Laporan Tugas Kecil 3 IF2211-Strategi Algoritma

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

Disusun Oleh:

Raden Haryosatyo Wisjnunandono

13520070

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2021/2022

BAB I

ALGORITMA BRANCH AND BOUND DALAM PERSOALAN 15-PUZZLE

1.1 Algoritma Branch and Bound Utama

Dalam persoalan penyelesaian 15-puzzle ini dapat digunakan algoritma branch and bound sebagai algoritma utama. Pada subbab ini akan dijelaskan algoritma utama branch and bound saja dan diasumsikan state awal adalah solusi dapat dicari (solveable). Untuk algoritma untuk menentukan apakah suatu puzzle dapat diselesaikan atau tidak akan dijelaskan pada subbab selanjutnya.

Algoritma branch and bound yang diimplementasikan dalam program ini adalah sebagai berikut:

1. Inisialisasi sebuah pohon ruang status dengan state awal sebagai rootnya.
2. Masukkan node root ke dalam antrian Q
3. Jika antrian Q kosong hentikan proses
4. Pilih dan keluarkan dari antrian Q node N yang memiliki cost untuk mencapai goal yang terkecil
5. Jika node N merupakan simpul solusi maka hentikan pencarian
6. Jika node N bukan merupakan solusi, bangkitkan anak-anaknya
    - Anak yang valid dari sebuah node adalah yang jika *empty slot*-nya digeser bentuk statenya tidak kembali ke state parent dari node dan movementnya valid (jika *empty slot* sudah berada di paling kanan puzzle maka tidak bisa digerakkan ke kanan)
7. Untuk setiap node anak M dari node N hitung cost untuk mencapai goal lalu hitung cost total nya dengan menjumlahkan cost menuju goal dengan depth dari node M.
8. Masukkan node M ke dalam antrian Q
9. Kembali ke langkah 3

**) note: Untuk algoritma penentuan solvabilitas dan penghitungan cost akan dijelaskan pada subbab selanjutnya.*

1.2 Algoritma Penghitungan Cost Sebuah Node Menuju Goal State

Pada subbab ini akan dijelaskan algoritma untuk mengetahui cost dari suatu node untuk menuju ke goal state. Adapun algoritmanya adalah sebagai berikut:

1. Buat sebuah larik 1 dimensi L
2. Masukkan elemen-elemen puzzle ke larik pada indeks i dengan rumus:
    - i = no baris*4 + no kolom, dengan i, no kolom, dan nomor baris dimulai dari 0
3. Inisialisasi varibel counter untuk menghitung banyaknya tile pada puzzle yang berada bukan pada tempatnya
4. Iterasi seluruh elemen pada larik, jika elemen larik di indeks i tidak sama dengan i+1 (kecuali untuk elemen larik di indeks i = 16), maka tambahkan 1 pada variabel counter
5. Setelah selesai melakukan iterasi, counter merupakan cost dari puzzle menuju goal state

6. Jumlahkan counter dengan depth dari node untuk mendapat total cost

1.3 Algoritma Pengecekan Solvabilitas Puzzle (Kurang(i) + X)

Pada subbab ini akan dijelskan algoritma pencarian kurang(i) dan penentuan X

1.3.1 Algoritma Kurang(i)

1. Buat sebuah larik 1 dimensi L
2. Masukkan elemen-elemen puzzle ke larik pada indeks i dengan rumus:
    - i = no baris*4 + no kolom, dengan i, no kolom, dan nomor baris dimulai dari 0
3. Inisialisasi varibel kurang untuk menghitung banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa.
4. Iterasi larik sebanyak 16 kali dengan iterator i, pada tiap iterasi tersebut iterasi larik lagi dari mulai i+1 sampai 16 dengan iterator j.
5. Untuk tiap iterasi j, tambahkan 1 pada variabel kurang jika terdapat ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa.

1.3.2 Algoritma Penentuan X

1. Tentukan letak tile kosong pada puzzle
2. Simpang posisi tile pada koordinat R,C dengan R adalah baris dan C adalah kolom tile
3. X adalah hasil dari penjumlahan R dan C yang dimodulo dengan 2

1.3.3 Algoritma Penentuan Solvabilitas

1. Jumlahkan variabel kurang pada subbab 1.3.1 dengan varibel X pada 1.3.2
2. Jika hasil penjumlahan genap maka puzzle tersebut dapat dipecahkan
3. Sebalikanya, jika hasil penjumlahan adalah ganjil maka puzzle tersebut tidak dapat dipecahkan.

BAB II

SOURCE CODE

2.1 Kelas Matrix Untuk Menyimpan Puzzle

```
class Matrix:
    #Konstruktor Kelas
    def __init__(self, source):
        self.arr = source
        #jika dibuat di awal(dari file)
        try:
            self.arr = []
            f = open("../test/"+source , "r")
            for line in f:
                self.arr.append(list(map(lambda x : int(x), line.split())))
        #jika dibuat dari pergerakan state sebelumnya
        except:
            self.arr = source

    def calculateCost(self):
        temp = self.toOneDim()
        misplaced = 0

        for i in range(16):
            #jika bukan 16 periksa apakah sudah benar posisinya
            if temp[i] != 16 and temp[i] != i+1:
                misplaced += 1

        return misplaced

    def printMatrix(self):
        for row in self.arr:
            for elem in row:
                if elem == 16:
                    elem = '#'
                print('%4s' % elem, end="")
            print()
```

```python
def toOneDim(self):
    temp = []
    for row in self.arr:
        for elem in row:
            temp.append(elem)
    return temp

def cekKurang(self):
    temp = self.toOneDim()
    kurang = 0
    for i in range(16):
        for j in range(i+1,16):
            if temp[j] < temp[i]:
                kurang+=1
    return kurang

def findEmptySlot(self):
    for i in range(4):
        for j in range(4):
            if self.arr[i][j] == 16:
                return (i, j)
```

```python
def cekSolvabilitas(self):
    kurang = self.cekKurang()
    x, y = self.findEmptySlot()
    par = (x+y) % 2
    total = kurang + par

    print("Kurang(i) =", kurang)
    print("X =", par)
    print("Total =", total, end="")

    isSolveable = total%2

    if (isSolveable == 0):
        print(" (even)\nSOLVEABLE!")
    else:
        print(" (odd)\nUNSOLVEABLE!")

    return isSolveable == 0
```

```
def moveEmpty(self, direction):
    dr = 0
    dc = 0
    if direction == "UP":
        dr = -1
    elif direction == "DOWN":
        dr = 1
    elif direction == "RIGHT":
        dc = 1
    elif direction == "LEFT":
        dc = -1

    row, col = self.findEmptySlot()
    newRow = row+dr
    newCol = col + dc

    if (newRow>=0 and newRow<4 and newCol>=0 and newCol<4):
        #Swap content in row, col w/ newRow, newCol
        nextStateMatrix =  copy.deepcopy(self.arr)
        temp = nextStateMatrix[newRow][newCol]
        nextStateMatrix[newRow][newCol] = 16
        nextStateMatrix[row][col] = temp
        return nextStateMatrix
    else:
        return None
```

2.2 Kelas Tree yang Menyimpang Pohon Ruang Status

```
class Tree:
    def __init__(self, matrix, parent=None, depth = 0, cost=0, move=""):
        self.parent = parent
        self.matrix = matrix
        self.depth = depth
        self.cost = cost
        self.move = move
```

2.3 Kelas PriorityQueue yang Menyimpan Antrian Q

```
class PriorityQueue(object):
    def __init__(self):
        self.queue = []

    # Periksa apakah queue kosong
    def isEmpty(self):
        return len(self.queue) == 0

    # Untuk memasukkan data ke queue
    def push(self, data):
        self.queue.insert(0, data)

    # Untuk mengeluarkan elemen dari queue
    def delete(self):
        try:
            min = 0
            for i in range(len(self.queue)):
                if self.queue[i].cost + self.queue[i].depth < self.queue[min].cost + self.queue[min].depth:
                    min = i
            item = self.queue[min]
            del self.queue[min]
            return item
        except IndexError:
            print()
            exit()
```

2.4 Main Program

```
import time
from tree import Tree
from priorqueue import PriorityQueue
from matrix import Matrix

# minta input testcase yang ingin dicoba kepada pengguna
doc = input("Enter File to Test: ")
print()

#periksa apakah puzzle sudah solved
def isCompleted(board):
    flatboard = board.toOneDim()

    for i in range(len(flatboard)):
        #solved jika elemen sudah terurut dari 1 sampai 16
        if flatboard[i]-1 != i:
            return False
    return True

directions = ["UP", "RIGHT", "DOWN", "LEFT"]

#inisialisasi board puzzle awal sebagai root node
root = Matrix(doc)

try:
    a = root.arr[0][1]
except:
    print("File not Exists!")
    exit()

root.printMatrix()

if not root.cekSolvabilitas():
    exit()
```

```
#inisialisasi priorityqueue
pq = PriorityQueue()
#inisialisasi state space tree
tree = Tree(root)

#masukkan root node ke tree
pq.push(tree)

#variable penyimpan node yang merupakan solusi
sol_node = None

completed = False
node_counter = 1
starttime = time.process_time_ns()
```

```
#main branch and bound algorithm
while not pq.isEmpty() and not completed:
    node = pq.delete()

    if (isCompleted(node.matrix)):
        sol_node = node
        completed = True

    else:
        for i,direction in enumerate(directions):
            #buat node anak yang arah pergerakannya bukan berlawanna dengan arah gerak node
            if (directions[(i+2)%4] != node.move):
                moved_puzzle = Matrix(node.matrix.moveEmpty(direction))
                #print(direction)
                #jika puzzle dapat digerakkan
                if (moved_puzzle != None and moved_puzzle.arr != None):
                #moved_puzzle.printMatrix()
                    goalCost = moved_puzzle.calculateCost()

                    child = Tree(moved_puzzle, node, node.depth+1, goalCost, direction)
                    #print("hello\n")
                    node_counter +=1
                    pq.push(child)
```

```
#telusuri pohon untuk mendapat langkah puzzle
startNode = sol_node
sol_array = []
print()
while (startNode != tree):
    sol_array.insert(0, startNode)
    startNode = startNode.parent

endtime = time.process_time_ns()


#output hasil
# print(len(sol_array))
move_monitor = ""
for i in range(len(sol_array)):
    print("STEP " + str(i+1) + ": " + sol_array[i].move)
    sol_array[i].matrix.printMatrix()
    move_monitor += sol_array[i].move[0] + " "
    print()

print("MOVES:", move_monitor)
print("Node generated:", node_counter)
print("time taken:", end="")
print(((endtime-starttime)/1000000), "ms")
```

BAB III

INPUT DAN OUTPUT

3.1 solveable_01.txt

Input File:

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

Output:

```
Enter File to Test: solveable_01.txt

   1   2   3   4
   5   6   7  11
   9  10  12   8
  13  14  15   #
Kurang(i) = 6
X = 0
Total = 6 (even)
SOLVEABLE!

STEP 1: LEFT
   1   2   3   4
   5   6   7  11
   9  10  12   8
  13  14   #  15

STEP 2: LEFT
   1   2   3   4
   5   6   7  11
   9  10  12   8
  13   #  14  15

STEP 3: UP
   1   2   3   4
   5   6   7  11
   9   #  12   8
  13  10  14  15

STEP 4: UP
   1   2   3   4
   5   #   7  11
   9   6  12   8
  13  10  14  15

STEP 5: RIGHT
   1   2   3   4
   5   7   #  11
   9   6  12   8
  13  10  14  15

STEP 6: RIGHT
   1   2   3   4
   5   7  11   #
   9   6  12   8
  13  10  14  15

STEP 7: DOWN
   1   2   3   4
   5   7  11   8
   9   6  12   #
  13  10  14  15

STEP 8: LEFT
   1   2   3   4
   5   7  11   8
   9   6   #  12
  13  10  14  15

STEP 9: UP
   1   2   3   4
   5   7   #   8
   9   6  11  12
  13  10  14  15

STEP 10: LEFT
   1   2   3   4
   5   #   7   8
   9   6  11  12
  13  10  14  15
```

```
STEP 11: DOWN
   1   2   3   4
   5   6   7   8
   9   #  11  12
  13  10  14  15

STEP 12: DOWN
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13   #  14  15

STEP 13: RIGHT
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13  14   #  15

STEP 14: RIGHT
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13  14  15   #

MOVES: L L U U R R D L U L D D R R
Node generated: 759
time taken:78.125 ms
```

3.2 solveable_02.txt

Input File:

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

Output:

```
Enter File to Test: solveable_02.txt

   1   2  12   3
   5   6   8   4
  13   9  11  15
  10   #   7  14
Kurang(i) = 26
X = 0
Total = 26 (even)
SOLVEABLE!

STEP 1: LEFT
   1   2  12   3
   5   6   8   4
  13   9  11  15
   #  10   7  14

STEP 2: UP
   1   2  12   3
   5   6   8   4
   #   9  11  15
  13  10   7  14

STEP 3: RIGHT
   1   2  12   3
   5   6   8   4
   9   #  11  15
  13  10   7  14

STEP 4: RIGHT
   1   2  12   3
   5   6   8   4
   9  11   #  15
  13  10   7  14
```

```
STEP 5: UP
   1   2  12   3
   5   6   #   4
   9  11   8  15
  13  10   7  14

STEP 6: UP
   1   2   #   3
   5   6  12   4
   9  11   8  15
  13  10   7  14

STEP 7: RIGHT
   1   2   3   #
   5   6  12   4
   9  11   8  15
  13  10   7  14

STEP 8: DOWN
   1   2   3   4
   5   6  12   #
   9  11   8  15
  13  10   7  14

STEP 9: LEFT
   1   2   3   4
   5   6   #  12
   9  11   8  15
  13  10   7  14

STEP 10: DOWN
   1   2   3   4
   5   6   8  12
   9  11   #  15
  13  10   7  14
```

```
STEP 11: DOWN
  1  2  3  4
  5  6  8 12
  9 11  7 15
 13 10  # 14

STEP 12: RIGHT
  1  2  3  4
  5  6  8 12
  9 11  7 15
 13 10 14  #

STEP 13: UP
  1  2  3  4
  5  6  8 12
  9 11  7  #
 13 10 14 15

STEP 14: UP
  1  2  3  4
  5  6  8  #
  9 11  7 12
 13 10 14 15

STEP 15: LEFT
  1  2  3  4
  5  6  #  8
  9 11  7 12
 13 10 14 15

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

```
STEP 17: LEFT
  1  2  3  4
  5  6  7  8
  9  #  11 12
 13 10 14 15

STEP 18: DOWN
  1  2  3  4
  5  6  7  8
  9 10 11 12
 13  #  14 15

STEP 19: RIGHT
  1  2  3  4
  9 10 11 12
 13 14  #  15

STEP 20: RIGHT
  1  2  3  4
  5  6  7  8
  9 10 11 12
 13 14 15  #

MOVES: L U R R U U R D L D D R U U L D L D R R
Node generated: 4371
time taken:1406.25 ms
```

3.3 solveable_03.txt

Input File:

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

Output:

```
Enter File to Test: solveable_03.txt

  1  2  3  4
  5  6  7  8
 11 12 15 14
 10  9 13  #
Kurang(i) = 12
X = 0
Total = 12 (even)
SOLVEABLE!

STEP 1: UP
  1  2  3  4
  5  6  7  8
 11 12 15  #
 10  9 13 14

STEP 2: LEFT
  1  2  3  4
  5  6  7  8
 11 12  #  15
 10  9 13 14

STEP 3: LEFT
  1  2  3  4
  5  6  7  8
 11  #  12 15
 10  9 13 14

STEP 4: LEFT
  1  2  3  4
  5  6  7  8
  #  11 12 15
 10  9 13 14
```

```
STEP 5: DOWN
  1  2  3  4
  5  6  7  8
 10 11 12 15
  #  9 13 14

STEP 6: RIGHT
  1  2  3  4
  5  6  7  8
 10 11 12 15
  9  # 13 14

STEP 7: RIGHT
  1  2  3  4
  5  6  7  8
 10 11 12 15
  9 13  # 14

STEP 8: RIGHT
  1  2  3  4
  5  6  7  8
 10 11 12 15
  9 13 14  #

STEP 9: UP
  1  2  3  4
  5  6  7  8
 10 11 12  #
  9 13 14 15

STEP 10: LEFT
  1  2  3  4
  5  6  7  8
 10 11  #  12
  9 13 14 15
```

```
STEP 11: LEFT
   1   2   3   4
   5   6   7   8
  10   #  11  12
   9  13  14  15

STEP 12: LEFT
   1   2   3   4
   5   6   7   8
   #  10  11  12
   9  13  14  15

STEP 13: DOWN
   1   2   3   4
   5   6   7   8
   9  10  11  12
   #  13  14  15

STEP 14: RIGHT
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13   #  14  15

STEP 15: RIGHT
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13  14   #  15

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

MOVES: U L L L D R R R U L L L D R R R
Node generated: 857
time taken:62.5 ms
```

3.4 unsolveable_01.txt

Input File:

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

Output:

```
Enter File to Test: unsolveable_01.txt

   1   3   4  15
   2   #   5  12
   7   6  11  14
   8   9  10  13
Kurang(i) = 37
X = 0
Total = 37 (odd)
UNSOLVEABLE!
```

3.5 unsolveable_02.txt

Input File:

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

Output:

```
Enter File to Test: unsolveable_02.txt

   1   2  14   3
   5   6   4   7
  15  10  11   #
   9  13   8  12
Kurang(i) = 30
X = 1
Total = 31 (odd)
UNSOLVEABLE!
```

BAB IV

KESIMPULAN

Permasalahan persoalan penyelesaian 15-puzzle dapat diselesaikan dengan menerapkan algoritma branch and bound. Tiap-tiap state board puzzle dapat dianggap sebagai node dari sebuah pohon ruang status. Anak yang valid dari sebuah node adalah yang jika empty slot-nya digeser bentuk statenya tidak kembali ke state parent dari node dan movementnya valid (jika empty slot sudah berada di paling kanan puzzle maka tidak bisa digerakkan ke kanan). Tiap pembangkitan node baru kita hitung cost dari node tersebut unutk mencapai goal state lalu masukkan node tersebut ke dalam antrian. Pengeluaran elemen dari antrian didasarkan pada cost dari node. Semakin kecil cost nya semakin diprioritaskan.

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		

Link drive:

https://drive.google.com/drive/folders/1ghl4MWQxBQ1F1s99ftQXORM46idzaHCY?usp=sharing

(buka dengan akun std)

Link GitHub

https://github.com/nandono206/15-Puzzle-Branch-and-Bound