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

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

readFile

```
11  def readFile(matrixPuzzle, fileName):
12      lines = []
13      dir = '../test/' + fileName
14      with open(dir, 'r') as f:
15          lines = f.readlines()
16
17      i = 0
18      for line in lines:
19          parse = line.split(' ', 4)
20          arrayPuzzle = []
21          arrayPuzzle.append(int(parse[0]))
22          arrayPuzzle.append(int(parse[1]))
23          arrayPuzzle.append(int(parse[2]))
24          arrayPuzzle.append(int(parse[3]))
25          matrixPuzzle.append(arrayPuzzle)
26
```

listToString

```
27  def listToString(li):
28      text = ""
29      for num in li:
30          text += str(num)
31          text += " "
32      text += "\n"
33      return text
34
```

writeFile

```
35  def writeFile(path, fileName, totalNode, time):
36      dir = '../test/' + fileName
37      f = open(dir, "w")
38
39      for matriks in path:
40          for row in matriks:
41              text = listToString(row)
42              f.write(text)
43          f.write("\n")
44
45      for row in GOAL_STATE:
46          text = listToString(row)
47          f.write(text)
48      f.write("\n")
49      f.write(f"Waktu yang dibutuhkan : {time}\n")
50      f.write(f"Jumlah node yang dibangktikan : {totalNode}\n")
51
```

estimateCost

```
53  def estimateCost(matrixPuzzle):
54      jumlahKurang = 0
55      for i in range(4):
56          for j in range(4):
57              if (matrixPuzzle[i][j] == 16):
58                  continue
59              if(matrixPuzzle[i][j] != GOAL_STATE[i][j]):
60                  jumlahKurang += 1
61      return jumlahKurang
62
```

kurang

```
63  def kurang(matrixPuzzle):
64      jumlahKurang = 0
65      for i in range(4):
66          for j in range(4):
67              num = matrixPuzzle[i][j]
68              for k in range(i, 4):
69                  for l in range(4):
70                      if (i == k and l < j):
71                          continue
72                      elif (num > matrixPuzzle[k][l]):
73                          jumlahKurang += 1
74      return jumlahKurang
75
```

isSolveable

```
76  def isSolveable(matrixPuzzle):
77      sum = kurang(matrixPuzzle)
78      for i in range(4):
79          for j in range(4):
80              if matrixPuzzle[i][j] == 16:
81                  sum += (i+j)%2
82      return sum % 2 == 0
83
```

Bangkitkan #1

```
85  def bangkitkan(queue, totalNode):
86      i0 = queue.getEmptyi()
87      j0 = queue.getEmptyj()
88      matrixRise = queue.getMatrix()
89      path = []
90      pathRise = queue.getPath()
91      if (len(pathRise) > 0):
92          for i in range(len(pathRise)):
93              tempPath = [[0 for j in range(4)] for i in range(4)]
94              for j in range(4):
95                  for k in range(4):
96                      tempPath[j][k] = pathRise[i][j][k]
97              path.append(tempPath)
98      path.append(queue.getMatrix())
99      queue.delete()
100
```

Bangkitkan #2 : Langkah ke Atas

```
101     try:
102         matrixUp = [[0 for j in range(4)] for i in range(4)]
103         for i in range(4):
104             for j in range(4):
105                 matrixUp[i][j] = matrixRise[i][j]
106         temp = matrixUp[i0][j0]
107         matrixUp[i0][j0] = matrixUp[i0-1][j0]
108         matrixUp[i0-1][j0] = temp
109
110         costUp = len(path) + estimateCost(matrixUp)
111
112         iUp = i0-1
113         jUp = j0
114         if (not matrixUp in path):
115             queue.insert(costUp, iUp, jUp, matrixUp, path)
116             totalNode += 1
117     except IndexError:
118         # print("index error")
119         pass
120
```

Bangkitkan #3 : Langkah ke Kanan

```
121     try:
122         matrixRight = [[0 for j in range(4)] for i in range(4)]
123         for i in range(4):
124             for j in range(4):
125                 matrixRight[i][j] = matrixRise[i][j]
126         temp = matrixRight[i0][j0]
127         matrixRight[i0][j0] = matrixRight[i0][j0+1]
128         matrixRight[i0][j0+1] = temp
129 
130         costRight = len(path) + estimateCost(matrixRight)
131 
132         iRight = i0
133         jRight = j0+1
134 
135         if (not matrixRight in path):
136             queue.insert(costRight, iRight, jRight, matrixRight, path)
137             totalNode += 1
138     except IndexError:
139         # print("index error")
140         pass
141 
```

Bangkitkan #4 : Langkah ke Bawah

```
142     try:
143         matrixDown = [[0 for j in range(4)] for i in range(4)]
144         for i in range(4):
145             for j in range(4):
146                 matrixDown[i][j] = matrixRise[i][j]
147         temp = matrixDown[i0][j0]
148         matrixDown[i0][j0] = matrixDown[i0+1][j0]
149         matrixDown[i0+1][j0] = temp
150 
151         costDown = len(path) + estimateCost(matrixDown)
152 
153         iDown = i0+1
154         jDown = j0
155 
156         if (not matrixDown in path):
157             queue.insert(costDown, iDown, jDown, matrixDown, path)
158             totalNode += 1
159     except IndexError:
160         # print("index error")
161         pass
162 
```

Bangkitkan #5 : Langkah ke Kiri

```
    try:
        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:
        # print("index error")
        pass

    return queue, totalNode

```

checkGoal

```
def checkGoal(matrixPuzzle):
    isSame = True
    for i in range(4):
        for j in range(4):
            if (matrixPuzzle[i][j] != GOAL_STATE[i][j]):
                isSame = False

    return isSame

```

branchAndBound

```
196  def branchAndBound(matrixPuzzle):
197      queue = PriorityQueue()
198      cost = estimateCost(matrixPuzzle)
199      totalNode = 1
200
201      for i in range(4):
202              for j in range(4):
203                  if matrixPuzzle[i][j] == 16:
204                      i0 = i
205                      j0 = j
206
207      queue.insert(cost, i0, j0, matrixPuzzle, [])
208      isFound = False
209      while(not isFound and not queue.isEmpty()):
210          queue,totalNode = bangkitkan(queue, totalNode)
211          if (checkGoal(queue.getMatrix())):
212              goalPath = queue.getPath()
213              isFound = True
214      if (isFound):
215          return goalPath, totalNode
216      else:
217          return [], totalNode
218
```

main

```
219  def main():
220      matrixPuzzle = []
221      readFile(matrixPuzzle, "3.txt")
222
223      if (isSolveable(matrixPuzzle)):
224          tStart = time.time()
225          goalPath, totalNode = branchAndBound(matrixPuzzle)
226          tEnd = time.time()
227          writeFile(goalPath, "solusi.txt", totalNode, tEnd-tStart)
228      else:
229          print("unsolveable")
230
231  main()
```

PriorityQueue.py #1

```python
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]:
                    min = i
            item = self.queue[min][0]
            return item
        except IndexError:
            print()
            exit()

```

PriorityQueue.py #2 : getMatrix

```python
    def getMatrix(self):
        try:
            min = 0
            for i in range(len(self.queue)):
                if self.queue[i][0] < self.queue[min][0]:
                    min = i
            item = self.queue[min][3]
            return item
        except IndexError:
            print()
            exit()

```

PriorityQueue.py #3 : getPath

```python
    def getPath(self):
        try:
            min = 0
            for i in range(len(self.queue)):
                if self.queue[i][0] < self.queue[min][0]:
                    min = i
            item = self.queue[min][4]
            return item
        except IndexError:
            print()
            exit()

```

PriorityQueue.py #4 : getEmptyi

```
47      def getEmptyi(self):
48          try:
49              min = 0
50              for i in range(len(self.queue)):
51                  if self.queue[i][0] < self.queue[min][0]:
52                      min = i
53              item = self.queue[min][1]
54              return item
55          except IndexError:
56              print()
57              exit()
58
```

PriorityQueue.py #5 : getEmptyj

```
59      def getEmptyj(self):
60          try:
61              min = 0
62              for i in range(len(self.queue)):
63                  if self.queue[i][0] < self.queue[min][0]:
64                      min = i
65              item = self.queue[min][2]
66              return item
67          except IndexError:
68              print()
69              exit()
70
```

PriorityQueue.py #6 : delete

```
71      def delete(self):
72          try:
73              min = 0
74              for i in range(len(self.queue)):
75                  if self.queue[i][0] < self.queue[min][0]:
76                      min = i
77              del self.queue[min]
78          except IndexError:
79              print()
80              exit()
81
```

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

```
2 5 3 4
1 10 6 8
9 16 7 11
13 14 15 12
```

Output solusi_1.txt

test > solusi_1.txt

```
2 5 3 4
1 10 6 8
9 16 7 11
13 14 15 12

2 5 3 4
1 16 6 8
9 10 7 11
13 14 15 12

2 16 3 4
1 5 6 8
9 10 7 11
13 14 15 12

16 2 3 4
1 5 6 8
9 10 7 11
13 14 15 12

1 2 3 4
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

test > 2.txt

```
2 16 3 4
1 5 6 8
9 10 7 11
13 14 15 12
```

Output solusi_2.txt

test > solusi_2.txt

```
2 16 3 4
1 5 6 8
9 10 7 11
13 14 15 12

16 2 3 4
1 5 6 8
9 10 7 11
13 14 15 12

1 2 3 4
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 : 0.0009846687316894531
Jumlah node yang dibangktikan : 22

```

Input 3.txt

test > 3.txt

```
1 2 3 4
5 6 7 8
10 13 11 12
9 16 14 15
```

Output solusi_3 .txt

test > solusi_3.txt

```
1 2 3 4
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

```

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 16 15

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

Waktu yang dibutuhkan : 0.0008742809295654297
Jumlah node yang dibangktikan : 21

```

Instansi Persoalan yang Tidak bisa diselesaikan

Input 4.txt

test > 4.txt

```
1   4 2 3 1
2   5 6 7 8
3   9 10 11 12
4   13 14 15 16
```

```
D:\Education\Formal\Institut Teknologi Bandung\Akademik\Semester IV\Strategi Algoritma\Tugas\Tugas Kecil\Tucil3_13520037\src>py Puzzle.py
unsolveable
```

Input 5.txt

test > 5.txt

```
1   1 4 2 3
2   5 10 13 11
3   12 14 15 6
4   16 8 7 9
```

```
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 :

AkmalArifin/IF2211-Tugas-Kecil-3-15-Puzzle

E. CEKLIST

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	√	
5. Bonus dibuat		√