LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA

Penyelesaian *Word Search Puzzle* dengan Algoritma *Brute Force*

Nama : Ghebyon Tohada Nainggolan
NIM : 13520079
Kelas : K-01
Bahasa : Python

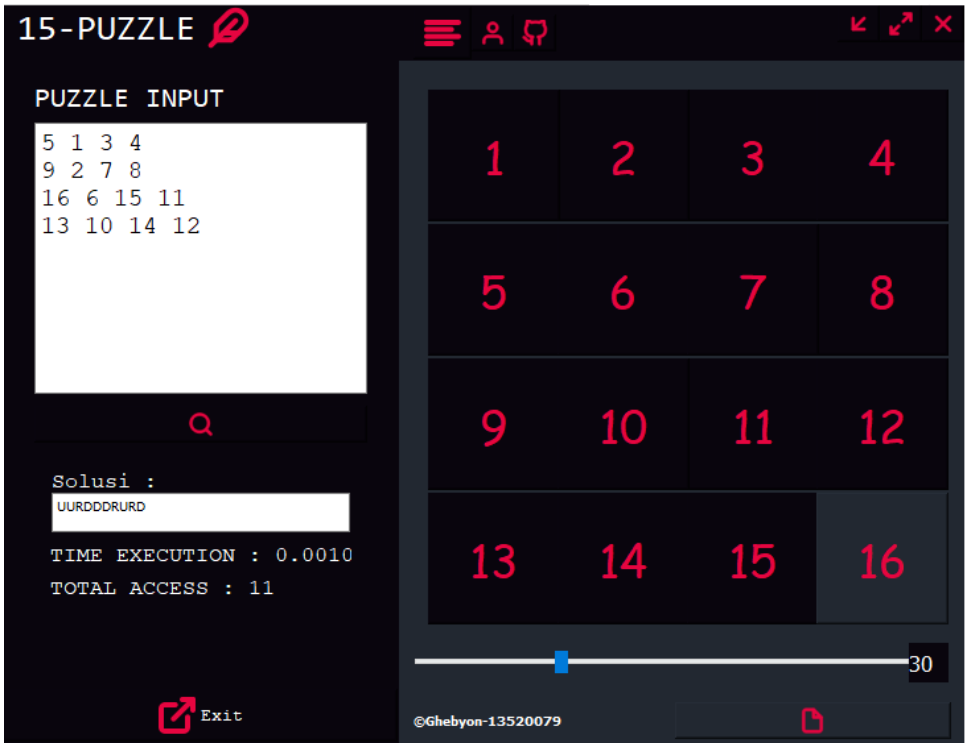
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021

A. ALGORITMA BRANCH AND BOUND

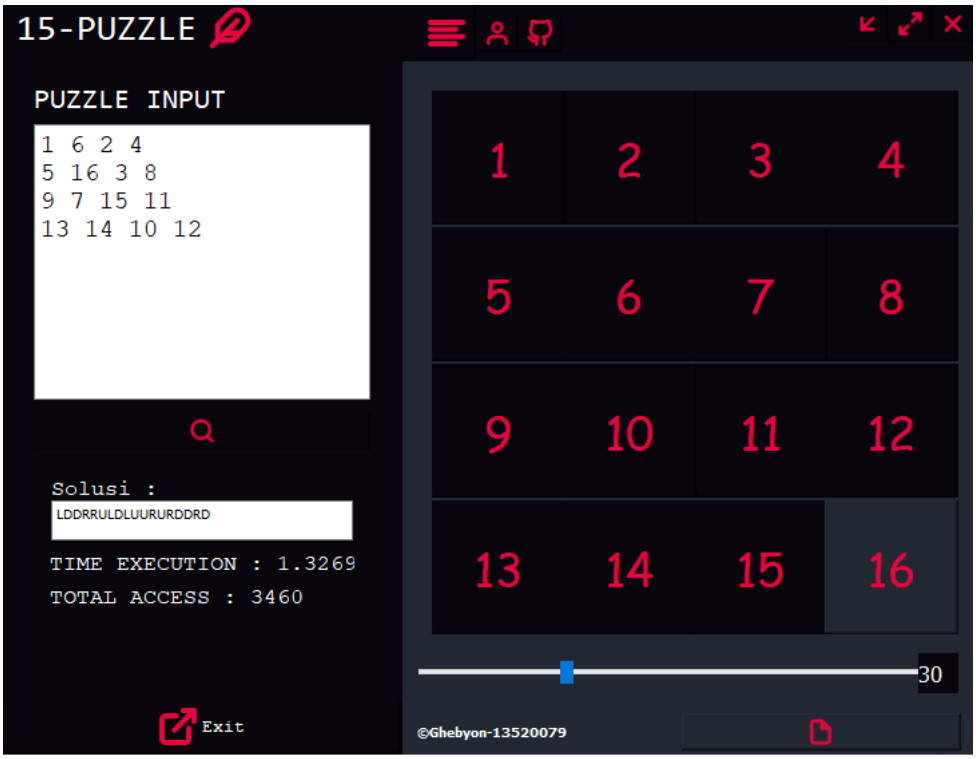
1. Sediakan list notVisited dan list visited.
2. Input suatu Puzzle dan dijadikan sebagai root Puzzle. Masukkan root Puzzle ke dalam list notVisited
3. Untuk setiap Puzzle pada list notVisited, hitung costnya dengan heuristik
   $c(x) = f(x) + g(x)$
   Keterangan :
   - $c(x)$ = cost untuk simpul x
   - $f(x)$ = cost untuk mencapai simpul x dari akar
   - $g(x)$ = taksiran panjang lintasan terpendek dari P ke simpul solusi

   Pilih Puzzle dengan cost terendah sebagai currentPuzzle
4. Periksa apakah puzzle merupakan solusi atau tidak
   4.1 Jika ya, maka pencarian dihentikan
   4.2 Jika tidak, lanjutkan
5. Tentukan pergerakan selanjutnya yang memungkinkan. Syarat pergerakan yang memungkinkan :
   - Slot kosong pada puzzle bergerak ke arah kiri, kanan, atas, atau bawah.
   - Pergerakan tersebut tidak menghasilkan Puzzle yang sudah pernah dikunjungi (tidak terdapat pada list visited)
6. Hapus currentPuzzle dari list notVisited dan masukkan ke dalam listVisited
7. Kembali ke langkah 2

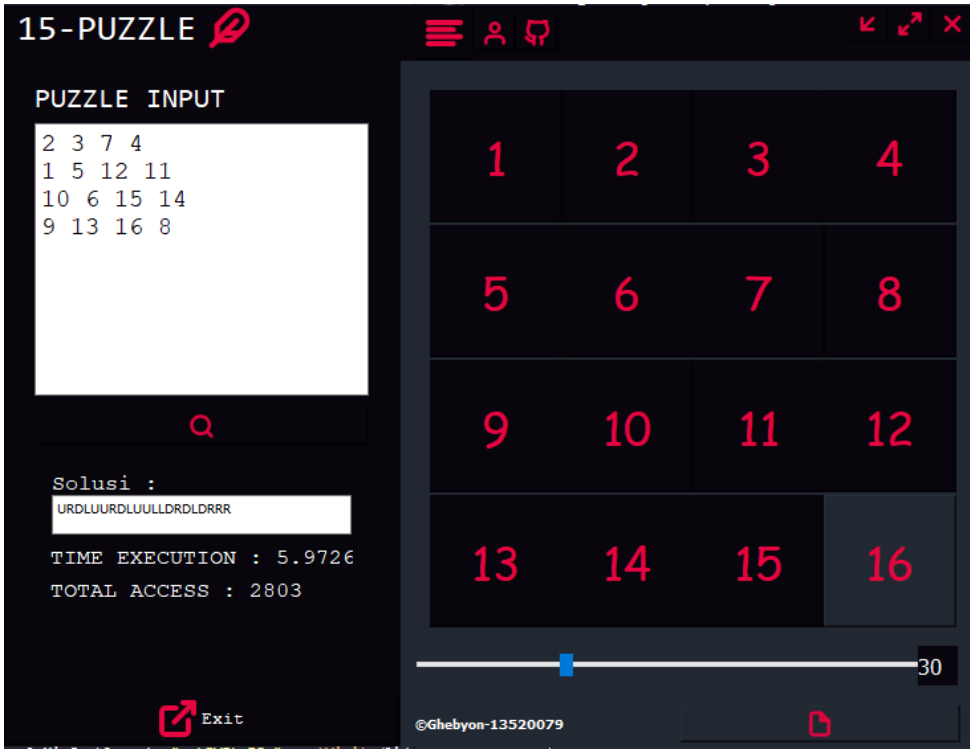
B. INPUT DAN OUTPUT



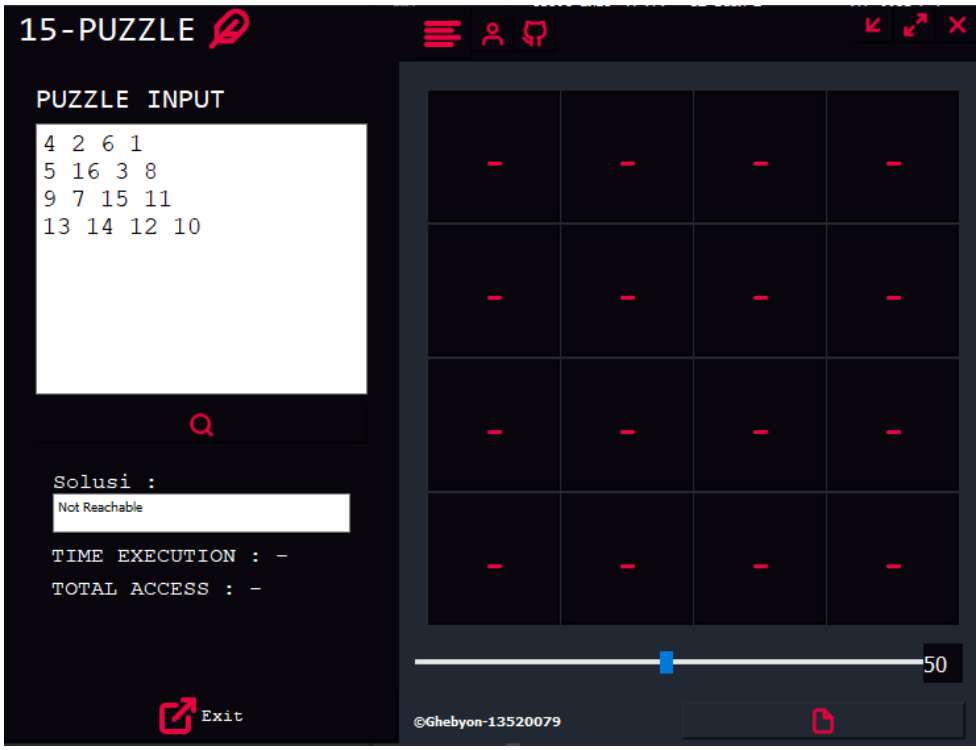
Gambar 1 Test Case Reachable 1



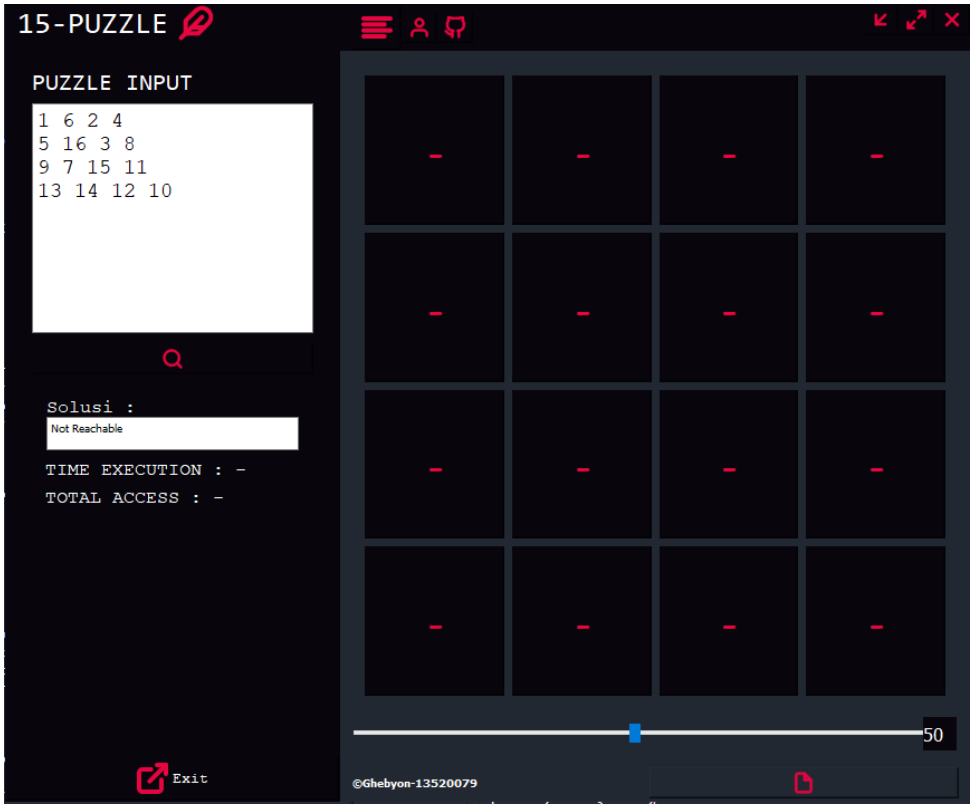
Gambar 2 Test Case Reachable 2



Gambar 3 Test Case Reachable 3



Gambar 4 Test Case Not Reachable 1



Gambar 5 Test Case Not Reachable 2

C. GITHUB

https://github.com/ghebyon/Tucil03-13520079

D. PENILAIAN MANDIRI

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	✓	

E. KODE PROGRAM

KELAS PUZZLE

```
class Puzzle:
    def __init__(self, level : int, puzzle : List, solusi):
        self.level = level
        self.puzzle = puzzle
        g_cost = 0
        for i in range(16):
            if(self.puzzle[i] != i+1 and self.puzzle[i] != 16):
                g_cost += 1
        self.cost = self.level + g_cost
        self.solusi = solusi    #List of Char : 'L', 'R', 'U', 'D'

    def reachable(self):
        count = 0
        for i in range(15):
            for j in range(i+1,16,1):
                if(self.puzzle[j] < self.puzzle[i]):
                    count += 1
        for i in range(15):
            if(self.puzzle[i] == 16 and i in [1,3,4,6,9,11,12,14]):
                count += 1
        if(count%2 == 0):
            return True
        else:
            return False

    def possibleMove(self):
        moveSet = ['L', 'U', 'D', 'R']
        blankPos = -1
        for i in range(16):
            if (self.puzzle[i] == 16):
                blankPos = i
                if (i%4 == 0):
                    moveSet.remove('L')
                if (i//4 == 0):
                    moveSet.remove('U')
                if (i//4 == 3):
```

```
                    moveSet.remove('D')
                if ((i+1)%4 == 0):
                    moveSet.remove('R')
        move_puzzle = {}
        for move in moveSet:
            tempPuzzle = self.puzzle[:]
            if(move == 'L'):
                tempValue = tempPuzzle[blankPos]
                tempPuzzle[blankPos] = tempPuzzle[blankPos-1]
                tempPuzzle[blankPos-1] = tempValue
            if(move == 'U'):
                tempValue = tempPuzzle[blankPos]
                tempPuzzle[blankPos] = tempPuzzle[blankPos-4]
                tempPuzzle[blankPos-4] = tempValue
            if(move == 'D'):
                tempValue = tempPuzzle[blankPos]
                tempPuzzle[blankPos] = tempPuzzle[blankPos+4]
                tempPuzzle[blankPos+4] = tempValue
            if(move == 'R'):
                tempValue = tempPuzzle[blankPos]
                tempPuzzle[blankPos] = tempPuzzle[blankPos+1]
                tempPuzzle[blankPos+1] = tempValue
            move_puzzle.update({move : tempPuzzle})
        return move_puzzle

    def isSolution(self):
        for i in range(16):
            if(self.puzzle[i] != i+1):
                return False
        return True
```

ALGORITMA BRANCH AND BOUND

```
def BranchnBound(notVisited : list): #parameter berisi list of object
Puzzle
    if(len(notVisited) > 0):
        visited = []
        countAccess = 0
```

```
        while(True):
            greaterLevel = notVisited[0].level
            min = notVisited[0].cost
            idxPuzzleMinCost = 0
            for i in range (len(notVisited)):
                if min > notVisited[i].cost:
                    min = notVisited[i].cost
                    greaterLevel = notVisited[i].level
                    idxPuzzleMinCost = i
                elif min == notVisited[i].cost:
                    if greaterLevel < notVisited[i].level:
                        min = notVisited[i].cost
                        greaterLevel = notVisited[i].level
                        idxPuzzleMinCost = i
            countAccess += 1
            a = 0
            for i in notVisited[idxPuzzleMinCost].puzzle:
                if i==16:
                    print(0,"\t", end="")
                else:
                    print(i,"\t", end="")
                if ((a+1)%4 == 0):
                    print()
                a += 1
            print("COST IS: ",notVisited[idxPuzzleMinCost].cost, ",
LEVEL IS:", notVisited[idxPuzzleMinCost].level)
            if (notVisited[idxPuzzleMinCost].isSolution()):
                finalSolution = notVisited[idxPuzzleMinCost].solusi
                return countAccess, finalSolution
            else:
                dictPossibleMove =
notVisited[idxPuzzleMinCost].possibleMove()
                for items in dictPossibleMove.items():
                    if(items[1] not in visited):
                        newLevel = notVisited[idxPuzzleMinCost].level +
1
                        newSolution =
notVisited[idxPuzzleMinCost].solusi + items[0]
                        newPuzzle =
```

```
Puzzle(newLevel,items[1],newSolution)
                        notVisited.append(newPuzzle)
                trashPuzzle = notVisited.pop(idxPuzzleMinCost)
                visited.append(trashPuzzle.puzzle)
```

F. TEST CASE

Reachable 1

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

Reachable 2

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

Reachable 3

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

Not Reachable 1

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

Not Reachable 2

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