

LAPORAN
TUGAS KECIL IF2211 STRATEGI ALGORITMA
SEMESTER II TAHUN 2021/2022



DISUSUN OLEH:
Yohana Golkaria Nainggolan 13520053

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

Daftar Isi

BAB I ALGORITMA BRANCH AND BOUND	3
BAB II SOURCE CODE	4
BAB III HASIL UJI PROGRAM	7
BAB IV ALAMAT PROGRAM	9
LAMPIRAN	10

BAB I ALGORITMA BRANCH AND BOUND

Algoritma *Branch and Bound* merupakan salah satu jenis strategi algoritma yang dapat digunakan untuk menjabarkan *state* dari suatu persoalan bila diketahui *state* tujuan dari persoalan yang akan dipecahkan. Efisiensi dari algoritma ini bisa berbeda bergantung pada fungsi heuristiknya. Semakin sedikit *state* yang dibangun, semakin efisien algoritma ini bekerja. Efisiensi algoritma dapat dilakukan dengan memilih *state* dengan *cost* terkecil.

Penggunaan algoritma *Branch and Bound* cocok digunakan untuk menyelesaikan permasalahan 15 Puzzle karena *state* akhir dari permainan 15 Puzzle sudah diketahui secara pasti, yaitu memosisikan 15 angka pada *puzzle* secara berurut di dalam 16 *tile* yang tersedia. Langkah-langkah penyelesaian 15 Puzzle adalah sebagai berikut:

1. Tentukan *state* awal dari teka-teki.
2. Hitung jumlah inversi serta posisi kotak kosong untuk menentukan apakah *state solvable* atau tidak.
3. Tentukan *state* yang mungkin dicapai dari *current state*. Jika tidak ada *state* yang memungkinkan, tidak perlu membangun *state*. *State* yang mungkin dicapai:
 - I. Kotak kosong bisa berpindah ke atas atau tidak
 - II. Kotak kosong bisa berpindah ke kiri atau tidak
 - III. Kotak kosong bisa berpindah ke kanan atau tidak
 - IV. Kotak kosong bisa berpindah ke bawah atau tidak

4. Hitung *cost* taksiran:

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

5. Ulangi langkah ketiga untuk *state* yang dipilih
6. Stop jika sudah mencapai *state* akhir.

BAB II SOURCE CODE

Program ditulis dalam bahasa Python dengan isi program sebagai berikut:

1. puzzleSolver.py

```
import copy

class StateSpaceTree:
    def __init__(self, root, parent = None, depth=0, move=""):
        self.root = root
        self.parent = parent
        self.depth = depth
        self.move = move

class Queue:
    def __init__(self, priority):
        self.queue = []
        self.function = priority

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

    def firstElmt(self):
        return self.queue[0]

    def push(self, item):
        idx = 0
        found = False

        while(not found and idx < len(self.queue)):
            if(self.function(item, self.queue[idx])):
                found = True
            else:
                idx+=1

        self.queue.insert(idx, item)

    def pop(self):
        self.queue.pop(0)

class Puzzle:
    def __init__(self, dir):
        self.board = []
        self.size = 0

        f = open(dir, "r")
        for line in f:
            self.board.append(list(map(lambda x : int(x), line.split())))
        self.size = len(self.board)

    def emptyCell(self):
        for i,row in enumerate(self.board):
            for j,value in enumerate(row):
                if(value==self.size**2):
                    return (i,j)

    def boardFlat(self):
        return [val for arr in self.board for val in arr]

    def move(self, a, b):
        (c, d) = self.emptyCell()
        if(c+a>0 and c+a<self.size and d+b>0 and d+b<self.size):
            moved_puzzle = copy.deepcopy(self)
            moved_puzzle.board[c][d], moved_puzzle.board[c+a][d+b] = moved_puzzle.board[c+a][d+b], moved_puzzle.board[c][d]
            return moved_puzzle
        else:
            return None

    def isSolveable(self):
        (c, d) = self.emptyCell()
        temp = self.boardFlat()
        x = (c+d) % 2
        total = 0
        for i in range(0,self.size**2):
            for j in range(i+1,self.size**2):
                if(temp[i]>temp[j]):
                    total+=1

        print("Jumlah inversi yang terjadi", total)
        print("Parity:", x)
        print("Alpha:", total + x, "(even)" if (total+x) % 2 == 0 else "(odd)")

        return (total + x) % 2 == 0

    def boardOutput(self):
        for row in self.board:
            for value in row:
                print('%4s' % (value if value!=self.size**2 else "#"), end=" ")
            print()
```

2. main.py

```

import time
import argparse
from puzzleSolver import Puzzle
from puzzleSolver import Queue
from puzzleSolver import StateSpaceTree

def tiles(puzzle):
    total = 0
    board = puzzle.boardFlat()

    for i in range(1, puzzle.size**2+1):
        if(board[i-1] != i):
            total+=1
    return total

def distance(puzzle):
    total = 0
    board = puzzle.boardFlat()

    for index in range(0, puzzle.size**2):
        currentElmt = board[index]
        if(currentElmt!=puzzle.size**2):
            cur_r, cur_c = (currentElmt-1) // puzzle.size, (currentElmt-1) % puzzle.size
            index_r, index_c = index // puzzle.size, index % puzzle.size

            total += abs(index_r - cur_r) + abs(index_c - cur_c)
    return total

def isSorted(puzzle):
    board = puzzle.boardFlat()

    for i in range(1, (puzzle.size**2)+1):
        if(board[i-1] != i):
            return False

    return True

def solution(solved):
    sol = []

    state = solved.parent
    prev = solved

    while(state != None):
        sol.insert(0,prev)
        prev = state
        state = state.parent
    return sol

argument_parser = argparse.ArgumentParser(prog='python main.py')
argument_parser.add_argument('filename', metavar='filename', type=str, help='Filename dari puzzle')
argument_parser.add_argument('-sh', '--shorthand', action='store_true', help='Menampilkan solusi')
argument_parser.add_argument('-md', '--manhattandist', action='store_true', help='Mengkalkulasi jarak')
argument = argument_parser.parse_args()

root = StateSpaceTree( Puzzle("../test/" + argument.filename) )

```

```

root.root.boardOutput()
print()

if(not root.root.isSolveable()):
    print("Puzzle is unsolveable.")
    exit()

print("Puzzle is solveable.")
print()

count = 1
cost = tiles
if(argument.manhattandist):
    cost = distance

pq = Queue(Lambda x,y : x.depth + cost(x.root) <= y.depth + cost(y.root))
pq.push(root)
solution_state = None
movesItem = [(-1,0), (0,-1), (1,0), (0,1)]
moves = ["Up", "Left", "Down", "Right"]

start = time.process_time_ns()

while(not pq.isEmpty()):
    current = pq.firstElmt()
    pq.pop()
    if(isSorted(current.root)):
        solution_state = current
        break
    for i, (dr, dc) in enumerate(movesItem):
        if(moves[(i+2)%4] != current.move):
            total = StateSpaceTree(current.root.move(dr, dc), parent=current, depth=current.depth+1, move=moves[i])
            if(total != None and total.root != None):
                count += 1
                pq.push( total )

solution_array = solution(solution_state)
end = time.process_time_ns()

if(not argument.shorthand):
    for index, state in enumerate(solution_array):
        print("Step", str(index+1) + ":", state.move)
        state.root.boardOutput()
        print()

print("Jumlah pergeseran yang terjadi:", len(solution_array))
solutionSH = ""
for i in range(len(solution_array)):
    solutionSH += solution_array[i].move[0] + " "
solutionSH += "Solved"
print(solutionSH)

print(count,"nodes generated")

timeTaken = end - start
print(timeTaken / 1000000, "ms taken")

```

BAB III HASIL UJI PROGRAM

1. 1.txt (solvable)

Hasil test case 1:

```
PS C:\Tucil-III-Stima\src> python main.py -sh 1.txt
 1   2   4   7
 5   6   #   3
 9  11  12   8
13  10  14  15

Jumlah inversi yang terjadi 21
Parity: 1
Alpha: 22 (even)
Puzzle is solveable.

Jumlah pergeseran yang terjadi: 11
R U L D R D L L D R R Solved
70 nodes generated
15.625 ms taken
```

2. 2.txt (unsolvable)

Hasil test case 2:

```
PS C:\Tucil-III-Stima\src> python main.py -sh 2.txt
13  10  11   6
 5   7   4   8
 1  12  14   9
 3  15   2   #

Jumlah inversi yang terjadi 59
Parity: 0
Alpha: 59 (odd)
Puzzle is unsolveable.
```

3. 3.txt (unsolvable)

Hasil test case 3:

```
PS C:\Tucil-III-Stima\src> python main.py -sh 3.txt
 2   1   3   4
 5   6   7   8
 9  10  11  12
13  14  15   #

Jumlah inversi yang terjadi 1
Parity: 0
Alpha: 1 (odd)
Puzzle is unsolveable.
```

4. 4.txt (solvable)

Hasil test case 4:

```
PS C:\Tucil-III-Stima\src> python main.py -sh 4.txt
 1  2 12  3
 5  6  8  4
13  9 11 15
10  #  7 14

Jumlah inversi yang terjadi 26
Parity: 0
Alpha: 26 (even)
Puzzle is solveable.

Jumlah pergeseran yang terjadi: 20
L U R R U U R D L D D R U U L D L D R R Solved
6566 nodes generated
43078.125 ms taken
```

5. 5.txt (solvable)

Hasil test case 5:

```
PS C:\Tucil-III-Stima\src> python main.py -sh 5.txt
 1  2  3  4
 5  6  #  8
 9 10  7 11
13 14 15 12

Jumlah inversi yang terjadi 15
Parity: 1
Alpha: 16 (even)
Puzzle is solveable.

Jumlah pergeseran yang terjadi: 3
D R D Solved
10 nodes generated
0.0 ms taken
```


BAB IV ALAMAT PROGRAM

<https://github.com/Yohanagn/Tucil-III-Stima>

LAMPIRAN

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		√