Laporan Tugas Kecil 3

Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*

IF2211 Strategi Algoritma

Oleh:

Nama:Fawwaz Anugrah Wiradhika Dharmasatya

NIM:13520086

Kelas:K02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

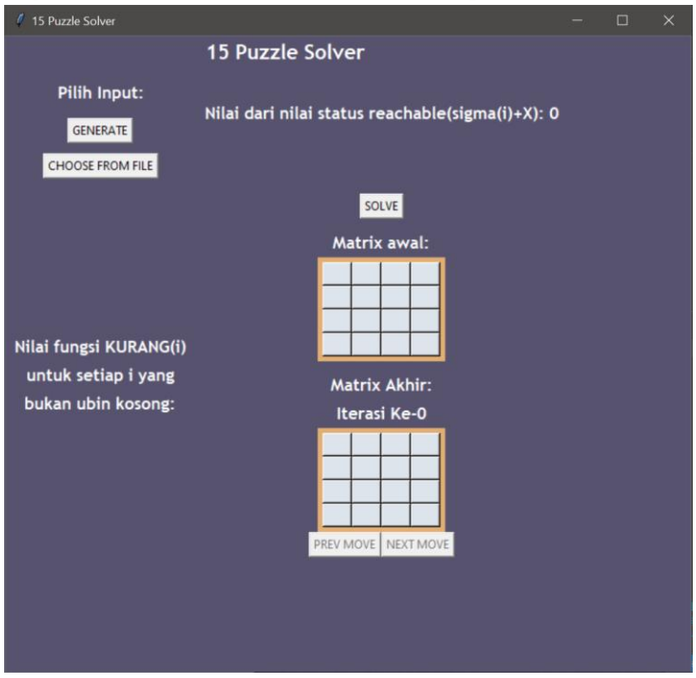
INSTITUT TEKNOLOGI BANDUNG

2022

I. Algoritma dan Cara Kerja

Implementasi penyelesaian 15 Puzzle dengan algoritma Branch and Bound ini diimplementasikan dalam bahasa Python. Program dimulai dengan menjalankan file main.py.

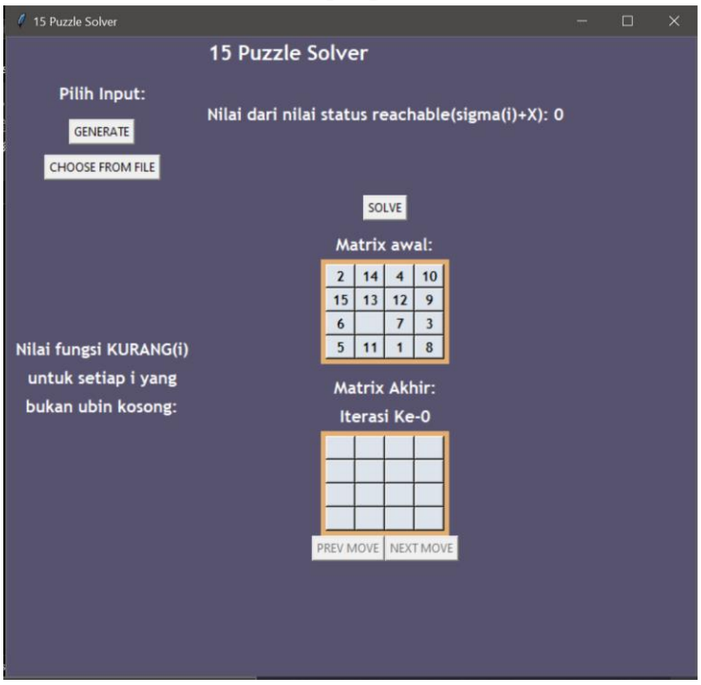
Setelah file tersebut dijalankan maka akan muncul GUI seperti berikut:



Gambar 1.1. Tampilan Awal Program

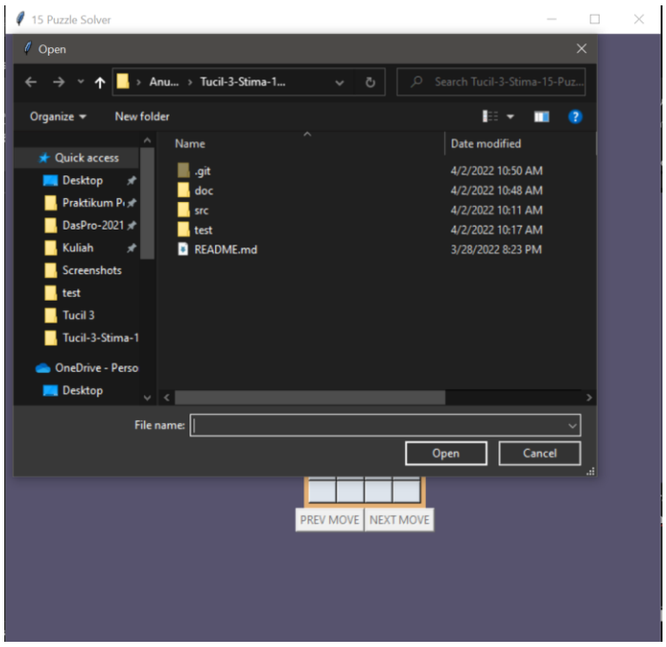
Cara Penggunaan:

1. Pilih jenis input matriks. Sesuai spek, matriks bisa dihasilkan secara acak oleh program atau dari file input.
    a) Jika ingin dihasilkan secara acak oleh program, tekan tombol GENERATE.



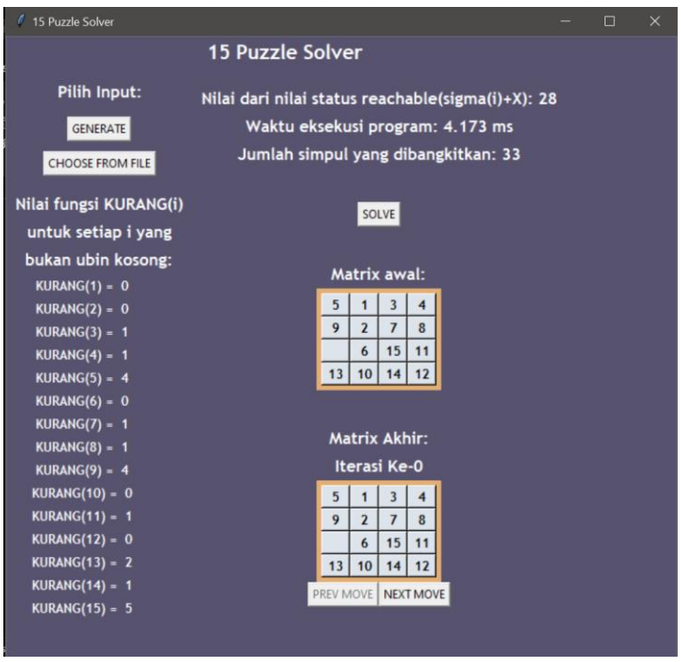
Gambar 1.2. Matriks Hasil Produksi Program

    b) Jika ingin memasukkan matriks dari file eksternal, tekan tombol CHOOSE FROM FILE. Tombol tersebut akan membuka *file dialog*. Pilih file yang ingin dibaca lalu tekan Open.

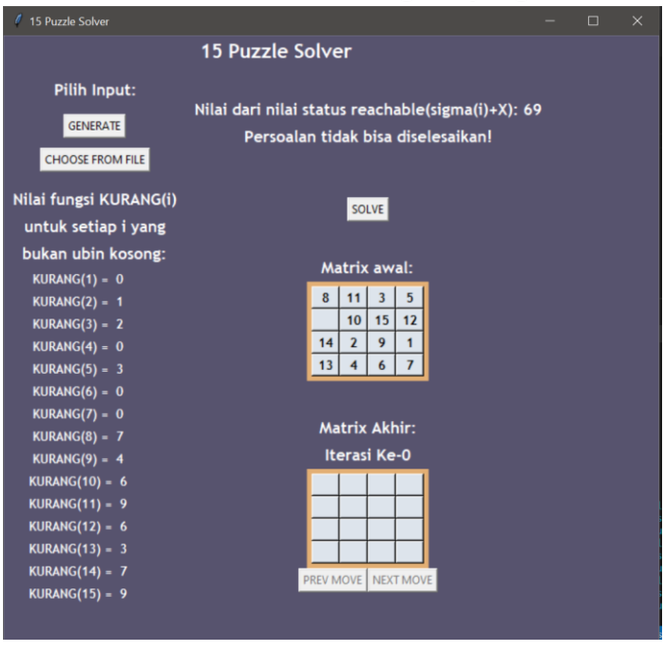


Gambar 1.3. *File Dialog* Untuk Memilih File Input

2. Tekan tombol SOLVE untuk menjalankan program. Tunggu hingga selesai.



Gambar 1.4. Contoh Hasil Eksekusi Program yang Bisa Diselesaikan



Gambar 1.5. Contoh Hasil Eksekusi Program yang Tidak Bisa Diselesaikan

3. Pada hasil akhir, bisa dilihat nilai fungsi KURANG(i) untuk setiap i, total nilai status *reachable*, matriks akhir dan status akhir. Jika matriks tidak bisa diselesaikan, maka akan menampilkan tulisan "Persoalan tidak bisa diselesaikan!", sedangkan bila matriks bisa diselesaikan maka akan menampilkan total waktu penyelesaian beserta jumlah simpul yang dibangkitkan.
4. Di matriks akhir, terdapat tombol PREV MOVE dan NEXT MOVE untuk melihat penyelesaian matriks langkah-per-langkah. Tombol PREV MOVE digunakan untuk melihat langkah sebelumnya, sedangkan tombol NEXT MOVE digunakan untuk melihat langkah setelahnya.

Cara Kerja Algoritma *Branch and Bound* untuk menyelesaikan 15 Puzzle:

1. Memeriksa apakah matriks bisa diselesaikan. Matriks direpresentasikan sebagai array of integer dengan 16 elemen.Matriks bisa diselesaikan apabila persamaan:

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

bernilai genap.

Fungsi KURANG(i) merupakan fungsi yang menghitung banyaknya ubin bernomor j sedemikian sehingga j<i namun POSISIS(j)>POSISI(i).

POSISI(i) adalah indeks elemen array bernilai i di posisi awal puzzle matriks yang ingin diselesaikan.

Untuk nilai X, diasumsikan indeks array dimulai dari 0, maka:

$$\text{X} = \begin{cases} 1 & ,\text{indeks} = 1,3,4,6,9,11,12,14 \\ 0 & ,\text{otherwise} \end{cases}$$

2. Jika matriks tidak bisa diselesaikan, maka akan menampilkan tulisan di layar "Persoalan tidak bisa diselesaikan!". Jika bisa diselesaikan, maka algoritma akan membangkitkan simpul akar yakni posisi awal matriks. Simpul ini, simpul-simpul cabangnya, serta cabang dari simpul-simpul cabangnya berisi 6 buah data, yakni:
   - Indeks simpul
   - Indeks parent simpul(bernilai -1 untuk simpul akar)
   - Posisi elemen matriks pada simpul tersebut
   - Besar *cost* dari simpul tersebut. Cost dihitung berdasarkan persamaan berikut:

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

$Keterangan$:
$\hat{c}(i)$ = cost dari simpul
$\hat{f}(\text{i})$ = kedalaman simpul dari simpul akar
$\hat{g}(\text{i})$ = jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir.

   - Nilai hash dari matriks. Nilai hash matriks diperoleh dengan mencari nilai hash dari string dengan pola "0<sel-0>0<sel-1>…0<sel-15>", dengan <sel-i> merupakan nilai sel pada indeks-i di matriks.
   - Kedalaman simpul dari simpul akar. Simpul akar memiliki kedalaman 0.

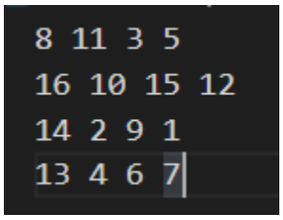
   Jika simpul akar ini merupakan solusi, maka solusi telah ditemukan. Jika tidak, masukkan simpul akar ke list berisi daftar simpul.
3. Jika simpul akar bukan solusi, bangkitkan simpul-simpul yang mungkin dari simpul akar, yakni (sesuai urutan) menggeser sel kosong ke arah: atas, kanan, bawah, kiri. Setiap pembangkitan simpul akan mengingkremen variabel jumlah_simpul. Jika pada salah satu pembangkitan simpul ditemukan solusi, maka pencarian selesai. Jika tidak ditemukan, masukkan semua simpul cabang ke list yang berisi daftar seluruh simpul serta list yang berisi simpul aktif.
4. Urutkan simpul aktif terurut membesar berdasarkan cost.
5. Ambil simpul di simpul aktif dengan bobot terkecil. Jika simpul ini merupakan simpul solusi pertama atau nilai cost nya lebih kecil dari nilai cost solusi sebelumnya, simpul

ini menjadi solusi dan simpul aktif lain yang cost nya lebih besar dari simpul ini akan “dimatikan”(dihapus dari list simpul aktif).

6. Jika simpul tersebut bukan merupakan simpul solusi, maka bangkitkan simpul-simpul yang mungkin dari simpul tersebut, yakni (sesuai urutan) menggeser sel kosong ke arah: atas, kanan, bawah, kiri. Setiap pembangkitan simpul akan mengingkremen variabel jumlah_simpul. Jika pada salah satu pembangkitan simpul ditemukan solusi, maka pencarian selesai. Untuk semua simpul cabang, jika simpul sudah pernah dibangkitkan sebelumnya, maka abaikan. Jika belum maka masukkan simpul cabang ke list yang berisi daftar seluruh simpul serta list yang berisi simpul aktif.
7. Kembali ke Langkah 4 bila solusi belum ditemukan.

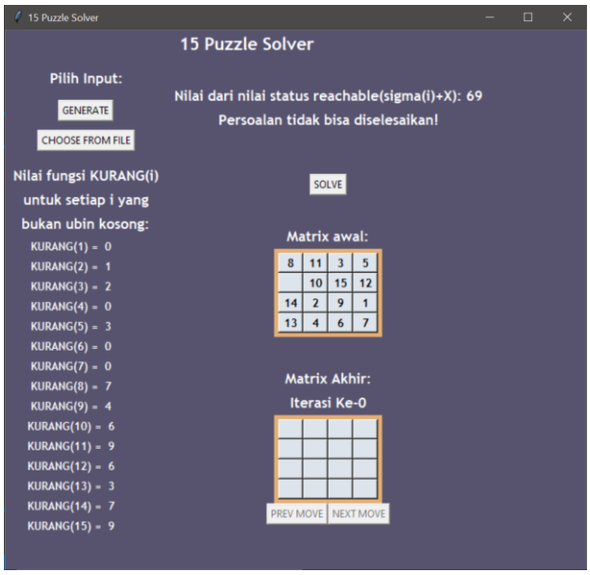
II. *Screenshot Input-Output*

1. TC1 (Unsolvable)
   - nama file: test-case-1(unsolvable).txt
   - Isi matriks:



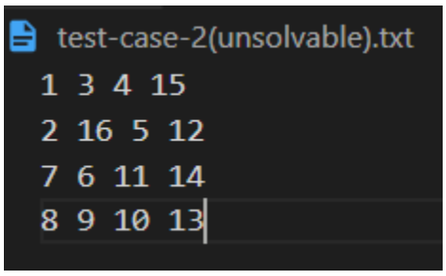
Gambar 2.1. Input TC 1

   - Hasil Eksekusi:



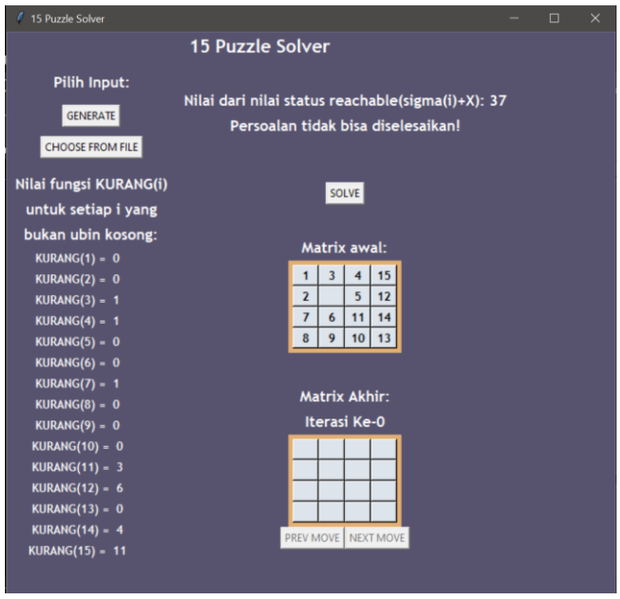
Gambar 2.2. Hasil Eksekusi TC 1

2. TC2 (Unsolvable)
   - nama file: test-case-2(unsolvable).txt
   - Isi matriks:



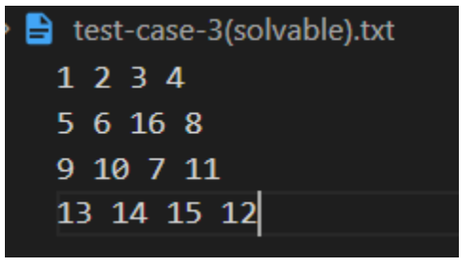
Gambar 2.3. Input TC 2

   - Hasil Eksekusi:



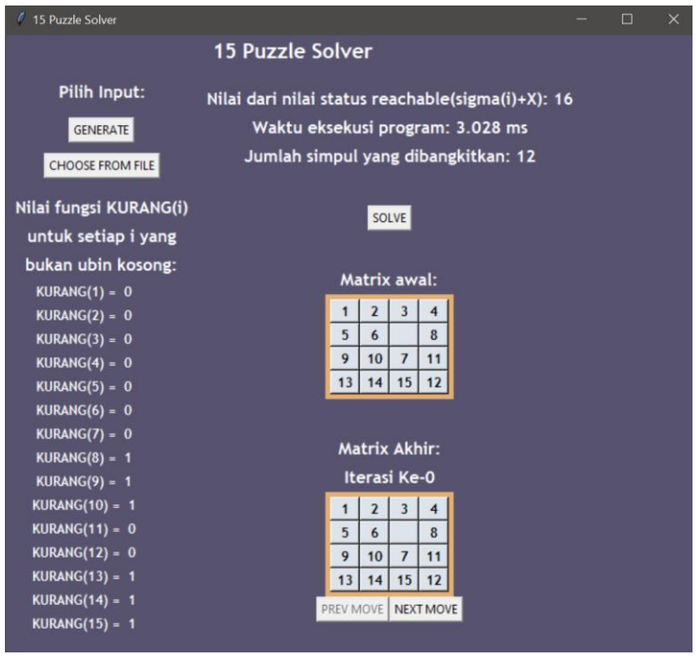
Gambar 2.4. Hasil Eksekusi TC 2

3. TC3 (Solvable)
    - nama file: test-case-3(solvable).txt
    - Isi matriks:



Gambar 2.5. Input TC 3

- Hasil Eksekusi:



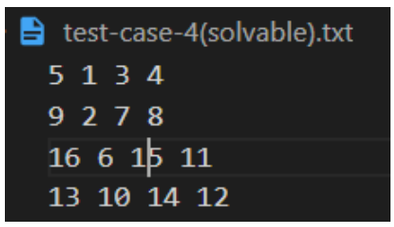
Gambar 2.6. Hasil Eksekusi TC 3

- Langkah Penyelesaian:

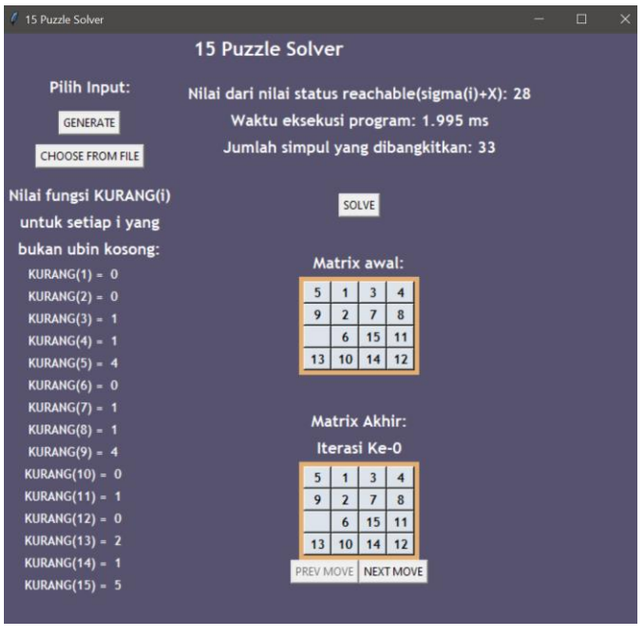
  Tabel 2.1. Langkah Penyelesaian TC 3

Langkah ke-	Matriks
0 (matriks awal)	
1	
2	
3 (selesai)	

4. TC4 (Solvable)
   - nama file: test-case-4(solvable).txt
   - Isi matriks:

     Gambar 2.7. Input TC 4
   - Hasil Eksekusi:



Gambar 2.8. Hasil Eksekusi TC 4

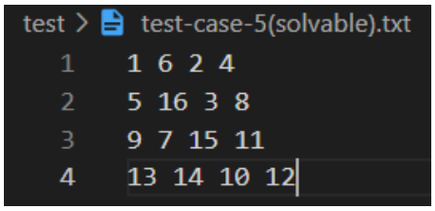
- Langkah Penyelesaian:

Tabel 2.2. Langkah Penyelesaian TC 4

Langkah ke-	Matriks
0 (matriks awal)	
1	
2	
3	
4	

5	
6	
7	
8	
9	
10 (selesai)	

5. TC5 (Solvable)
   - nama file: test-case-5(solvable).txt
   - Isi matriks:



Gambar 2.9. Input TC 5

   - Hasil Eksekusi:

Gambar 2.10. Hasil Eksekusi TC 5

- Langkah Penyelesaian:

Tabel 2.3. Langkah Penyelesaian TC 5

Langkah ke-	Matriks
0 (matriks awal)	
1	
2	
3	

4	 Matrix Akhir: Iterasi Ke-4 1 6 2 4 9 5 3 8 13 7 15 11 14 10 12 PREV MOVE NEXT MOVE
5	 Matrix Akhir: Iterasi Ke-5 1 6 2 4 9 5 3 8 13 7 15 11 14 10 12 PREV MOVE NEXT MOVE
6	 Matrix Akhir: Iterasi Ke-6 1 6 2 4 9 5 3 8 13 7 11 14 10 15 12 PREV MOVE NEXT MOVE
7	 Matrix Akhir: Iterasi Ke-7 1 6 2 4 9 5 3 8 13 7 11 14 10 15 12 PREV MOVE NEXT MOVE
8	 Matrix Akhir: Iterasi Ke-8 1 6 2 4 9 5 3 8 13 10 7 11 14 15 12 PREV MOVE NEXT MOVE
9	 Matrix Akhir: Iterasi Ke-9 1 6 2 4 9 5 3 8 13 10 7 11 14 15 12 PREV MOVE NEXT MOVE

10	
11	
12	
13	
14	
15	
16	

17		
18 (selesai)		

III. Checklist Penyelesaian

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 data uji	✓	
5.Bonus dibuat	✓	

IV. Kode Program

Tabel 4.1. *Source Code* file main.py (Program utama yang memanggil dan menjalankan GUI)

```
from GUI import GUI
#Program Utama 15 Puzzle Solver Berbasis GUI
if __name__ == '__main__':
    gui = GUI()
    gui.mainloop()
```

Tabel 4.2. *Source Code* file GUI.py(Bagian yang mengurusi input-ouput dan GUI)

```
import tkinter as tk
from tkinter import filedialog
from TSP15Puzzle import TSP15Puzzle
import random
import os
import time
import threading
#sumber template
GUI:https://gist.github.com/RamonWill/0422b061464097a7a0162f33e4c13a2e
class GUI(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.title("15 Puzzle Solver")
```

```
        self.geometry("700x700")
        self.configure(background="#57536E")
        self.matriks = None
        self.solver = None
        self.filabel = []#daftar label fungsi kurang(i)
        self.solution = None
        self.iteration = 0
        self.time_label = None
        self.node_label = None

        title_styles = {"font": ("Trebuchet MS Bold",
16),"foreground":"white","background":"#57536E"}
        input_text_styles =  {"font": ("Trebuchet MS Bold",
13),"foreground":"white","background":"#57536E"}

        kurang_text_styles =  {"font": ("Trebuchet MS Bold",
13),"foreground":"white","background":"#57536E"}
        self.kurang_normal_text_styles = {"font": ("Trebuchet MS Bold",
10),"foreground":"white","background":"#57536E"}

        self.start_matrix_text_styles = {"font": ("Trebuchet MS Bold",
13),"foreground":"white","background":"#57536E"}
        matrix_cell_text_styles = {"font": ("Trebuchet MS Bold",
10),"foreground":"black","background":"#dde4ec"}

        sigma_text_styles = {"font": ("Trebuchet MS Bold",
13),"foreground":"white","background":"#57536E"}
        #judul
        title_text = tk.Label(self,title_styles,text="15 Puzzle
Solver",justify="center")
        title_text.grid(row=0,column=0,columnspan=3)

        #frame input
        input_frame =
tk.Frame(self,bg="#57536E",height=600,width=400,borderwidth=5)
        input_frame.grid(row=1,column=0)

        input_label = tk.Label(input_frame,input_text_styles,text="Pilih
Input:",justify="center",width=18)
        input_label.grid(row=0,column=0,pady=6)

        generate_button =
tk.Button(input_frame,text="GENERATE",command=lambda:self.generate())
        generate_button.grid(row=1,column=0,pady=5)

        choose_from_button = tk.Button(input_frame,text="CHOOSE FROM
FILE",command=lambda:self.getMatrixFromFile())
        choose_from_button.grid(row=2,column=0,pady=5)

        #sigma frame
        self.sigma_frame =
tk.Frame(self,bg="#57536E",height=600,width=300,borderwidth=5)
        self.sigma_frame.grid(row=1,column=1)

        self.sigma_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="Nilai dari
nilai status reachable(sigma(i)+X): 0",justify="left")
        self.sigma_label.grid(row=0,column=0)
```

```
        self.warning_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="",justify="le
ft")
        self.warning_label.grid(row=1,column=0)
        #frame nilai fungsi kurang(i)
        self.kurang_frame =
tk.Frame(self,bg="#57536E",height=600,width=300,borderwidth=5)
        self.kurang_frame.grid(row = 2,column=0,rowspan=3)

        kurang_label_1 =
tk.Label(self.kurang_frame,kurang_text_styles,text="Nilai fungsi
KURANG(i)",justify="center")
        kurang_label_1.grid(row=0,column=0,columnspan=2)

        kurang_label_2 =
tk.Label(self.kurang_frame,kurang_text_styles,text="untuk setiap i
yang",justify="center")
        kurang_label_2.grid(row=1,column=0,columnspan=2)

        kurang_label_3 =
tk.Label(self.kurang_frame,kurang_text_styles,text="bukan ubin
kosong:",justify="center")
        kurang_label_3.grid(row=2,column=0,columnspan=2)

        #tombol solve
        solve_button =
tk.Button(self,text="SOLVE",command=lambda:threading.Thread(target=self.sol
ve()).start())
        solve_button.grid(row=2,column=1,pady=5)


        #menampilkan matriks awal
        start_matrix_frame =
tk.Frame(self,bg="#57536E",height=600,width=300,borderwidth=5)
        start_matrix_frame.grid(row=3,column=1)

        start_matrix_label =
tk.Label(start_matrix_frame,self.start_matrix_text_styles,text="Matrix
awal:",justify="center")
        start_matrix_label.grid(row=0,column=0)

        matrix_frame =
tk.Frame(start_matrix_frame,bg="#e3af74",height=200,width=200,borderwidth=5
)
        matrix_frame.grid(row=1,column=0)

        self.startMatrixCell = []
        k = 0
        for i in range(4):
            for j in range(4):
                cell = tk.Label(matrix_frame,matrix_cell_text_styles,text="
",justify="center",relief="raised",padx=5,pady=1,width=2)
                cell.grid(row=i,column=j)
                self.startMatrixCell.append(cell)
                k+=1
        #template buat nampilin matriks akhir
```

```
        self.end_matrix_frame =
tk.Frame(self,bg="#57536E",height=600,width=300,borderwidth=5)
        self.end_matrix_frame.grid(row=4,column=1)

        self.end_matrix_label =
tk.Label(self.end_matrix_frame,self.start_matrix_text_styles,text="Matrix
Akhir:",justify="center")
        self.end_matrix_label.grid(row=0,column=0,columnspan=2)

        self.iteration_label =
tk.Label(self.end_matrix_frame,self.start_matrix_text_styles,text="Iterasi
Ke-0",justify="center")
        self.iteration_label.grid(row=1,column=0,columnspan=2)

        self.e_matrix_frame =
tk.Frame(self.end_matrix_frame,bg="#e3af74",height=200,width=200,borderwidt
h=5)
        self.e_matrix_frame.grid(row=2,column=0,columnspan=2)

        self.endMatrixCell = []
        k = 0
        for i in range(4):
            for j in range(4):
                cell =
tk.Label(self.e_matrix_frame,matrix_cell_text_styles,text="
",justify="center",relief="raised",padx=5,pady=1,width=2)
                cell.grid(row=i,column=j)
                self.endMatrixCell.append(cell)
                k+=1
        #prev button
        self.prev_button = tk.Button(self.end_matrix_frame,text="PREV
MOVE",command=lambda:self.prev(),state="disabled")
        self.prev_button.grid(row=3,column=0)
        #next button
        self.next_button = tk.Button(self.end_matrix_frame,text="NEXT
MOVE",command=lambda:self.next(),state="disabled")
        self.next_button.grid(row=3,column=1)
    def generate(self):
        #menghasilkan matriks 15 puzzle acak
        self.matriks = [0 for i in range(16)]
        for i in range(16):
            angka = random.randint(1,16)
            while angka in self.matriks:
                angka = random.randint(1,16)
            self.matriks[i] = angka
        self.solver = TSP15Puzzle(self.matriks)
        for k in range(16):
            if(self.matriks[k]==16):
                self.startMatrixCell[k]['text'] = " "
            else:
                self.startMatrixCell[k]['text'] = str(self.matriks[k])
        if(len(self.filabel)>0):
            for i in range(len(self.filabel)):
                self.filabel[i]['text'] = ""
        if(self.time_label!=None):
            self.time_label['text'] = ""
        if(self.node_label!=None):
            self.node_label['text'] = ""
```

```
        self.sigma_label['text'] = "Nilai dari nilai status
reachable(sigma(i)+X): 0"
        for i in range(16):
            self.endMatrixCell[i]['text'] = " "
        self.warning_label['text'] = ""
        self.iteration_label['text'] = "Iterasi Ke-0"
        self.next_button['state'] = "disabled"
        self.prev_button['state'] = "disabled"
    def getMatrixFromFile(self):
        #memilih file input dari file
        filename = filedialog.askopenfilename(initialdir=os.getcwd())
        if(filename!=None):
            self.matriks = [0 for i in range(16)]
            i = 0
            f = open(filename, "r")
            for line in f:
                subarr = line.rstrip('\n').split(" ")
                for num in subarr:
                    self.matriks[i] = int(num)
                    i+=1
            f.close()
            self.solver = TSP15Puzzle(self.matriks)
            for k in range(16):
              #  print(type(self.startMatrixCell[k]))
                if(self.matriks[k]==16):
                    self.startMatrixCell[k]['text'] = " "
                else:
                    self.startMatrixCell[k]['text'] = str(self.matriks[k])
            if(len(self.filabel)>0):
                for i in range(len(self.filabel)):
                    self.filabel[i]['text'] = ""
            if(self.time_label!=None):
                self.time_label['text'] = ""
            if(self.node_label!=None):
                self.node_label['text'] = ""
            self.sigma_label['text'] = "Nilai dari nilai status
reachable(sigma(i)+X): 0"
            for i in range(16):
                self.endMatrixCell[i]['text'] = " "
            self.warning_label['text'] = ""
            self.iteration_label['text'] = "Iterasi Ke-0"
            self.next_button['state'] = "disabled"
            self.prev_button['state'] = "disabled"
    def solve(self):
        #menyelesaikan puzzle
        if(self.matriks!=None and self.solver!=None):
            self.warning_label['text'] = ""
            self.iteration_label['text'] = "Iterasi Ke-0"
            self.solver.setStartTime()
            #hitung yang kurang(i)
            status_number = 0
            for i in range(1,16):
                kurang_number = self.solver.KURANG(i)
                status_number += kurang_number
                fi_label =
tk.Label(self.kurang_frame,self.kurang_normal_text_styles,text="KURANG(%d)
=  %d" % (i, kurang_number),justify="center")
                fi_label.grid(row=i+2,column=0)
```

```
                self.filabel.append(fi_label)
            #Hitung nilai sigma(kurang-i)+x
            get_1_position = [ 1, 3, 4, 6, 9, 11, 12, 14 ] #jika kotak
kosong berada di indeks ini, maka nilai status_number += 1
            status_number += self.solver.KURANG(16)
            if(self.matriks.index(16) in get_1_position):
                status_number += 1
            self.sigma_label['text'] = "Nilai dari nilai status
reachable(sigma(i)+X): "+str(status_number)
            #mengecek status reachable
            if(status_number % 2 != 0):#kalau ganjil maka tidak reachable
                self.warning_label['text'] = "Persoalan tidak bisa
diselesaikan!"
            else:
                #menyelesaikan puzzle
                jumlah_simpul = self.solver.solve()
                #menghandle kasus waktu penyelesaian terlalu
lama(dinonaktifkan secara default, bisa diaktifkan dengan menghilangkan
                # tanda # di file TSP15Puzzle.py pada line 15,134,135,136)
                if(jumlah_simpul==None):
                    self.warning_label['text'] = "Persoalan membutuhkan
waktu yang lama untuk diselesaikan! (melebihi 8 menit!)"
                else:
                    #menampilkan waktu eksekusi program
                    time_elapsed = self.solver.getElapsedTime()
                    if(time_elapsed<1000):#kurang dari 1 detik
                        self.time_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="Waktu
eksekusi program: %s ms" % (round(time_elapsed,3)),justify="left")
                    elif(time_elapsed<1000*60):#kurang dari 1 menit
                        self.time_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="Waktu
eksekusi program: %d detik %s ms" %
(time_elapsed//1000,round(time_elapsed%1000,3)),justify="left")
                    elif(time_elapsed<1000*60*60):#kurang dari 1 jam
                        menit = time_elapsed//(1000*60)
                        remainder = time_elapsed%(1000*60)
                        detik = remainder //1000
                        ms = round(remainder % 1000,3)
                        self.time_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="Waktu
eksekusi program: %d menit %d detik %s ms" %
(menit,detik,ms),justify="left")
                    else: #lebih dari 1 jam
                        jam = time_elapsed//(1000*60*60)
                        rem_1 = time_elapsed%(1000*60*60)
                        menit = rem_1//(1000*60)
                        remainder = rem_1%(1000*60)
                        detik = remainder //1000
                        ms = round(remainder % 1000,3)
                        self.time_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="Waktu
eksekusi program: %d jam %d menit %d detik %s ms" %
(jam,menit,detik,ms),justify="left")
                    self.time_label.grid(row=1,column=0)
                    #menampilkan jumlah simpul yang dibangkitkan
```

```
                    self.node_label =
tk.Label(self.sigma_frame,self.start_matrix_text_styles,text="Jumlah simpul
yang dibangkitkan: %d" % (jumlah_simpul))
                    self.node_label.grid(row=2,column=0)
                    self.solution = self.solver.get_solution()
                    self.iteration = 0
                    initial_action = self.solution[self.iteration]
                    for k in range(16):
                        if(initial_action[2][k]==16):
                            self.endMatrixCell[k]['text'] = " "
                        else:
                            self.endMatrixCell[k]['text'] =
str(initial_action[2][k])
                    self.prev_button['state'] = "disabled"
                    if(len(self.solution)==1):
                        self.next_button['state'] = "disabled"
                    else:
                        self.next_button['state'] = "active"
    def next(self):
        #menuju move selanjutnya
        if(self.solution!=None):
            if(self.iteration<len(self.solution)):
                self.iteration+=1
                action = self.solution[self.iteration]
                self.iteration_label['text'] = "Iterasi Ke-
"+str(self.iteration)
                for k in range(16):
                    if(action[2][k]==16):
                        self.endMatrixCell[k]['text'] = " "
                    else:
                        self.endMatrixCell[k]['text'] = str(action[2][k])
                if(self.iteration==len(self.solution)-1):
                    self.next_button['state'] = "disabled"
                    if(len(self.solution)>1):
                        self.prev_button['state'] = "active"
                else:
                    self.next_button['state'] = "active"
                    self.prev_button['state'] = "active"
    def prev(self):
        #menuju move sebelumnya
        if(self.solution!=None):
            if(self.iteration>0):
                self.iteration-=1
                action = self.solution[self.iteration]
                self.iteration_label['text'] = "Iterasi Ke-
"+str(self.iteration)
                for k in range(16):
                    if(action[2][k]==16):
                        self.endMatrixCell[k]['text'] = " "
                    else:
                        self.endMatrixCell[k]['text'] = str(action[2][k])
                if(self.iteration==0):
                    self.prev_button['state'] = "disabled"
                    self.next_button['state'] = "active"
                else:
                    self.prev_button['state'] = "active"
                    self.next_button['state'] = "active"
```

Tabel 4.3. *Source Code* TSP15Puzzle.py (Bagian utama program yang berisi algoritma *Branch and Bound* untuk menyelesaikan 15 Puzzle)

```
import time
import copy
from operator import itemgetter

from numpy import array_equal
class TSP15Puzzle:
    def __init__(self,matrix):
        self.matrix = matrix
        self.startTime = None # waktu mulai dalam ms
        self.endTime = None #waktu algoritma selesai
        self.solution = []
        self.endNode = None
        self.simpul = []#simpul yang sudah pernah dibangkitkan
        self.simpul_hidup = []
        #self.MAXTIME = 8*60*1000 #waktu maksimum komputasi (dalam ms)
    def get_matrix(self):
        return self.matrix
    def getElapsedTime(self):
        #mengembalikan waktu algoritma dalam ms
        return self.endTime-self.startTime
    def cetakMatrix(self,matrix=None):
        if(matrix == None):
            matrix = self.matrix
        #mencetak matriks awal
        for i in range(16):
            print(matrix[i],end=" ")
            if(i==3 or i==7 or i==11):
                print()
        print()
    def KURANG(self,i):
        #menghitung jumlah nilai di depan posisi nomor i yang nilainya
lebih kecil dari i
        count = 0
        start_idx = self.matrix.index(i)
        for n in range(start_idx+1,16):
            if(self.matrix[n]<i and self.matrix[n]!=0):
                count += 1
        return count
    def getCost(self, tempMatrix):
        #mendapatkan cost suatu cabang
        #cari f(i)
        #langsung cari g(i)
        cost = 0
        #while(idx_parent!=0):#ulangi sampai nyampe ke root
        #    #idx_parent =
self.simpul[self.getIDX(idx_parent,self.simpul)][1]
        #    idx_parent = self.simpul[idx_parent][1]
        #    cost+=1
        #g(i)
        for i in range(16):
            if(tempMatrix[i] != 16 and tempMatrix[i]!=i+1):
                cost+=1
        return cost
    def g(self,matrix):
        #mengembalikan cost mencapai simpul tujuan dari simpul i
```

```
        #dihitung dengan menghitung jumlah label yang tidak berada di
posisi akhirnya
        cost = 0
        for i in range(1,17):
            if(matrix[i-1]!=i):
                cost += 1
        return cost

    def solve(self):
        #menyelesaikan puzzle dengan algoritma branch and bound
        #struktur data simpul: simpul = (indeks,parent,isi,
cost,hash_value,kedalaman)->disimpan di atribut kelas
        #return (jumlah_simpul_yang_berhasil_dibangkitkan)
        self.solution = []
        #inisialisasi simpul pertama
        i = 0
        first_node = [i,-1,self.matrix,0,self.hashing(self.matrix),0]
        self.simpul.append(first_node)
        jumlah_simpul = 1
        if(self.g(first_node[2])==0):
            self.endNode = copy.deepcopy(first_node)
            self.endTime = time.time()*1000
            self.getPath(self.endNode)
            return jumlah_simpul
        i+=1
        # melakukan algoritma branch and bound
        # inisialisasi simpul awal
        idx_kosong = self.matrix.index(16)
        found = False
        # 1->ke atas
        if(idx_kosong//4 > 0 and not found):#bukan di baris pertama
            temp = copy.deepcopy(self.matrix)
            temp[idx_kosong],temp[idx_kosong-4] = temp[idx_kosong-
4],temp[idx_kosong]
            node = [i,0,temp,1+self.getCost(temp),self.hashing(temp),1]
            self.simpul_hidup.append(node)
            self.simpul.append(node)
            i+=1
            jumlah_simpul+=1
            if(self.g(node[2])==0):
                self.endNode = copy.deepcopy(node)
                #solusi ketemu
                found = True
        # 2->ke kanan
        if(idx_kosong%4 != 3 and not found):#bukan di kolom terakhir
            temp = copy.deepcopy(self.matrix)
            temp[idx_kosong],temp[idx_kosong+1] =
temp[idx_kosong+1],temp[idx_kosong]
            node = [i,0,temp,1+self.getCost(temp),self.hashing(temp),1]
            self.simpul_hidup.append(node)
            self.simpul.append(node)
            i+=1
            jumlah_simpul+=1
            if(self.g(node[2])==0):
                self.endNode = copy.deepcopy(node)
                #solusi ketemu
                found = True
        # 3->ke bawah
```

```
        if(idx_kosong//4 != 3 and not found):#bukan di baris terakhir
            temp = copy.deepcopy(self.matrix)
            temp[idx_kosong],temp[idx_kosong+4] =
temp[idx_kosong+4],temp[idx_kosong]
            node = [i,0,temp,1+self.getCost(temp),self.hashing(temp),1]
            self.simpul_hidup.append(node)
            self.simpul.append(node)
            i+=1
            jumlah_simpul+=1
            if(self.g(node[2])==0):
                self.endNode = copy.deepcopy(node)
                #solusi ketemu
                found = True
        # 4->ke kiri
        if(idx_kosong%4 != 0 and not found):#bukan di kolom pertama
            temp = copy.deepcopy(self.matrix)
            temp[idx_kosong],temp[idx_kosong-1] = temp[idx_kosong-
1],temp[idx_kosong]
            node = [i,0,temp,1+self.getCost(temp),self.hashing(temp),1]
            self.simpul_hidup.append(node)
            self.simpul.append(node)
            i+=1
            jumlah_simpul+=1
            if(self.g(node[2])==0):
                self.endNode = copy.deepcopy(node)
                #solusi ketemu
                found = True
        while(len(self.simpul_hidup)>0 and not found):#selama masih ada
simpul hidup
            #temp_time = time.time()*1000
            #if(temp_time-self.startTime>self.MAXTIME):
            #    return None
            temp = copy.deepcopy(self.simpul_hidup)
            self.simpul_hidup = sorted(temp,key=itemgetter(3))#urutkan dari
cost yang terkecil
            node = self.simpul_hidup.pop(0)
            if(self.g(node[2])==0):
                if(self.endNode==None):
                    self.endNode = copy.deepcopy(node)
                    self.pruning(self.endNode[3])
                    if(len(self.simpul_hidup)==0):
                        break
                    #solusi ketemu
                elif(node[3]<self.endNode[3]):
                    self.endNode = copy.deepcopy(node)
                    self.pruning(self.endNode[3])
                    if(len(self.simpul_hidup)==0):
                        break
                    #solusi ketemu
            #generasikan simpul lain
            idx_kosong = node[2].index(16)
            f_i = self.simpul[node[0]][5]+1
            # 1->ke atas
            if(idx_kosong//4 > 0):#bukan di baris pertama
                temp = copy.deepcopy(node[2])
                temp[idx_kosong],temp[idx_kosong-4] = temp[idx_kosong-
4],temp[idx_kosong]
                jumlah_simpul+=1
```

```
                hash_value = self.hashing(temp)
                if self.checkUnique(hash_value):
                    #dapatkan kedalaman matriks atasannya
                    #f_i = self.simpul[node[0]][5]+1
                    temp_node =
[i,node[0],temp,f_i+self.getCost(temp),hash_value,f_i]
                    self.simpul_hidup.append(temp_node)
                    self.simpul.append(temp_node)
                    i+=1
            # 2->ke kanan
            if(idx_kosong%4 != 3):#bukan di kolom terakhir
                temp = copy.deepcopy(node[2])
                temp[idx_kosong],temp[idx_kosong+1] =
temp[idx_kosong+1],temp[idx_kosong]
                jumlah_simpul+=1
                hash_value = self.hashing(temp)
                if self.checkUnique(hash_value):
                    #f_i = self.simpul[node[0]][5]+1
                    temp_node =
[i,node[0],temp,f_i+self.getCost(temp),hash_value,f_i]
                    self.simpul_hidup.append(temp_node)
                    self.simpul.append(temp_node)
                    i+=1
            # 3->ke bawah
            if(idx_kosong//4 != 3):#bukan di baris terakhir
                temp = copy.deepcopy(node[2])
                temp[idx_kosong],temp[idx_kosong+4] =
temp[idx_kosong+4],temp[idx_kosong]
                jumlah_simpul+=1
                hash_value = self.hashing(temp)
                if self.checkUnique(hash_value):
                    #f_i = self.simpul[node[0]][5]+1
                    temp_node =
[i,node[0],temp,f_i+self.getCost(temp),hash_value,f_i]
                    self.simpul_hidup.append(temp_node)
                    self.simpul.append(temp_node)
                    i+=1
            # 4->ke kiri
            if(idx_kosong%4 != 0):#bukan di kolom pertama
                temp = copy.deepcopy(node[2])
                temp[idx_kosong],temp[idx_kosong-1] = temp[idx_kosong-
1],temp[idx_kosong]
                jumlah_simpul+=1
                hash_value = self.hashing(temp)
                if self.checkUnique(hash_value):
                    #f_i = self.simpul[node[0]][5]+1
                    temp_node =
[i,node[0],temp,f_i+self.getCost(temp),hash_value,f_i]
                    self.simpul_hidup.append(temp_node)
                    self.simpul.append(temp_node)
                    i+=1
        #mendapatkan path rute
        self.getPath(self.endNode)
        self.endTime = time.time()*1000
        return jumlah_simpul
    def getIDX(self,node_idx,matrix):
        #mendapatkan index suatu node
        i = 0
```

```
        for elmt in matrix:
            if(elmt[0]==node_idx):#indeksnya sama
                return i
            i+=1
    def getPath(self,node):
        #mendapatkan path dari end node ke root
        node_idx = self.getIDX(node[0],self.simpul)
        self.solution.append(self.simpul[node_idx])
        while(node_idx!=0):#bukan root,loop sampe ketemu loop
            node_idx = self.getIDX(node[1],self.simpul)
            node = self.simpul[node_idx]
            self.solution.append(node)
        self.solution = sorted(self.solution,key=itemgetter(0))
    def showStep_CLI(self):
        i = 1
        for idx in range(len(self.solution)):
            print("Langkah "+str(i)+":")
            self.cetakMatrix(self.solution[idx][2])
            i += 1
    def checkUniquePakeMatrix(self,matrix):
    #memeriksa apakah suatu matrix sudah ada di self.simpul atau belum
dengan cek manual isi matriks
        for simpul in self.simpul:
            if self.compareMatrix(simpul[2],matrix):
                return False
        return True
    def checkUnique(self,hash_matriks):
        #memeriksa apakah suatu matrix sudah ada di self.simpul atau belum
        for simpul in self.simpul:
            if simpul[4]==hash_matriks:
                return False
        return True
    def compareMatrix(self,mat1,mat2):
    #membandingkan 2 buah matrix
        for i in range(len(mat1)):
            if(mat1[i]!=mat2[i]):
                return False
        return True
    def setStartTime(self):
        #mengassign nilai waktu mulai
        self.startTime = time.time()*1000
    def get_solution(self):
        #mendapatkan solusi 15 puzzle
        return self.solution
    def hashing(self,object):
        #combine tiap element di object jadi sebuah string
        string = ""
        for char in object:
            string+= "0"
            string+= str(char)
        return hash(string)
    def pruning(self,cost):
        #memangkas cabang aktif yang cost nya lebih besar dari cost
        temp = copy.deepcopy(self.simpul_hidup)
        for simpul in self.simpul_hidup:
            if(simpul[3]>cost):
                idx = self.getIDX(simpul[0],temp)
                temp.pop(idx)
```

```
        self.simpul_hidup = copy.deepcopy(temp)
```

V. Instantiasi Contoh Kasus

- Catatan: Angka 16 di file test case menandakan sel kosong

1) TC1(nama file: test-case-1(unsolvable).txt):

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

2) TC2(nama file: test-case-2(unsolvable).txt):

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

3) TC3(nama file: test-case-3(solvable).txt):

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

4) TC4(nama file: test-case-4(solvable).txt):

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

5) TC5(nama file: test-case-5(solvable).txt):

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

VI. Alamat Kode Program

Alamat repositori kode program:
https://github.com/Wiradhika6051/Tucil-3-Stima-15-Puzzle/tree/main