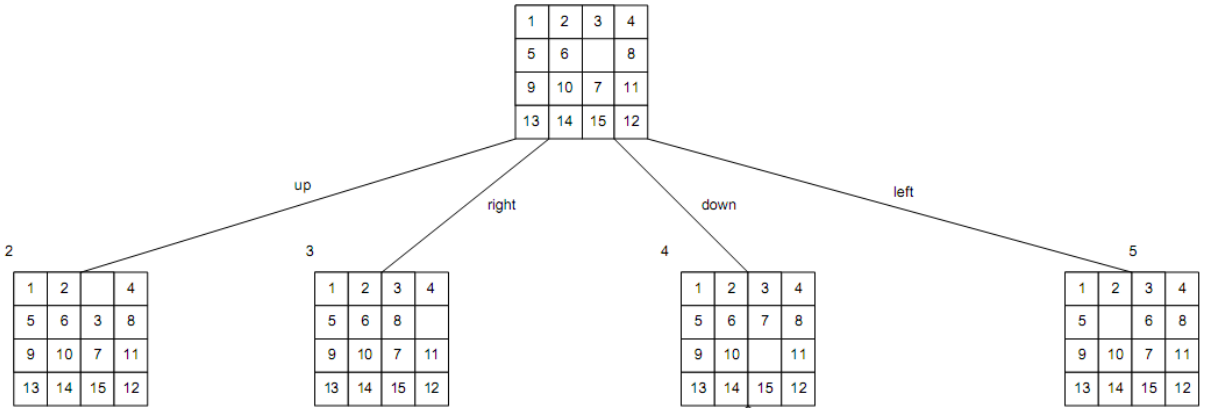


Maka state goal hanya bisa dicapai dari status awal jika $\sum_{i=1}^{16} KURANG(i) + X$ bernilai genap.

Untuk menghitung cost pada setiap ekspansi simpul, digunakan

$$\hat{c}(i) = \hat{f}(i) + \hat{g}(i)$$

$\hat{c}(i)$ = ongkos (cost) untuk simpul i
$\hat{f}(i)$ = panjang lintasan dari simpul akar ke P
$\hat{g}(i)$ = jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir



C(2) = 1+4 = 5 C(2) = 1+4 = 5 C(2) = 1+4 = 5 C(2) = 1+4 = 5

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

Goal

Simpul yang memiliki cost terkecil akan diekspansi terlebih dahulu dan seterusnya sampai state awal mencapai state tujuan.

BAB III
IMPLEMENTASI

Instansiasi class Puzzle15

```
from copy import deepcopy
import numpy as np

class Puzzle15:

    rows = 4
    columns = 4
    arrOfKurang = []
    nodeHeight = 0
    cost = 0
    arrOfRute = []

    def __init__(self, arrPuzzle):
        self.arrPuzzle = arrPuzzle
        for i in range(self.rows):
            for j in range(self.columns):
                if(self.arrPuzzle[i][j] == 16):
                    self.emptyCell = i*10 + j

    def __eq__(self, other):
        return (self.arrPuzzle == other.arrPuzzle)

    def addRute(self, rute):
        self.arrOfRute.append(rute)
...
```

Method untuk mencetak matriks

```
def printPuzzle(self):
    for i in range (self.rows):
        for j in range (self.columns):
            if (self.arrPuzzle[i][j] != 16):
                if(self.arrPuzzle[i][j] < 10):
                    print(" " + str(self.arrPuzzle[i][j]), end=" ")
                else:
                    print(self.arrPuzzle[i][j], end=" ")
            else:
                print("  ", end=" ")
        print()
    print()
```

Method untuk menggerakan ubin kosong

```
def moveUp(self):
    if(self.getEmptyRow() != 0): # Jika ubin kosong tidak di baris pertama
        temp = # Empty Cell
        # Empty Cell = # Ubin di atas Empty Cell
        # Ubin di atas Empty Cell = temp
        self.emptyCell -= 10 # Posisi current Empty Cell

def moveDown(self):
    if(self.getEmptyRow() != 3): #Jika ubin kosong tidak di baris terakhir
        temp = # Empty Cell
        # Empty Cell = # Ubin di kanan Empty Cell
        # Ubin di kanan Empty Cell = temp
        self.emptyCell += 10 # Posisi current Empty Cell

def moveLeft(self):
    if(self.getEmptyColumn() != 0): # Jika ubin kosong tidak di kolom pertama
        temp = # Empty Cell
        # Empty Cell = # Ubin di kiri Empty Cell
        # Ubin di kiri Empty Cell = temp
        self.emptyCell -= 1 # Posisi current Empty Cell

def moveRight(self):
    if(self.getEmptyColumn() != 3): # Jika ubin kosong tidak di kolom terakhir
        temp = # Empty Cell
        # Empty Cell = # Ubin di kanan Empty Cell
        # Ubin di kanan Empty Cell = temp
        self.emptyCell += 1 # Posisi current Empty Cell
```

Method menghitung Kurang(i)

```
def kurang(self):
    temp = np.array(self.arrPuzzle)
    temp = temp.flatten()
    for i in range(0, 16):
        count = 0
        for j in range(i+1, 16):
            if(temp[i] > temp[j]):
                count += 1
        self.arrOfKurang.insert(i, count)

def sumOfKurang(self):
    sum = 0
    for i in range(16):
        sum += self.arrOfKurang[i]
    return sum
```

Method Kurang(i) + X

```
def KurangX(self):
    X = (self.getEmptyRow() + self.getEmptyColumn()) % 2
    sumX = self.sumOfKurang() + X
    return sumX
```

Fungsi untuk menghitung cost

```
def cost(P, Goal):
    count = 0
    for i in range(4):
        for j in range(4):
            if (P.arrPuzzle[i][j] != Goal.arrPuzzle[i][j]):
                count += 1
    return (P.nodeHeight + count)
```

Fungsi lainnya

```
def insertPrio(Q, P): # Ekspansi simpul ke dalam Queue sesuai cost
    for i in range(len(Q)):
        if(Q[i].getCost() > P.getCost()):
            Q.insert(i, P)
            return
    Q.append(P)

def isEqual(P, Q): # Mengecek apakah simpul P telah dibangkitkan pada Queue
    for i in range(len(Q)):
        if (P == Q[i]):
            return True
    return False
```

Prosedur membaca inputan dari file.txt serta instansiasi class

```
fileFound = False
while not(fileFound):
    try:
        file = input("Masukkan nama file: ")
        ins = open( file, "r" )
        data = []
        for line in ins:
            number_strings = line.split()
            numbers = [int(n) for n in number_strings]
            data.append(numbers)
        ins.close()
        fileFound = True
    except FileNotFoundError:
        print("File tidak ditemukan!")
        print()

Puzzle = Puzzle15(data)
```

Algoritma *Branch and Bound*

```
startTime = time.time() # Waktu mulai
NULL = 16
Goal = Puzzle15([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,NULL]])

allQueue = []
Queue = []
i = 1

Queue.append(Puzzle)
allQueue.append(Puzzle)

while(Queue[0] != Goal):
    current = Queue.pop(0)

    up = deepcopy(current)
    up.arrOfRute = deepcopy(current.arrOfRute)
    up.moveUp()
    if(not(isEqual(up, allQueue))):
        up.setHeight(i)
        up.setCost(cost(up, Goal))
        up.addRute('up')
        insertPrio(Queue, up)
        insertPrio(allQueue, up)

    # Langkah ekspansi simpul down
    # Langkah ekspansi simpul left
    # Langkah ekspansi simpul right

# Queue[0] == Goal
endTime = time.time() # Waktu akhir
executionTime = (endTime-startTime)*1000000.00 # Waktu eksekusi dalam
mikrosekon
```

BAB IV
UJI KASUS

Telah dibuat 5 file untuk uji kasus dengan 2 kasus tidak dapat diselesaikan dan 3 sisanya bisa diselesaikan

File testcase1.txt

```
1 2 3 4
5 6 7 8
9 16 10 11
13 14 15 12
```

File testcase2.txt

```
1 2 3 4
5 6 16 12
9 10 8 7
13 14 11 15
```

File testcase3.txt

```
1 2 3 16
5 6 7 15
9 10 8 4
13 14 12 11
```

File testcase4.txt

```
1 3 4 15
2 16 5 12
7 6 11 14
8 9 10 13
```

File testcase5.txt

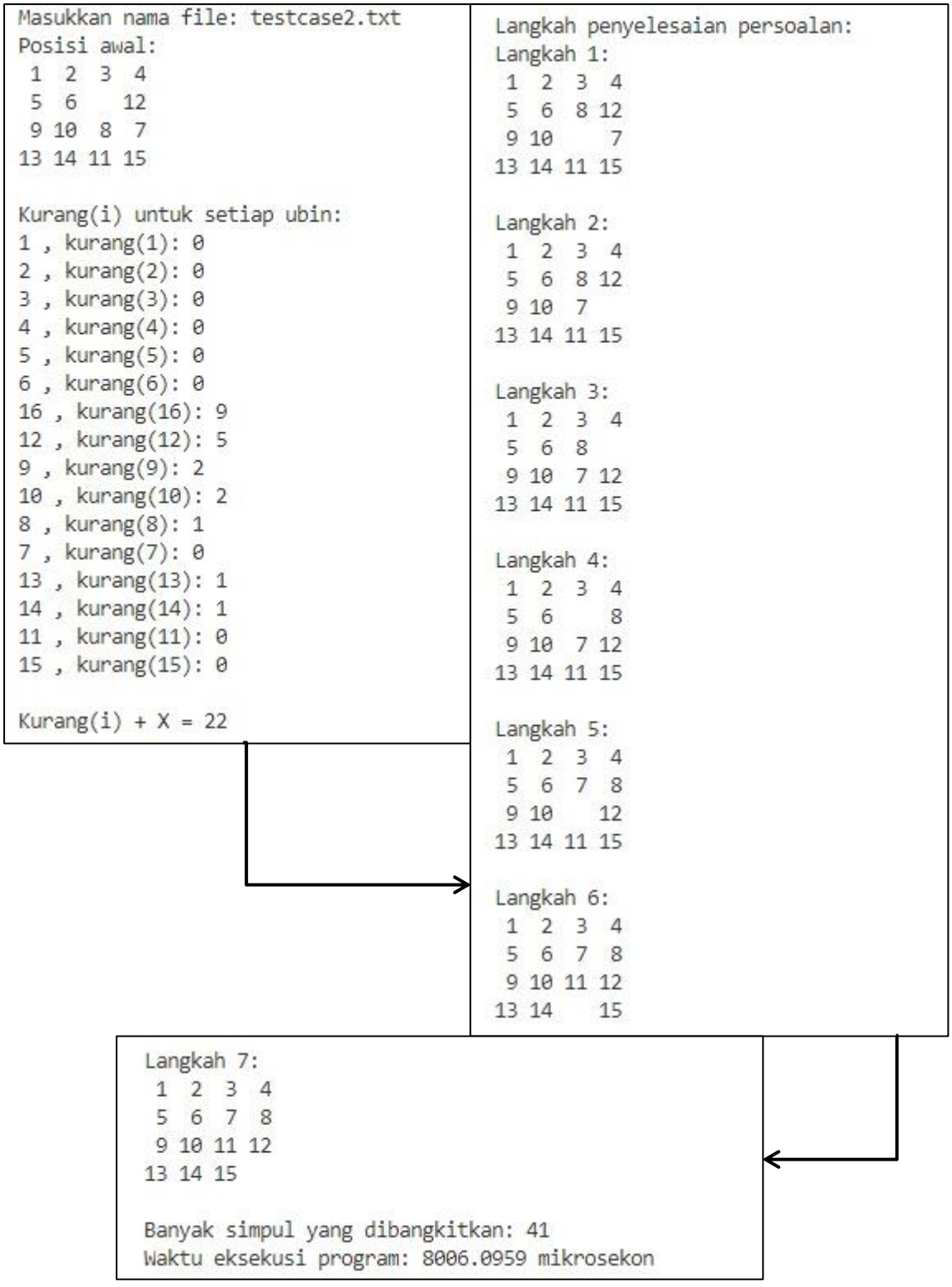
```
4 16 12 11
2 3 10 14
5 1 13 6
8 15 9 7
```

- *Screen-shot* input-output program untuk testcase1.txt

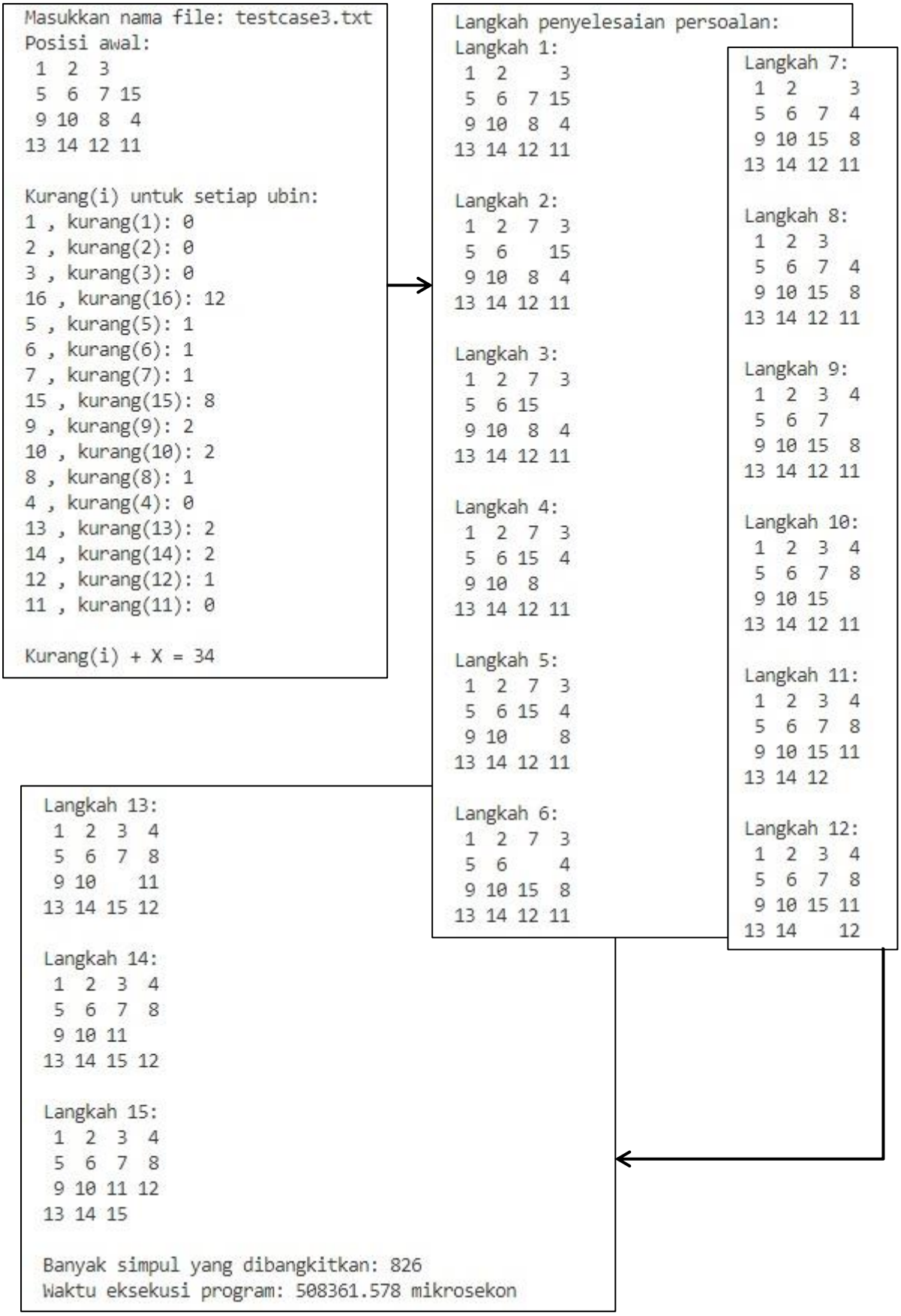
```
Masukkan nama file: testcase1.txt
Posisi awal:
 1  2  3  4
 5  6  7  8
 9    10 11
13 14 15 12

Kurang(i) untuk setiap ubin:
1 , kurang(1): 0
2 , kurang(2): 0
3 , kurang(3): 0
4 , kurang(4): 0
5 , kurang(5): 0
6 , kurang(6): 0
7 , kurang(7): 0
8 , kurang(8): 0
9 , kurang(9): 0
16 , kurang(16): 6
10 , kurang(10): 0
11 , kurang(11): 0
13 , kurang(13): 1
14 , kurang(14): 1
15 , kurang(15): 1
12 , kurang(12): 0

Kurang(i) + X = 10
```

```
Langkah penyelesaian persoalan:
Langkah 1:
 1  2  3  4
 5  6  7  8
 9 10    11
13 14 15 12

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

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

Banyak simpul yang dibangkitkan: 10
Waktu eksekusi program: 1998.9014 mikrosekon
```

- *Screen-shot* input-output program untuk testcase2.txt



- *Screen-shot* input-output program untuk testcase3.txt



- *Screen-shot* input-output program untuk testcase4.txt

```
Masukkan nama file: testcase4.txt
Posisi awal:
 1  3  4 15
 2     5 12
 7  6 11 14
 8  9 10 13

Kurang(i) untuk setiap ubin:
1 , kurang(1): 0
3 , kurang(3): 1
4 , kurang(4): 1
15 , kurang(15): 11
2 , kurang(2): 0
16 , kurang(16): 10
5 , kurang(5): 0
12 , kurang(12): 6
7 , kurang(7): 1
6 , kurang(6): 0
11 , kurang(11): 3
14 , kurang(14): 4
8 , kurang(8): 0
9 , kurang(9): 0
10 , kurang(10): 0
13 , kurang(13): 0

Kurang(i) + X = 37

Karena Kurang(i) + X adalah ganjil, maka persoalan tidak dapat diselesaikan
```

- *Screen-shot* input-output program untuk testcase5.txt

```
Masukkan nama file: testcase5.txt
Posisi awal:
 4    12 11
 2  3 10 14
 5  1 13  6
 8 15  9  7

Kurang(i) untuk setiap ubin:
4 , kurang(4): 3
16 , kurang(16): 14
12 , kurang(12): 10
11 , kurang(11): 9
2 , kurang(2): 1
3 , kurang(3): 1
10 , kurang(10): 6
14 , kurang(14): 7
5 , kurang(5): 1
1 , kurang(1): 0
13 , kurang(13): 4
6 , kurang(6): 0
8 , kurang(8): 1
15 , kurang(15): 2
9 , kurang(9): 1
7 , kurang(7): 0

Kurang(i) + X = 61

Karena Kurang(i) + X adalah ganjil, maka persoalan tidak dapat diselesaikan
```

Spesifikasi komputer/laptop yang digunakan untuk mengeksekusi program ini adalah sebagai berikut : Intel Core i7-5500U 2.4GHz processor, Nvidia Geforce 940M 2GB graphics card, 8GB DDR3 memory (RAM), 1 TB HDD, 64-bit Operating System, x64 based processor.

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil *running*	✓	
3. Program dapat menerima input dan menuliskan output	✓	
4. Luaran sudah benar untuk semua *n*	✓	

DAFTAR PUSTAKA

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