Tugas Kecil 3 Strategi Algoritma Semester II/2022:
Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*

Oleh: Nathanael Santoso

Bab I
Algoritma

1.1. Pendahuluan

Algoritma Branch and Bound merupakan algoritma Best First Search yang menerapkan bounding yaitu operasi mematikan simpul pohon yang memenuhi suatu kriteria heuristik. Seperti pada Best First Search, setiap simpul dalam algoritma Branch and Bound akan diberikan sebuah nilai yang disebut *cost*. Nilai tersebut ditentukan secara heuristik berdasarkan hasil yang diinginkan. Berbeda dengan Best First Search, jika sudah ditemukan simpul solusi, maka semua simpul dalam antrian pencarian yang *cost*-nya lebih tinggi dari *cost* simpul solusi akan dimatikan.

Persoalan 15-Puzzle merupakan sebuah persoalan dimana terdapat 15 petak bernomor terurut dari 1 sampai 15 dan satu petak kosong di dalam sebuah wadah berukuran 4 petak kali 4 petak. Sebuah petak dapat digeser ke dalam petak kosong sehingga posisi awal petak menjadi petak kosong dan posisi petak kosong terisi oleh petak awal. Tujuan dari permainan ini adalah mengurutkan petak sehingga nomor petak terurut dari 1 sampai lima belas dalam wadah dan petak kosong berada pada pojok kanan bawah. Persoalan 15-Puzzle ini dapat diselesaikan oleh algoritma Branch and Bound menggunakan fungsi *cost* yang berupa pertambahan dari kedalaman sebuah simpul dan jumlah petak yang tidak sesuai dengan solusi. Namun sebelum itu, terdapat preprocessing terhadap posisi awal menggunakan fungsi $\sum Kurang(i) + X$ di mana variabel i adalah petak dalam posisi awal. Nilai fungsi $Kurang(i)$ adalah total dari semua petak j dalam posisi awal yang posisi j dalam posisi tujuan lebih kecil dan posisi i dalam posisi tujuan, namun posisi j dalam posisi awal lebih besar dari posisi i dalam posisi awal. Nilai variabel X adalah 1 jika petak kosong terletak pada posisi 2, 4, 5, 7, 11, 13, 14, atau 16 dan untuk sisanya X bernilai 0. Jika fungsi $\sum Kurang(i) + X$ bernilai genap, maka posisi tujuan dapat dicapai dan algoritma dapat dijalankan. Tetapi, jika bernilai ganjil, maka posisi tujuan tidak bisa tercapai.

1.2. Alur Umum

Mengikuti penjelasan pada pendahuluan, program akan mengikuti alur sebagai berikut:

- Mendapatkan posisi awal 15-Puzzle dari input atau membuat posisi awal acak menggunakan algoritma *pseudorandom*.
- Melakukan Pengecekan terhadap posisi awal menggunakan fungsi $\sum Kurang(i) + X$, dan apabila bisa dicapai, maka akan melanjutkan program. Jika tidak dapat dicapai, tidak akan melanjutkan perhitungan menggunakan algoritma Branch and Bound.
- Menerapkan algoritma Branch and Bound untuk mencari solusi penyelesaian terbaik.

1.3. Penjelasan Implementasi

1.3.1. Konstanta

- VERBOSE: True jika ada masukan argument '-v' atau '-verbose', False jika tidak. Program akan menampilkan luaran tambahan jika True.
- DEPTH: Jika masukan argument angka, maka Depth = Argumen, None jika tidak
- ENDSTATE: Posisi tujuan dari 15-Puzzle

1.3.2. Fungsi Umum

1.3.2.1. displayPuzzle(node: Node)

Fungsi ini menampilkan 15-Puzzle pada terminal dengan warna yang bagus dan tabel yang cantik.

1.3.2.2. getScramble()

Fungsi ini mengembalikan sebuah posisi awal teracak yang dibuat dengan aturan pseudorandom sebagai berikut:

- Menetapkan variabel kedalaman Depth sama dengan konstanta argument DEPTH. Jika konstanta argumen DEPTH tidak ada, maka Depth adalah angka acak di antara 12 dan 18
- Dengan posisi tujuan sebagai basis, dipilih arah secara acak berdasarkan posisi petak kosong dengan syarat arah tersebut mungkin dilakukan (e.x. jika di pinggir kanan, tidak mungkin melakukan pergeseran ke kiri karena petak di sebelah kanan petak kosong tidak ada.)
- Mengulangi pengambilan arah sampai jumlah arah sama dengan Depth dengan syarat arah tidak boleh berkebalikan dengan arah yang diambil sebelumnya (e.x. Jika arah sebelumnya kiri, tidak akan mengambil kanan)
- Menyusun petak berdasarkan arah-arah yang telah diambil.

1.3.2.3. Reachable(endpos, curpos)

Fungsi ini merupakan implementasi dari fungsi $\sum Kurang(i) + X$ dan akan menampilkan tabel nilai setiap petak dalam curpos dan X beserta nilai total fungsi. Jika posisi akhir endpos dapat dicapai, maka akan mengembalikan True dan jika tidak maka False.

1.3.2.4. getMoves(node: Node)

Fungsi ini mengembalikan posisi yang dapat dicapai dari suatu posisi awal masukan dengan syarat posisi yang dikembalikan memungkinkan dicapai dengan aturan pergerakan petak 15-Puzzle dan posisi dengan arah berkebalikan dari posisi awal. (e.x. Jika posisi awal dicapai dengan pergerakan petak ke kiri, maka tidak akan mengembalikan posisi hasil pergerakan petak ke kanan)

1.3.2.5. loading()

Menampilkan loading bar untuk persoalan yang komputasinya panjang.

1.3.3. Kelas Node

1.3.3.1. Atribut

- Position: Array of Array of integer
- Parent: Node
- Child: array of Node
- Cost: integer
- Direction: string
- Bounded: Boolean
- Solution: Boolean

1.3.3.2. Metode

- Ctor(Position, Parent=None, Direction=None, Bounded=False, Solution=False):
  Inisialisasi Node dengan Position, Parent, Direction, Bounded, dan Solution. Cost Diinitialisasi dengan perhitungan f() dan g().
- f():
  Menghitung kedalaman Node menggunakan rekursi Parent
- g(endpos)
  Menghitung Jumlah petak dalam Position yang tidak sesuai posisinya dengan posisi akhir endpos
- Getters, Setters, and Print Function

1.3.4. Kelas BranchAndBound

1.3.4.1. Atribut

- Root: Node
- Queue: Array of Node
- Solutions: Array of Node
- Paths: Array of Array of Node
- Handler: Thread.Event() object (For handling keyboard interrupt)
- NodeCount: integer
- Starttime: Double
- Endtime: Double

1.3.4.2. Method

- Ctor(Position: Array of Array of integer):
  Initialisasi Node dengan Root yang merupakan Node(Position), Queue, Solutions, dan Paths, dengan array kosong, handler dengan Thread.Event() Object, dan NodeCount, Starttime, dan Endtime dengan 0. Kemudian Jika Reachable(Root.Position) maka akan membuat thread proses self.solve() baru yang dijalankan pada latar belakang. Jika sudah selesai, maka akan menampilkan luaran Solusi Terbaik/ Beberapa Solusi Terbaik Paths, Jumlah Simpul yang dibangkitkan NodeCount, dan Waktu yang diperlukan (Endtime - Starttime) * 1000.
- raiseNodes(root_node: Node):
  Memanggil getMoves(root_node) untuk mendapatkan posisi yang mungkin tercapai dari simpul root_node, membangkitkan anak simpul berdasarkan posisi tersebut, dan memasukkan anak simpul ke dalam Queue. Anak simpul kemudian diperiksa apakan

*cost*nya lebih tinggi dari *cost* simpul-simpul dalam Solutions (jika ada) dan jika lebih tinggi, maka akan di-Bound dan dikeluarkan dari Queue. Kemudian, root_node dikeluarkan dari Queue.

- findLowestCost():
  Menemukan Node dengan *cost* terkecil dalam Queue yang tidak Bounded, jika tidak ketemu maka mengembalikan None.
- getSolved():
  Mencari Node solusi dalam Queue. Jika ditemukan, maka semua node dalam Queue dengan nilai *cost* lebih besar dari solusi akan di-Bound dan dikeluarkan dari Queue. Solusi kemudian dimasukkan ke dalam himpunan solusi Solutions, atribut Solution-nya menjadi True, dan dikeluarkan dari Queue.
- solve():
  Memanggil getSolved(), root_node = findLowestCost(), dan raiseNodes(root_node: Node) secara sekuensial dalam loop tak terhingga sampai findLowestCost mengembalikan None, kemudian keluar dari loop.
  Himpunan Solusi kemudian diambil solusi dengan *cost* terkecil dan semua solusi yang memiliki *cost* lebih besar di-Bound, atribut Solution-nya dijadikan False, dan dikeluarkan dari himpunan solusi.
  Kemudian, untuk setiap solusi dalam Solutions dibentuk sebuah array of Node yang terdiri dari setiap simpul dari akar sampai solusi secara sekuensial. Setiap array tersebut kemudian dimasukkan ke dalam Paths.
- signal_handler(signum, frame):
  Untuk menangani KeyboardInterrupt pada thread process.
- printTree(node: Node = None, spaces = 0):
  Mengeluarkan pohon ruang status dari Objek BranchAndBound secara rekursif.

Bab II
Implementasi

2.1. Konstanta dan Fungsi

```
import os
import sys
import random
import time
import threading
import signal
import pandas as pd
from tabulate import tabulate
from colorama import Fore, Style
from copy import deepcopy

# Global Constants
VERBOSE = False
CUR_PATH = os.path.dirname(os.path.abspath(__file__))
DEPTH = None

if len(sys.argv) > 2:
    if sys.argv[1] == "-verbose" or sys.argv[1] == "-v":
        VERBOSE = True
    if sys.argv[2].isnumeric():
        DEPTH = int(sys.argv[2])
        NEW_PATH = None
    else:
        NEW_PATH = CUR_PATH[:-3] + "test\\" + sys.argv[2]
        if not os.path.exists(NEW_PATH):
            print("File Doesn't Exist, Running Default Script")
            NEW_PATH = None
elif len(sys.argv) > 1:
    if sys.argv[1] == "-verbose" or sys.argv[1] == "-v":
        VERBOSE = True
        NEW_PATH = None
    elif sys.argv[1].isnumeric():
        DEPTH = int(sys.argv[1])
        NEW_PATH = None
    else:
        NEW_PATH = CUR_PATH[:-3] + "test\\" + sys.argv[1]
        if not os.path.exists(NEW_PATH):
            print("File Doesn't Exist, Running Default Script")
            NEW_PATH = None

ENDSTATE = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 0]]

# unicode for terminal drawing
BOTTOM_LEFT = '\u2514'
```

```
BOTTOM_RIGHT = '\u2518'
TOP_LEFT = '\u250C'
TOP_RIGHT = '\u2510'

CROSSROAD = '\u253C'
TOP_T_JUNCTION = '\u252C'
BOTTOM_T_JUNCTION = '\u2534'
LEFT_T_JUNCTION = '\u251C'
RIGHT_T_JUNCTION = '\u2524'
VERTICAL = Style.BRIGHT + Fore.BLUE + '\u2502' + Fore.RESET +
Style.RESET_ALL
HORIZONTAL = '\u2500'

# Separator Line Constants
TOP_LINE = Style.BRIGHT + Fore.BLUE + TOP_LEFT + HORIZONTAL + HORIZONTAL +
HORIZONTAL + HORIZONTAL + TOP_T_JUNCTION + HORIZONTAL + HORIZONTAL +
HORIZONTAL + HORIZONTAL + TOP_T_JUNCTION + HORIZONTAL + HORIZONTAL +
HORIZONTAL + HORIZONTAL + TOP_T_JUNCTION + HORIZONTAL + HORIZONTAL +
HORIZONTAL + HORIZONTAL + TOP_RIGHT + Fore.RESET + Style.RESET_ALL
MIDDLE_LINE = Style.BRIGHT + Fore.BLUE + LEFT_T_JUNCTION + HORIZONTAL +
HORIZONTAL + HORIZONTAL + HORIZONTAL + CROSSROAD + HORIZONTAL + HORIZONTAL +
HORIZONTAL + HORIZONTAL + CROSSROAD + HORIZONTAL + HORIZONTAL + HORIZONTAL +
HORIZONTAL + CROSSROAD + HORIZONTAL + HORIZONTAL + HORIZONTAL + HORIZONTAL +
RIGHT_T_JUNCTION + Fore.RESET + Style.RESET_ALL
BOTTOM_LINE = Style.BRIGHT + Fore.BLUE + BOTTOM_LEFT + HORIZONTAL +
HORIZONTAL + HORIZONTAL + HORIZONTAL + BOTTOM_T_JUNCTION + HORIZONTAL +
HORIZONTAL + HORIZONTAL + HORIZONTAL + BOTTOM_T_JUNCTION + HORIZONTAL +
HORIZONTAL + HORIZONTAL + HORIZONTAL + BOTTOM_T_JUNCTION + HORIZONTAL +
HORIZONTAL + HORIZONTAL + HORIZONTAL + BOTTOM_RIGHT + Fore.RESET +
Style.RESET_ALL

# Output Constants
LEFT_BRACKET = Style.BRIGHT + Fore.BLUE + HORIZONTAL + HORIZONTAL +
RIGHT_T_JUNCTION + Fore.RESET + Style.RESET_ALL
RIGHT_BRACKET = Style.BRIGHT + Fore.BLUE + LEFT_T_JUNCTION + HORIZONTAL +
HORIZONTAL + Fore.RESET + Style.RESET_ALL

# 15-Puzzle Print
def displayPuzzle(node: Node):
    matrix = node.position
    print(TOP_LINE)
    for i, array in enumerate(matrix):
        for j, num in enumerate(array):
            print(VERTICAL, end=' ')
            if ENDSTATE[i][j] == num:
                print(Style.BRIGHT + Fore.GREEN, end='')
            else:
                print(Style.BRIGHT + Fore.RED, end='')
```

```
            if num == 0:
                print(Fore.RESET + Style.RESET_ALL, end='   ')
            elif num < 10:
                print(str(num) + Fore.RESET + Style.RESET_ALL, end='  ')
            else:
                print(str(num) + Fore.RESET + Style.RESET_ALL, end=' ')
        print(VERTICAL)
        if matrix.index(array) == 3:
            print(BOTTOM_LINE)
        else:
            print(MIDDLE_LINE)

# PseudoRandom Scramble
def getScramble():
    scramble = list()
    matrix = deepcopy(ENDSTATE)

    if DEPTH == None:
        depth = random.randint(12, 18)
    else:
        depth = DEPTH

    for i, array in enumerate(matrix):
        if 0 in array:
            j = array.index(0)
            idx_i = i
            idx_i_copy = i
            idx_j = j
            idx_j_copy = j

    while len(scramble) < depth:
        rdm = None
        while True:
            rdm = random.randint(0, 3)
            if len(scramble) > 0 and scramble[-1] == (2 + rdm) % 4:
                continue
            match rdm:
                case 0 if idx_i_copy < 3:
                    idx_i_copy += 1
                    break
                case 1 if idx_j_copy > 0:
                    idx_j_copy -= 1
                    break
                case 2 if idx_i_copy > 0:
                    idx_i_copy -= 1
                    break
                case 3 if idx_j_copy < 3:
                    idx_j_copy += 1
```

```
                    break
        scramble.append(rdm)

    for dir in scramble:
        match dir:
            case 0:
                matrix[idx_i][idx_j], matrix[idx_i + 1][idx_j] =
matrix[idx_i + 1][idx_j], matrix[idx_i][idx_j]
                idx_i += 1
            case 1:
                matrix[idx_i][idx_j], matrix[idx_i][idx_j - 1] =
matrix[idx_i][idx_j - 1], matrix[idx_i][idx_j]
                idx_j -= 1
            case 2:
                matrix[idx_i][idx_j], matrix[idx_i - 1][idx_j] =
matrix[idx_i - 1][idx_j], matrix[idx_i][idx_j]
                idx_i -= 1
            case 3:
                matrix[idx_i][idx_j], matrix[idx_i][idx_j + 1] =
matrix[idx_i][idx_j + 1], matrix[idx_i][idx_j]
                idx_j += 1

    return matrix

def Reachable(endpos, curpos):
    endpos_flat = [item for array in endpos for item in array]
    curpos_flat = [item for array in curpos for item in array]
    count = 0

    d = dict()
    for i in range(1, 16):
        d[str(i)] = 0
    d['0'] = 0
    d['X'] = 0

    for i, num in enumerate(curpos_flat):
        kurang = 0
        idx_i = endpos_flat.index(num)
        for j in range(i + 1, len(curpos_flat)):
            idx_j = endpos_flat.index(curpos_flat[j])
            if idx_j < idx_i:
                kurang += 1
        d[str(num)] = kurang
        count += kurang

    if curpos_flat.index(0) in [1, 3, 4, 6, 9, 11, 12, 14]:
        d['X'] = 1
        count += 1
```

```
    d['Total'] = count
    df = pd.DataFrame(d, index=[0])
    print(tabulate(df.T))
    print("The Value Of Reachability is:", count)

    if count % 2 == 0:
        print("Hence the End State is Reachable")
        return True
    else:
        print("Hence the End State is Unreachable")
        return False

def getMoves(node: Node):
    matrix = node.getPosition()
    prevdir = node.getDirection()

    match prevdir:
        case "UP":
            prevmove = 0
        case "RIGHT":
            prevmove = 1
        case "DOWN":
            prevmove = 2
        case "LEFT":
            prevmove = 3
        case _:
            prevmove = -1

    for i, array in enumerate(matrix):
        if 0 in array:
            j = array.index(0)
            # Move Up
            if i < 3 and prevmove != 2:
                u = deepcopy(matrix)
                u[i][j], u[i + 1][j] = u[i + 1][j], u[i][j]
            else:
                u = None
            # Move Right
            if j > 0 and prevmove != 3:
                r = deepcopy(matrix)
                r[i][j], r[i][j - 1] = r[i][j - 1], r[i][j]
            else:
                r = None
            # Move Down
            if i > 0 and prevmove != 0:
                d = deepcopy(matrix)
                d[i][j], d[i - 1][j] = d[i - 1][j], d[i][j]
```

```
            else:
                d = None
            # Move Left
            if j < 3 and prevmove != 1:
                l = deepcopy(matrix)
                l[i][j], l[i][j + 1] = l[i][j + 1], l[i][j]
            else:
                l = None

    return [u, r, d, l]

def loading():
    for c in ['   ', '.  ', '.. ', '...']:
        sys.stdout.write('Calculating Best Route' + c + '\r')
        sys.stdout.flush()
        time.sleep(0.1)
```

2.2. Kelas Node

```
class Node:
    def __init__(self, _position, _parent = None, _direction = None,
_bounded = False, _solution = False):
        self.position = _position
        self.parent = _parent
        if self.parent != None:
            self.cost = self.f() + self.g(ENDSTATE)
        else:
            self.cost = None
        self.direction = _direction
        self.bounded = _bounded
        self.solution = _solution
        self.child = list()

    def f(self):
        if self.parent == None:
            return 0
        else:
            return 1 + self.parent.f()

    def g(self, endpos):
        endpos_flat = [item for array in endpos for item in array]
        curpos_flat = [item for array in self.position for item in array]
        count = 0
        for i, num in enumerate(curpos_flat):
            if num != 0 and i != endpos_flat.index(num):
                count += 1
        return count
```

```
    def addChild(self, _child):
        self.child.append(_child)

    def getParent(self):
        return self.parent

    def getChildren(self):
        return self.child

    def getPosition(self):
        return self.position

    def getDirection(self):
        return self.direction

    def getCost(self):
        return self.cost

    def isBounded(self):
        return self.bounded

    def toggleBounded(self):
        self.bounded = not self.bounded

    def toggleSolution(self):
        self.solution = not self.solution

    def __repr__(self, indent=0):
        space = " "
        first  = "| " + str(self.position[0][0]).rjust(2) + " " +
str(self.position[0][1]).rjust(2) + " " + str(self.position[0][2]).rjust(2)
+ " " + str(self.position[0][3]).rjust(2) + " | " + str(self.direction) +
",\n"
        second = "| " + str(self.position[1][0]).rjust(2) + " " +
str(self.position[1][1]).rjust(2) + " " + str(self.position[1][2]).rjust(2)
+ " " + str(self.position[1][3]).rjust(2) + " | " + str(self.cost) + ",\n"
        third  = "| " + str(self.position[2][0]).rjust(2) + " " +
str(self.position[2][1]).rjust(2) + " " + str(self.position[2][2]).rjust(2)
+ " " + str(self.position[2][3]).rjust(2) + " | " + ("Bounded" if
self.bounded else "UnBounded") + ",\n"
        fourth = "| " + str(self.position[3][0]).rjust(2) + " " +
str(self.position[3][1]).rjust(2) + " " + str(self.position[3][2]).rjust(2)
+ " " + str(self.position[3][3]).rjust(2) + " | " + ("Solution" if
self.solution else "Not Solution")
        return space * indent + first + space * indent + second + space *
indent + third + space * indent + fourth
```

2.3. Kelas BranchAndBound

```
class BranchAndBound:
    def __init__(self, _position):
        self.root = Node(_position)
        self.queue = list()
        self.solutions = list()
        self.paths = list()
        self.handler = threading.Event()
        self.nodecount = 0
        self.starttime = 0
        self.endtime = 0
        if Reachable(ENDSTATE, self.root.position):
            self.queue.append(self.root)
            self.nodecount += 1
            signal.signal(signal.SIGINT, self.signal_handler)
            process = threading.Thread(target=self.solve)
            if VERBOSE:
                process.daemon = True
                print("\nOrder of Node Expansion:")
                process.start()
                while process.is_alive():
                    time.sleep(1)
            else:
                process.start()
                while process.is_alive():
                    loading()
                sys.stdout.write('                          ')

            if self.handler.is_set():
                sys.exit("\nProcess Interupted, Exiting...")

            for i, path in enumerate(self.paths):
                if len(self.paths) == 1:
                    print("\nSingle Best Solution:")
                else:
                    print("\nBest Solution " + str(i + 1) + ":\n")
                if VERBOSE:
                    print("--- Initial State ---")
                    print(tabulate(self.root.position, tablefmt="grid"))
                else:
                    print(LEFT_BRACKET + "INITIAL STATE" + RIGHT_BRACKET)
                    displayPuzzle(self.root)
                for node in path:
                    if VERBOSE:
                        print("--- " + node.direction + " ---")
                        print(tabulate(node.position, tablefmt="grid"))
                    else:
```

```
                        print(LEFT_BRACKET + node.direction + RIGHT_BRACKET)
                        displayPuzzle(node)

            if VERBOSE:
                print("\n--- Total Nodes: " + str(self.nodecount) + " ---")
                print("\n--- Time Taken : {:.3f}ms ---".format((self.endtime
- self.starttime) * 1000))
            else:
                print("\n" + LEFT_BRACKET + "Total Nodes: " +
str(self.nodecount) + RIGHT_BRACKET)
                print("\n" + LEFT_BRACKET + "Time Taken :
{:.3f}ms".format((self.endtime - self.starttime) * 1000) + RIGHT_BRACKET)

        else:
            print("\nEnd State Unreachable, Exiting...")
            sys.exit()

    def raiseNodes(self, root_node: Node):
        posarr = getMoves(root_node)
        for i, pos in enumerate(posarr):
            if pos != None:
                match i:
                    case 0:
                        child = Node(pos, root_node, 'UP')
                    case 1:
                        child = Node(pos, root_node, 'RIGHT')
                    case 2:
                        child = Node(pos, root_node, 'DOWN')
                    case 3:
                        child = Node(pos, root_node, 'LEFT')
                if VERBOSE:
                    print(tabulate(child.position, tablefmt="grid"))
                root_node.addChild(child)
                self.queue.append(child)
                self.nodecount += 1
                for solution in self.solutions:
                    if solution.getCost() < child.getCost() and child in
self.queue:
                        child.toggleBounded()
                        self.queue.remove(child)
        self.queue.remove(root_node)

    def findLowestCost(self):
        best = None
        for node in self.queue:
            if not node.isBounded():
                if best == None:
                    best = node
```

```
                elif best.getCost() > node.getCost():
                    best = node
        return best

    def getSolved(self):
        solution = None
        for node in self.queue:
            if node.position == ENDSTATE:
                solution = node
                solution.toggleSolution()
            if solution != None and solution not in self.solutions:
                for node in self.queue:
                    if node.getCost() > solution.getCost() and not
node.isBounded():
                        node.toggleBounded()
                        self.queue.remove(node)
                self.solutions.append(solution)
                self.queue.remove(solution)

    def solve(self):
        self.starttime = time.time()

        while not self.handler.is_set():
            self.getSolved()
            next = self.findLowestCost()
            if next == None:
                break
            else:
                self.raiseNodes(next)

        self.endtime = time.time()

        mincost = None
        for solution in self.solutions:
            if mincost == None:
                mincost = solution.getCost()
            elif mincost > solution.getCost():
                mincost = solution.getCost()

        for solution in self.solutions:
            if solution.getCost() > mincost:
                solution.toggleSolution()
                solution.toggleBounded()
                self.solutions.remove(solution)

        for solution in self.solutions:
            path = list()
            node = solution
```

```
            while node.getParent() != None:
                path.append(node)
                node = node.getParent()
            path.reverse()
            self.paths.append(path)

    def signal_handler(self, signum, frame):
        self.handler.set()

    def printTree(self, node: Node = None, spaces = 0):
        if node == None:
            print("\nState Space Tree:\n")
            print('+')
            print(self.root)
            self.printTree(self.root, 2)
        else:
            for child in node.getChildren():
                print(' ' * spaces + '+')
                print(child.__repr__(spaces))
                self.printTree(child, spaces + 2)
```

2.4. Main

```
if __name__ == "__main__":
    if len(sys.argv) > 1 and NEW_PATH != None:
        with open(NEW_PATH, 'r') as f:
            lines = f.readlines()
            array = [[int(num) for num in line.split()] for line in lines]
    else:
        array = getScramble()

    if VERBOSE:
        print("--- INPUT ---")
        print(tabulate(array, tablefmt="grid"))
    else:
        print(LEFT_BRACKET + "INPUT" + RIGHT_BRACKET)
        displayPuzzle(array)

    bnb = BranchAndBound(array)

    if VERBOSE:
        bnb.printTree()
        print("\nSolutions:")
        for i, path in enumerate(bnb.paths):
            print("Solution no." + str(i + 1))
            for node in path:
                print(node)
            print("This Solution Has " + str(len(path)) + " Moves")
```

Bab III
Hasil Pengujian

3.1. Known Case

Input:

known_test.txt

Tucil_3_Stima > test > known_test.txt

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

Ouput:

```
C:\Github\Tucil_3_Stima\src>python BranchAndBound.py known_test.txt
INPUT
```

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

```
-----  --
1      0
2      0
3      0
4      0
5      0
6      0
7      0
8      1
9      1
10     1
11     0
12     0
13     1
14     1
15     1
0      9
X      1
Total  16
-----  --
The Value Of Reachability is: 16
Hence the End State is Reachable
```

```
Single Best Solution:
INITIAL STATE
```

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

UP

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

LEFT

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

UP

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

```
Total Nodes: 14
Time Taken : 0.000ms
```

3.2. Reachable

3.2.1. First Case

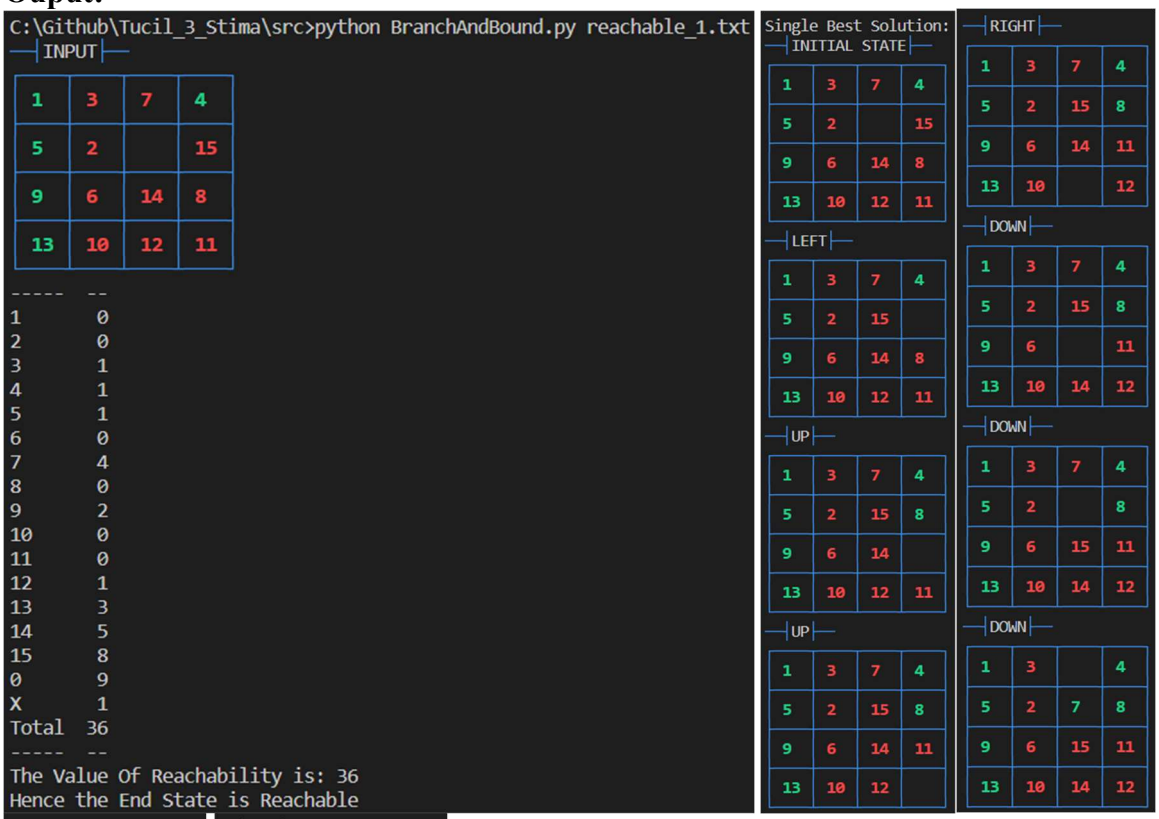
Input:

reachable_1.txt

Tucil_3_Stima > test > reachable_1.txt

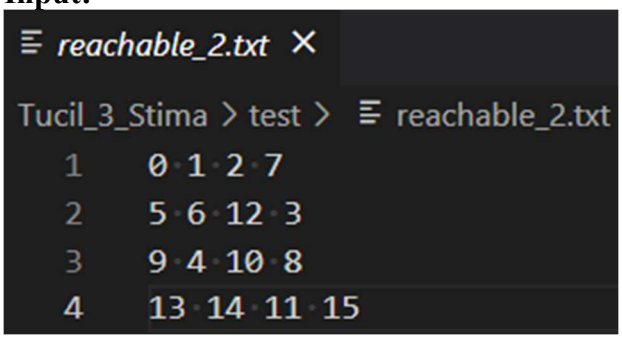
```
1 3 7 4
5 2 0 15
9 6 14 8
13 10 12 11
```

Ouput:

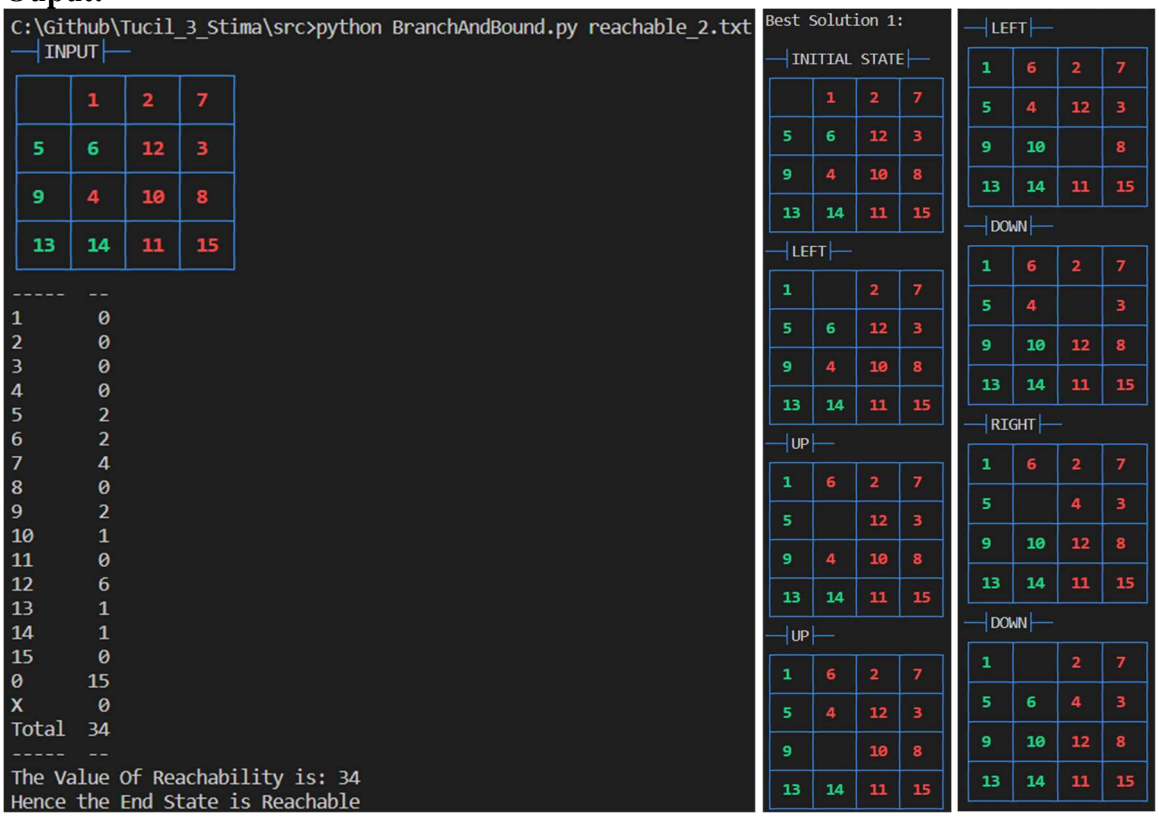

C:\Github\Tucil_3_Stima\src>python BranchAndBound.py reachable_1.txt
INPUT
1 3 7 4
5 2 15
9 6 14 8
13 10 12 11
1 0
2 0
3 1
4 1
5 1
6 0
7 4
8 0
9 2
10 0
11 0
12 1
13 3
14 5
15 8
0 9
X 1
Total 36
The Value Of Reachability is: 36
Hence the End State is Reachable
Single Best Solution:
INITIAL STATE
1 3 7 4
5 2 15
9 6 14 8
13 10 12 11
LEFT
1 3 7 4
5 2 15
9 6 14 8
13 10 12 11
UP
1 3 7 4
5 2 15 8
9 6 14
13 10 12 11
UP
1 3 7 4
5 2 15 8
9 6 14 11
13 10 12
RIGHT
1 3 7 4
5 2 15 8
9 6 14 11
13 10 12
DOWN
1 3 7 4
5 2 15 8
9 6 11
13 10 14 12
DOWN
1 3 7 4
5 2 8
9 6 15 11
13 10 14 12
DOWN
1 3 4
5 2 7 8
9 6 15 11
13 10 14 12
RIGHT
1 3 4
5 2 7 8
9 6 15 11
13 10 14 12
UP
1 2 3 4
5 7 8
9 6 15 11
13 10 14 12
UP
1 2 3 4
5 6 7 8
9 15 11
13 10 14 12
UP
1 2 3 4
5 6 7 8
9 10 15 11
13 14 12
LEFT
1 2 3 4
5 6 7 8
9 10 15 11
13 14 12
DOWN
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
LEFT
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
Total Nodes: 1126
Time Taken : 62.026ms

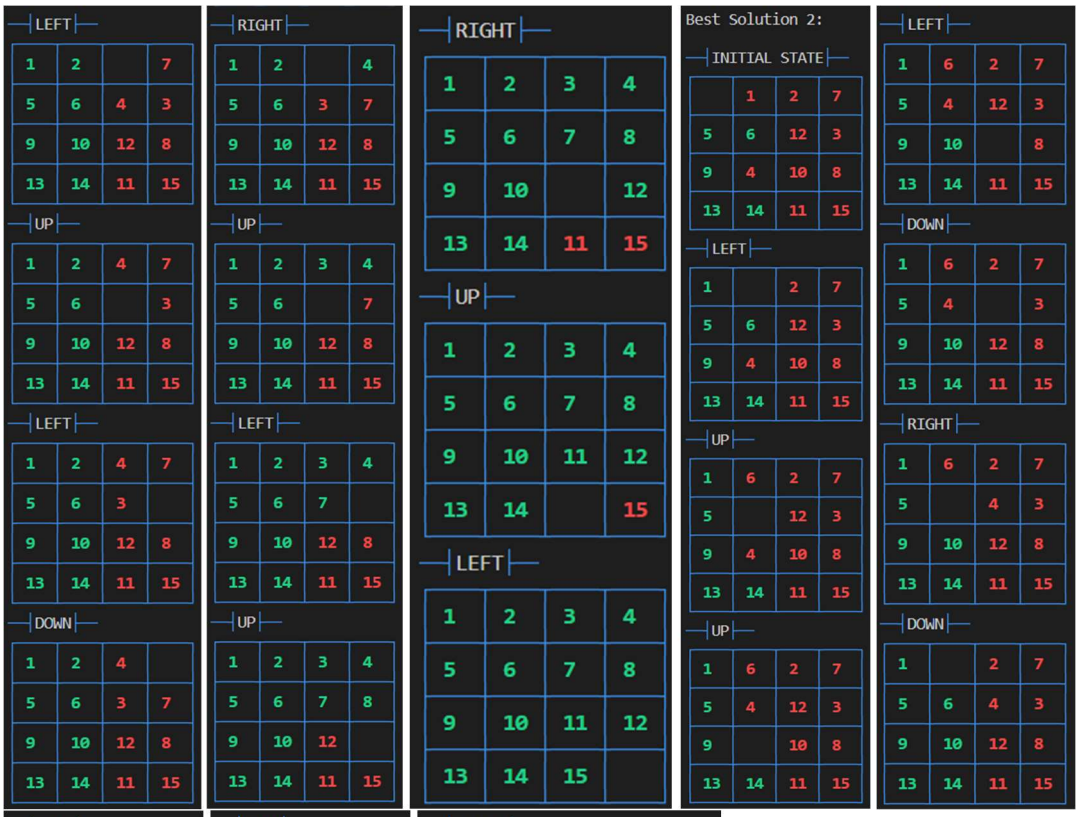
3.2.2. Second Case

Input:



Ouput:

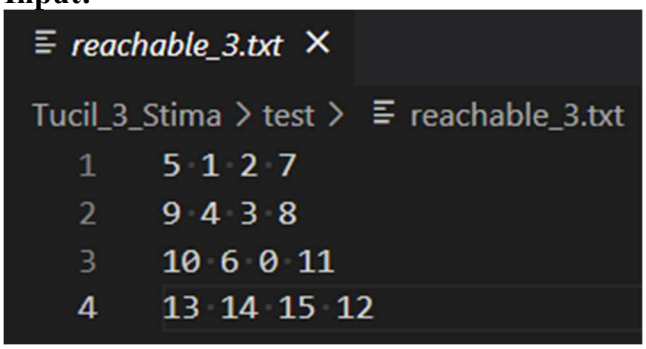



LEFT
1 2 7
5 6 4 3
9 10 12 8
13 14 11 15
UP
1 2 4 7
5 6 3
9 10 12 8
13 14 11 15
LEFT
1 2 4 7
5 6 3
9 10 12 8
13 14 11 15
DOWN
1 2 4
5 6 3 7
9 10 12 8
13 14 11 15
RIGHT
1 2 4
5 6 3 7
9 10 12 8
13 14 11 15
UP
1 2 3 4
5 6 7
9 10 12 8
13 14 11 15
LEFT
1 2 3 4
5 6 7
9 10 12 8
13 14 11 15
UP
1 2 3 4
5 6 7 8
9 10 12
13 14 11 15
RIGHT
1 2 3 4
5 6 7 8
9 10 12
13 14 11 15
UP
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
LEFT
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
Best Solution 2:
INITIAL STATE
1 2 7
5 6 12 3
9 4 10 8
13 14 11 15
LEFT
1 2 7
5 6 12 3
9 4 10 8
13 14 11 15
UP
1 6 2 7
5 12 3
9 4 10 8
13 14 11 15
UP
1 6 2 7
5 4 12 3
9 10 8
13 14 11 15
LEFT
1 6 2 7
5 4 12 3
9 10 8
13 14 11 15
DOWN
1 6 2 7
5 4 3
9 10 12 8
13 14 11 15
RIGHT
1 6 2 7
5 4 3
9 10 12 8
13 14 11 15
DOWN
1 2 7
5 6 4 3
9 10 12 8
13 14 11 15

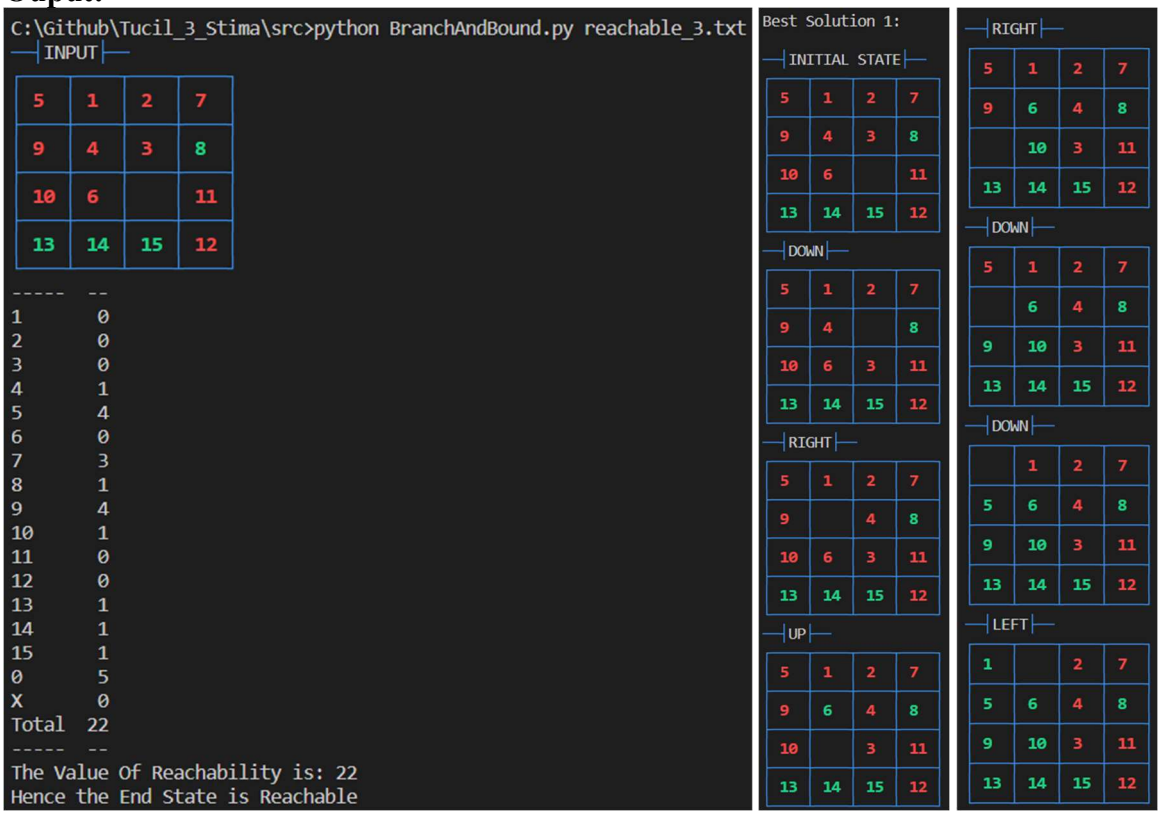

LEFT
1 2 7
5 6 4 3
9 10 12 8
13 14 11 15
LEFT
1 2 7
5 6 4 3
9 10 12 8
13 14 11 15
UP
1 2 7 3
5 6 4
9 10 12 8
13 14 11 15
RIGHT
1 2 7 3
5 6 4
9 10 12 8
13 14 11 15
DOWN
1 2 3
5 6 7 4
9 10 12 8
13 14 11 15
LEFT
1 2 3
5 6 7 4
9 10 12 8
13 14 11 15
UP
1 2 3 4
5 6 7
9 10 12 8
13 14 11 15
UP
1 2 3 4
5 6 7 8
9 10 12
13 14 11 15
RIGHT
1 2 3 4
5 6 7 8
9 10 12
13 14 11 15
UP
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
LEFT
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
Total Nodes: 3091
Time Taken : 385.997ms

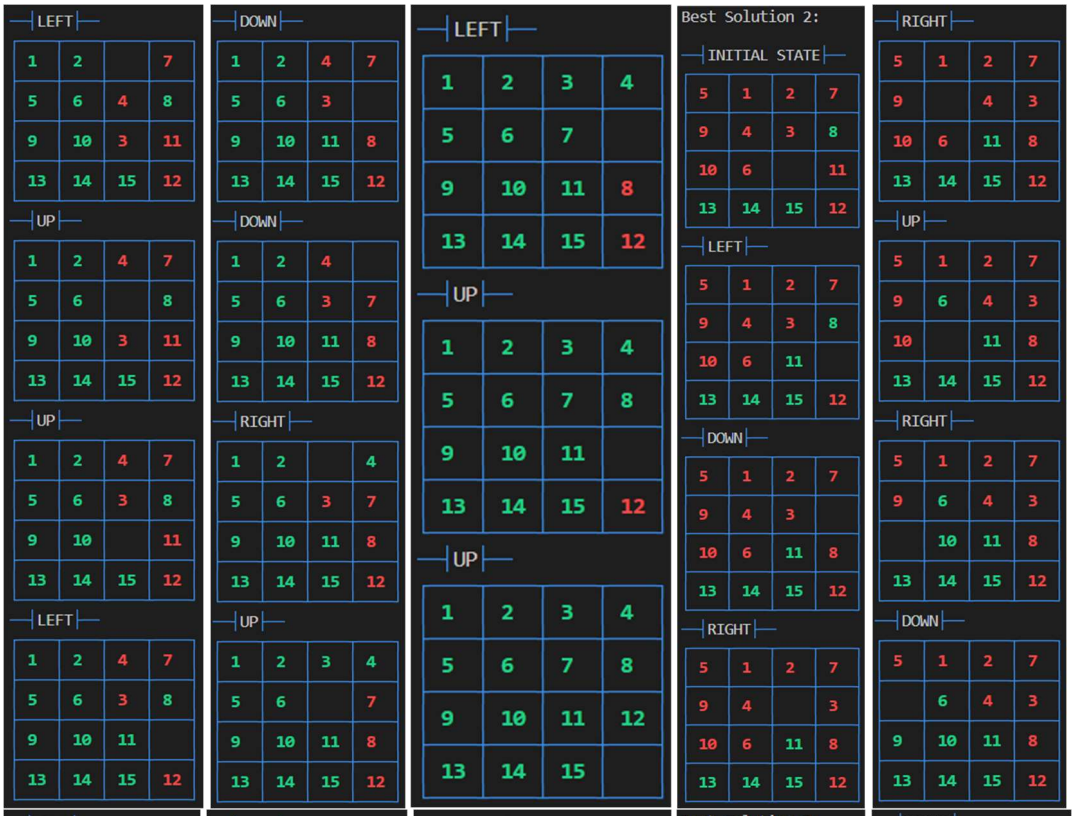
3.2.3. Third Case

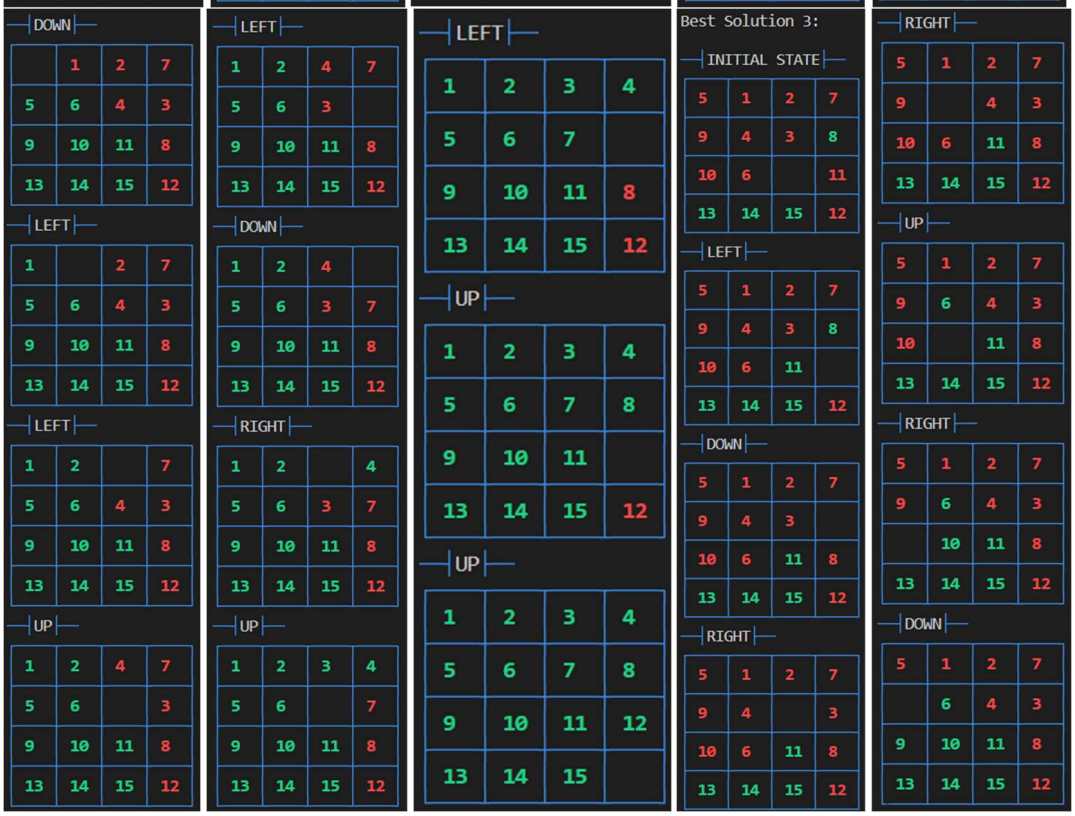
Input:



Ouput:




LEFT
1 2 7
5 6 4 8
9 10 3 11
13 14 15 12
UP
1 2 4 7
5 6 8
9 10 3 11
13 14 15 12
UP
1 2 4 7
5 6 3 8
9 10 11
13 14 15 12
LEFT
1 2 4 7
5 6 3 8
9 10 11
13 14 15 12
DOWN
1 2 4 7
5 6 3
9 10 11 8
13 14 15 12
DOWN
1 2 4
5 6 3 7
9 10 11 8
13 14 15 12
RIGHT
1 2 4
5 6 3 7
9 10 11 8
13 14 15 12
UP
1 2 3 4
5 6 7
9 10 11 8
13 14 15 12
LEFT
1 2 3 4
5 6 7
9 10 11 8
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
Best Solution 2:
INITIAL STATE
5 1 2 7
9 4 3 8
10 6 11
13 14 15 12
LEFT
5 1 2 7
9 4 3 8
10 6 11
13 14 15 12
DOWN
5 1 2 7
9 4 3
10 6 11 8
13 14 15 12
RIGHT
5 1 2 7
9 4 3
10 6 11 8
13 14 15 12
RIGHT
5 1 2 7
9 4 3
10 6 11 8
13 14 15 12
UP
5 1 2 7
9 6 4 3
10 11 8
13 14 15 12
RIGHT
5 1 2 7
9 6 4 3
10 11 8
13 14 15 12
DOWN
5 1 2 7
6 4 3
9 10 11 8
13 14 15 12

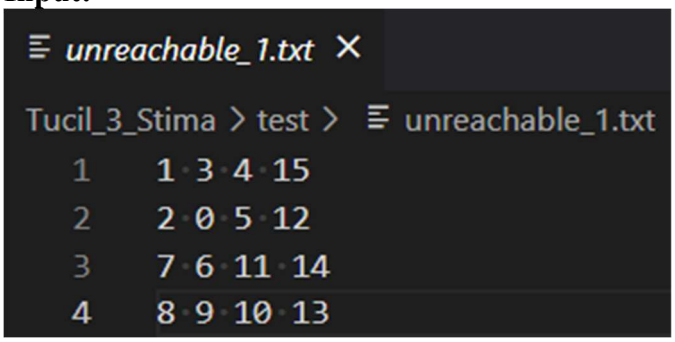

DOWN
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
LEFT
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
LEFT
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
UP
1 2 4 7
5 6 3
9 10 11 8
13 14 15 12
LEFT
1 2 4 7
5 6 3
9 10 11 8
13 14 15 12
DOWN
1 2 4
5 6 3 7
9 10 11 8
13 14 15 12
RIGHT
1 2 4
5 6 3 7
9 10 11 8
13 14 15 12
UP
1 2 3 4
5 6 7
9 10 11 8
13 14 15 12
LEFT
1 2 3 4
5 6 7
9 10 11 8
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
Best Solution 3:
INITIAL STATE
5 1 2 7
9 4 3 8
10 6 11
13 14 15 12
LEFT
5 1 2 7
9 4 3 8
10 6 11
13 14 15 12
DOWN
5 1 2 7
9 4 3
10 6 11 8
13 14 15 12
RIGHT
5 1 2 7
9 4 3
10 6 11 8
13 14 15 12
RIGHT
5 1 2 7
9 4 3
10 6 11 8
13 14 15 12
UP
5 1 2 7
9 6 4 3
10 11 8
13 14 15 12
RIGHT
5 1 2 7
9 6 4 3
10 11 8
13 14 15 12
DOWN
5 1 2 7
6 4 3
9 10 11 8
13 14 15 12


DOWN
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
LEFT
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
LEFT
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
LEFT
1 2 7
5 6 4 3
9 10 11 8
13 14 15 12
UP
1 2 7 3
5 6 4
9 10 11 8
13 14 15 12
RIGHT
1 2 7 3
5 6 4
9 10 11 8
13 14 15 12
DOWN
1 2 3
5 6 7 4
9 10 11 8
13 14 15 12
LEFT
1 2 3
5 6 7 4
9 10 11 8
13 14 15 12
UP
1 2 3 4
5 6 7
9 10 11 8
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
UP
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
Total Nodes: 3009
Time Taken : 460.043ms

3.3. Unreachable

3.3.1. First Case

Input:



Ouput:

```
C:\Github\Tucil_3_Stima\src>python BranchAndBound.py unreachable_1.txt
──┤INPUT├──
```

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

```
-----  --
1       0
2       0
3       1
4       1
5       0
6       0
7       1
8       0
9       0
10      0
11      3
12      6
13      0
14      4
15     11
0      10
X       0
Total  37
-----  --
The Value Of Reachability is: 37
Hence the End State is Unreachable

End State Unreachable, Exiting...
```

3.3.2. Second Case

Input:

unreachable_2.txt

Tucil_3_Stima > test > unreachable_2.txt

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

Ouput:

```
C:\Github\Tucil_3_Stima\src>python BranchAndBound.py unreachable_2.txt
┤INPUT├
```

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

```
-----  -
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     1
0      3
X      1
Total  5
-----  -
The Value Of Reachability is: 5
Hence the End State is Unreachable

End State Unreachable, Exiting...
```

Bab IV
Kesimpulan

4.1. Kesimpulan

- Persoalan 15-Puzzle dapat diselesaikan dengan algoritma Branch and Bound.
- Solusi yang dihasilkan adalah solusi optimal karena cost bersifat admissible

4.2. Catatan

- Berkas teks terlampir pada repository github dengan format "{nama_file_masukan}.txt" pada folder test.
- Luaran yang lebih lengkap terdapat pada repository github dengan format "{nama_file_masukan}.out" pada folder test. Terdapat informasi tambahan urutan pembangkitan simpul, visualisasi pohon ruang simpul, dan Solusi dalam bentuk Node.

4.3. Tabel Kelengkapan

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		✓

4.4. Link Penting

Link Github:
https://github.com/nart4hire/Tucil_3_Stima