**LAPORAN TUGAS KECIL 3**

Mata Kuliah IF2211 Strategi Algoritma

A screenshot of a computer

Description automatically generated with medium confidence

Nama Penulis:

Aji Andhika Falah 13520012

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**202****2**

# BAB I Algoritma *Branch and Bound*

Algoritma *Branch and Bound* adalah algoritma yang membagi suatu masalah menjadi masalah-masalah yang lebih kecil (*Branch*) dengan sebuah batasan (*Bound*) untuk mencapai solusi optimal. Algoritma ini membentuk sebuah pohon ruang status, yang setiap simpul diberi nilai *cost*. Algoritma ini akan selalu mengolah simpul yang memiliki *cost* terendah agar mendapatkan solusi optimal.

Untuk mencari solusi optimal dari sebuah 15-*puzzle* akan digunaka algoritma *Branch and Bound* dengan langkah-langkah:

1. Sebelum masuk algoritma hitung , dengan Kurang(i) adalah banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa. Sedangkan X akan bernilai 1 jika tile kosong (di dalam program adalah tile bernomor 16) berada di tile yang diarsir, dan 0 jika tidak.

A picture containing shoji, crossword puzzle, building, clipart

Description automatically generated

Jika bernilai ganjil, maka puzzle tidak bisa diselesaikan. Sedangkan jika bernilai genap, maka puzzle bisa diselesaikan. Jika tidak bisa diselesaikan, maka berhenti.

1. Masukkan simpul akar ke dalam antrian Q. Q adalah sebuah *priority queue*. Jika simpul akar adalah memiliki *current puzzle state* yang merupakan solusi *puzzle* (*goal node*), maka solusi telah ditemukan. Setiap simpul akan berisi *cost*, *level*, *address to parent node*, *current puzzle state*, dan *move* yang dilakukan. *cost* di node adalah jumlah tile yang tidak seperti di solusi *puzzle*, sehingga Q akan memiliki priority sesuai dengan *cost + level*.
2. Jika Q kosong, Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai nilai ‘cost’ ĉ(i) paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu.
4. Jika simpul i memiliki *current puzzle state* yang memiliki solusi *puzzle*, berarti solusi sudah ditemukan.
5. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Anaknya adalah semua gerakan yang mungkin dan bukan kebalikan dari gerakan sebelumnya.
6. Untuk setiap anak j dari simpul i, hitung ĉ(j), dan masukkan semua anak-anak tersebut ke dalam Q.
7. Kembali ke langkah 3.

# BAB 2 *Source Program*

Program ini ditulis dalam bahasa Python. Terdapat 2 file, main.py dan BnB.py. File main.py akan menerima input dan mengirimkan informasi untuk diolah, sedangkan BnB.py berisi fungsi dan prosedur untuk algoritma *Branch and Bound* tersebut.

**File main.py**

import BnB

import os.path

loop = True

while (loop):

    puzzle = []

    kurang = 1

    option = input("Read File or Randomized?\n(1) Read File\n(2) Randomized\nInput: ")

    print("")

    while (option != "1" and option != "2"):

        option = input("Input error, Read File or Randomized?\n(1) Read File\n(2) Randomized\nInput: ")

        print("")

    if (int(option) == 1):

        puzzle = BnB.readFile()

        kurang = BnB.printPuzzleKurang(puzzle)

    elif (int(option) == 2):

        repeat = True

        puzzle = BnB.randomPuzzle()

        kurang = BnB.printPuzzleKurang(puzzle)

        while (repeat):

            option2 = input("Is this okay? (Y/N): ")

            if (option2 == "Y" or option2 == "y"):

                repeat = False

            elif (option2 == "N" or option2 == "n"):

                puzzle = BnB.randomPuzzle()

                kurang = BnB.printPuzzleKurang(puzzle)

            else:

                print("\nInput error")

        print("")

    if (kurang%2 == 0):

        print("Loading...\n")

        BnB.solve(puzzle)

    else:

        print("\nPuzzle cannot be solved\n")

    option3 = input("Want to do another? (Y/N): ")

    repeat = True

    while (repeat):

        if (option3 == "Y" or option3 == "y"):

            repeat = False

        elif (option3 == "N" or option3 == "n"):

            loop = False

            repeat = False

        else:

            option3 = input("Input error. Want to do another? (Y/N): ")

    print("")

**File BnB.py**

import os

import random

import time

from heapq import heappush, heappop

from tokenize import Token

# Class for priority queue

class PQueue:

    # Initiate a Priority Queue

    def \_\_init\_\_(self):

        self.heap = []

    # Push into the queue

    def push(self, k):

        heappush(self.heap, k)

    # Pop from the queue

    def pop(self):

        return heappop(self.heap)

    # Check if the queue is empty

    def empty(self):

        if not self.heap:

            return True

        else:

            return False

# Class for nodes

class Node:

    # Initiate a node

    def \_\_init\_\_(self, cost, level, parent, puzzle, move):

        self.cost = cost

        self.level = level

        self.parent = parent

        self.puzzle = puzzle

        self.move = move

    # overide the function Less Than (<) for the priority queue

    def \_\_lt\_\_(self, nxt):

        return self.cost + self.level < nxt.cost + nxt.level

# Function for reading files

def readFile():

    puzzle = []

    puzzleRow = []

    filename = input("Please input filename: ")

    path = os.path.join((os.path.abspath(os.path.join(os.getcwd(), os.pardir))), "test", filename)

    while (not os.path.exists(path)):

        filename = input("\nFilename not found. Please reinput filename: ")

        path = os.path.join((os.path.abspath(os.path.join(os.getcwd(), os.pardir))), "test", filename)

    file = open(path).read()

    array = file.split('\n')

    for i in range (len(array)):

        elements = array[i].split(" ")

        puzzleRow = []

        for j in range (len(elements)):

            puzzleRow.append(int(elements[j]))

        puzzle.append(puzzleRow)

    return (puzzle)

# Function to randomized a puzzle

def randomPuzzle():

    puzzle = []

    puzzleRow = []

    check = [False for i in range (16)]

    for i in range (4):

        puzzleRow = []

        for j in range(4):

            n = random.randint(1,16)

            while (check[n-1]):

                n = random.randint(1,16)

            check[n-1] = True

            puzzleRow.append(n)

        puzzle.append(puzzleRow)

    return (puzzle)

# Function that counts KURANG(i) + X

def kurang(puzzle) :

    kurang = 0

    before = 0

    print("KURANG(i):")

    for e in range(1, len(puzzle) \* len(puzzle[0]) + 1):

        found = False

        grid = False

        ii = 0

        jj = 0

        while (ii < len(puzzle) and not found):

            jj=0

            while (jj < len(puzzle[0]) and not found):

                if (puzzle[ii][jj] == e):

                    found = True

                else:

                    jj += 1

            if (not found):

                ii += 1

        for i in range (len(puzzle)):

            for j in range (len(puzzle[i])):

                if (puzzle[i][j] < e and ((i > ii) or (i == ii and j > jj))) :

                    kurang += 1

                if (e == 16 and puzzle[i][j] == 16 and ((i%2 == 1 and j%2 == 0) or (i%2 == 0 and j%2 == 1))):

                    kurang += 1

                    grid = True

        if (grid):

            print("Tile " + str(e) + ": " + str(kurang - before - 1))

            before = kurang

        else:

            print("Tile " + str(e) + ": " + str(kurang - before))

            before = kurang

    return kurang

# Function that prints out the puzzle and KURANG(i) + X

def printPuzzleKurang(puzzle):

    print("Puzzle:")

    x = '\n'.join([''.join(['{:4}'.format(element) for element in row]) for row in puzzle])

    print(x)

    print("")

    nkurang = kurang(puzzle)

    print("\nKURANG(i) + X =", nkurang,"\n")

    return nkurang

# Function that counts how many tiles are not in the correct places

def countWrong(puzzle):

    cost = 0

    for i in range (len(puzzle)):

        for j in range (len(puzzle[0])):

            if (puzzle[i][j] != (i\*len(puzzle[0]) + j + 1)):

                cost += 1

    return (cost)

# Function that search the location of the empty tile (16 tile)

def search16(puzzle):

    i = 0

    j = 0

    found = False

    while (i < len(puzzle) and not found):

        j = 0

        while (j < len(puzzle[0]) and not found):

            if (puzzle[i][j] == 16):

                found = True

            else:

                j += 1

        if (not found):

            i += 1

    return ([i, j])

# Function that switch a tile with an empty tile

def switchEmpty(puzzle, i, j):

    empty = search16(puzzle)

    tileSwitched = [empty[0] + i, empty[1] + j]

    temp = puzzle[empty[0]][empty[1]]

    puzzle[empty[0]][empty[1]] = puzzle[tileSwitched[0]][tileSwitched[1]]

    puzzle[tileSwitched[0]][tileSwitched[1]] = temp

    return (puzzle)

# Function that prints out the puzzle movements from start to finish

def printFinal(node):

    i = 0

    if (node.parent != None):

        i = printFinal(node.parent)

        i += 1

    print("Puzzle move " + str(i) + ": " )

    x = '\n'.join([''.join(['{:4}'.format(element) for element in row]) for row in node.puzzle])

    print(x, "\n")

    return (i)

# Function that checks if a move is the reverse of the move before that

def notReverse(i, j):

    return (not((i == 0 and j == 2) or (i == 2 and j == 0) or (i == 1 and j == 3) or (i == 3 and j == 1)))

# Procedure that solves the puzzle with Branch and Bound

def solve(puzzle):

    startTimer = time.perf\_counter()

    finalPuzzle = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

    rowMove = [ 1, 0, -1, 0 ]  # Empty slot move 0 = down, 1 = left, 2 = up, 3 = right

    colMove = [ 0, -1, 0, 1 ]

    pqueue = PQueue()

    totalNodes = 1

    # Make root node

    root = Node(countWrong(puzzle), 0, None, puzzle, None)

    pqueue.push(root)

    # Gets node with the lowest cost

    while (not pqueue.empty()):

        pnode = pqueue.pop()

        current = pnode.puzzle

        if (current == finalPuzzle):

            break

        else:

            # Move the empty tile to 4 different ways

            for i in range (4):

                empty = search16(current)

                # Check if move is valid

                if (empty[0] + rowMove[i] >= 0 and empty[0] + rowMove[i] < len(current) and empty[1] + colMove[i] >= 0 and empty[1] + colMove[i] < len(current[0]) and notReverse(i, pnode.move)):

                    # Make the moved puzzle

                    switchedPuzzle = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

                    for ii in range (len(current)):

                        for jj in range (len(current[0])):

                            switchedPuzzle[ii][jj] = current[ii][jj]

                    switchedPuzzle = switchEmpty(switchedPuzzle, rowMove[i], colMove[i])

                    # Make the node and push it to priority queue

                    child = Node(countWrong(switchedPuzzle), pnode.level + 1, pnode, switchedPuzzle, i)

                    pqueue.push(child)

                    totalNodes += 1

    endTimer = time.perf\_counter()

    printFinal(pnode)

    print(totalNodes, "nodes were created in the algorithm")

    print(f"The elapsed time is {endTimer - startTimer:0.4f} seconds")

    print("")

# BAB 3 *Screenshot input* dan *output*

**1. failed1.txt**

*input:*

Text

Description automatically generated

*output:*

A screenshot of a computer

Description automatically generated with medium confidence

**2. failed2.txt**

*input:*

Text

Description automatically generated

*output:*

A screenshot of a computer

Description automatically generated with medium confidence

**3. success1.txt**

*input:*

Text

Description automatically generated

*output:*

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated



**4. success2.txt**

*input:*

A screenshot of a computer

Description automatically generated with medium confidence

*output:*

A picture containing calendar

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing calendar

Description automatically generated

**5. success3.txt**

*input:*

Text

Description automatically generated

*output:*

A picture containing calendar

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

# BAB 4 *GitHub* Program dan Berkas Teks

Untuk program dan berkas teks saya, bisa dilihat di link berikut:

<https://github.com/Enderageous/Tucil3_13520012>

# 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 |  | √ |