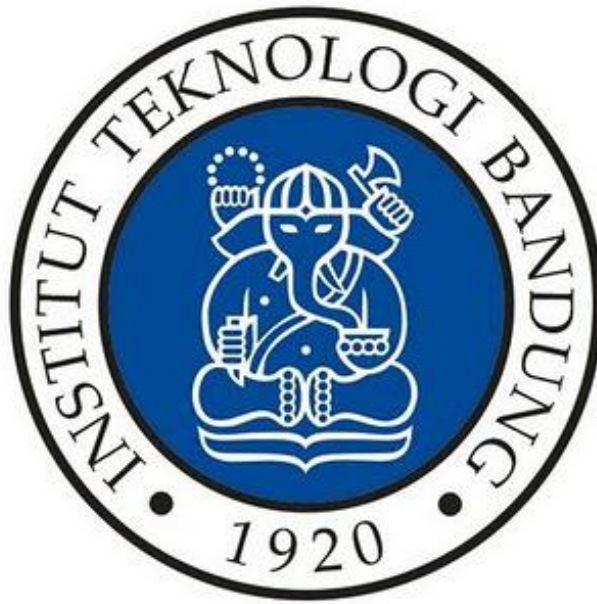


Tugas Kecil 3

**IF2211 Strategi Algoritma - Penyelesaian Perosalan 15-Puzzle dengan
Algoritma Branch and Bound**



Oleh

**Bayu Samudra
13520128**

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021**

Daftar Isi

Daftar Isi	2
Bab 1 Pendahuluan	3
Bab 2 Deskripsi Algoritma	4
Bab 3 Implementasi Program	6
Bab 4 Hasil Eksekusi Program	17
4.1. Instansiasi Masalah 1	17
4.2. Instansiasi Masalah 2	17
4.3. Instansiasi Masalah 3	19
4.4. Instansiasi Masalah 4	22
4.5. Instansiasi Masalah 5	26
Tautan Penting	28
Checklist Kemampuan Pustaka	28
Daftar Pustaka	30

Bab 1

Pendahuluan

15-Puzzle problem adalah permainan untuk menyusun kumpulan angka pada sebuah tabel hingga menuju titik akhir yang digunakan. Permainan akan dimulai dengan angka acak yang biasanya akan dicari jalan untuk menuju titik yang ditunjukkan oleh gambar 1 berikut:

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

Gambar 1.1 Tujuan akhir dari 15-Puzzle Problem

Sebuah instansiasi puzzle bisa saja tidak memiliki solusi untuk mencapai tujuan akhir. Untuk mendeteksi hal tersebut, perlu dihitung jumlah dari fungsi kurang dari setiap ubin tidak kosong.

Fungsi kurang didefinisikan sebagai jumlah ubin yang posisinya berada lebih besar dibandingkan dengan posisinya dan juga nilai ubin tersebut lebih besar dari ubin acuan. Jumlah tersebut dijumlahkan dengan paritas dari ubin kosong. Untuk menghitung paritas, perhatikan gambar 1.2.

Gambar 1.2. Posisi Paritas

Bila ubin kosong berada pada posisi ubin hitam, nilai paritas diberikan nilai 1, sebaliknya diberikan nilai 0.

Sebuah 15-puzzle dapat diselesaikan apabila jumlah fungsi kurang ditambah paritas merupakan nilai genap. Oleh karena itu, untuk mencegahnya pencarian tanpa solusi, dapat dilakukan validasi solusi puzzle.

Bab 2

Deskripsi Algoritma

Pada tugas ini, saya menggunakan algoritma *Branch and Bound*. Pencarian solusi dilakukan dengan langkah sebagai berikut

Langkah 1

Pada mulanya, perlu dilakukannya validasi apakah sebuah instansiasi masukan memiliki solusi. Hal tersebut dapat dilakukan dengan memeriksa nilai berikut

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

Fungsi kurang didefinisikan sebagaimana pada bab sebelumnya dan X merupakan paritas dari ubin kosong. Apabila nilai hasil di atas adalah genap, maka instansiasi memiliki solusi. Akan tetapi, untuk sebaliknya tidak. Untuk persoalan yang memiliki solusi maka dicarilah solusinya dengan melakukan langkah selanjutnya.

Langkah 2

Pada langkah ini, akan dibuat sebuah *PriorityQueue*. Aturan yang diterapkan adalah memprioritaskan sebuah node yang memiliki cost paling rendah. Apabila terdapat dua buah node yang memiliki cost yang sama, maka diprioritaskan node yang memiliki tingkatan lebih tinggi. Cost dari sebuah node didefinisikan sebagai jumlah kolom pada suatu instansiasi yang tidak sesuai dengan tujuan akhir.

Selain itu, perlu dibuat sebuah list yang menyimpan instansiasi yang telah diproses. List ini dibuat secara terurut berdasarkan instansiasi pada list tersebut. Sebuah instansiasi dikatakan lebih besar dari yang lainnya apabila terdapat pada posisi terkecil nilai yang lebih besar dari instansiasi lainnya. Pencarian sebuah adanya instansiasi dilakukan secara pencarian biner. Pemasukan data dilakukan dengan cara melakukan insertion sort.

Langkah 3

Apabila tipe data sudah siap, masukan instansiasi pembangkit (instansiasi awal yang merupakan input) ke dalam *PriorityQueue*. Selama solusi belum ditemukan, lakukan langkah berikut:

1. Ambil elemen head dari *PriorityQueue* sebut saja elemen tersebut adalah *e1*

2. Periksa apakah `el` telah berada pada list yang menyimpan instansiasi yang telah diproses? Bila sudah ada, proses kembali elemen head lainnya. Hal tersebut merupakan proses bounding terhadap elemen yang sudah pernah diproses.
3. Periksa apakah elemen `el` sudah merupakan solusi? Apabila merupakan solusi, maka loncat ke langkah 4.
4. Masukkan elemen `el` pada list penyimpan instansiasi yang telah diproses.
5. Lakukan ekspansi node dari elemen `el`. Hal ini dapat dilakukan dengan melakukan semua *legal move* pada posisi tersebut. *Legal move* didefinisikan sebagai berikut
 - a. Untuk *move UP*, ubin kosong harus tidak berada baris paling atas (baris indeks 0)
 - b. Untuk *move DOWN*, ubin kosong harus tidak berada pada baris paling bawah (baris indeks 3)
 - c. Untuk *move LEFT*, ubin kosong tidak boleh berada pada kolom paling kiri (kolom indeks 0)
 - d. Untuk *move RIGHT*, ubin kosong tidak boleh berada pada kolom paling kanan (kolom indeks 3)
6. Untuk setiap ekspansi yang valid, catat aksi yang dilakukan dan node tersebut dimasukkan ke dalam `PriorityQueue`. Untuk ekspansi yang tidak valid, dilakukan *bounded*.

Langkah 4

Tampilkan solusi pada tempat yang diinginkan dengan menampilkan daftar aksi yang dilakukan pada setiap move.

Bab 3

Implementasi Program

Berikut ini adalah implementasi dari ide pada bab 2 menggunakan bahasa python:

```
from datetime import datetime, timedelta
import sys

def main(filePath):
    file = open(filePath)

    contents = file.read()
    input = []

    for i in contents.split():
        if i == "-":
            input.append(16)
        else:
            input.append(int(i))

    print()

    table = Table(input)
    solver = Solver(table)
    start = datetime.now()
    try:
        print("\x1B[35mMatriks Input:\x1B[0m")
        printTable(table)
        print()

        solver.solve()

        print("\x1B[35mLangkah Penyelesaian:\x1B[0m\n")
        penjelajah = Node(table)
        cnt = 1
        for i in solver.getSolutions()[0].getMove():
            print("\x1B[36mLangkah\x1B[0m \x1B[33m%d\x1B[0m: \x1B[34m%s\x1B[0m" %
                  (cnt, i))
            penjelajah = penjelajah.move(i)
            printTable(penjelajah.getTable())
            print()
```

```

        cnt += 1

    end = datetime.now()
    time = end - start
    printTime(time)
    print("Jumlah Langkah : \x1B[33m%d\x1B[0m langkah" %
solver.getMinimalCost())
    print("Jumlah Node: \x1B[33m%d\x1B[0m node" % solver.getNodeNumber())
    print()

except UnsolveableException:
    print("Hasil Proses:")
    print("\x1B[31mSolusi tidak ditemukan\x1B[0m")
    print()

    end = datetime.now()
    time = end - start
    printTime(time)
finally:
    print("\x1B[35mData Instans:\x1B[0m")
    print("Jumlah Kurang(i) = \x1B[33m%d\x1B[0m" % table.fungsiKurang())
    print("Jumlah Kurang(i) + paritas = \x1B[33m%d\x1B[0m" %
table.solvePoint())

def printTable(table: Table):
    num = 0

    for i in table.getTable():
        if((num + 1) % 4 != 0):
            if i < 10:
                print(" %d" % i, end=" ")
            else:
                print("%d" % i, end=" ")
        else:
            if i < 10:
                print(" %d" % i)
            else:
                print("%d" % i)

    num += 1

```

```

def printTime(ns: timedelta):
    if ns.microseconds < 1000:
        print("⌚ Time : \x1B[33m%d\x1B[0m us" % (ns.microseconds))
    elif ns.microseconds < 1_000_000:
        print("⌚ Time : \x1B[33m%d\x1B[0m ms" % (ns.microseconds / 1000))
    else:
        print("⌚ Time : \x1B[33m%d\x1B[0m s" % (ns.seconds))

    print()

if __name__ == "__main__":
    print("""\x1B[32m
_____
/_ | ____|      | __ \ \      \      | |
| | | ____| |__ ) |    _   _   | | ____
| |__ \ \ ____| ____/ | | | _ / _ / | _ \ \
| |__ ) |      | | | | _ / / / / | | _ /
|_| ____/      |_| \ \__,_/_/_/_/_|_| \__|

\x1B[36mVersi 1.0.0\x1B[0m
""")
    if len(sys.argv) > 1:
        filePath = sys.argv[1]
        print("Membaca file \x1B[33m%s\x1B[0m" % filePath)
    else:
        filePath = input("Silahkan masukan file path : ")

    main(filePath)

class MoveException(Exception):
    def __init__(self, message: str, move: str) -> None:
        super().__init__(message)
        self.__move = move

    def getMove(self) -> str:
        return self.__move

class Node:
    def __init__(self, table: Table, move: list = []) -> None:
        self.__move = move

```



```

        self.__table = table
        self.__level = len(move)
def __eq__(self, __o: object) -> bool:
    return self.cost() == __o.cost() and self.getLevel() == __o.getLevel()

def __ge__(self, __o:object) -> bool:
    return self.cost() >= __o.cost() or \
        (self.cost() == __o.cost() and self.getLevel() <= __o.getLevel())

def __gt__(self, __o:object) -> bool:
    return self.cost() > __o.cost() or \
        (self.cost() == __o.cost() and self.getLevel() < __o.getLevel())

def __le__(self, __o:object) -> bool:
    return self.cost() <= __o.cost() or \
        (self.cost() == __o.cost() and self.getLevel() >= __o.getLevel())

def __lt__(self, __o:object) -> bool:
    return self.cost() < __o.cost() or \
        (self.cost() == __o.cost() and self.getLevel() > __o.getLevel())

def getTable(self) -> Table:
    return self.__table

def getLevel(self) -> int:
    return self.__level
def getMove(self) -> list:
    return self.__move
def cost(self) -> int:
    return self.__table.prediction() + self.getLevel()

def isSolution(self) -> bool:
    return self.__table.isSolution()
def moveUp(self):
    return Node(self.__table.toTop(), [*self.__move, "UP"])

def moveDown(self):
    return Node(self.__table.toBottom(), [*self.__move, "DOWN"])

def moveLeft(self):
    return Node(self.__table.toLeft(), [*self.__move, "LEFT"])

```

```

def moveRight(self):
    return Node(self.__table.toRight(), [*self.__move, "RIGHT"])

def move(self, move: str):
    if move == "UP":
        return self.moveUp()
    elif move == "DOWN":
        return self.moveDown()
    elif move == "LEFT":
        return self.moveLeft()
    elif move == "RIGHT":
        return self.moveRight()
    else:
        raise Exception("Move action unknown")

from decimal import Underflow

class PrioQueue:
    def __init__(self) -> None:
        self.__queue = []
    def push(self, a: Node):
        self.__queue.append(a)

        for i in range(len(self.__queue)-1, 0, -1):
            if self.__queue[i] < self.__queue[i-1]:
                self.__queue[i], self.__queue[i-1] = self.__queue[i-1],
self.__queue[i]
            else:
                break
    def pop(self) -> Node:
        if len(self.__queue) == 0:
            raise Underflow("PriorityQueue Underflow")

        res = self.__queue[0]
        self.__queue = self.__queue[1:]
        return res

    def filterQueue(self, cost):
        result = PrioQueue()

```

```

    for i in self.__queue:
        if i.cost() <= cost:
            result.push(i)

    return result

def isEmpty(self):
    return len(self.__queue) == 0

class Solver:
    def __init__(self, start: Table, all=False) -> None:
        self.__start = Node(start)
        self.__prio = PrioQueue()
        self.__state = StateSave()
        self.__prio.push(
            self.__start
        )
        self.__solution: list[Node] = []
        self.__costLimit = -1
        self.__nodeNumber = 1
        self.__allSolution = all

    def solve(self):
        if not self.__start.getTable().isSolveable():
            raise UnsolveableException(self.__start)

        while not self.__prio.isEmpty():
            currentState = self.__prio.pop()

            if currentState.isSolution():
                if self.__costLimit > currentState.cost():
                    self.__solution.clear()

                self.__solution.append(currentState)
                self.__costLimit = currentState.cost()

            if not self.__allSolution:
                return

            self.__prio = self.__prio.filterQueue(self.__costLimit)
            continue

```

```

elif self.__costLimit != -1 and \
    currentState.cost() >= self.__costLimit:
    continue

self.__state.addState(currentState.getTable())
moves = ["UP", "RIGHT", "DOWN", "LEFT"]

for i in moves:
    try:
        newNode = currentState.move(i)
        if self.__state.isStateExist(newNode.getTable()):
            continue
        self.__prio.push(newNode)
        self.__nodeNumber += 1
    except MoveException as e:
        pass
    except Exception as e:
        raise e

def getSolutions(self):
    return self.__solution

def getMinimalCost(self):
    return self.__costLimit

def getNodeNumber(self):
    return self.__nodeNumber

class StateSave:
def __init__(self) -> None:
    self.__list = []
def addState(self, table: Table):
    if self.isStateExist(table): return
    self.__list.append(table)
    for i in range(len(self.__list)-1, 0, -1):
        if self.__list[i] < self.__list[i-1]:
            self.__list[i], self.__list[i-1] = self.__list[i-1], self.__list[i]
        else:
            break
def isStateExist(self, table: Table):

```

```

l = 0
r = len(self.__list)

while(l < r):
    mid = (l+r) // 2

    if self.__list[mid] == table:
        return True
    elif self.__list[mid] > table:
        r = mid
    else:
        l = mid + 1

return False

def getLength(self):
    return len(self.__list)

class Table:
    def __init__(self, table: list) -> None:
        self.__table = []
        cnt = 0

        for i in table:
            self.__table.append(i)

            if i == 16:
                self.__pos = cnt

            cnt += 1
    def __eq__(self, __o: object) -> bool:
        return self.__table == __o.getTable()
    def __ge__(self, __o:object) -> bool:
        return self.__table >= __o.getTable()

    def __gt__(self, __o:object) -> bool:
        return self.__table > __o.getTable()

    def __le__(self, __o:object) -> bool:
        return self.__table <= __o.getTable()

```

```

def __lt__(self, __o:object) -> bool:
    return self.__table < __o.getTable()

def getTable(self) -> list:
    return self.__table
def fungsiKurang(self) -> int:
    kurang = 0

    for i in range(len(self.__table)):
        for j in range(i+1, len(self.__table)):
            if self.__table[i] > self.__table[j]:
                kurang += 1

    return kurang

def solvePoint(self) -> int:
    kurang = (self.__pos + ((self.__pos // 4) % 2)) % 2

    return kurang + self.fungsiKurang()

def isSolveable(self) -> bool:
    return (self.solvePoint() % 2) == 0
def prediction(self) -> int:
    score = 0
    for i in range(len(self.__table)):
        if i + 1 != self.__table[i] and self.__table[i] != 16:
            score += 1

    return score
def toLeft(self):
    j = self.__pos % 4

    if j == 0:
        raise MoveException("The empty slot is in the leftmost", "LEFT")

    cp = self.__table.copy()
    cp[self.__pos-1], cp[self.__pos] = cp[self.__pos], cp[self.__pos-1]

    return Table(cp)

```

```

def toRight(self):
    j = self.__pos % 4

    if j == 3:
        raise MoveException("The empty slot is in the rightmost", "RIGHT")

    cp = self.__table.copy()
    cp[self.__pos+1], cp[self.__pos] = cp[self.__pos], cp[self.__pos+1]

    return Table(cp)
def toTop(self):
    i = self.__pos // 4

    if i == 0:
        raise MoveException("The empty slot is in the top", "UP")

    cp = self.__table.copy()
    topIdx = self.__pos - 4

    cp[topIdx], cp[self.__pos] = cp[self.__pos], cp[topIdx]

    return Table(cp)

def toBottom(self):
    i = self.__pos // 4

    if i >= 3:
        raise MoveException("The empty slot is in the bottom", "DOWN")

    cp = self.__table.copy()
    bottomIdx = self.__pos + 4

    cp[bottomIdx], cp[self.__pos] = cp[self.__pos], cp[bottomIdx]
    return Table(cp)
def isSolution(self) -> bool:
    return self.prediction() == 0

class UnsolveableException(Exception):
    def __init__(self, table: Table) -> None:
        super().__init__("Unsolveable Exception")
        self.__table = table

```

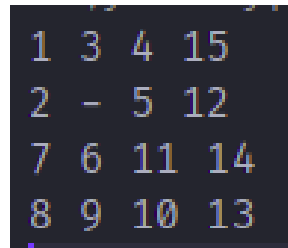
```
def getTable(self) -> Table:  
    return self.__table
```


Bab 4

Hasil Eksekusi Program

4.1. Instansiasi Masalah 1

Berikut ini adalah instansiasi yang saya gunakan pada permasalahan pertama



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

Gambar 4.1. Instansiasi permasalahan 1

Instansiasi ini merupakan instansiasi yang tidak memiliki solusi. Berikut ini adalah hasil eksekusi program



```
(venv) ~ - Stima-Puzzle [main] ⚡ ./run.sh test/tc/tc1.txt

15-Puzzle
Versi 1.0.0
Membaca file test/tc/tc1.txt
Matriks Input:
1 3 4 15
2 16 5 12
7 6 11 14
8 9 10 13

Hasil Proses:
Solusi tidak ditemukan

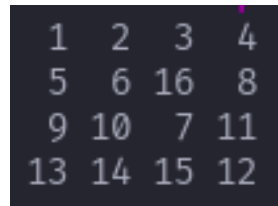
⌚ Time : 85 us

Data Instans:
Jumlah Kurang(i) = 37
Jumlah Kurang(i) + paritas = 37
```

Gambar 4.2. Hasil Pemrosesan Permasalahan 1

4.2. Instansiasi Masalah 2

Berikut ini adalah instansiasi yang saya gunakan pada permasalahan kedua



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

Gambar 4.1. Instansiasi permasalahan 2

Instansiasi ini merupakan instansiasi yang memiliki solusi sebanyak 3 langkah. Berikut ini adalah hasil eksekusi program

```

(venv) └─ Stima-Puzzle [main] ⚡ ./run.sh test/tc/tc2.txt

15-Puzzle

Versi 1.0.0

Membaca file test/tc/tc2.txt

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

Langkah Penyelesaian:

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

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

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

⌚ Time : 555 us

Jumlah Langkah : 3 langkah
Jumlah Node: 10 node

Data Instans:
Jumlah Kurang(i) = 15
Jumlah Kurang(i) + paritas = 16

```

Gambar 4.4. Hasil pemrosesan instansiasi kedua

4.3. Instansiasi Masalah 3

Berikut ini adalah instansiasi permasalahan ketiga

```

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

```

Gambar 4.5. Instansiasi Permasalahan 3

Berikut ini adalah hasil pemrosesan diatas

```

(venv) └─ Stima-Puzzle [main] ⚡ ./run.sh test/tc/tc3.txt

15-Puzzle

Versi 1.0.0

Membaca file test/tc/tc3.txt

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

Langkah Penyelesaian:

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

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

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

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

```

Langkah 5: LEFT

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

Langkah 6: LEFT

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

Langkah 7: UP

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

Langkah 8: UP

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

Langkah 9: UP

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

Langkah 10: RIGHT

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

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

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

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

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

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

⌚ Time : 4 ms

Jumlah Langkah : 15 langkah
Jumlah Node: 40 node

Data Instans:
Jumlah Kurang(i) = 35
Jumlah Kurang(i) + paritas = 36
(venv) └─ Stima-Puzzle [main] ⚡ □
```

Gambar 4.6. Hasil Pemrosesan ketiga

4.4. Instansiasi Masalah 4

Berikut ini adalah instansiasi permasalahan 4

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

Gambar 4.7. Input Permasalahan 4

Berikut ini adalah solusi dari permasalahan diatas:

```
(venv) Stima-Puzzle [main] ⚡ ./run.sh test/tc/tc4.txt
```

15-Puzzle

Versi 1.0.0

Membaca file test/tc/tc4.txt

Matriks Input:

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

Langkah Penyelesaian:

Langkah 1: UP

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

Langkah 2: RIGHT

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

Langkah 3: RIGHT

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

Langkah 4: UP

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

Langkah 5: UP

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

Langkah 6: LEFT

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

Langkah 7: DOWN

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

Langkah 8: LEFT

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

Langkah 9: UP

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

Langkah 10: LEFT

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

Langkah 11: DOWN

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


Langkah 12: RIGHT

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

Langkah 13: RIGHT

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

Langkah 14: DOWN

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

Langkah 15: DOWN

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

Langkah 16: LEFT

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

Langkah 17: LEFT

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

Langkah 18: UP

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

```

Langkah 19: RIGHT
 1  2  3  4
 5  6  7  8
 9 16 11 12
13 10 14 15

Langkah 20: DOWN
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 16 14 15

Langkah 21: RIGHT
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 16 15

Langkah 22: RIGHT
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

⌚ Time : 277 ms

Jumlah Langkah : 22 langkah
Jumlah Node: 4379 node

Data Instans:
Jumlah Kurang(i) = 26
Jumlah Kurang(i) + paritas = 26

```

Gambar 4.8. Hasil dari pemrosesan

4.5. Instansiasi Masalah 5

Berikut ini adalah instansiasi permasalahan kelima

```

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

```

Gambar 4.9. Instansiasi Permasalahan

Berikut ini adalah hasil pemrosesan dari instansiasi diatas. Diketahui bahwa instansiasi tersebut tidak memiliki solusi.

```
15-Puzzle  
Versi 1.0.0  
Membaca file test/tc/tc5.txt  
Matriks Input:  
5 1 3 16  
9 2 6 4  
13 10 11 8  
14 15 12 7  
Hasil Proses:  
Solusi tidak ditemukan  
⌚ Time : 78 us  
Data Instans:  
Jumlah Kurang(i) = 38  
Jumlah Kurang(i) + paritas = 39
```

Gambar 4.10. Hasil pemrosesan instansiasi kelima

Tautan Penting

Hasil program pada laporan ini dapat anda akses pada link berikut ini:

https://github.com/bayusamudra5502/Tucil3_13520128

Checklist Kemampuan Pustaka

Berikut ini adalah daftar cek dari kemampuan pustaka yang dibuat

Poin	Ya	Tidak
1. Program Hasil dikompilasi	v	
2. Program hasil <i>running</i>	v	
3. Program dapat menerima input dan menulis output	v	
4. Luaran sudah benar untuk semua data uji	v	
5. Bonus dibuat		v

Daftar Pustaka

Tim Dosen IF2211. 2022. “Slide Kuliah IF2211 Strategi Algoritma”. Bandung: Institut Teknologi Bandung. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/stima21-22.htm>
(Diakses pada 2 April 2022)