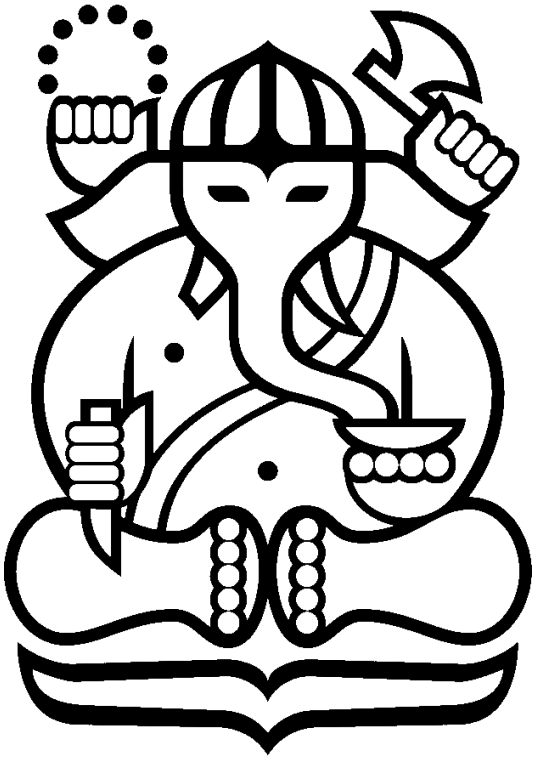
Laporan Tugas Kecil 3 IF2211 Strategi Algoritma
Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound



Disusun oleh:
13520110 - Farrel Ahmad

2 April 2022

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022

Daftar Isi

Algoritma Branch and Bound pada Penyelesaian 15-Puzzle **3**

Input/Output Program **5**

Checklist **8**

Kode Program **9**

Test Case **18**

Daftar Pustaka **19**

Algoritma *Branch and Bound* pada Penyelesaian 15-Puzzle

Algoritma branch and bound adalah algoritma yang digunakan untuk persoalan optimasi (meminimalkan/memaksimalkan). Algoritma ini sebenarnya mirip dengan BFS hanya saja dikombinasikan dengan *least cost search*. Secara akses iterasi mirip seperti BFS, akan tetapi pemilihan simpul untuk iterasi berikutnya berdasarkan cost yang diketahui, misal berdasarkan cost terkecil atau terbesar tergantung persoalan. Setiap simpul memiliki taksiran nilai cost c(i). c(i) adalah cost dengan taksiran tertentu yang termurah ke simpul tujuan melalui simpul i.

Algoritma branch and bound dapat digunakan untuk penyelesaian 15-Puzzle. Cost pada setiap simpul berdasarkan rumus berikut

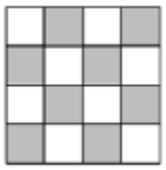
$$c(i) = f(i) + g(i)$$

Dengan c(i) adalah cost pada simpul i, f(i) adalah cost dari simpul awal ke simpul i dan g(i) adalah perkiraan cost simpul dari simpul i ke simpul tujuan. Pada persoalan 15 Puzzle, taksiran f(i) adalah kedalaman dari tree dari simpul awal ke simpul i dan taksiran g(i) adalah jumlah ubin tidak kosong yang posisinya tidak sesuai dengan posisi akhir.

Program penyelesaian 15 puzzle ini dalam branch and bound ini diimplementasikan dalam bahasa Python. Class Solver adalah suatu kelas yang dapat menyelesaikan 15 puzzle. Kelas tersebut memiliki atribut matriks, states yang merupakan priority queue (berdasarkan cost c) dari tuple [nomor node, state matriks, cost c, cost f, perintah], search log (state matriks apa saja yang pernah diperiksa), start time (waktu mulai), stop time (waktu selesai). Method yang dimiliki salah satunya adalah method solve, yaitu untuk menyelesaikan dan menampilkan step 15 puzzle.

Algoritma branch and bound yang diimplementasikan adalah sebagai berikut:

1. Input matriks ke program (lewat manual user input, file, ataupun random).
2. Menghitung jumlah Sigma(Kurang(i)) + X kemudian mengecek apakah hasilnya genap atau ganjil. X bernilai 1 jika matriks kosong (nomor 16) berada pada posisi di arsir dan bernilai 0 di luar arsir. Jika hasil Sigma(Kurang(i)) + X adalah genap maka reachable, jika ganjil maka tidak reachable (tidak ada solusi). Jika tidak reachable akan menampilkan pesan “Solution is not reachable !”



Gambar 1 Posisi Arsir

3. Jika reachable maka melakukan while loop dengan kondisi selama posisi matriks sekarang belum mencapai solusi.

4. Mengecek apakah matriks dapat bergerak ke atas dan prev_move bukan ke bawah. Jika tidak lanjut ke langkah berikutnya. Jika iya (membangunkan simpul) pindahkan elemen kosong (atau nomor 16 dalam program) ke atas untuk sementara kemudian mengecek apakah state matriks sekarang sebelumnya sudah dicek. Jika sudah, tidak dimasukan ke dalam states priority queue, jika belum maka masukan matriks ke states queue beserta informasi cost c, cost f, dan prev_move yaitu ke atas. Pindahkan nomor 16 ke bawah ke posisi awalnya.
5. Mengecek apakah matriks dapat bergerak ke bawah dan prev_move bukan ke atas. Jika tidak lanjut ke langkah berikutnya. Jika iya (membangunkan simpul) pindahkan elemen kosong (atau nomor 16 dalam program) ke bawah untuk sementara kemudian mengecek apakah state matriks sekarang sebelumnya sudah dicek. Jika sudah, tidak dimasukan ke dalam states priority queue, jika belum maka masukan ke states queue beserta informasi cost c, cost f, dan prev_move yaitu ke bawah. Pindahkan nomor 16 ke atas ke posisi awalnya.
6. Mengecek apakah matriks dapat bergerak ke kiri dan prev_move bukan ke kanan. Jika tidak lanjut ke langkah berikutnya. Jika iya (membangunkan simpul) pindahkan elemen kosong (atau nomor 16 dalam program) ke kiri untuk sementara kemudian mengecek apakah state matriks sekarang sebelumnya sudah dicek. Jika sudah, tidak dimasukan ke dalam states priority queue, jika belum maka masukan ke states queue beserta informasi cost c, cost f, dan prev_move yaitu ke kiri. Pindahkan nomor 16 ke kanan ke posisi awalnya.
7. Mengecek apakah matriks dapat bergerak ke kanan dan prev_move bukan ke kiri. Jika tidak lanjut ke langkah berikutnya. Jika iya (membangunkan simpul) pindahkan elemen kosong (atau nomor 16 dalam program) ke kanan untuk sementara kemudian mengecek apakah state matriks sekarang sebelumnya sudah dicek. Jika sudah, tidak dimasukan ke dalam states priority queue, jika belum maka masukan ke states queue beserta informasi cost c, cost f, dan prev_move yaitu ke kiri. Pindahkan nomor 16 ke kiri ke posisi awalnya.
8. Mengambil elemen pertama states priority queue karena merupakan simpul dengan cost terkecil.
9. Jika matriks pada state tersebut sudah diperiksa (ada di list search log) maka buang simpul tersebut dan ambil elemen berikutnya. Lakukan hingga elemen state dengan cost terkecil belum diperiksa.
10. Setelah diambil, pop elemen state pada priority queue yang sudah diambil
11. Elemen yang diambil adalah cost terkecil sehingga menjadi state matriks berikutnya
12. Selama belum mencapai matriks solusi, ulangi langkah 4-11.
13. Setelah ditemukan, tampilkan informasi akhir dari matriks, yaitu langkah menuju solusi, dan waktu yang diperlukan untuk menemukan solusi dalam milisekon (ms).

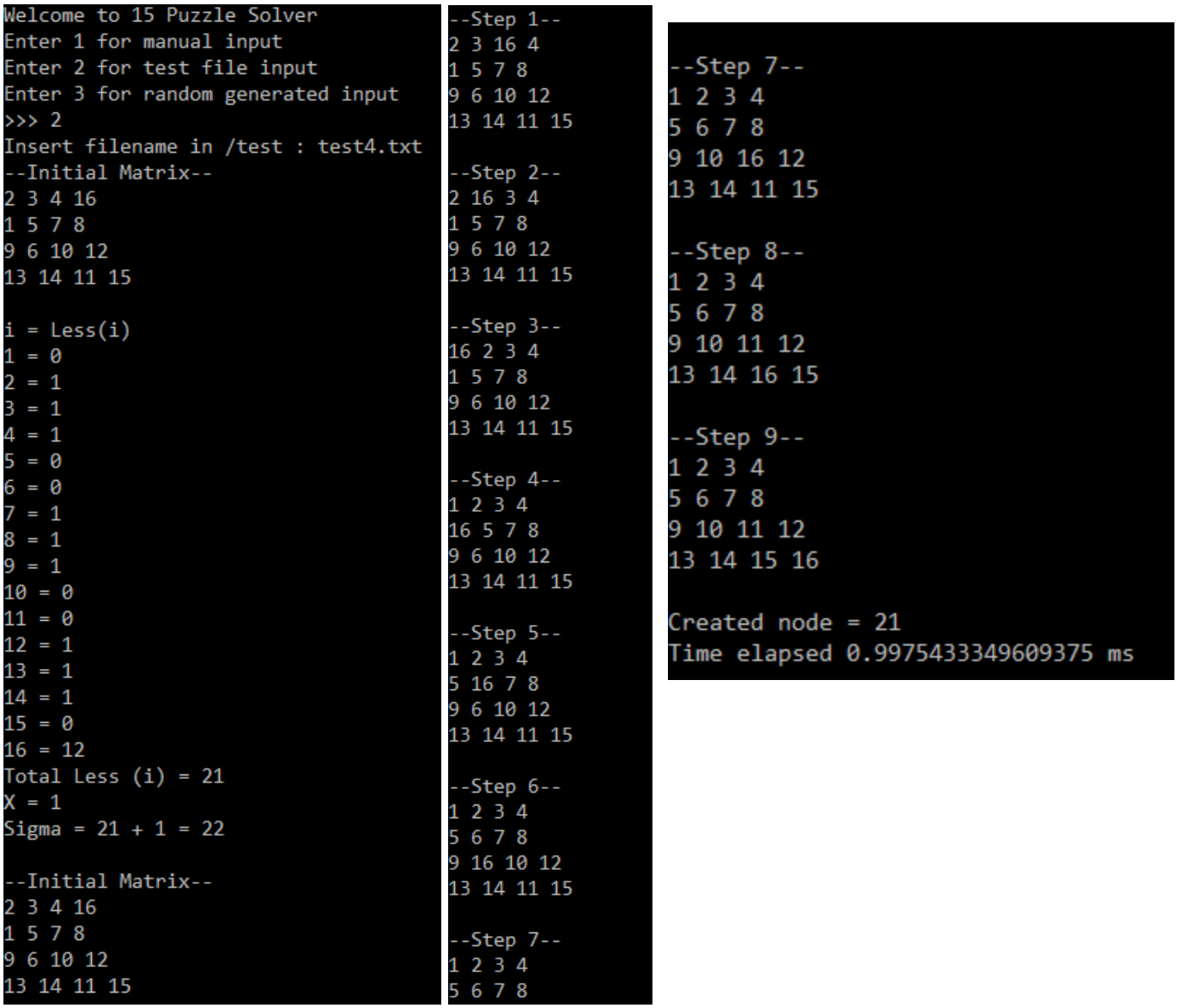
Input/Output Program

```
Welcome to 15 Puzzle Solver
Enter 1 for manual input
Enter 2 for test file input
Enter 3 for random generated input
>>> 1
4 2 1 3
8 7 6 5
9 16 11 12
13 14 10 15
--Initial Matrix--
4 2 1 3
8 7 6 5
9 16 11 12
13 14 10 15

i = Less(i)
1 = 0
2 = 1
3 = 0
4 = 3
5 = 0
6 = 1
7 = 2
8 = 3
9 = 0
10 = 0
11 = 1
12 = 1
13 = 1
14 = 1
15 = 0
16 = 6
Total Less (i) = 20
X = 1
Sigma = 20 + 1 = 21

Solution is not reachable !
```

Gambar 2 Input Output Program Tidak Ada Solusi



Gambar 3 Input Output Solusi Ditemukan

Checklist

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		✓

Kode Program

LINK GITHUB : https://github.com/farrel-a/Tucil3_13520110

Solver.py

```
import os
import platform
from copy import deepcopy
import random
import time

class Puzzle_15_Solver:
    # CTOR
    def __init__(self):
        # PRIVATE ATTRIBUTES
        self._matrix = [
                        [1,2,3,4],
                        [5,6,7,8],
                        [9,10,11,12],
                        [13,14,15,16]
                        ]
        self._states = []
        self._searchLog = []
        self._startTime = 0.0
        self._stopTime = 0.0

    # PRIVATE METHODS
    def _validMatrix(self, matrixInput):
        # check whether matrix is valid 15-puzzle matrix
        arr = []
        if (len(matrixInput)) != 4:
            return False
        else:
            for i in range(len(matrixInput)):
                if len(matrixInput[i]) != 4:
                    return False
                else:
                    for a in matrixInput[i]:
                        arr.append(a)
            arr.sort()
            for i in range(16):
                if arr[i] != i+1:
                    return False
            return True

    def _getIndex(self, a):
        # get index of element a in self._matrix
        for i in range(4):
            for j in range(4):
                if (self._matrix[i][j] == a):
                    return i, j
        raise Exception("Index not found")
```

```
    def _getIndexMatInput(self, mat, a):
        # get index from a given matrix input
        for i in range(4):
            for j in range(4):
                if (mat[i][j] == a):
                    return i, j
        raise Exception("Index not found")

    def _less(self, a):
        # for each b where position(b) > position(a) but value(b) < value(a)
        result = 0
        i, j = self._getIndex(a)
        j+=1
        while(i < 4):
            while (j < 4):
                if (self._matrix[i][j] < a):
                    result += 1
                j += 1
            i += 1
            j = 0
        return result

    def _getAllLess(self, show = False):
        # get all _less in self._matrix
        result = 0
        a = 1
        arr = []
        for i in range(4):
            for j in range(4):
                less = self._less(self._matrix[i][j])
                if show:
                   arr.append([self._matrix[i][j], less])
                result += less
                a += 1
        if show:
            arr.sort(key = lambda x : x[0])
            for a in arr:
                print(f"{a[0]} = {a[1]}")

        return result

    def _getX(self):
        # get X from self._matrix
        X = 0
        idx_16i, idx_16j = self._getIndex(16)
        if ((idx_16i == 0 or idx_16i == 2) and (idx_16j == 1 or idx_16j == 3)):
            X = 1
        elif ((idx_16i == 1 or idx_16i == 3) and (idx_16j == 0 or idx_16j == 2)):
            X = 1
        return X
```

```
    def _reachAble(self):
        # Check whether solution is reachable
        all_less = self._getAllLess()
        X = self._getX()
        total = all_less + X
        # if even then reachable
        return total%2 == 0

    def _gCost(self):
        # compute g cost of current self._matrix
        a = 1
        cost = 0
        for i in range(4):
            for j in range(4):
                if (self._matrix[i][j] != 16 and self._matrix[i][j] != a):
                    cost += 1
                a += 1
        return cost

    def _isTarget(self):
        # check whether current self._matrix is solution
        a = 1
        for i in range(4):
            for j in range(4):
                if (self._matrix[i][j] != a):
                    return False
                a += 1
        return True

    def _ableMoveUp(self):
        # check whether able to move up
        i, _ = self._getIndex(16)
        return True if i-1 >= 0 else False

    def _ableMoveDown(self):
        # check whether able to move down
        i, _ = self._getIndex(16)
        return True if i+1 <= 3 else False

    def _ableMoveRight(self):
        # check whether able to move right
        _, j = self._getIndex(16)
        return True if j+1 <= 3 else False

    def _ableMoveLeft(self):
        # check whether able to move left
        _, j = self._getIndex(16)
        return True if j-1 >= 0 else False

    def _moveUp(self):
        if (self._ableMoveUp):
            i, j = self._getIndex(16)
```

```
            temp = self._matrix[i-1][j]
            self._matrix[i-1][j] = 16
            self._matrix[i][j] = temp
        else:
            print("Unable to move up")

    def _moveDown(self):
        if (self._ableMoveDown):
            i, j = self._getIndex(16)
            temp = self._matrix[i+1][j]
            self._matrix[i+1][j] = 16
            self._matrix[i][j] = temp
        else:
            print("Unable to move down")

    def _moveLeft(self):
        if (self._ableMoveLeft):
            i, j = self._getIndex(16)
            temp = self._matrix[i][j-1]
            self._matrix[i][j-1] = 16
            self._matrix[i][j] = temp
        else:
            print("Unable to move left")

    def _moveRight(self):
        if (self._ableMoveRight):
            i, j = self._getIndex(16)
            temp = self._matrix[i][j+1]
            self._matrix[i][j+1] = 16
            self._matrix[i][j] = temp
        else:
            print("Unable to move right")

    def _displayStep(self):
        # display solution step
        step = []
        self._searchLog.reverse()
        prev_i, prev_j = self._getIndexMatInput(self._searchLog[0], 16)
        step.append(deepcopy(self._searchLog[0]))
        for i in range(1, len(self._searchLog)):
            curr_i, curr_j = self._getIndexMatInput(self._searchLog[i], 16)
            if (abs(curr_i-prev_i) == 1 and abs(curr_j - prev_j) == 0):
                step.append(deepcopy(self._searchLog[i]))
            elif(abs(curr_i-prev_i) == 0 and abs(curr_j - prev_j) == 1):
                step.append(deepcopy(self._searchLog[i]))
            prev_i, prev_j = curr_i, curr_j
        self._searchLog.reverse()
        step.reverse()
        step_num = 1
        for mat in step:
            self._matrix = mat
            print(f"--Step {step_num}--")
```

```
            self.displayMatrix()
            print()
            step_num += 1

    # PUBLIC METHODS
    def solve(self):
        # solve self._matrix
        # initial information
        print("--Initial Matrix--")
        self.displayMatrix()
        print()
        print("i = Less(i)")
        print(f"Total Less (i) = {self._getAllLess(show = True)}")
        print(f"X = {self._getX()}")
        print(f"Sigma = {self._getAllLess()} + {self._getX()} = {self._getAllLess() +
self._getX()}")
        print()

        if (self._reachAble()):
            print("--Initial Matrix--")
            self.displayMatrix()
            print()

            # algorithm start (start time)
            self._startTime = time.time()
            fCost = 0
            created_node = 1
            created_nodes = []
            prev_move = ""
            while(not(self._isTarget())):
                mat_states = [states[1] for states in self._states]
                if (self._ableMoveUp() and prev_move != "down"):
                    # able to move up and prev_move not "down", move up and add to queue in
self._states
                    self._moveUp()
                    if (self._matrix not in self._searchLog and self._matrix not in
mat_states):
                        gCost = self._gCost()
                        cCost = fCost + 1 + gCost
                        created_node+=1
                        created_nodes.append(created_node)
                        mat = deepcopy(self._matrix)
                        self._states.append([created_node, mat, cCost, fCost+1, "up"])
                        self._states.sort(key = lambda x : x[2])
                        self._moveDown()
                    else:
                        self._moveDown()

                if (self._ableMoveDown() and prev_move != "up"):
                    # able to move down and prev_move not "up", move down add to queue in
self._states
                    self._moveDown()
```

```
                    if (self._matrix not in self._searchLog and self._matrix not in
mat_states):
                        gCost = self._gCost()
                        cCost = fCost + 1 + gCost
                        created_node+=1
                        created_nodes.append(created_node)
                        mat = deepcopy(self._matrix)
                        self._states.append([created_node, mat, cCost, fCost+1, "down"])
                        self._states.sort(key = lambda x : x[2])
                        self._moveUp()
                    else:
                        self._moveUp()

                if (self._ableMoveLeft() and prev_move != "right"):
                    # able to move left and prev_move not "right", move left and add to queue
in self._states
                    self._moveLeft()
                    if (self._matrix not in self._searchLog and self._matrix not in
mat_states):
                        gCost = self._gCost()
                        cCost = fCost + 1 + gCost
                        created_node+=1
                        created_nodes.append(created_node)
                        mat = deepcopy(self._matrix)
                        self._states.append([created_node, mat, cCost, fCost+1, "left"])
                        self._states.sort(key = lambda x : x[2])
                        self._moveRight()
                    else:
                        self._moveRight()

                if (self._ableMoveRight() and prev_move != "left"):
                    # able to move right and prev_move not "left", move right and add to queue
in self._states
                    self._moveRight()
                    if (self._matrix not in self._searchLog and self._matrix not in
mat_states):
                        gCost = self._gCost()
                        cCost = fCost + 1 + gCost
                        created_node+=1
                        created_nodes.append(created_node)
                        mat = deepcopy(self._matrix)
                        self._states.append([created_node, mat, cCost, fCost+1, "right"])
                        self._states.sort(key = lambda x : x[2])
                        self._moveLeft()
                    else:
                        self._moveLeft()

                cmd_arr = self._states[0]
                fCost = cmd_arr[3]
                prev_move = cmd_arr[4]

                # if state already checked
```

```
            while (cmd_arr[1] in self._searchLog):
                self._states.pop(0)
                cmd_arr = deepcopy(self._states[0])
                fCost = cmd_arr[3]
                prev_move = cmd_arr[4]
            if (cmd_arr[1] in self._searchLog):
                raise Exception("duplicate")
            self._states.pop(0)
            self._matrix = deepcopy(cmd_arr[1])
            self._searchLog.append(deepcopy(cmd_arr[1]))

        # algorithm finish (stop time)
        self._stopTime = time.time()

        # display step and time elapsed
        if (len(self._searchLog) != 0):
            self._displayStep()
        print(f"Created node = {created_node}")
        timeElapsed = self._stopTime - self._startTime
        timeElapsedMS = timeElapsed*1000.00
        print(f"Time elapsed {timeElapsedMS} ms")

    else:
        print("Solution is not reachable !")

def setMatrix(self, matrixInput):
    # set self._matrix to a given matrixInput
    if (self._validMatrix(matrixInput)):
        self._matrix = matrixInput
    else:
        print("Invalid Matrix Input")

def displayMatrix(self):
    # display self._matrix
    for i in range(len(self._matrix)):
        for j in range(len(self._matrix[i])):
            print(self._matrix[i][j], end=" ")
        print()

def readMatrixFromInput(self):
    # read matrix from user keyboard input
    mat = []
    for i in range(4):
        inputArr = input().split()
        arr = [int(e) for e in inputArr]
        mat.append(arr)
    self.setMatrix(mat)

def readMatrixFromFile(self):
    # read matrix from file in /test
    OS = platform.system()
    filename = input("Insert filename in /test : ")
```

```
        if (OS == "Windows"):
            path = f"..\\test\\{filename}"
        elif (OS == "Linux" or OS == "Darwin"):
            path = f"../test/{filename}"
        else:
            raise Exception("Program only supports Windows, Linux, and Mac OS")

        if os.path.isfile(path):
            f = open(path, "r")
            mat = [[int(n) for n in line.split(" ")] for line in f]
            self.setMatrix(mat)
        else:
            raise Exception("File not found !")

    def readMatrixRandom(self):
        # read 15 puzzle matrix from random generated number
        list_num = [i+1 for i in range(16)]
        for i in range(4):
            for j in range(4):
                num = random.choice(list_num)
                self._matrix[i][j] = num
                list_num.remove(num)
```

main.py

```
from Solver import Puzzle_15_Solver

if __name__ == "__main__":
    # Instantiate puzzle 15 solver object
    solver = Puzzle_15_Solver()

    # Main Menu
    print("Welcome to 15 Puzzle Solver")
    print("Enter 1 for manual input")
    print("Enter 2 for test file input")
    print("Enter 3 for random generated input")
    try:
        user_input = int(input(">>> "))
        if (user_input == 1):
            solver.readMatrixFromInput()
            solver.solve()
        elif (user_input == 2):
            solver.readMatrixFromFile()
            solver.solve()
        elif (user_input == 3):
            solver.readMatrixRandom()
            solver.solve()
        else:
            print("invalid input")
    except ValueError:
        print("invalid input")
```

Test Case

test1.txt (not solvable)

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

test2.txt (not solvable)

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

test3.txt (solvable)

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

test4.txt (solvable)

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

test5.txt (solvable)

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

Daftar Pustaka

Munir, Rinaldi. 2021. *Algoritma Branch & Bound (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf. Diakses pada 2 April 2022.

Python Documentation. 2022. *random*. https://docs.python.org/3/library/random.html. Diakses pada 2 April 2022.

Python Documentation. 2022. *os*. https://docs.python.org/3/library/os.html. Diakses pada 31 Maret 2022.

Python Documentation. 2022. *platform.* https://docs.python.org/3/library/platform.html. Diakses pada 31 Maret 2022