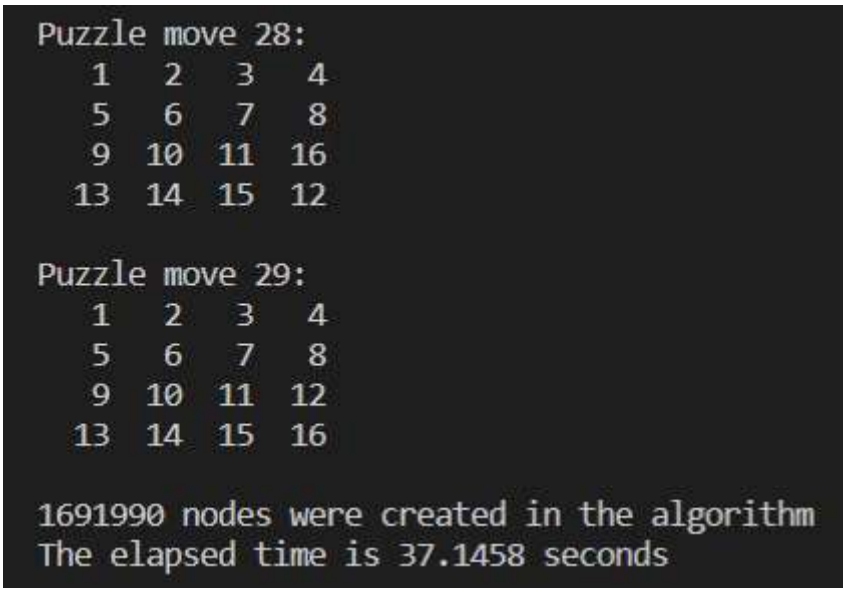
LAPORAN TUGAS KECIL 3

Mata Kuliah IF2211 Strategi Algoritma



Nama Penulis:

Aji Andhika Falah 13520012

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

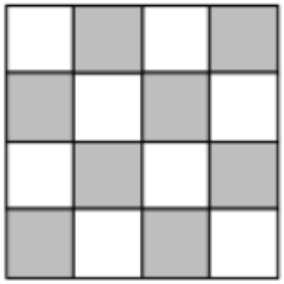
BAB I

Algoritma *Branch and Bound*

Algoritma *Branch and Bound* adalah algoritma yang membagi suatu masalah menjadi masalah-masalah yang lebih kecil (*Branch*) dengan sebuah batasan (*Bound*) untuk mencapai solusi optimal. Algoritma ini membentuk sebuah pohon ruang status, yang setiap simpul diberi nilai *cost*. Algoritma ini akan selalu mengolah simpul yang memiliki *cost* terendah agar mendapatkan solusi optimal.

Untuk mencari solusi optimal dari sebuah 15-*puzzle* akan digunaka algoritma *Branch and Bound* dengan langkah-langkah:

1. Sebelum masuk algoritma hitung $\sum_{i=1}^{16} Kurang(i) + X$, dengan Kurang(i) adalah banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa. Sedangkan X akan bernilai 1 jika tile kosong (di dalam program adalah tile bernomor 16) berada di tile yang diarsir, dan 0 jika tidak.

   Jika $\sum_{i=1}^{16} Kurang(i) + X$ bernilai ganjil, maka puzzle tidak bisa diselesaikan. Sedangkan jika bernilai genap, maka puzzle bisa diselesaikan. Jika tidak bisa diselesaikan, maka berhenti.
2. Masukkan simpul akar ke dalam antrian Q. Q adalah sebuah *priority queue*. Jika simpul akar adalah memiliki *current puzzle state* yang merupakan solusi *puzzle* (*goal node*), maka solusi telah ditemukan. Setiap simpul akan berisi *cost*, *level*, *address to parent node*, *current puzzle state*, dan *move* yang dilakukan. *cost* di node adalah jumlah tile yang tidak seperti di solusi *puzzle*, sehingga Q akan memiliki priority sesuai dengan *cost + level*.
3. Jika Q kosong, Stop.
4. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai nilai ‘cost’ ĉ(i) paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu.
5. Jika simpul i memiliki *current puzzle state* yang memiliki solusi *puzzle*, berarti solusi sudah ditemukan.
6. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Anaknya adalah semua gerakan yang mungkin dan bukan kebalikan dari gerakan sebelumnya.
7. Untuk setiap anak j dari simpul i, hitung ĉ(j), dan masukkan semua anak-anak tersebut ke dalam Q.
8. Kembali ke langkah 3.

BAB 2

Source Program

Program ini ditulis dalam bahasa Python. Terdapat 2 file, main.py dan BnB.py. File main.py akan menerima input dan mengirimkan informasi untuk diolah, sedangkan BnB.py berisi fungsi dan prosedur untuk algoritma *Branch and Bound* tersebut.

File main.py

```
import BnB
import os.path

loop = True

while (loop):
    puzzle = []
    kurang = 1

    option = input("Read File or Randomized?\n(1) Read File\n(2)
Randomized\nInput: ")
    print("")
    while (option != "1" and option != "2"):
        option = input("Input error, Read File or Randomized?\n(1) Read
File\n(2) Randomized\nInput: ")
        print("")

    if (int(option) == 1):
        puzzle = BnB.readFile()

        kurang = BnB.printPuzzleKurang(puzzle)

    elif (int(option) == 2):
        repeat = True
        puzzle = BnB.randomPuzzle()
        kurang = BnB.printPuzzleKurang(puzzle)

        while (repeat):
            option2 = input("Is this okay? (Y/N): ")
            if (option2 == "Y" or option2 == "y"):
                repeat = False
            elif (option2 == "N" or option2 == "n"):
                puzzle = BnB.randomPuzzle()
                kurang = BnB.printPuzzleKurang(puzzle)
            else:
                print("\nInput error")
        print("")

    if (kurang%2 == 0):
```

```
        print("Loading...\n")
        BnB.solve(puzzle)
    else:
        print("\nPuzzle cannot be solved\n")

    option3 = input("Want to do another? (Y/N): ")
    repeat = True
    while (repeat):
        if (option3 == "Y" or option3 == "y"):
            repeat = False
        elif (option3 == "N" or option3 == "n"):
            loop = False
            repeat = False
        else:
            option3 = input("Input error. Want to do another? (Y/N): ")
    print("")
```

File BnB.py

```
import os
import random
import time
from heapq import heappush, heappop
from tokenize import Token


# Class for priority queue
class PQueue:

    # Initiate a Priority Queue
    def __init__(self):
        self.heap = []

    # Push into the queue
    def push(self, k):
        heappush(self.heap, k)

    # Pop from the queue
    def pop(self):
        return heappop(self.heap)

    # Check if the queue is empty
    def empty(self):
        if not self.heap:
            return True
        else:
            return False
```

```
# Class for nodes
class Node:

    # Initiate a node
    def __init__(self, cost, level, parent, puzzle, move):
        self.cost = cost
        self.level = level
        self.parent = parent
        self.puzzle = puzzle
        self.move = move

    # overide the function Less Than (<) for the priority queue
    def __lt__(self, nxt):
        return self.cost + self.level < nxt.cost + nxt.level

# Function for reading files
def readFile():
    puzzle = []
    puzzleRow = []

    filename = input("Please input filename: ")
    path = os.path.join((os.path.abspath(os.path.join(os.getcwd(),
os.pardir))), "test", filename)
    while (not os.path.exists(path)):
        filename = input("\nFilename not found. Please reinput filename: ")
        path = os.path.join((os.path.abspath(os.path.join(os.getcwd(),
os.pardir))), "test", filename)

    file = open(path).read()
    array = file.split('\n')
    for i in range (len(array)):
        elements = array[i].split(" ")
        puzzleRow = []
        for j in range (len(elements)):
            puzzleRow.append(int(elements[j]))
        puzzle.append(puzzleRow)

    return (puzzle)

# Function to randomized a puzzle
def randomPuzzle():
    puzzle = []
    puzzleRow = []
    check = [False for i in range (16)]

    for i in range (4):
        puzzleRow = []
```

```
        for j in range(4):
            n = random.randint(1,16)
            while (check[n-1]):
                n = random.randint(1,16)
            check[n-1] = True
            puzzleRow.append(n)
        puzzle.append(puzzleRow)

    return (puzzle)

# Function that counts KURANG(i) + X
def kurang(puzzle) :
    kurang = 0
    before = 0
    print("KURANG(i):")
    for e in range(1, len(puzzle) * len(puzzle[0]) + 1):
        found = False
        grid = False
        ii = 0
        jj = 0
        while (ii < len(puzzle) and not found):
            jj=0
            while (jj < len(puzzle[0]) and not found):
                if (puzzle[ii][jj] == e):
                    found = True
                else:
                    jj += 1
            if (not found):
                ii += 1
        for i in range (len(puzzle)):
            for j in range (len(puzzle[i])):
                if (puzzle[i][j] < e and ((i > ii) or (i == ii and j > jj))) :
                    kurang += 1
                if (e == 16 and puzzle[i][j] == 16 and ((i%2 == 1 and j%2 ==
0) or (i%2 == 0 and j%2 == 1))):
                    kurang += 1
                    grid = True
        if (grid):
            print("Tile " + str(e) + ": " + str(kurang - before - 1))
            before = kurang
        else:
            print("Tile " + str(e) + ": " + str(kurang - before))
            before = kurang

    return kurang

# Function that prints out the puzzle and KURANG(i) + X
def printPuzzleKurang(puzzle):
```

```
    print("Puzzle:")
    x = '\n'.join([''.join(['{:4}'.format(element) for element in row]) for
row in puzzle])
    print(x)

    print("")
    nkurang = kurang(puzzle)
    print("\nKURANG(i) + X =", nkurang,"\n")

    return nkurang


# Function that counts how many tiles are not in the correct places
def countWrong(puzzle):
    cost = 0

    for i in range (len(puzzle)):
        for j in range (len(puzzle[0])):
            if (puzzle[i][j] != (i*len(puzzle[0]) + j + 1)):
                cost += 1
    return (cost)

# Function that search the location of the empty tile (16 tile)
def search16(puzzle):
    i = 0
    j = 0
    found = False

    while (i < len(puzzle) and not found):
        j = 0
        while (j < len(puzzle[0]) and not found):
            if (puzzle[i][j] == 16):
                found = True
            else:
                j += 1
        if (not found):
            i += 1

    return ([i, j])

# Function that switch a tile with an empty tile
def switchEmpty(puzzle, i, j):
    empty = search16(puzzle)
    tileSwitched = [empty[0] + i, empty[1] + j]
    temp = puzzle[empty[0]][empty[1]]
    puzzle[empty[0]][empty[1]] = puzzle[tileSwitched[0]][tileSwitched[1]]
    puzzle[tileSwitched[0]][tileSwitched[1]] = temp
```

```
    return (puzzle)

# Function that prints out the puzzle movements from start to finish
def printFinal(node):
    i = 0

    if (node.parent != None):
        i = printFinal(node.parent)
        i += 1

    print("Puzzle move " + str(i) + ": " )
    x = '\n'.join(['' .join(['{:4}'.format(element) for element in row]) for
row in node.puzzle])
    print(x, "\n")

    return (i)

# Function that checks if a move is the reverse of the move before that
def notReverse(i, j):
    return (not((i == 0 and j == 2) or (i == 2 and j == 0) or (i == 1 and j ==
3) or (i == 3 and j == 1)))

# Procedure that solves the puzzle with Branch and Bound
def solve(puzzle):
    startTimer = time.perf_counter()

    finalPuzzle = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15,
16]]
    rowMove = [ 1, 0, -1, 0 ]  # Empty slot move 0 = down, 1 = left, 2 = up, 3
= right
    colMove = [ 0, -1, 0, 1 ]
    pqueue = PQueue()
    totalNodes = 1

    # Make root node
    root = Node(countWrong(puzzle), 0, None, puzzle, None)
    pqueue.push(root)

    # Gets node with the lowest cost
    while (not pqueue.empty()):
        pnode = pqueue.pop()
        current = pnode.puzzle
        if (current == finalPuzzle):
            break
        else:
            # Move the empty tile to 4 different ways
            for i in range (4):
                empty = search16(current)
```

```
                # Check if move is valid
                if (empty[0] + rowMove[i] >= 0 and empty[0] + rowMove[i] <
len(current) and empty[1] + colMove[i] >= 0 and empty[1] + colMove[i] <
len(current[0]) and notReverse(i, pnode.move)):
                    # Make the moved puzzle
                    switchedPuzzle = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11,
12], [13, 14, 15, 16]]
                    for ii in range (len(current)):
                        for jj in range (len(current[0])):
                            switchedPuzzle[ii][jj] = current[ii][jj]
                    switchedPuzzle = switchEmpty(switchedPuzzle, rowMove[i],
colMove[i])
                    # Make the node and push it to priority queue
                    child = Node(countWrong(switchedPuzzle), pnode.level + 1,
pnode, switchedPuzzle, i)
                    pqueue.push(child)
                    totalNodes += 1

    endTimer = time.perf_counter()
    printFinal(pnode)
    print(totalNodes, "nodes were created in the algorithm")
    print(f"The elapsed time is {endTimer - startTimer:0.4f} seconds")
    print("")
```

BAB 3

Screenshot input dan *output*

1. failed1.txt

input:

```
Read File or Randomized?
(1) Read File
(2) Randomized
Input: 1

Please input filename: failed1.txt
```

output:

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

KURANG(i):
Tile 1: 0
Tile 2: 0
Tile 3: 2
Tile 4: 0
Tile 5: 0
Tile 6: 4
Tile 7: 3
Tile 8: 4
Tile 9: 5
Tile 10: 3
Tile 11: 7
Tile 12: 9
Tile 13: 9
Tile 14: 7
Tile 15: 2
Tile 16: 8

KURANG(i) + X = 63


Puzzle cannot be solved

Want to do another? (Y/N):
```

2. failed2.txt

input:

```
Read File or Randomized?
(1) Read File
(2) Randomized
Input: 1

Please input filename: failed2.txt
```

output:

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

KURANG(i):
Tile 1: 0
Tile 2: 0
Tile 3: 1
Tile 4: 0
Tile 5: 4
Tile 6: 1
Tile 7: 1
Tile 8: 0
Tile 9: 0
Tile 10: 4
Tile 11: 5
Tile 12: 11
Tile 13: 1
Tile 14: 3
Tile 15: 4
Tile 16: 11

KURANG(i) + X = 47


Puzzle cannot be solved

Want to do another? (Y/N):
```

3. success1.txt

input:

```
Read File or Randomized?
(1) Read File
(2) Randomized
Input: 1

Please input filename: success1.txt
```

output:

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

KURANG(i):
Tile 1: 0
Tile 2: 0
Tile 3: 2
Tile 4: 2
Tile 5: 1
Tile 6: 0
Tile 7: 0
Tile 8: 5
Tile 9: 0
Tile 10: 1
Tile 11: 1
Tile 12: 0
Tile 13: 1
Tile 14: 1
Tile 15: 5
Tile 16: 15

KURANG(i) + X = 34

Loading...
```

```
Puzzle move 0:
 16   3   4   8
  1   5   2   6
  7  10  15  11
  9  13  14  12

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Puzzle move 17:
   1   2   3   4
   5   6  16   8
   9  10   7  11
  13  14  15  12

Puzzle move 18:
   1   2   3   4
   5   6   7   8
   9  10  16  11
  13  14  15  12

Puzzle move 19:
   1   2   3   4
   5   6   7   8
   9  10  11  16
  13  14  15  12

Puzzle move 20:
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13  14  15  16

1776 nodes were created in the algorithm
The elapsed time is 0.0264 seconds
```

```
Want to do another? (Y/N):
```

4. success2.txt

input:

```
Read File or Randomized?
(1) Read File
(2) Randomized
Input: 1

Please input filename: success2.txt
```

output:

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

KURANG(i):
Tile 1: 0
Tile 2: 0
Tile 3: 0
Tile 4: 0
Tile 5: 1
Tile 6: 1
Tile 7: 0
Tile 8: 3
Tile 9: 4
Tile 10: 1
Tile 11: 4
Tile 12: 1
Tile 13: 4
Tile 14: 2
Tile 15: 2
Tile 16: 14

KURANG(i) + X = 38

Loading...
```

```
Puzzle move 0:
  1  16   2   3
  5   9   8  11
  6  13   4  10
 14  15  12   7

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Puzzle move 17:
   5   1   2   3
   6   8   4  16
   9  10   7  11
  13  14  15  12

Puzzle move 18:
   5   1   2   3
   6   8  16   4
   9  10   7  11
  13  14  15  12

Puzzle move 19:
   5   1   2   3
   6  16   8   4
   9  10   7  11
  13  14  15  12

Puzzle move 20:
   5   1   2   3
  16   6   8   4
   9  10   7  11
  13  14  15  12
```

```
Puzzle move 21:
 16   1   2   3
  5   6   8   4
  9  10   7  11
 13  14  15  12

Puzzle move 22:
  1  16   2   3
  5   6   8   4
  9  10   7  11
 13  14  15  12

Puzzle move 23:
  1   2  16   3
  5   6   8   4
  9  10   7  11
 13  14  15  12

Puzzle move 24:
  1   2   3  16
  5   6   8   4
  9  10   7  11
 13  14  15  12

Puzzle move 25:
  1   2   3   4
  5   6   8  16
  9  10   7  11
 13  14  15  12

Puzzle move 26:
  1   2   3   4
  5   6  16   8
  9  10   7  11
 13  14  15  12

Puzzle move 27:
  1   2   3   4
  5   6   7   8
  9  10  16  11
 13  14  15  12
```

```
Puzzle move 28:
   1   2   3   4
   5   6   7   8
   9  10  11  16
  13  14  15  12

Puzzle move 29:
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13  14  15  16

1691990 nodes were created in the algorithm
The elapsed time is 37.3939 seconds

Want to do another? (Y/N): 
```

5. success3.txt

input:

```
Read File or Randomized?
(1) Read File
(2) Randomized
Input: 1

Please input filename: success3.txt
```

output:

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

KURANG(i):
Tile 1: 0
Tile 2: 0
Tile 3: 0
Tile 4: 0
Tile 5: 4
Tile 6: 3
Tile 7: 3
Tile 8: 1
Tile 9: 8
Tile 10: 3
Tile 11: 3
Tile 12: 2
Tile 13: 12
Tile 14: 9
Tile 15: 6
Tile 16: 3

KURANG(i) + X = 58

Loading...
```

```
KURANG(i) + X = 58

Loading...

Puzzle move 0:
  13   9   5   1
  14   6   7   2
  15  10  11   3
  16  12   8   4

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Puzzle move 17:
   5   1  16   2
   9   6   7   3
  13  10  11   4
  14  15  12   8

Puzzle move 18:
   5   1   2  16
   9   6   7   3
  13  10  11   4
  14  15  12   8

Puzzle move 19:
   5   1   2   3
   9   6   7  16
  13  10  11   4
  14  15  12   8
```

```
Puzzle move 20:
   5   1   2   3
   9   6   7   4
  13  10  11  16
  14  15  12   8

Puzzle move 21:
   5   1   2   3
   9   6   7   4
  13  10  11   8
  14  15  12  16

Puzzle move 22:
   5   1   2   3
   9   6   7   4
  13  10  11   8
  14  15  16  12

Puzzle move 23:
   5   1   2   3
   9   6   7   4
  13  10  11   8
  14  16  15  12

Puzzle move 24:
   5   1   2   3
   9   6   7   4
  13  10  11   8
  16  14  15  12

Puzzle move 25:
   5   1   2   3
   9   6   7   4
  16  10  11   8
  13  14  15  12

Puzzle move 26:
   5   1   2   3
  16   6   7   4
   9  10  11   8
  13  14  15  12
```

```
Puzzle move 27:
 16   1   2   3
  5   6   7   4
  9  10  11   8
 13  14  15  12

Puzzle move 28:
  1  16   2   3
  5   6   7   4
  9  10  11   8
 13  14  15  12

Puzzle move 29:
  1   2  16   3
  5   6   7   4
  9  10  11   8
 13  14  15  12

Puzzle move 30:
  1   2   3  16
  5   6   7   4
  9  10  11   8
 13  14  15  12

Puzzle move 31:
  1   2   3   4
  5   6   7  16
  9  10  11   8
 13  14  15  12

Puzzle move 32:
  1   2   3   4
  5   6   7   8
  9  10  11  16
 13  14  15  12

Puzzle move 33:
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  14  15  16
```

```
7224480 nodes were created in the algorithm
The elapsed time is 158.3336 seconds

Want to do another? (Y/N): n
```

BAB 4

GitHub Program dan Berkas Teks

Untuk program dan berkas teks saya, bisa dilihat di link berikut:

https://github.com/Enderageous/Tucil3_13520012

Lampiran

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		√