Tugas Kecil 3 IF2211 Strategi Algoritma

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

Disusun oleh:

13520016 - Gagas Praharsa Bahar

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH DAN CARA KERJA ALGORITMA

1.1 Algoritma Branch and Bound

Algoritma *branch and bound* adalah salah satu variasi dari BFS yang menggunakan elemen dari least cost search. Algoritma ini digunakan untuk persoalan optimasi (meminimalkan atau memaksimalkan suatu fungsi objektif, yang tidak melanggar batasan (constraints) persoalan).

Inti dari algoritma ini adalah setiap simpul diberi sebuah nilai *cost*, yaitu nilai taksiran lintasan termurah ke simpul status tujuan yang melalui simpul status *i*, dan simpul yang dikembangkan berikutnya ditentukan berdasarkan nilai *cost*.

1.2 15 Puzzle

15 Puzzle adalah sebuah permainan puzzle geser yang memiliki 15 kotak bernomor 1-15 pada matriks 4x4, dengan satu kotak kosong. Kotak yang bersebelahan dengan kotak kosong dapat dipindah dengan menggesernya menuju kotak kosong.

1.3 Algoritma Penyelesaian 15 Puzzle dengan Pendekatan Branch and Bound

Dalam menyelesaikan permasalahan *15 Puzzle* secara algoritmik, penulis menggunakan pendekatan *branch and bound*. Adapun langkah-langkah penyelesaian permasalahan dalam algoritma dapat dijelaskan secara deskriptif sebagai berikut:

1. Terima input dari file teks berupa puzzle yang sesuai dengan ketentuan yang ditentukan sebelumnya.
2. Tampung input pada struktur data (array 2d) agar dapat diakses oleh program.
3. Tentukan apakah matriks yang diinput dapat diselesaikan dengan menggunakan perhitungan $\sum kurang + X$ harus genap. Apabila ganjil, maka matriks tidak dapat diselesaikan.
4. Inisialisasi node berupa objek dengan matriks awal dan cost awal, serta queue kosong untuk menempatkan node yang belum di ekspan dan array kosong untuk menempatkan node yang sudah dikunjungi. Cost dihitung dengan rumus c(p) = f(p) + g(p) dengan f(p) adalah depth dari node, dan g(p) adalah taksiran lintasan terpendek (yang dalam penyelesaian ini dihitung dengan manhattan distance)
5. Lakukan loop untuk semua node saat queue masih ada anggotanya, dengan mengambil queue paling depan (yang dimasukkan serupa dengan priorityqueue berdasarkan cost) lalu melakukan ekspansi terhadap node tersebut.
6. Loop akan berhenti apabila g(p) untuk sebuah node solusi sama dengan 0.

BAB 2
INPUT OUTPUT PROGRAM

2.1 Input Program

Terdapat dua jenis input untuk program 15 puzzle yang dibuat oleh penulis, yaitu dari file dan juga dari terminal/console langsung. Contoh file input adalah sebagai berikut:

```
Choose your input option:
1. File input
2. Console input
1
Enter your desired filename: test1.txt
INITIAL MATRIX:
2       3       4       11
1       5       10      8
9       6       12      15
13      14      0       7
```

Program akan mencari pada folder test untuk file dengan nama yang diberikan. Apabila file tersebut tidak ada, maka akan terdapat sebuah pesan error dan program akan keluar.

```
Choose your input option:
1. File input
2. Console input
1
Enter your desired filename: a
No file with the desired name found.
```

Untuk pilihan console input, pengguna harus memberikan matriks 4x4 dengan angka yang valid dari 0-15, dengan 0 melambangkan kotak kosong.

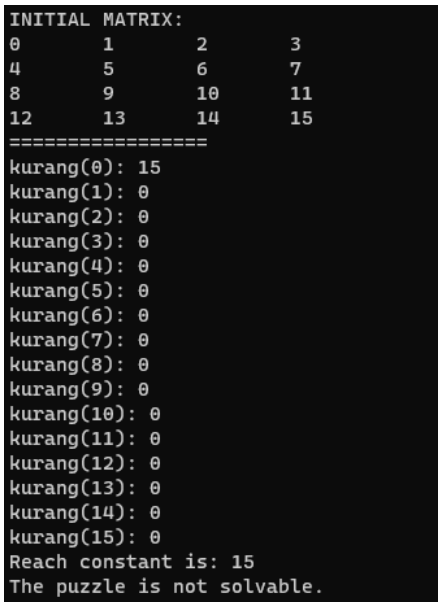
```
Enter your 4x4 matrix (valid values only 0 to 15):
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
INITIAL MATRIX:
0       1       2       3
4       5       6       7
8       9       10      11
12      13      14      15
```

2.2 Output Program

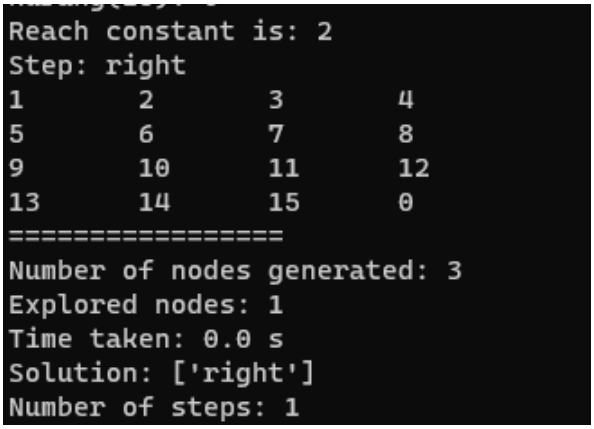
Setelah melakukan input untuk matriks awal, maka program akan berjalan dan mencoba untuk menyelesaikan puzzle yang dilambangkan oleh puzzle tersebut.

Pertama-tama, program akan menampilkan matriks awal sebelum diubah, beserta nilai kurang dari seluruh elemen dan juga nilai reach constant. Setelah itu, akan dilakukan pengecekan apakah puzzle tersebut dapat diselesaikan atau tidak.

Apabila puzzle tersebut tidak dapat diselesaikan (dapat diketahui dari nilai $\sum kurang + X$), maka akan ada beberapa kemungkinan output yang berbeda.



Apabila puzzle dapat diselesaikan, maka program akan mencoba untuk menyelesaikan puzzle tersebut. Setelah berhasil, program akan menampilkan step penyelesaian, node yang dibangkitkan, node yang di *explore*, time taken, array solution, beserta number of steps yang diperlukan. Contoh pada file yang hanya membutuhkan satu step:



BAB 3

SOURCE CODE PROGRAM

3.1. Repository Program

Repository program dapat diakses melalui:

- Github: https://github.com/gagaspbahar/15-puzzle-bnb

3.2. Source Code Program

3.2.1. app.py

```
from lib.puzzle import *

if __name__ == '__main__':
    print('''
  __   _____     _____                      _
 /_ | | ____|   |  __ \                    | |
  | | | |__     | |__) |  _   _   ____  ____ | |   __
  | | |___ \    |  ___/  | | | | |_  / |_  / | |  / _ \\
  | |  ___) |   | |      | |_| |  / /   / /  | | |  __/
  |_| |____/    |_|       \__,_| /___| /___| |_|  \___|

                                                    ''')
    print("Choose your input option: ")
    print("1. File input")
    print("2. Console input")
    option = int(input())
    if(option == 1):
        filename = input("Enter your desired filename: ")
        startingMatrix = fileInput(filename)
    elif(option == 2):
        print("Enter your 4×4 matrix (valid values only 0 to 15): ")
        startingMatrix = consoleInput()
    else:
        print("Invalid input.")
        quit()
    print("INITIAL MATRIX: ")
    printMatrix(startingMatrix)
    print("═══════════════")
    solve(startingMatrix)
```

3.2.2. puzzle.py

```
import copy
import heapq as hq
import time

class PuzzleNode:
  test_dir = "test/"

  def __init__(self, level, cost, matrix):
    self.level = level
    self.matrix = matrix
    self.cost = cost
    self.moves = []

  def getMatrix(self):
    return self.matrix

  def getLevel(self):
    return self.level

  def getCost(self):
    return self.cost

  def getFunction(self):
    return self.level + self.cost

  def getMoves(self):
    return self.moves

  def appendMoves(self, direction):
    self.moves.append(direction)

  def moveUp(self):
    for i in range(len(self.matrix)):
      for j in range(len(self.matrix[i])):
        if(self.matrix[i][j] == 0):
          if(i == 0):
            return -1
          else:
            self.matrix[i][j] = self.matrix[i-1][j]
            self.matrix[i-1][j] = 0
            return 0
```

```
def moveDown(self):
  for i in range(len(self.matrix)):
    for j in range(len(self.matrix[i])):
      if(self.matrix[i][j] == 0):
        if(i == len(self.matrix)-1):
          return -1
        else:
          self.matrix[i][j] = self.matrix[i+1][j]
          self.matrix[i+1][j] = 0
          return 0

def moveLeft(self):
  for i in range(len(self.matrix)):
    for j in range(len(self.matrix[i])):
      if(self.matrix[i][j] == 0):
        if(j == 0):
          return -1
        else:
          self.matrix[i][j] = self.matrix[i][j-1]
          self.matrix[i][j-1] = 0
          return 0

def moveRight(self):
  for i in range(len(self.matrix)):
    for j in range(len(self.matrix[i])):
      if(self.matrix[i][j] == 0):
        if(j == len(self.matrix[i])-1):
          return -1
        else:
          self.matrix[i][j] = self.matrix[i][j+1]
          self.matrix[i][j+1] = 0
          return 0

def move(self, direction):
  if(direction == "up"):
    status = self.moveUp()
  elif(direction == "down"):
    status = self.moveDown()
  elif(direction == "left"):
    status = self.moveLeft()
  elif(direction == "right"):
    status = self.moveRight()
  else:
```

```
      print("Invalid direction.")
    return status

  def calcCost(self):
    # using manhattan distance
    ans = 0
    for i in range(4):
      for j in range(4):
        num = getAbsolute(i,j) + 1
        if(num == 16):
          num = 0
        temp = getPosition(self.matrix, num)
        i1 = temp[0]
        j1 = temp[1]
        ans += abs(i1-i) + abs(j1-j)
    self.cost = ans

  def incrementLevel(self):
    self.level += 1

  def getChild(self):
    direction = ["up", "down", "left", "right"]
    children = []
    for i in range(4):
      temp = copy.deepcopy(self)
      status = temp.move(direction[i])
      if(status == 0):
        temp.incrementLevel()
        temp.calcCost()
        temp.appendMoves(direction[i])
        children.append(temp)
    return children

  def __lt__(self, other):
    return self.getFunction() <= other.getFunction()

def printMatrix(matrix):
  print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in matrix]))


def fileInput(filename):
  matrix = []
  try:
```

```
    file = open(PuzzleNode.test_dir + filename, "r")
  except:
    print("No file with the desired name found.")
    quit()
  for line in file:
    line = line.strip()
    matrix.append(line.split(" "))
  for i in range(len(matrix)):
    for j in range(len(matrix[i])):
      matrix[i][j] = int(matrix[i][j])
      if(matrix[i][j] == 16):
        matrix[i][j] = 0
  return matrix

def consoleInput():
  matrix = [list(map(int, input().split(" "))) for j in range(4)]
  return matrix


def findX(matrix):
  for i in range(4):
    for j in range(4):
      if(matrix[i][j] == 0):
        if((i%2 == 0 and j%2 == 1) or (i%2 == 1 and j%2 == 0)):
          return 1
        else:
          return 0

def getAbsolute(i, j):
  return i*4 + j

def getPosition(matrix, n):
  for i in range(4):
    for j in range(4):
      if(matrix[i][j] == n):
        return [i, j]

def findKurang(matrix):
  # Banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i)
  kurang = 0
  for n in range(16):
    currentKurang = 0
    for i in range(4):
```

```
      for j in range(4):
        # n itu i
        # getabsolute i j itu j
        m = getAbsolute(i, j)
        check = 16 if n == 0 else n
        temp = getPosition(matrix, n)
        checkI = temp[0]
        checkJ = temp[1]
        tempM = getPosition(matrix, m)
        checkMI = tempM[0]
        checkMJ = tempM[1]
        m = 16 if m == 0 else m
        if((m < check) and (getAbsolute(checkMI, checkMJ) > getAbsolute(checkI, checkJ))):
          kurang += 1
          currentKurang += 1
    print("kurang(" + str(n) + "):", currentKurang)
  return kurang

def reach(matrix):
  reachConst = findKurang(matrix) + findX(matrix)
  return reachConst

def isReachable(matrix):
  reachConst = reach(matrix)
  print("Reach constant is:", str(reachConst))
  if(reachConst % 2 == 1):
    print("The puzzle is not solvable.")
    return False
  else:
    return True

def solve(initialMatrix):
  if(isReachable(initialMatrix)):
    initialNode = PuzzleNode(0, 0, initialMatrix)
    initialNode.calcCost()
    queue = []
    hq.heapify(queue)
    hq.heappush(queue, initialNode)
    explored = []
    nodeCount = 0
    exploredNodeCount = 0
    now = time.time()
    while(queue):
```

```
      currentNode = hq.heappop(queue)
      if(currentNode.getCost() == 0):
        ans = currentNode.getMoves()
        break
      explored.append(currentNode)
      exploredNodeCount += 1
      children = currentNode.getChild()
      for child in children:
        hq.heappush(queue, child)
        nodeCount += 1
    done = time.time()
    printSolution(initialMatrix, ans)
    print("Number of nodes generated:", nodeCount)
    print("Explored nodes:", exploredNodeCount)
    print("Time taken:", str(done-now), "s")
    print("Solution:", ans)
    print("Number of steps:", len(ans))

def printSolution(initialMatrix, moves):
  matrix = PuzzleNode(0, 0, initialMatrix)
  for move in moves:
    print("Step:", move)
    if(move == "up"):
      matrix.moveUp()
    elif(move == "down"):
      matrix.moveDown()
    elif(move == "left"):
      matrix.moveLeft()
    elif(move == "right"):
      matrix.moveRight()
    printMatrix(matrix.getMatrix())
    print("═══════════════")
```

BAB 4

MASUKAN DAN LUARAN PROGRAM

4.1 test1.txt

<table>
<tr><th>Input</th><th>Output</th><th>steps</th></tr>
<tr>
<td>

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

</td>
<td>

```
INITIAL MATRIX:
2       3       4       11
1       5       10      8
9       6       12      15
13      14      0       7
=================
kurang(0): 1
kurang(1): 0
kurang(2): 1
kurang(3): 1
kurang(4): 1
kurang(5): 0
kurang(6): 0
kurang(7): 0
kurang(8): 2
kurang(9): 2
kurang(10): 4
kurang(11): 7
kurang(12): 1
kurang(13): 1
kurang(14): 1
kurang(15): 3
Reach constant is: 26
```

```
Number of nodes generated: 5750
Explored nodes: 2043
Time taken: 0.3451075553894043 s
```

```
Number of steps: 21
```

```
Solution: ['right', 'up', 'left', 'up', 'right',
 'up', 'left', 'left', 'left', 'down', 'right',
'down', 'right', 'down', 'right', 'up', 'up', 'l
eft', 'down', 'right', 'down']
```

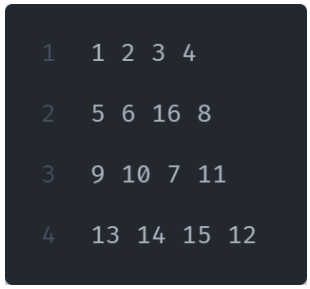
</td>
<td>
Step: right
2 3 4 11
1 5 10 8
9 6 12 15
13 14 7 0
==========
Step: up
2 3 4 11
1 5 10 8
9 6 12 0
13 14 7 15
==========
Step: left
2 3 4 11
1 5 10 8
9 6 0 12
13 14 7 15
==========
Step: up
2 3 4 11
1 5 0 8
9 6 10 12
13 14 7 15
==========
Step: right
2 3 4 11
1 5 8 0
9 6 10 12
13 14 7 15
==========
Step: up
2 3 4 0
1 5 8 11
9 6 10 12
13 14 7 15
==========
Step: left
2 3 0 4
1 5 8 11
9 6 10 12
13 14 7 15
==========
Step: left
</td>
</tr>
</table>

		2 0 3 4 1 5 8 11 9 6 10 12 13 14 7 15 ========== Step: left 0 2 3 4 1 5 8 11 9 6 10 12 13 14 7 15 ========== Step: down 1 2 3 4 0 5 8 11 9 6 10 12 13 14 7 15 ========== Step: right 1 2 3 4 5 0 8 11 9 6 10 12 13 14 7 15 ========== Step: down 1 2 3 4 5 6 8 11 9 0 10 12 13 14 7 15 ========== Step: right 1 2 3 4 5 6 8 11 9 10 0 12 13 14 7 15 ========== Step: down 1 2 3 4 5 6 8 11 9 10 7 12 13 14 0 15 ========== Step: right 1 2 3 4 5 6 8 11 9 10 7 12 13 14 15 0 ========== Step: up 1 2 3 4 5 6 8 11 9 10 7 0

		13 14 15 12 ========== Step: up 1 2 3 4 5 6 8 0 9 10 7 11 13 14 15 12 ========== Step: left 1 2 3 4 5 6 0 8 9 10 7 11 13 14 15 12 ========== Step: down 1 2 3 4 5 6 7 8 9 10 0 11 13 14 15 12 ========== Step: right 1 2 3 4 5 6 7 8 9 10 11 0 13 14 15 12 ========== Step: down 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 ==========

4.2 test2.txt

Input	Output	steps




```
INITIAL MATRIX:
1       2       3       4
5       6       0       8
9       10      7       11
13      14      15      12
=================
kurang(0): 9
kurang(1): 0
kurang(2): 0
kurang(3): 0
kurang(4): 0
kurang(5): 0
kurang(6): 0
kurang(7): 0
kurang(8): 1
kurang(9): 1
kurang(10): 1
kurang(11): 0
kurang(12): 0
kurang(13): 1
kurang(14): 1
kurang(15): 1
Reach constant is: 16
```

```
Number of nodes generated: 11
Explored nodes: 3
Time taken: 0.0 s
Solution: ['down', 'right', 'down']
Number of steps: 3
```

```
Step: down
1       2       3       4
5       6       7       8
9       10      0       11
13      14      15      12
=================
Step: right
1       2       3       4
5       6       7       8
9       10      11      0
13      14      15      12
=================
Step: down
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      0
=================
```

4.3 test3.txt

Input	Output	steps
1 1 3 4 15 2 2 0 5 12 3 7 6 11 14 4 8 9 10 13	INITIAL MATRIX: 1 3 4 15 2 0 5 12 7 6 11 14 8 9 10 13 ========== kurang(0): 10 kurang(1): 0 kurang(2): 0 kurang(3): 1 kurang(4): 1 kurang(5): 0 kurang(6): 0 kurang(7): 1 kurang(8): 0 kurang(9): 0 kurang(10): 0 kurang(11): 3 kurang(12): 6 kurang(13): 0 kurang(14): 4 kurang(15): 11 Reach constant is: 37 The puzzle is not solvable.	No steps.

4.4 test4.txt

Input	Output	steps
1 2 3 4 11 1 5 10 8 9 6 12 15 13 14 16 7	INITIAL MATRIX: 10 2 4 8 1 5 3 0 9 7 6 12 13 14 11 15 ========== kurang(0): 8 kurang(1): 0 kurang(2): 1 kurang(3): 0 kurang(4): 2 kurang(5): 1 kurang(6): 0 kurang(7): 1 kurang(8): 5 kurang(9): 2 kurang(10): 9 kurang(11): 0 kurang(12): 1 kurang(13): 1 kurang(14): 1 kurang(15): 0 Reach constant is: 32 Number of nodes generated: 10981 Explored nodes: 3832 Time taken: 0.6908278465270996 s Solution: ['up', 'left', 'left', 'left', 'down', 'right', 'up', 'right', 'down', 'down', 'left', 'up', 'right', 'down', 'down', 'right'] Number of steps: 16	Step: up 10 2 4 0 1 5 3 8 9 7 6 12 13 14 11 15 ========== Step: left 10 2 0 4 1 5 3 8 9 7 6 12 13 14 11 15 ========== Step: left 10 0 2 4 1 5 3 8 9 7 6 12 13 14 11 15 ========== Step: left 0 10 2 4 1 5 3 8 9 7 6 12 13 14 11 15 ========== Step: down 1 10 2 4 0 5 3 8 9 7 6 12 13 14 11 15 ========== Step: right 1 10 2 4 5 0 3 8 9 7 6 12 13 14 11 15 ========== Step: up 1 0 2 4 5 10 3 8 9 7 6 12 13 14 11 15 ========== Step: right 1 2 0 4 5 10 3 8

		9 7 6 12 13 14 11 15 ========== Step: down 1 2 3 4 5 10 0 8 9 7 6 12 13 14 11 15 ========== Step: down 1 2 3 4 5 10 6 8 9 7 0 12 13 14 11 15 ========== Step: left 1 2 3 4 5 10 6 8 9 0 7 12 13 14 11 15 ========== Step: up 1 2 3 4 5 0 6 8 9 10 7 12 13 14 11 15 ========== Step: right 1 2 3 4 5 6 0 8 9 10 7 12 13 14 11 15 ========== Step: down 1 2 3 4 5 6 7 8 9 10 0 12 13 14 11 15 ========== Step: down 1 2 3 4 5 6 7 8 9 10 11 12 13 14 0 15 ========== Step: right 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 ==========

4.5 test5.txt

Input	Output	steps
1 2 3 4 11 1 5 10 8 9 6 12 15 13 14 16 7	INITIAL MATRIX: 1 2 3 4 5 6 7 0 9 10 12 8 11 13 14 15 ========== kurang(0): 8 kurang(1): 0 kurang(2): 0 kurang(3): 0 kurang(4): 0 kurang(5): 0 kurang(6): 0 kurang(7): 0 kurang(8): 0 kurang(9): 1 kurang(10): 1 kurang(11): 0 kurang(12): 2 kurang(13): 0 kurang(14): 0 kurang(15): 0 Reach constant is: 12 Number of nodes generated: 134668 Explored nodes: 45540 Time taken: 8.73783254623413 s Solution: ['down', 'left', 'down', 'left', 'left', 'up', 'right', 'down', 'right', 'up', 'left', 'left', 'down', 'right', 'right', 'right'] Number of steps: 16	Step: down 1 2 3 4 5 6 7 8 9 10 12 0 11 13 14 15 ========== Step: left 1 2 3 4 5 6 7 8 9 10 0 12 11 13 14 15 ========== Step: down 1 2 3 4 5 6 7 8 9 10 14 12 11 13 0 15 ========== Step: left 1 2 3 4 5 6 7 8 9 10 14 12 11 0 13 15 ========== Step: left 1 2 3 4 5 6 7 8 9 10 14 12 0 11 13 15 ========== Step: up 1 2 3 4 5 6 7 8 0 10 14 12 9 11 13 15 ========== Step: right 1 2 3 4 5 6 7 8 10 0 14 12 9 11 13 15 ==========

		Step: down 1 2 3 4 5 6 7 8 10 11 14 12 9 0 13 15 ========== Step: right 1 2 3 4 5 6 7 8 10 11 14 12 9 13 0 15 ========== Step: up 1 2 3 4 5 6 7 8 10 11 0 12 9 13 14 15 ========== Step: left 1 2 3 4 5 6 7 8 10 0 11 12 9 13 14 15 ========== Step: left 1 2 3 4 5 6 7 8 0 10 11 12 9 13 14 15 ========== Step: down 1 2 3 4 5 6 7 8 9 10 11 12 0 13 14 15 ========== Step: right 1 2 3 4 5 6 7 8 9 10 11 12 13 0 14 15 ========== Step: right 1 2 3 4 5 6 7 8 9 10 11 12 13 14 0 15 ========== Step: right 1 2 3 4 5 6 7 8

		9 10 11 12 13 14 15 0 ==========

REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf

LAMPIRAN

Checklist Fitur

Poin	Ya	Tidak
1. Program berhasil dikompilasi	v	
2. Program berhasil *running*	v	
3. Program dapat menerima input dan menuliskan output	v	
4. Luaran sudah benar untuk semua data uji	v	
5. Bonus dibuat		v

Testcase yang dipakai

test1.txt	2 3 4 11 1 5 10 8 9 6 12 15 13 14 16 7
test2.txt	1 2 3 4 5 6 16 8 9 10 7 11 13 14 15 12
test3.txt	1 3 4 15 2 0 5 12 7 6 11 14 8 9 10 13
test4.txt	10 2 4 8 1 5 3 16 9 7 6 12 13 14 11 15
test5.txt	1 2 3 4 5 6 7 16 9 10 12 8 11 13 14 15