LAPORAN TUGAS KECIL

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

Ditujukan untuk memenuhi salah satu tugas kecil mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2021/2022

Disusun oleh:

Andhika Arta Aryanto 13520081

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

I. PENJELASAN ALGORITMA PROGRAM

1.1 Branch and Bound

Algoritma Branch and Bound adalah algoritma yang biasnya digunakan untuk persoalan optimasi dengan menimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar batasan dari persoalan. Pada dasarnya, Branch and Bound adalah algoritma BFS digabungkan dengan least cost search, dengan setiap simpul akan diberi sebuah nilai *cost* dan pembangkitan simpul selanjutnya akan berdasarkan *cost* tersebut. Untuk persoalan maksimasi akan mengambil *cost* terbesar dan sebaliknya untuk minimasi

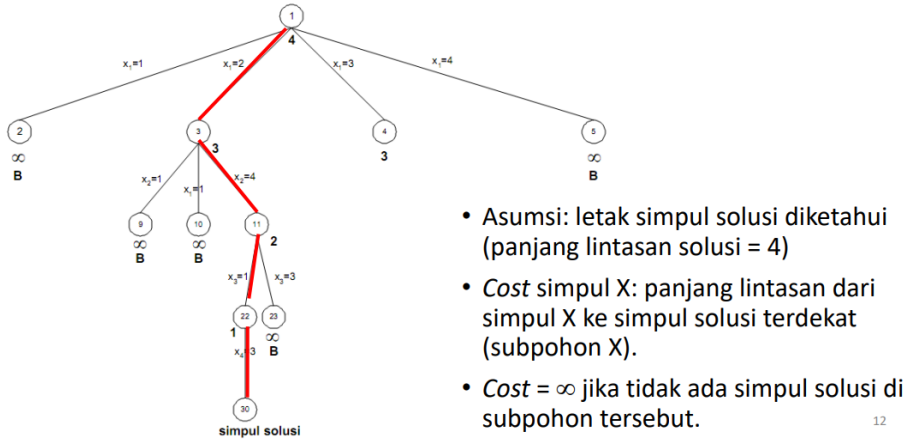


Foto 1. Contoh Penyelesaian 4-Ratu dengan BnB

Pada algoritma BnB akan ada suatu fungsi pembatas yang akan memangkas jalur yang dianggap tidak lagi mengarah pada solusi, berikut beberapa kriteria pemangkasan secara umum :

- Nilai simpul tidak lebih baik dari nilai terbaik sejauh ini (the best solution so far)
- Simpul tidak merepresentasikan solusi yang ‘feasible’ karena ada batasan yang dilanggar
- Solusi pada simpul tersebut hanya terdiri atas satu titik → tidak ada pilihan lain; bandingkan nilai fungsi obyektif dengan solusi terbaik saat ini, yang terbaik yang diambil

1.2 Menyelesaikan persoalan 15-Puzzle dengan Branch and Bound

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

(a) Susunan awal

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

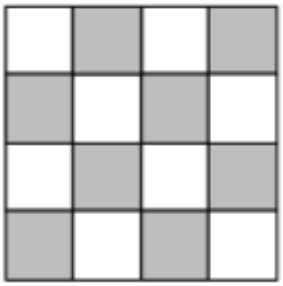
(b) Susunan akhir

Foto 2. Permainan 15-Puzzle

Bisa dilihat dari gambar diatas, permainan 15-Puzzle adalah sebuah permainan puzzle geser dimana terdapat 15 ubin persegi dari nomor 1 – 15 dengan awal posisi acak dalam sebuah *grid 4x4* sehingga menyisakan 1 ubin kosong. Pemain lalu akan mencoba memecahkan puzzle sehingga menjadi seperti susunan akhir pada Foto 2 diatas.

Permainan Puzzle ini bisa diselesaikan dengan beberapa algoritma, misalnya BFS, DFS, ataupun BnB. Pada tugas kecil kali ini akan dibuat sebuah program yang akan menyelesaikan puzzle tersebut dengan algoritma BnB. Beberapa hal yang perlu diperhatikan adalah, dapat ditentukan apakah sebuah puzzle dapat diselesaikan atau tidak bergantung dengan posisi awal dari puzzle tersebut. Berikut cara penentuan :

- Teorema yang digunakan adalah, susunan akhir dari puzzle bisa dicapai jika ∑ KURANG(i) + X dari sebuah puzzle bernilai genap
- X akan bernilai 1 jika posisi kosong ada pada sel yang diarsir, 0 jika tidak :

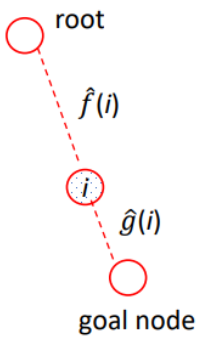


- KURANG(i) adalah banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI (j) > POSISI(i), dengan POSISI adalah posisi suatu nomor pada Puzzle tersebut. Dalam bahasa mudahnya , misal KURANG(4) adalah banyaknya ubin yang nilainya lebih kecil dari 4 (1,2,3) yang posisinya lebih besar dari 4 atau bisa dibilang berada di “sebelah kanan” ubin 4. Perhatikan gambar berikut :

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

  Di puzzle ini, KURANG(4) bernilai 1 karena terdapat satu ubin yang bernilai lebih kecil dan berada di POSISI lebih dari 4, yaitu ubin 2
- Seperti yang sudah disinggung sebelumnya, suatu puzzle dapat diselesaikan hanya bila penjumlahan kedua hal ini bernilai genap

Setelah mengetahui *constraint* yang bisa menentukan apakah suatu *puzzle* dapat diselesaikan akan tidak, akan dibahas mengenai Cost dari simpul hidup pada persoalan ini. Umumnya pada kebanyakan persoalan, letak simpul solusi tidak diketahui sehingga biasanya Cost simput merupakan taksiran dimana $\hat{c}(i) = f(i) + \hat{g}(i)$ dimana $\hat{c}$ adalah ongkos untuk simpul i, f adalah ongkos untuk mencapai simpul i dari akar dan $\hat{g}$ adalah ongkos mencapai simpul tujuan dari simpul i



Dari sini, didapatkan bahwa cost untuk suatu simpul P pada 15-puzle adalah :$\hat{c}$(P) = f(P) + $\hat{g}$(P) denfan f(P) adalah panjang lintasan dari simpul akar ke simpul P dan g(P) adalah jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir (jumlah ubin yang tidak berada di tempat seharusnya)

1.3 Implementasi penyelesaian persoalan 15-Puzzle dengan Branch and Bound

Setelah sudah dibahas mengenai penyelesaian Branch and Bound pada bab 1.2, akan dijelaskan mengenai bagaimana implementasi dari algoritma ini ke program python. Pertama – tama, implementasi dilakukan dengan membuat sebuah Class puzzle yang akan menampung matrix sebagai representasi lokasi ubin pada puzzle, orang tua dari puzzle tersebut (alias move – move yang dilakukan untuk mencapai posisi itu), index dari lokasi "kosong" pada puzzle (pada program ini lokasi kosong dilambangkan dengan angka 0) serta fcost dan gcost dari puzzle. Algoritma Branch and Bound dan pembangkitan node dengan cost terkecil akan dibantu dengan library bawaan python yaitu PriorityQueue

Berikut langkah – langkah yang dilakukan :

1. Program meminta input dari user, terdapat 2 pilihan yaitu membuat Puzzle secara random atau memasukkan puzzle dari file ".txt"
2. Dibuat objek puzzle berdasarkan keinginan user, setelah itu nilai ∑ KURANG(i) + X dari puzzle dihitung
3. Nilai keseluruhan KURANG(I) dan ∑ KURANG(i) + X akan ditampilkan ke console dan program akan berhenti apabila puzzle tidak bisa diselesaikan (bernilai genap_
4. Jika puzzle bisa diselesaikan, akan dialkukan inisialisasi untuk beberapa hal yaitu : visited yaitu dictionary yang berisi Puzzle apa saja yang sudah pernah dikunjungi, array solutions yang menampung step – step yang dilakukan apabila puzzle berhasil diselesaikan serta matQueue yang memanfaatkan PriorityQueue dari Python
5. Pertama – tama, puzzle posisi awal akan mulai dimasukkan ke dalam Queue dan mulai di proses, program akan masuk ke *while* loop yang tidak akan berhenti sampai queue tersebut kosong. Puzzle juga dimasukkan pada dictionary visited untuk memastikan program tidak mengulangi posisi yang sama
6. Dalam *while loop* akan diambil Puzzle dengan cost terkecil

7. Pengambilan puzzle dengan cost terkecil ini memanfaatkan .get dari PriorityQueue dan denganmendefinisikan fungsi lessthan untuk puzzle dengan membandingkan cost dari puzzle tersebut. Puzzle yang “terkecil” didefinisikan dengan mencari puzzle dengan fcost + gcost terkecil atau apabila cost sama akan dicari puzzle dengan fcost terbesar. Hal ini merupakan heuristik yang dilakukan karena apabila cost 2 puzzle sama , puzzle dengan fcost yang lebih besar berarti gcost dari puzzle tersebut lebih kecil dan hal ini berarti puzzle tersebut lebih dekat dengan node solusi
8. Program akan melakukan pengecekan gerakan apa saja yang mungkin dari Puzzle tersebut dengan melihat posisi dari lokasi kosong, lalu program akan melakukan gerakan yang valid pada puzzle tersebut
9. Misal pada state Puzzle sekarang , gerakan yang mungkin adalah atas kiri bawah, Program akan melakukan pemanggilan pada fungsi pembantu yang melakukan hal itu, lalu program akan mengecek apakah puzzle setelah digerakan terdapat pada dictionary visited yang sudah kita buat tadi, apabila tidak terdapat, puzzle baru ini akan dimasukan pada queue. Hal ini apabila dilihat pada algoritma pada bagian pohon Branch and Bound merupakan bagian dimana anak dari suatu *node* yang sedang diproses dibangkitkan
10. Setelah itu , langkah 5-9 akan terus dilakukan sampai didapatkan suatu puzzle yang “solved”, setelah puzzle solved, semua parent dari puzzle tersebut dimasukkan ke array solusi, jadi misal suatu puzzle selesai dengan gerakan “up”,”down”,”left”, array solusi ini berisi = [[“up”,”down”,”left”]]. Program hanya akan mencari satu solusi dari puzzle tersebut.
11. Setelah itu program akan memunculkan output berupa urutan puzzle dari posisi awal sampai posisi akhir, solusi gerakan yang didapat, waktu eksekusi program, serta node yang dibangkitkan.

Pada program ini, beberapa puzzle yang memiliki step terlalu banyak akan memakan waktu dan space terlalu lama sehingga susah diselesaikan. Misal asumsi terdapat sebuah puzzle yang membutuhkan minimal 29 step untuk diselesaikan, node yang akan dibangkitkan adalah bisa sebanyak ratusan ribu – 1 juta sehingga dibutuhkan beberapa optimasi pada program. Berikut beberapa optimasi yang diimplementasikan :

- Penghitungan gcost diefisensikan sehingga hanya membutuhkan O(1) , hal ini dilakukan dengan daripada setiap Puzzle baru dihitung gcostnya, lebih baik apabila gcost dihitung tiap dilakukan setiap puzzle digerakan, sehingga program hanya perlu mengetahui gcost puzzle pada posisi paling awal. Karena pada esensinya gcost hanya berupa +1 (apabila ubin yang awalnya berada pada posisi benar menjadi salah), 0, dan -1(ubin yang digerakan awalnya posisi salah menjadi benar). Hal ini lumayan berdampak karena bayangkan apabila dibangkitkan 1 juta node, akan ada perbedaan yang lumayan.
- Penggunaan library python PriorityQueue juga dictionary sehingga mempercepat insertion untuk node juga dengan dictionary pengecekan apakah suatu puzzle pernah dikunjungi sebelumnya sangat cepat
- Heuristik berupa apabila saat pemilihan mana node dengan cost terkecil, apabila banyak node bernilai sama, akan dicari node dengan fcost paling besar, hal ini memastikan node yang diambil selalu node yang paling dekat dengan solusi

II. SOURCE CODE

2.1 Kode Program Dalam Python

- **File main.py :**

```
from puzzle import *
import numpy as np
from queue import PriorityQueue
import timeit

print("SELAMAT DATANG DI 15-PUZZLE-SOLVER :D\n")
print("Terdapat 2 Pilihan untuk 15-Puzzle : ")
print("1. Randomly Generated Puzzle")
print("2. Puzzle from .txt file\n")
cc = int(input("Silahkan Masukkan pilihan anda : "))

if (cc == 1) :
    puzzle = Puzzle()
if (cc == 2) :
    filename = "./Test/" + input("input filename : ")
    with open(filename, 'r') as f :
        inputMatrix = [[int(num) for num in line.split(' ')] for line in f]
    puzzle = Puzzle(inputMatrix, [])

start = timeit.default_timer()
puzzle.setGcost(puzzle.countGCost())

print("Matrix Awal :")
puzzle.printMatrix()
print("=" * 35)
print("Nilai untuk setiap fungsi Kurang (i) : \n")
for i in range(1,17) :
    count = puzzle.checkKurangIndividual(i)
    if(i < 10) :
        print(f"i = {i}  | Kurang {i} = {count}")
    else :
        print(f"i = {i} | Kurang {i} = {count}")
print("=" * 35)
print(f"Nilai untuk Σ Kurang(i) + X = {puzzle.checkKurang() +
puzzle.checkEmpty()}")

if(not puzzle.isReachable()) :
    print("Puzzle tidak dapat di Solve\n")
else :
    print("\nPuzzle bisa di solve, steps : ")
    visited = {}
```

```
solutions = []
matQueue = PriorityQueue()
matQueue.put(puzzle)
nodeVisited = 0

while(not matQueue.empty()) :
    currentNode = matQueue.get()
    visited[str(currentNode)] = True

    if(currentNode.isSolved()) :
        solutions.append(currentNode.parents)
        stop = timeit.default_timer()
        break
    else :
        availableMoves = currentNode.checkAvailableMove()
        for move in availableMoves :
            if move == "up" :
                new = currentNode.moveUp()
                if(str(new) not in visited) :
                    nodeVisited += 1
                    matQueue.put(new)
            if move == "right" :
                new = currentNode.moveRight()
                if(str(new) not in visited) :
                    nodeVisited += 1
                    matQueue.put(new)
            if move == "down" :
                new = currentNode.moveDown()
                if(str(new) not in visited) :
                    nodeVisited += 1
                    matQueue.put(new)
            if move == "left" :
                new = currentNode.moveLeft()
                if(str(new) not in visited) :
                    nodeVisited += 1
                    matQueue.put(new)


puzzle.stepSoFar(solutions[0])

print("-" * 35)
print(f"• Solusi yang didapat = {solutions}")
print(f"• Banyak step = {len(currentNode.parents)} step")
print(f"• Waktu Eksekusi : {stop - start} detik")
print(f"• Node yang dibangkitkan : {nodeVisited} node")
print("-" * 35)
```

- **File puzzle.py :**

```
import numpy as np
import copy

class Puzzle :
    def __init__(self, matrix = [], parents = [], gcost = 0, fcost = 0) :
        if(len(matrix) == 0) :
            self.randomPuzzleGenerator()
        else :
            self.matrix = np.array(matrix)
        self.parents = parents
        self.kosong = []
        self.fCost = fcost
        self.gCost = gcost
        self.findEmpty()

    def __lt__(self,other) :
        if(self.findcost() == other.findcost()) :
            return self.fCost >= other.fCost
        return self.findcost() <= other.findcost()

    def __str__ (self) :
        newpuzzle = (np.reshape(self.matrix,16)).tolist()
        asStr = ''
        for i in range (16) :
            asStr += f"{newpuzzle[i]}"
        return asStr

    def randomPuzzleGenerator(self) :
        self.matrix = np.arange(0,16)
        np.random.shuffle(self.matrix)
        self.matrix = np.reshape(self.matrix,(4,4))

    def setGcost(self, newGcost) :
        self.gCost = newGcost

    def ispuzzleValid(self) :
        if(self.matrix.shape != (4,4)) :
            return False
        newpuzzle = np.sort(np.reshape(self.matrix,16))
        for i in range(16) :
            if(i != newpuzzle[i]) :
                return False
        return True

    def checkKurangIndividual(self,number) :
        count = 0
```

```
        newpuzzle = (np.reshape(self.matrix,16)).tolist()
        if(number == 16) :
            foundIndex = self.kosong[0] * 4 + self.kosong[1]
        else :
            foundIndex = newpuzzle.index(number)

        for i in range(foundIndex,16) :
            if(newpuzzle[i] < number and newpuzzle[i] != 0) :
                count += 1
        return count


    def checkKurang(self) :
        #implementasi Kurang[i]
        count = 0
        newpuzzle = np.reshape(self.matrix,16)
        for i in range (15) :
            if(newpuzzle[i] != 0) :
                for j in range(i+1,16) :
                    if(newpuzzle[j] < newpuzzle[i] and newpuzzle[j] != 0) :
                        count += 1
            else :
                for j in range(i+1,16) :
                    count += 1
        return count

    def checkEmpty(self) :
        #mengecek apakah kosong ada di daerah arsir atau tidak
        newpuzzle = (np.reshape(self.matrix,16))
        newpuzzle = newpuzzle.tolist()
        index = newpuzzle.index(0)
        if(index in {0,2,5,7,8,10,13,15}):
            return 0
        elif(index in {1,3,4,6,9,11,12,14}) :
            return 1

    def isReachable(self) :
        return (self.checkEmpty() + self.checkKurang()) % 2 == 0

    def findEmpty(self) :
        newpuzzle = (np.reshape(self.matrix,16)).tolist()
        index = newpuzzle.index(0)
        self.kosong = [index//4 , index % 4]

    def countGCost(self) :
        cost = 0
        pembanding = np.arange(1,17)
        newpuzzle = (np.reshape(self.matrix,16))
```

```
        for i in range(0,16) :
            if(newpuzzle[i] != 0 and newpuzzle[i] != pembanding[i]) :
                cost += 1
        return cost

    def findcost(self) :
        return self.fCost + self.gCost

    def checkAvailableMove(self) :
        availableMoves = []
        if(self.kosong[0] != 0) :
            availableMoves.append("up")
        if(self.kosong[1] != 3) :
            availableMoves.append("right")
        if(self.kosong[0] != 3) :
            availableMoves.append("down")
        if(self.kosong[1] != 0) :
            availableMoves.append("left")
        return availableMoves

    #Fungsi gerakin puzzle
    def moveUp(self) :
        copy = np.ndarray.copy(self.matrix)
        i,j = self.kosong[0], self.kosong[1]

        indexBefore = ((i-1) * 4) + j
        indexAfter = i*4 + j
        gcostafter = self.gCost
        if(copy[i-1][j] == indexBefore + 1) :
            gcostafter += 1
        elif(copy[i-1][j]  == indexAfter + 1) :
            gcostafter -= 1

        newFcost = self.fCost + 1
        copy[i][j], copy[i-1][j] =  copy[i-1][j], copy[i][j]
        child = Puzzle(copy, self.parents + ["up"], gcostafter, newFcost)
        return child
    def moveDown(self) :
        copy = np.ndarray.copy(self.matrix)
        i,j = self.kosong[0], self.kosong[1]

        indexBefore = ((i+1) * 4) + j
        indexAfter = i*4 + j
        gcostafter = self.gCost
        if(copy[i+1][j] == indexBefore + 1) :
            gcostafter += 1
        elif(copy[i+1][j] == indexAfter + 1) :
            gcostafter -= 1
```

```
        copy[i][j], copy[i+1][j] =  copy[i+1][j], copy[i][j]
        newFcost = self.fCost + 1
        child = Puzzle(copy, self.parents + ["down"], gcostafter, newFcost)
        return child
    def moveLeft(self) :
        copy = np.ndarray.copy(self.matrix)
        i,j = self.kosong[0], self.kosong[1]
        gcostafter = self.gCost
        indexBefore = (i * 4) + (j - 1)
        indexAfter = i*4 + j

        if(copy[i][j-1] == indexBefore + 1) :
            gcostafter += 1
        elif(copy[i][j-1] == indexAfter + 1) :
            gcostafter -= 1

        newFcost = self.fCost + 1
        copy[i][j], copy[i][j-1] =  copy[i][j-1], copy[i][j]
        child = Puzzle(copy, self.parents + ["left"], gcostafter, newFcost)

        return child
    def moveRight(self) :
        copy = np.ndarray.copy(self.matrix)
        i,j = self.kosong[0], self.kosong[1]
        gcostafter = self.gCost
        indexBefore = (i * 4) + (j+1)
        indexAfter = i*4 + j

        if(copy[i][j+1] == indexBefore + 1) :
            gcostafter += 1
        elif(copy[i][j+1] == indexAfter + 1) :
            gcostafter -= 1

        copy[i][j], copy[i][j+1] =  copy[i][j+1], copy[i][j]
        newFcost = self.fCost + 1
        child = Puzzle(copy, self.parents + ["right"], gcostafter, newFcost)
        return child

    def isSolved(self) :
        return self.gCost == 0

    def stepSoFar(self,solution) :
        copied = copy.deepcopy(self)
        print("\nPosisi Awal : ")
        self.printMatrix()
        print("")
        i = 1
```

```
        for move in solution :
            print("=" * 35)
            if(move == "up") :
                print(f"Step ke - {i} : UP, Puzzle Menjadi :")
                copied = copied.moveUp()
            if(move == "down") :
                print(f"Step ke - {i} : DOWN, Puzzle Menjadi :")
                copied = copied.moveDown()
            if(move == "right") :
                print(f"Step ke - {i} : RIGHT, Puzzle Menjadi :")
                copied = copied.moveRight()
            if(move == "left") :
                print(f"Step ke - {i} : LEFT, Puzzle Menjadi :")
                copied = copied.moveLeft()
            if(i == len(solution)) :
                print("Posisi Akhir, Puzzle Solved ")
            copied.printMatrix()
            i += 1

    def printMatrix(self) :
        print("╔══╦══╦══╦══╗")
        for i in range(4):
            for j in range(4):
                print("║ ",end="")
                if(self.matrix[i][j] == 0) :
                    print(" ", end= "")
                else :
                    print(self.matrix[i][j], end="")
                if(self.matrix[i][j] < 10):
                    print(" ", end="")
            print("║")
            if(i != 3):
                print("╠══╬══╬══╬══╣")
        print("╚══╩══╩══╩══╝")
```

III. HASIL PERCOBAAN

3.1 Berkas teks instansiasi 5 Buah 15 Puzzle

Berikut file.txt yang akan digunakan dalam percobaan :

a. **unreachable1.txt :**

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

b. **unreachable2.txt :**

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

c. **3step.txt :**

1 2 3 4

5 6 0 8

9 10 7 11

13 14 15 12

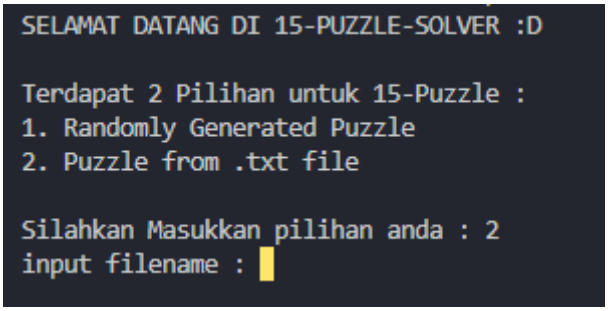
d. **5step.txt :**

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

e. **16step.txt :**

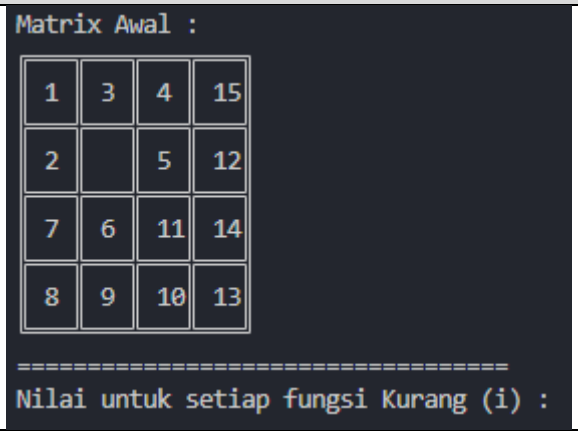
10 2 4 8
1 5 3 0
9 7 6 12
13 14 11 15

3.2 Interaksi Input/Output



3.3 Instansiasi 5 Persoalan 15-puzzle

Percobaan 1 (Puzzle tidak bisa di solve)

Nama File	unreachable1.txt
Bentuk Awal	
Hasil	
	

```
i = 1  | Kurang 1 = 0
i = 2  | Kurang 2 = 0
i = 3  | Kurang 3 = 1
i = 4  | Kurang 4 = 1
i = 5  | Kurang 5 = 0
i = 6  | Kurang 6 = 0
i = 7  | Kurang 7 = 1
i = 8  | Kurang 8 = 0
i = 9  | Kurang 9 = 0
i = 10 | Kurang 10 = 0
i = 11 | Kurang 11 = 3
i = 12 | Kurang 12 = 6
i = 13 | Kurang 13 = 0
i = 14 | Kurang 14 = 4
i = 15 | Kurang 15 = 11
i = 16 | Kurang 16 = 10
==================================
Nilai untuk Σ Kurang(i) + X = 37
Puzzle tidak dapat di Solve
```

Percobaan 2 (Puzzle tidak bisa di solve)

Nama File	unreachable2.txt
Bentuk Awal	
Hasil	

```
i = 1  | Kurang 1 = 0
i = 2  | Kurang 2 = 0
i = 3  | Kurang 3 = 2
i = 4  | Kurang 4 = 0
i = 5  | Kurang 5 = 4
i = 6  | Kurang 6 = 3
i = 7  | Kurang 7 = 6
i = 8  | Kurang 8 = 4
i = 9  | Kurang 9 = 2
i = 10 | Kurang 10 = 1
i = 11 | Kurang 11 = 10
i = 12 | Kurang 12 = 6
i = 13 | Kurang 13 = 0
i = 14 | Kurang 14 = 10
i = 15 | Kurang 15 = 11
i = 16 | Kurang 16 = 10
==================================
Nilai untuk Σ Kurang(i) + X = 69
Puzzle tidak dapat di Solve
```

Percobaan 3

Nama File	3step.txt
Bentuk Awal	

Hasil

```
==================================
Nilai untuk setiap fungsi Kurang (i) :

i = 1  | Kurang 1 = 0
i = 2  | Kurang 2 = 0
i = 3  | Kurang 3 = 0
i = 4  | Kurang 4 = 0
i = 5  | Kurang 5 = 0
i = 6  | Kurang 6 = 0
i = 7  | Kurang 7 = 0
i = 8  | Kurang 8 = 1
i = 9  | Kurang 9 = 1
i = 10 | Kurang 10 = 1
i = 11 | Kurang 11 = 0
i = 12 | Kurang 12 = 0
i = 13 | Kurang 13 = 1
i = 14 | Kurang 14 = 1
i = 15 | Kurang 15 = 1
i = 16 | Kurang 16 = 9
==================================
Nilai untuk Σ Kurang(i) + X = 16
```

Puzzle bisa di solve, steps :

Posisi Awal :

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

==========

Step ke - 1 : DOWN, Puzzle Menjadi :

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

==========

==========

Step ke - 2 : RIGHT, Puzzle Menjadi :

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

==========

Step ke - 3 : DOWN, Puzzle Menjadi :

Posisi Akhir, Puzzle Solved

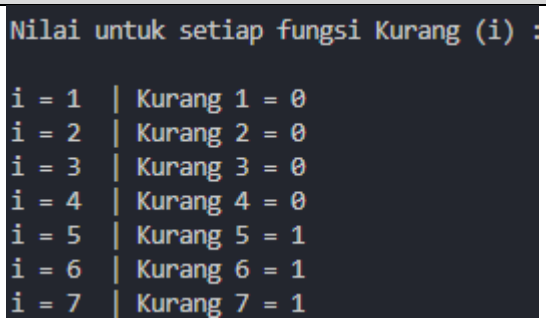
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- Solusi yang didapat = [['down', 'right', 'down']]
- Banyak step = 3 step
- Waktu Eksekusi : 0.005287699983455241 detik
- Node yang dibangkitkan : 9 node

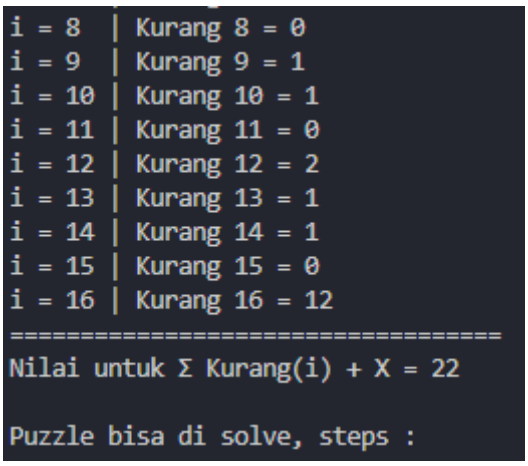
Percobaan 4

Nama File	5step.txt
Bentuk Awal	

Hasil



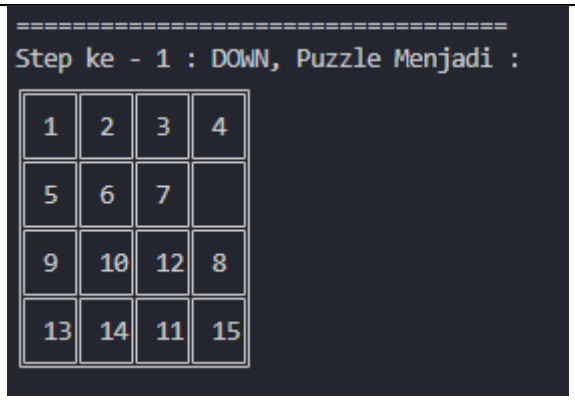
```
Nilai untuk setiap fungsi Kurang (i) :

i = 1  | Kurang 1 = 0
i = 2  | Kurang 2 = 0
i = 3  | Kurang 3 = 0
i = 4  | Kurang 4 = 0
i = 5  | Kurang 5 = 1
i = 6  | Kurang 6 = 1
i = 7  | Kurang 7 = 1
```

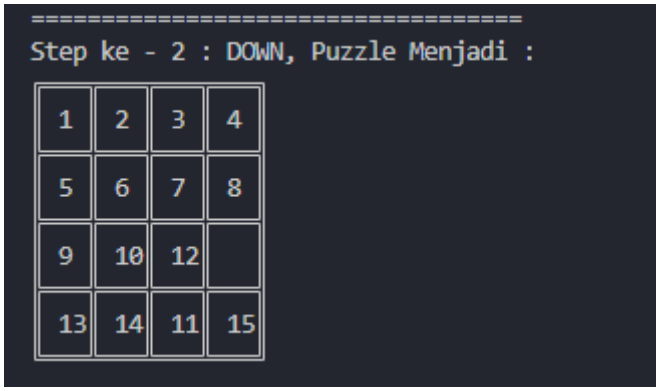


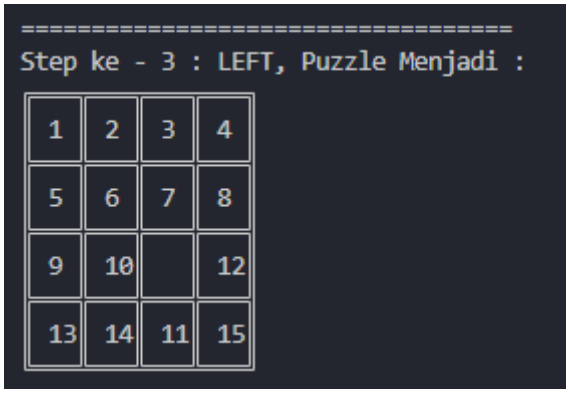
```
i = 8  | Kurang 8 = 0
i = 9  | Kurang 9 = 1
i = 10 | Kurang 10 = 1
i = 11 | Kurang 11 = 0
i = 12 | Kurang 12 = 2
i = 13 | Kurang 13 = 1
i = 14 | Kurang 14 = 1
i = 15 | Kurang 15 = 0
i = 16 | Kurang 16 = 12
==================================
Nilai untuk Σ Kurang(i) + X = 22

Puzzle bisa di solve, steps :
```

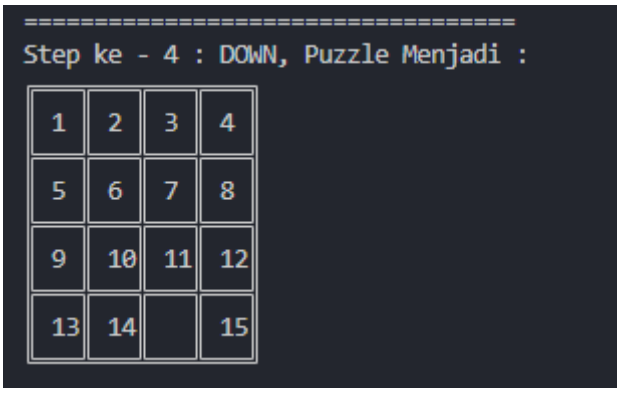
```
Posisi Awal :
```

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


==========
Step ke - 1 : DOWN, Puzzle Menjadi :
1 2 3 4
5 6 7
9 10 12 8
13 14 11 15


==========
Step ke - 2 : DOWN, Puzzle Menjadi :
1 2 3 4
5 6 7 8
9 10 12
13 14 11 15


==========
Step ke - 3 : LEFT, Puzzle Menjadi :
1 2 3 4
5 6 7 8
9 10 12
13 14 11 15


==========
Step ke - 4 : DOWN, Puzzle Menjadi :
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15

```
Step ke - 5 : RIGHT, Puzzle Menjadi :
Posisi Akhir, Puzzle Solved
```

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

```
-----------------------------------
• Solusi yang didapat = [['down', 'down', 'left', 'down', 'right']]
• Banyak step = 5 step
• Waktu Eksekusi : 0.005192500015255064 detik
• Node yang dibangkitkan : 11 node
-----------------------------------
```

Percobaan 5

Nama File	16step.txt
Bentuk Awal	

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

Hasil

```
==================================
Nilai untuk setiap fungsi Kurang (i) :

i = 1  | Kurang 1 = 0
i = 2  | Kurang 2 = 1
i = 3  | Kurang 3 = 0
i = 4  | Kurang 4 = 2
i = 5  | Kurang 5 = 1
i = 6  | Kurang 6 = 0
i = 7  | Kurang 7 = 1
i = 8  | Kurang 8 = 5
i = 9  | Kurang 9 = 2
i = 10 | Kurang 10 = 9
i = 11 | Kurang 11 = 0
i = 12 | Kurang 12 = 1
i = 13 | Kurang 13 = 1
i = 14 | Kurang 14 = 1
i = 15 | Kurang 15 = 0
i = 16 | Kurang 16 = 8
==================================
Nilai untuk Σ Kurang(i) + X = 32

Puzzle bisa di solve, steps :
```

Posisi Awal :

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

==========

Step ke - 1 : UP, Puzzle Menjadi :

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

==========

Step ke - 2 : LEFT, Puzzle Menjadi :

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

==========

Step ke - 3 : LEFT, Puzzle Menjadi :

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

==========

Step ke - 4 : LEFT, Puzzle Menjadi :

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

==========

Step ke - 5 : DOWN, Puzzle Menjadi :

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

==========

Step ke - 6 : RIGHT, Puzzle Menjadi :

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

==========

Step ke - 7 : UP, Puzzle Menjadi :

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

==========

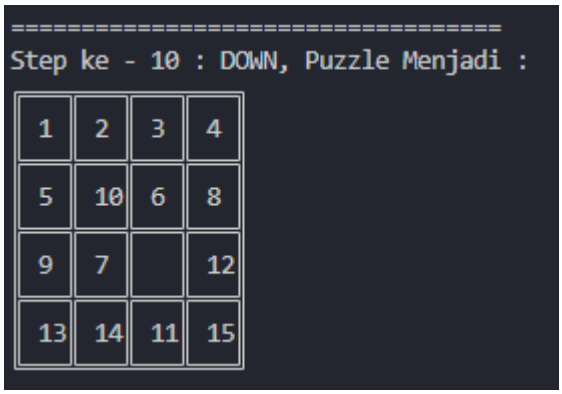
Step ke - 8 : RIGHT, Puzzle Menjadi :

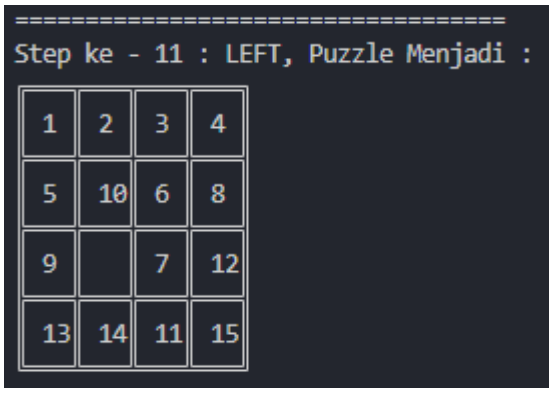
1	2		4
5	10	3	8
9	7	6	12
13	14	11	15

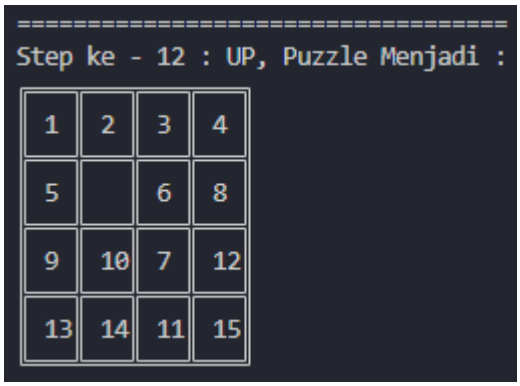
==========

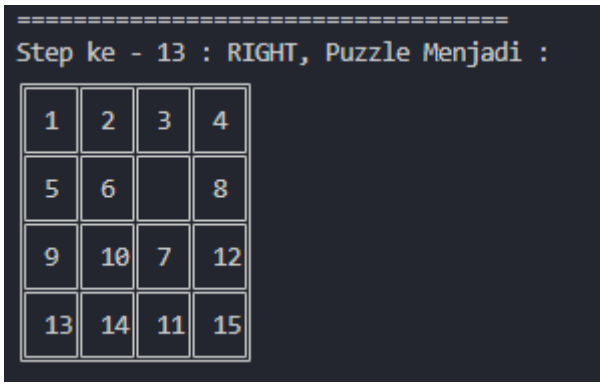
Step ke - 9 : DOWN, Puzzle Menjadi :

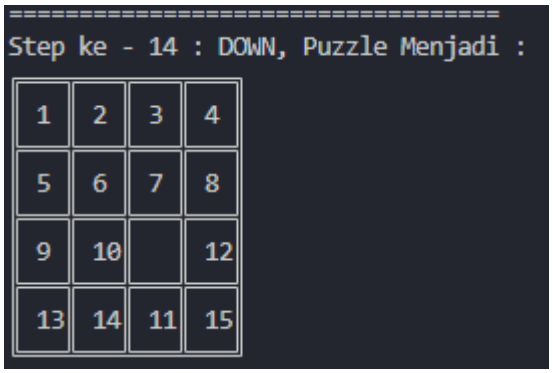
1	2	3	4
5	10		8
9	7	6	12
13	14	11	15

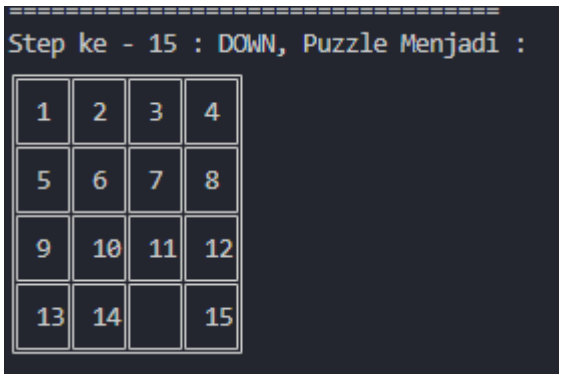


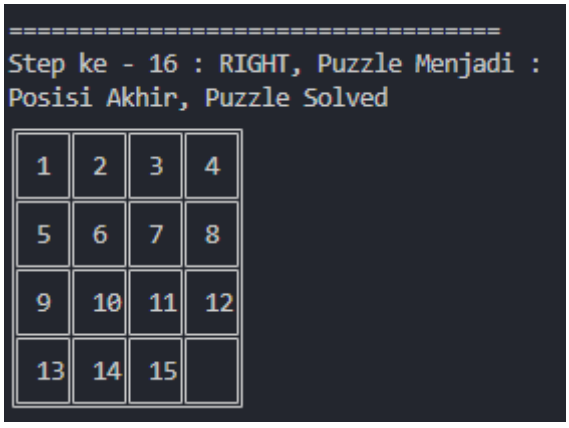













```
-----------------------------------
• Solusi yang didapat = [['up', 'left', 'left', 'left', 'down', 'right', 'up', 'right', 'down', 'down', 'left', 'up', 'right', 'down', 'down', 'right']]
• Banyak step = 16 step
• Waktu Eksekusi : 0.012928200012538582 detik
• Node yang dibangkitkan : 418 node
-----------------------------------
```

IV. LAMPIRAN

4.1 Alamat Github

https://github.com/dhikaarta/15-Puzzle-Solver

4.2 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		✓

V. REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf

https://www.geeksforgeeks.org/priority-queue-in-python/

https://www.w3schools.com/python/python_classes.asp

https://www.w3schools.com/python/python_file_handling.asp

https://numpy.org/doc/stable/reference/