**LAPORAN TUGAS KECIL 3  
IF2211 STRATEGI ALGORITMA**

Penyelesaian Persoalan 15-Puzzle Dengan Algoritma Branch and Bound

Logo

Description automatically generated

NIM : 13520085  
Nama : Ubaidillah Ariq Prathama  
Kelas : K01

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

1. Algoritma Branch and Bound Untuk Menyelesaikan Permasalahan

15-Puzzle adalah teka-teki geser yang memiliki 15 persegi bernomor 1 sampai 15 dalam bingkai dengan tinggi 4 kotak dan lebar 4 kotak, menyisakan satu posisi ubin kosong. Kotak di baris atau kolom yang sama dari posisi terbuka dapat dipindahkan dengan menggesernya secara horizontal atau vertikal. Tujuan akhir dari game ini adalah membuat kotak tersebut terurut kembali setelah beberapa kali memindahkan.

Algoritma Branch and Bound merupakan algoritma yang membagi permasalahan menjadi sub masalah lebih kecil yang mengarah ke solusi dengan pencabangan (branching) dan melakukan pembatasan (bounding) untuk mencapai solusi optimal. Pencabangan (branching) yaitu proses membentuk permasalahan ke dalam bentuk struktur pohon pencarian (search tree). Proses Pencabangan dilakukan untuk membangun semua cabang pohon yang menuju solusi, sedangkan proses pembatasan dilakukan dengan menghitung estimasi nilai (cost) simpul dengan memperhatikan batas. Pada dasarnya Branch and Bound merupakan BFS dengan optimasi.

15-Puzzle dapat diselesaikan menggunakan algoritma Branch and Bound. Langkah-langkah dalam menyelesaikan persoalan ini adalah :

1. Memeriksa apakah puzzle dapat diselesaikan. Hal ini dapat dilakukan dengan menggunakan fungsi Kurang(i) dan Arsir(x,y) dengan definisi :

Kurang(i) = Banyaknya ubin bernomor j sedemikian sehingga j < i dan Posisi(j) > Posisi(i). Posisi(i) = posisi ubin bernomor i pada susunan yang diperiksa.

Arsir(x,y) = Bernilai 0 jika x-y genap dan bernilai 1 jika x-y ganjil, x dan y di sini merupakan koordinat awal ubin yang kosong.

Jika ganjil maka puzzle tidak mungkin diselesaikan. Jika genap maka puzzle dapat diselesaikan. Hal ini diimplementasikan dalam fungsi lokasi, kurang, totalKurang.

1. Menghitung fungsi cost dari root. Fungsi cost dapat didefinisikan sebagai banyaknya ubin yang belum berada pada posisi yang benar ditambah kedalaman dari search tree. Untuk kasus root, kedalamannya adalah 0. Hal ini dapat diimplementasikan dengan looping matriks dan diimplementasikan dalam fungsi cost.
2. Mencari child yang mungkin dari parent. Terdapat 4 kemungkinan arah berpindahnya kotak kosong (atas, kanan, bawah, kiri). Constraintnya adalah setelah berpindah ubin kosong masih harus berada di dalam kotak 4x4 dan tidak boleh kembali ke posisi awal sebelum parentnya (repetisi). Hal ini dapat dicek dengan mudah dengan percabangan dan diimplementasikan dalam fungsi nextNode.
3. Child yang memungkinkan dari parent akan dicari costnya. Child ini akan dikonstruksi menjadi sebuah class Node yang berisi (parent, id, matrix, cost, block\_pos, level). Setelah itu akan dipush ke dalam priority queue yang akan memprioritaskan Node dengan cost terendah. Priority queue ini diimplementasikan menggunakan struktur data bawaan python yaitu heap. Hal ini diimplementasikan dalam file priorityQueue.py dan fungsi nextNode.
4. Jika semua child dari parent sudah dicek dan dimasukkan ke dalam priority queue, pencarian dilanjutkan. Keluarkan (pop) elemen terkecil dari priority queue dan elemen ini akan menjadi parent yang baru. Lakukan terus hingga mendapatkan solusi. Hal ini diimplementasikan dengan while loop pada file main.py
5. Hasil Pengujian

Calendar

Description automatically generatedText

Description automatically generatedPengujian test1.txt

Calendar

Description automatically generatedText, calendar

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedPengujian test2.txt

A picture containing text, calculator

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText, calendar

Description automatically generatedText

Description automatically generatedA picture containing text, calculator

Description automatically generatedText

Description automatically generatedText, calendar

Description automatically generatedPengujian test3.txt

Text, calendar

Description automatically generatedText, calendar

Description automatically generatedText, calendar

Description automatically generatedText

Description automatically generated

Text

Description automatically generated with medium confidencePengujian test4.txt

Text

Description automatically generatedPengujian test5.txt

1. Checklist Program

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1.Program berhasil dikompilasi | v |  |
| 2. Program berhasil running | v |  |
| 3. Program dapat menerima input dan menuliskan output | v |  |
| 4. Luaran sudah benar untuk semua data uji | v |  |
| 5.Bonus dibuat |  | v |

1. Kode Program

File priorityQueue.py

from heapq import heappush, heappop

class Node:

    def \_\_init\_\_(self, parent,id, matrix, cost, block\_pos, level):

        self.parent = parent

        self.id = id

        self.matrix = matrix

        self.cost = cost

        self.block\_pos = block\_pos

        self.level = level

    def \_\_lt\_\_(self, other):

        if self.cost == other.cost:

            return self.level < other.level

        return self.cost < other.cost

class PriorityQueue:

    def \_\_init\_\_(self):

        self.heap = []

    def push(self, k):

        heappush(self.heap, k)

    def pop(self):

        return heappop(self.heap)

    def empty(self):

        return not self.heap

File branchAndBound.py

from sympy import false, true

from priorityQueue import PriorityQueue, Node

from copy import deepcopy

pq = PriorityQueue()

id = 1

done = False

def lokasi(matriks, x) :

    for i in range (4) :

        for j in range (4) :

            if matriks[i][j] == x :

                return 4\*i+j

def kurang(matriks, i) :

    count = 0

    for j in range (1, i):

        if lokasi(matriks, i) < lokasi(matriks, j):

            count += 1

    return count

def totalKurang(node) :

    sum = 0

    for i in range (1, 17):

        sum += kurang(node.matrix, i)

    if (node.block\_pos[0] - node.block\_pos[1]) % 2 != 0 :

        sum += 1

    return sum

def cost(node):

    count=0

    for i in range (4):

        for j in range (4):

            if node.matrix[i][j] != 4\*i + j + 1 and node.matrix[i][j] != 0:

                count+=1

    node.cost = count + node.level

def isSolution(node):

    solution = true

    for i in range (4) :

        for j in range (4) :

            if(node.matrix[i][j] != 4\*i+j+1) :

                solution = false

    return solution

def printNode(node):

    print("Simpul ke-" + str(node.id))

    print("Cost : " + str(node.cost))

    print("Level : " + str(node.level))

    for i in range (21) :

        print("-", end="")

    print("")

    for i in range (4) :

        for j in range (4) :

            print("|", end="")

            if(node.matrix[i][j] >= 10):

                if node.matrix[i][j] == 16 :

                    print("  -", end=" ")

                else :

                    print(" " + str(node.matrix[i][j]), end=" ")

            else :

                print("  " + str(node.matrix[i][j]), end=" ")

        print("|", end="\n")

        for i in range (21) :

            print("-", end="")

        print("")

    print("")

def printSolution(node):

    listNode = []

    while node.parent != None :

        listNode.append(node)

        node = node.parent

    listNode.append(node)

    for i in range (len(listNode)-1, -1, -1):

        printNode(listNode[i])

    done = true

def readFile(fileName):

    dir = ".\\test\\"

    dir += fileName

    file = open(dir, "r")

    matriks = [[int(num) for num in line.split(' ')] for line in file]

    return matriks

def nextNode(node):

    global id

    global pq

    if node.block\_pos[0] != 0 and (node.parent == None or (node.parent).block\_pos != [node.block\_pos[0]-1, node.block\_pos[1]]):

        id+=1

        newNodeMatrix1 = deepcopy(node.matrix)

        newNodeMatrix1[node.block\_pos[0]][node.block\_pos[1]] = node.matrix[node.block\_pos[0]-1][node.block\_pos[1]]

        newNodeMatrix1[node.block\_pos[0]-1][node.block\_pos[1]] = 16

        newNode = Node(node, id, newNodeMatrix1, 0, [node.block\_pos[0] - 1, node.block\_pos[1]], node.level + 1)

        cost(newNode)

        if isSolution(newNode):

            print("Solusi ditemukan")

            printSolution(newNode)

        pq.push(newNode)

    if node.block\_pos[1] != 3 and (node.parent == None or (node.parent).block\_pos != [node.block\_pos[0], node.block\_pos[1]+1]):

        id+=1

        newNodeMatrix2 = deepcopy(node.matrix)

        newNodeMatrix2[node.block\_pos[0]][node.block\_pos[1]] = node.matrix[node.block\_pos[0]][node.block\_pos[1]+1]

        newNodeMatrix2[node.block\_pos[0]][node.block\_pos[1]+1] = 16

        newNode = Node(node, id, newNodeMatrix2, 0, [node.block\_pos[0], node.block\_pos[1]+1], node.level + 1)

        cost(newNode)

        if isSolution(newNode):

            print("Solusi ditemukan")

            printSolution(newNode)

        pq.push(newNode)

    if node.block\_pos[0] != 3 and (node.parent == None or (node.parent).block\_pos != [node.block\_pos[0]+1, node.block\_pos[1]]):

        id+=1

        newNodeMatrix3 = deepcopy(node.matrix)

        newNodeMatrix3[node.block\_pos[0]][node.block\_pos[1]] = node.matrix[node.block\_pos[0]+1][node.block\_pos[1]]

        newNodeMatrix3[node.block\_pos[0]+1][node.block\_pos[1]] = 16

        newNode = Node(node, id, newNodeMatrix3, 0, [node.block\_pos[0] + 1, node.block\_pos[1]], node.level + 1)

        cost(newNode)

        if isSolution(newNode):

            print("Solusi ditemukan")

            printSolution(newNode)

        pq.push(newNode)

    if node.block\_pos[1] != 0 and (node.parent == None or (node.parent).block\_pos != [node.block\_pos[0], node.block\_pos[1]-1]):

        id+=1

        newNodeMatrix4 = deepcopy(node.matrix)

        newNodeMatrix4[node.block\_pos[0]][node.block\_pos[1]] = node.matrix[node.block\_pos[0]][node.block\_pos[1]-1]

        newNodeMatrix4[node.block\_pos[0]][node.block\_pos[1]-1] = 16

        newNode = Node(node, id, newNodeMatrix4, 0, [node.block\_pos[0], node.block\_pos[1]-1], node.level + 1)

        cost(newNode)

        if isSolution(newNode):

            print("Solusi ditemukan")

            printSolution(newNode)

        pq.push(newNode)

File main.py

import branchAndBound as bnb

from branchAndBound import pq, done

from priorityQueue import Node

from time import time

fileName = input("Masukkan nama file : ")

start\_time = time()

initial\_matrix = bnb.readFile(fileName)

for i in range (4) :

    for j in range (4) :

        if(initial\_matrix[i][j] == 16) :

            initial\_block\_pos = [i, j]

            break

root = Node(None, 0 , initial\_matrix, 0, initial\_block\_pos, 0)

bnb.cost(root)

pq.push(root)

if(bnb.totalKurang(root) % 2 == 0):

    while(not pq.empty() and not done):

        node = pq.pop()

        if bnb.isSolution(node):

            print("Solusi ditemukan")

            break

        bnb.nextNode(node)

else :

    print("Tidak ada solusi yang mungkin")

end\_time = time()

print("Waktu Eksekusi : " + str(end\_time - start\_time)+ " detik")

1. Test Case Berupa Teks

test1.txt

1 2 3 4

5 6 16 8

9 10 7 11

13 14 15 12

test2.txt

1 7 2 4

9 5 3 8

13 16 11 12

10 6 14 15

test3.txt

1 2 3 4

5 6 7 8

16 13 9 10

11 12 15 14

test4.txt

1 3 4 15

2 16 5 12

7 6 11 14

8 9 10 13

test5.txt

7 2 10 9

6 16 1 3

4 5 12 11

15 8 14 13

1. Link Github

https://github.com/ubaidalih/Tucil3\_13520085