# LAPORAN TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

# Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound



## **Disusun oleh:**

Muhammad Akmal Arifin 13520037

Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung

2022

## A. Algoritma Branch and Bound

Algoritma merupakan sekumpulan instruksi yang terstruktur dan terbatas yang diimplementasikan ke dalam bentuk program komputer untuk menyelesaikan suatu masalah komputasi tertentu. Algoritma dalam pemrograman berarti menjelaskan langkah-langkah yang diperlukan dalam penyelesaian suatu permasalahan. Strategi algoritma merupakan berbagai algoritma yang dilakukan secara umum untuk memecahkan persoalan secara algoritmis, sehingga dapat diterapkan pada bermacam-macam persoalan. Salah satu strategi algoritma adalah algoritma *Branch and Bound*.

Algoritma *Branch and Bound* merupakan salah satu strategi algoritma yang cara pengerjannya menggunakan konsep *Breath First Search* akan tetapi menggunakan *Priority Queue* dalam menentukan *node* mana yang akan dibangkitkan selanjutnya. Priority Queue ditentukan oleh nilai *cost*, semakin kecil nilai *cost* maka semakin awal dieksekusi. *Cost* setiap persoalan bisa jadi berbeda-beda, dalam persoalan ini *cost* merupakan jumlah langkah yang diperlukan untuk mencapai susunan tersebut dari state awal ditambah dengan jumlah ubin tidak kosong yang belum sesuai dengan *goal state*.

Secara garis besar, algoritma *Branch and Bound* yang digunakan untuk penyelesaian permasalahan *15-Puzzle* pada program ini adalah

- 1. Membaca masukan state awal *puzzle* pada *file*
- 2. Melakukan pengecekan apakah *puzzle* dari *state* awal tersebut dapat mencapai *goal state*
- 3. Apabila *puzzle* dapat diselesaikan, maka dilakukan iterasi pembangkitan sampai menemukan *goal state*
- 4. Iterasi pembangkitan dilakukan dengan membangkitkan *puzzle* untuk seluruh arah dari ubin yang kosong (atas, kanan, bawah, kiri) serta dilakukan perhitungan *cost* tiap susunan *puzzle* dan ditambahkan ke dalam *priority queue*
- 5. Setelah itu, susunan *puzzle* yang akan dibangkitkan selanjutnya adalah susunan *puzzle* dengen nilai *cost* terkecil
- 6. Ketika sudah mendapatkan susunan yang sesuai dengan *goal state*, maka program selesai

#### B. SOURCE CODE PROGRAM

Program penyelesaian *15-Puzzle* dalam tugas ini menggunakan bahasa pemrograman Python. Berikut hasil tangkapan layer *source code* program.

#### Import and Constant Variable

```
import time
from PriorityQueue import PriorityQueue

# Constant Value
GOAL_STATE = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
UP = 1
RIGHT = 2
DOWN = 3
LEFT = 4
10
```

#### readFile

```
def readFile(matrixPuzzle, fileName):
    lines = []
    dir = '../test/' + fileName
    with open(dir, 'r') as f:
        lines = f.readlines()

i = 0
for line in lines:
    parse = line.split(' ', 4)
    arrayPuzzle = []
    arrayPuzzle.append(int(parse[0]))
    arrayPuzzle.append(int(parse[1]))
    arrayPuzzle.append(int(parse[2]))
    arrayPuzzle.append(int(parse[3]))
    matrixPuzzle.append(arrayPuzzle)
```

#### listToString

#### writeFile

#### estimateCost

#### kurang

#### *isSolveable*

#### Bangkitkan #1

#### Bangkitkan #2 : Langkah ke Atas

#### Bangkitkan #3: Langkah ke Kanan

```
try:

matrixRight = [[0 for j in range(4)] for i in range(4)]

for i in range(4):

for j in range(4):

matrixRight[i0][j] = matrixRise[i][j]

temp = matrixRight[i0][j0]

matrixRight[i0][j0] = matrixRight[i0][j0+1]

matrixRight[i0][j0+1] = temp

costRight = len(path) + estimateCost(matrixRight)

iRight = i0

jRight = j0+1

if (not matrixRight in path):

queue.insert(costRight, iRight, jRight, matrixRight, path)

totalNode += 1

except IndexError:

# print("index error")

pass
```

#### Bangkitkan #4: Langkah ke Bawah

```
try:
    matrixDown = [[0 for j in range(4)] for i in range(4)]
for i in range(4):
    for j in range(4):
        matrixDown[i][j] = matrixRise[i][j]
temp = matrixDown[i0][j0]
matrixDown[i0][j0] = matrixDown[i0+1][j0]
matrixDown[i0+1][j0] = temp

costDown = len(path) + estimateCost(matrixDown)

iDown = i0+1
jDown = j0

if (not matrixDown in path):
    queue.insert(costDown, iDown, jDown, matrixDown, path)
    totalNode += 1
except IndexError:
    # print("index error")
pass
```

## Bangkitkan #5 : Langkah ke Kiri

```
matrixLeft = [[0 for j in range(4)] for i in range(4)]
    for i in range(4):
       for j in range(4):
            matrixLeft[i][j] = matrixRise[i][j]
    temp = matrixLeft[i0][j0]
   matrixLeft[i0][j0] = matrixLeft[i0][j0-1]
   matrixLeft[i0][j0-1] = temp
    costLeft = len(path) + estimateCost(matrixLeft)
   iLeft = i0
   jLeft = j0-1
   if (not matrixLeft in path):
       queue.insert(costLeft, iLeft, jLeft, matrixLeft, path)
       totalNode += 1
except IndexError:
    pass
return queue, totalNode
```

#### checkGoal

#### **branchAndBound**

```
def branchAndBound(matrixPuzzle):
196
          queue = PriorityQueue()
          cost = estimateCost(matrixPuzzle)
          totalNode = 1
          for i in range(4):
                  for j in range(4):
                      if matrixPuzzle[i][j] == 16:
                          i\theta = i
                          j0 = j
          queue.insert(cost, i0, j0, matrixPuzzle, [])
          isFound = False
          while(not isFound and not queue.isEmpty()):
              queue,totalNode = bangkitkan(queue, totalNode)
              if (checkGoal(queue.getMatrix())):
                  goalPath = queue.getPath()
                  isFound = True
          if (isFound):
              return goalPath, totalNode
          else:
              return [], totalNode
```

#### main

```
def main():
    matrixPuzzle = []
    readFile(matrixPuzzle, "3.txt")

if (isSolveable(matrixPuzzle)):
    tStart = time.time()
    goalPath, totalNode = branchAndBound(matrixPuzzle)

tEnd = time.time()
    writeFile(goalPath, "solusi.txt", totalNode, tEnd-tStart)
else:
    print("unsolveable")

main()
```

#### PriorityQueue.py #1

```
class PriorityQueue(object):
   def __init__(self):
       self.queue = []
   def isEmpty(self):
       return len(self.queue) == 0
   def insert(self, dataCost, i, j, dataMatrix, dataPath):
        self.queue.append([dataCost, i, j, dataMatrix, dataPath])
   def getCost(self):
        try:
            min = 0
            for i in range(len(self.queue)):
                if self.queue[i][0] < self.queue[min][0]:</pre>
                    min = i
            item = self.queue[min][0]
            return item
        except IndexError:
            print()
            exit()
```

#### PriorityQueue.py #2 : getMatrix

#### PriorityQueue.py #3: getPath

#### PriorityQueue.py #4: getEmptyi

#### PriorityQueue.py #5: getEmptyj

#### PriorityQueue.py #6: delete

#### PriorityQueue.py #7: print

```
82  def print(self):
83     print(self.getCost())
84     print(self.getMatrix())
85     print(self.getPath())
```

#### C. SCREENSHOT INPUT DAN OUTPUT

## Instansi Persoalan yang Bisa Diselesaikan

## Input 1.txt

```
test > ≡ 1.txt

1 2 5 3 4

2 1 10 6 8

3 9 16 7 11

4 13 14 15 12
```

## Output solusi\_1.txt

```
test > 

solusi_1.txt
     2 5 3 4
  1
     1 10 6 8
     9 16 7 11
     13 14 15 12
     2 5 3 4
     1 16 6 8
 8 9 10 7 11
     13 14 15 12
     2 16 3 4
     1568
 13
    9 10 7 11
     13 14 15 12
     16 2 3 4
     1568
     9 10 7 11
     13 14 15 12
     1234
     16 5 6 8
     9 10 7 11
     13 14 15 12
```

```
1 2 3 4
5 16 6 8
 9 10 7 11
 13 14 15 12
 1 2 3 4
5 6 16 8
9 10 7 11
 13 14 15 12
 1 2 3 4
5 6 7 8
9 10 16 11
 13 14 15 12
 1 2 3 4
5 6 7 8
 9 10 11 16
 13 14 15 12
 1 2 3 4
5 6 7 8
 9 10 11 12
 13 14 15 16
 Waktu yang dibutuhkan : 0.0
 Jumlah node yang dibangktikan : 46
```

## Input 2.txt

## Output solusi\_2.txt

```
test > ≡ solusi_2.txt
  1 2 16 3 4
 2 1568
  3 9 10 7 11
  4 13 14 15 12
  6 16 2 3 4
  7 1568
    9 10 7 11
 9 13 14 15 12
    1234
 12 16 5 6 8
    9 10 7 11
 14 13 14 15 12
 16 1234
    5 16 6 8
   9 10 7 11
    13 14 15 12
 21 1 2 3 4
 22 5 6 16 8
 23 9 10 7 11
 24 13 14 15 12
```

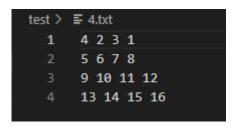
## Input 3.txt

## Output solusi\_3 .txt

```
test > ≡ solusi_3.txt
      1 2 3 4
  1
      5 6 7 8
      10 13 11 12
      9 16 14 15
      1 2 3 4
      5 6 7 8
      10 16 11 12
      9 13 14 15
      1 2 3 4
      5 6 7 8
      16 10 11 12
      9 13 14 15
      1 2 3 4
      5 6 7 8
      9 10 11 12
      16 13 14 15
      1 2 3 4
      5 6 7 8
      9 10 11 12
      13 16 14 15
```

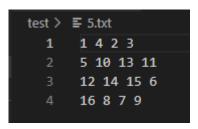
## Instansi Persoalan yang Tidak bisa diselesaikan

## Input 4.txt



D:\Education\Formal\Institut Teknologi Bandung\Akademik\Semester IV\Strategi Algoritma\Tugas\Tugas Kecil\Tucil3\_13520037\src>py Puzzle.py unsolveable

## Input 5.txt



D:\Education\Formal\Institut Teknologi Bandung\Akademik\Semester IV\Strategi Algoritma\Tugas\Tugas Kecil\Tucil3\_13520037\src>py Puzzle.py unsolveable

# D. ALAMAT DRIVE KODE PROGRAM

Alamat drive untuk kode program ini adalah:

# <u>AkmalArifin/IF2211-Tugas-Kecil-3-15-Puzzle</u>

# E. CEKLIST

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