**LAPORAN TUGAS KECIL**

**Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound***

Ditujukan untuk memenuhi salah satu tugas kecil mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2021/2022

Disusun oleh:

**Rania Dwi Fadhilah (K1) 13520142**



**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2022**

## DAFTAR ISI

[**DAFTAR ISI i**](#_Toc99889158)

[**I. ALGORITMA PROGRAM 1**](#_Toc99889159)

[**1.1 PENJELASAN ALGORITMA 1**](#_Toc99889160)

[**2.1 IMPLEMENTASI 2**](#_Toc99889161)

[**II. HASIL PERCOBAAN 10**](#_Toc99889162)

[**2.1 INTERAKSI I/O 10**](#_Toc99889163)

[**2.2 UJI COBA TXT 12**](#_Toc99889164)

[**2.3 UJI COBA MANUAL 26**](#_Toc99889165)

[**LAMPIRAN ii**](#_Toc99578667)

## I. ALGORITMA PROGRAM

## 1.1 PENJELASAN ALGORITMA

Algoritma *Branch and Bound* merupakan algoritma yang kerap digunakan untuk menyelesaikan persoalan optimasi, dimana program harus meminimalkan atau memaksimalkan suatu fungsi objektif tertentu tanpa melanggar batasan persoalan. Pada dasarnya, algoritma ini merupakan gabungan dari algoritma BFS (*Breadth First Search)* dengan *least cost search*. Oleh karena itu, cara kerja algoritma ini adalah memberikan nilai *cost* pada setiap simpulnya, kemudian mengekspansi simpul berdasarkan urutan nilai *cost* yang paling kecil (apabila ingin meminimalkan) ataupun yang paling besar (apabila ingin memaksimalkan), bukan berdasarkan urutan pembangkitannya. Salah satu contoh aplikasi dari algoritma ini adalah pada permainan 15-puzzle. Pada tugas kecil ini, kita diminta untuk mengimplementasikan penyelesaian persoalan 15-Puzzle dengan menggunakan algoritma ini.

Berikut adalah cara kerja program *Branch and Bound* dalam menyelesaikan persoalan 15-puzzle :

1. Membaca *puzzle* menjadi *array* 4 x 4 dalam bentuk kelas Puzzle yang juga menyimpan nilai :

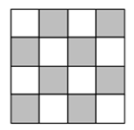
*cost* (harga jalur),

*path* (urutan jalur 🡪 c/: [[UP], [LEFT]]),

letak basis,

bentuk atas, bawah, kanan, dan kirinya.

1. Menghitung nilai kurang(i) dari puzzle dengan cara menghitung jumlah elemen setelah posisi(i) yang memiliki nilai lebih dari elemen pada posisi(i).
2. Menghitung nilai X,



a) Apabila basis berada di kotak terarsir, X = 0

b) Apabila basis berada di kotak tidak terarsir, X = 1

1. Apabila nilai kurang(i) + X = ganjil, maka *puzzle* tidak dapat diselesaikan
2. Apabila nilai kurang(i) + X = genap, maka pencarian dimulai dengan menginisialisasi *heapqueue*, memasukkan puzzle acuan pertama ke *dictionary* checked puzzle.
3. Apabila puzzle sudah dalam keadaan selesai, maka pencarian dihentikan dan lanjut ke tahap 9
4. Apabila puzzle belum dalam keadaan selesai, maka akan dilakukan :
5. Inisialisasi *path* dengan path terakhir yang dilalui oleh puzzle untuk menghindari perpindahan balik, seperti sebelumnya UP kemudian DOWN dan sebelumnya LEFT kemudian RIGHT yang akan mengembalikan puzzle ke posisi semula
6. Apabila posisi baris basis tidak = 0 dan path terakhirnya bukan “DOWN”, dilakukan pencarian ke atas dengan cara menginisialisasi *puzzle* baru dan menduplikasi matriks *puzzle* yang sedang dicari dan menukar posisi basisnya dengan matriks di atasnya.
7. Matriks diubah ke dalam bentuk list sementara untuk dicek apakah matriks tersebut sudah pernah diselidiki atau belum pada *dictionary* checked puzzle.
8. Apabila matriks puzzle belum pernah diselidiki, maka *path*-nya ditambahkan, basisnya diinisialisasi, dan *cost*-nya dihitung dengan rumus berikut :
9. Puzzledimasukkan pada *heapqueue* dengan *tuple value* berupa <*cost* Puzzle, Puzzle>.
10. Apabila *puzzle* sudah selesai, maka pencarian dihentikan
11. Apabila posisi baris basis tidak = 3 dan path terakhirnya bukan “UP”, dilakukan pencarian ke bawah dengan cara menginisialisasi *puzzle* baru dan menduplikasi matriks *puzzle* yang sedang dicari dan menukar posisi basisnya dengan matriks di bawahnya. Kemudian, melakukan tahap c,d,e,f.
12. Apabila posisi kolom basis tidak = 0 dan path terakhirnya bukan “RIGHT”, dilakukan pencarian ke kiri dengan cara menginisialisasi *puzzle* baru dan menduplikasi matriks *puzzle* yang sedang dicari dan menukar posisi basisnya dengan matriks di kirinya. Kemudian, melakukan tahap c,d,e,f.
13. Apabila posisi kolom basis tidak = 3 dan path terakhirnya bukan “LEFT”, dilakukan pencarian ke kanan dengan cara menginisialisasi *puzzle* baru dan menduplikasi matriks *puzzle* yang sedang dicari dan menukar posisi basisnya dengan matriks di kanannya. Kemudian, melakukan tahap c,d,e,f.
14. Puzzle yang berada di *heapqueue* dibangkitkan menjadi *puzzle* acuan. Puzzle yang dibangkitkan merupakan *puzzle* dengan nilai *cost* terkecil yang ada pada *heapqueue*.
15. Puzzle acuan dijadikan list dan dimasukan ke *dictionary* checked puzzle.
16. Kembali ke langkah 6
17. Menuliskan solusi pada layar dengan cara mengiterasi *path* dari *puzzle* tersebut

## 2.1 IMPLEMENTASI

a. puzzle.py

|  |
| --- |
| import heapq  import numpy as np  import time  # PUZZLE CLASS  class Puzzle :      goalstate = [[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]]      # 1. Puzzle Initialization      def \_\_init\_\_(self):          self.mtx = np.empty((4,4), int) # Create Empty Matrix          self.cost = 0 # Cost of the puzzle          self.zero = (0,0) # Position of the basis          self.path = [] # Path of the puzzle          self.up, self.down, self.left, self.right = None, None, None, None # Up, Down, Left, Right      # 2. Cost Counter      def countCost(self):          count = 0          for i in range (4) :              for j in range (4) :                  # Count number of matrix that is not in position                  if (self.mtx[i][j] != Puzzle.goalstate[i][j]) :                      count += 1            return count      # 3. Handle <      def \_\_lt\_\_ (self,other) :          return True  # IS FINISH BOOLEAN  def isFinish(puzzle):      if (puzzle.countCost() == 0) :          return True      else :          return False  # CHECK PUZZLE VALIDITY  def puzzleValid(puzzle) :      # 1. Change 2d array -> list      puzz = (np.reshape(puzzle.mtx, 16)).tolist()      # 2. Check if 1-16 is in order      for i in range (1,17) :          if (SearchI(puzz, i) == -1) :              return False        return True  # READ PUZZLE FROM FILE  def readPuzzleFromFile(filename):      # 1. Initialize Puzzle      Puz15 = Puzzle()      # 2. Open Puzzle File      try :          f = open(filename, "r")          # 3. Read Puzzle          try :              for i in range(4):                  temp = f.readline()                  Puz15.mtx[i] = temp.split()                  for j in range(4):                      Puz15.mtx[i][j] = int(Puz15.mtx[i][j])                      if (Puz15.mtx[i][j] == 0 or Puz15.mtx[i][j] == 16) :                          Puz15.mtx[i][j] = 16                          Puz15.zero = (i,j)              f.close()              Puz15.cost = Puz15.countCost()              # 4. Return Puzzle If Puzzle is Valid              if (puzzleValid(Puz15)) :                  return Puz15              else :                  print("  ║         Invalid input                                                    ║")                  return None          except :              print("  ║         Invalid input                                                    ║")              return None      except :          print("  ║         File not found                                                   ║")          return None  # READ PUZZLE FROM CONSOLE  def readPuzzleFromConsole():      try :          # 1. Initialize Puzzle          Puz15 = Puzzle()          # 2. Read Puzzle          print("  ║         Input Puzzle (4x4), consists of :                                ║")          print("  ║         a. Number 1 - 15                                                 ║")          print("  ║         b. 0 or 16 as basis                                              ║")          for i in range(4):              print("                                >> ", end="")              temp = str(input())              Puz15.mtx[i] = temp.split()              for j in range(4):                  Puz15.mtx[i][j] = int(Puz15.mtx[i][j])                  if (Puz15.mtx[i][j] == 0 or Puz15.mtx[i][j] == 16) :                      Puz15.mtx[i][j] = 16                      Puz15.zero = (i,j)            Puz15.cost = Puz15.countCost()          # 3. Return Puzzle If Puzzle is Valid          if (puzzleValid(Puz15)) :                  return Puz15          else :              print("  ║         Invalid input                                                    ║")              return None      except :          print("  ║         Invalid input                                                    ║")          return None  # SEARCH INDEX OF NUM  def SearchI(puzz, num) :      for i in range(len(puzz)) :          if (puzz[i] == num) :              return i      return -1  # PRINT PUZZLE  def printPuzzle(puzz):      print("                              ╔════╦════╦════╦════╗")      for i in range(4):          print("                              ", end = "")          for j in range(4):              print("║ ",end="")              if (puzz[i][j] == 16) :                  print("   ", end = "")                  continue              print("%02d " % puzz[i][j], end = "")          print("║")          if(i != 3):              print("                              ╠════╬════╬════╬════╣")      print("                              ╚════╩════╩════╩════╝")  # CONVERT PUZZLE TO LIST  def puzzleToList(Puzzle) :      puzz = ""      temp = (np.reshape(Puzzle, 16)).tolist()      for i in range(16) :          puzz += str(temp[i])      return puzz  # FIND KURANG(I) + X (BASIS POSITION)  # TO CHECK IF PUZZLE IS REACHABLE  def Reachable(Puzzle) :      # 1. Convert Puzzle to List      puzz = (np.reshape(Puzzle.mtx, 16)).tolist()        # 2. Find Kurang(i)      count = 0      print("                                ╔════╦═══════════╗     ")      print("                                ║  i ║ kurang(i) ║     ")      print("                                ╠════╬═══════════╣     ")      for i in range (1,17) :          temp = 0          index = SearchI(puzz, i)          for j in range (index, 16) :              if (i > puzz[j]) :                  temp += 1          count += temp          print("                                ║ ", end = "")          if (i < 10) :              print(" %d" % i, end = "")          else :              print("%d" % i, end = "")          print(" ║    ", end="")          if (temp < 10) :              print(" %d" % temp, end = "")          else :              print("%d" % temp, end = "")          print("     ║")      print("                                ╚════╩═══════════╝")        # 3. Find basis position      if ((Puzzle.zero[0] + Puzzle.zero[1]) % 2 != 0) :          count += 1        return count  # SOLVE PUZZLE  def solvePuzzle(PuzzPar) :      # 1. Initialize Time      now = time.time()      # 2. Initialize Puzzle      currPuzz = PuzzPar      # 3. Initialize Checked Dictionary      checked = {}      puzz = puzzleToList(currPuzz.mtx)      checked[puzz] = True      # 4. Initialize HeapQueue      pq = []      # 5. Initialize node      simpul = 0      # 6. Loop until Puzzle = Goal State      while (not (isFinish(currPuzz))) :          # Check last path          if (len(currPuzz.path) != 0) :              path = currPuzz.path[len(currPuzz.path)-1]          else :              path = None          # Check Up          if (currPuzz.zero[0] != 0 and path != "DOWN") :              currPuzz.up = Puzzle()              currPuzz.up.mtx = np.copy(currPuzz.mtx)              currPuzz.up.mtx[currPuzz.zero[0]][currPuzz.zero[1]] = currPuzz.up.mtx[currPuzz.zero[0]-1][currPuzz.zero[1]]              currPuzz.up.mtx[currPuzz.zero[0]-1][currPuzz.zero[1]] = 16              temp = puzzleToList(currPuzz.up.mtx)              if (temp not in checked) :                  currPuzz.up.path = currPuzz.path + ["UP"]                  currPuzz.up.zero = [currPuzz.zero[0]-1, currPuzz.zero[1]]                  currPuzz.up.cost = currPuzz.up.countCost() + len(currPuzz.up.path)                  heapq.heappush(pq, (currPuzz.up.cost, currPuzz.up))                  simpul += 1                  if (isFinish(currPuzz.up)) :                      currPuzz = currPuzz.up                      break          # Check Down          if (currPuzz.zero[0] != 3 and path != "UP") :              currPuzz.down = Puzzle()              currPuzz.down.mtx = np.copy(currPuzz.mtx)              currPuzz.down.mtx[currPuzz.zero[0]][currPuzz.zero[1]] = currPuzz.down.mtx[currPuzz.zero[0]+1][currPuzz.zero[1]]              currPuzz.down.mtx[currPuzz.zero[0]+1][currPuzz.zero[1]] = 16              temp = puzzleToList(currPuzz.down.mtx)              if (temp not in checked) :                  currPuzz.down.path = currPuzz.path + ["DOWN"]                  currPuzz.down.zero = [currPuzz.zero[0]+1, currPuzz.zero[1]]                  currPuzz.down.cost = currPuzz.down.countCost() + len(currPuzz.down.path)                  heapq.heappush(pq, (currPuzz.down.cost, currPuzz.down))                  simpul += 1                  if (isFinish(currPuzz.down)) :                      currPuzz = currPuzz.down                      break          # Check Left          if (currPuzz.zero[1] != 0 and path != "RIGHT") :              currPuzz.left = Puzzle()              currPuzz.left.mtx = np.copy(currPuzz.mtx)              currPuzz.left.mtx[currPuzz.zero[0]][currPuzz.zero[1]] = currPuzz.left.mtx[currPuzz.zero[0]][currPuzz.zero[1]-1]              currPuzz.left.mtx[currPuzz.zero[0]][currPuzz.zero[1]-1] = 16              temp = puzzleToList(currPuzz.left.mtx)              if (temp not in checked) :                  currPuzz.left.path = currPuzz.path + ["LEFT"]                  currPuzz.left.zero = [currPuzz.zero[0], currPuzz.zero[1]-1]                  currPuzz.left.cost = currPuzz.left.countCost() + len(currPuzz.left.path)                  heapq.heappush(pq, (currPuzz.left.cost, currPuzz.left))                  simpul += 1                  if (isFinish(currPuzz.left)) :                      currPuzz = currPuzz.left                      break          # Check Right          if (currPuzz.zero[1] != 3 and path != "LEFT") :              currPuzz.right = Puzzle()              currPuzz.right.mtx = np.copy(currPuzz.mtx)              currPuzz.right.mtx[currPuzz.zero[0]][currPuzz.zero[1]] = currPuzz.right.mtx[currPuzz.zero[0]][currPuzz.zero[1]+1]              currPuzz.right.mtx[currPuzz.zero[0]][currPuzz.zero[1]+1] = 16              temp = puzzleToList(currPuzz.right.mtx)              if (temp not in checked) :                  currPuzz.right.path = currPuzz.path + ["RIGHT"]                  currPuzz.right.zero = [currPuzz.zero[0], currPuzz.zero[1]+1]                  currPuzz.right.cost = currPuzz.right.countCost() + len(currPuzz.right.path)                  heapq.heappush(pq, (currPuzz.right.cost, currPuzz.right))                  simpul += 1                  if (isFinish(currPuzz.right)) :                      currPuzz = currPuzz.right                      break            currPuzz = heapq.heappop(pq)[1]          puzz = puzzleToList(currPuzz.mtx)          checked[puzz] = True      t = time.time() - now      # 7. Print Solution Puzzle      start = PuzzPar.mtx      zero = PuzzPar.zero      for i in range (len(currPuzz.path)) :          # Change Position          if (currPuzz.path[i] == "UP") :              start[zero[0]][zero[1]] = start[zero[0]-1][zero[1]]              start[zero[0]-1][zero[1]] = 16              zero = (zero[0]-1, zero[1])          elif (currPuzz.path[i] == "DOWN") :              start[zero[0]][zero[1]] = start[zero[0]+1][zero[1]]              start[zero[0]+1][zero[1]] = 16              zero = (zero[0]+1, zero[1])          elif (currPuzz.path[i] == "LEFT") :              start[zero[0]][zero[1]] = start[zero[0]][zero[1]-1]              start[zero[0]][zero[1]-1] = 16              zero = (zero[0], zero[1]-1)          elif (currPuzz.path[i] == "RIGHT") :              start[zero[0]][zero[1]] = start[zero[0]][zero[1]+1]              start[zero[0]][zero[1]+1] = 16              zero = (zero[0], zero[1]+1)            # Print Puzzle          print("                                  STEP %d = %s" % (i+1, currPuzz.path[i]))          printPuzzle(start)      return currPuzz, t, simpul |

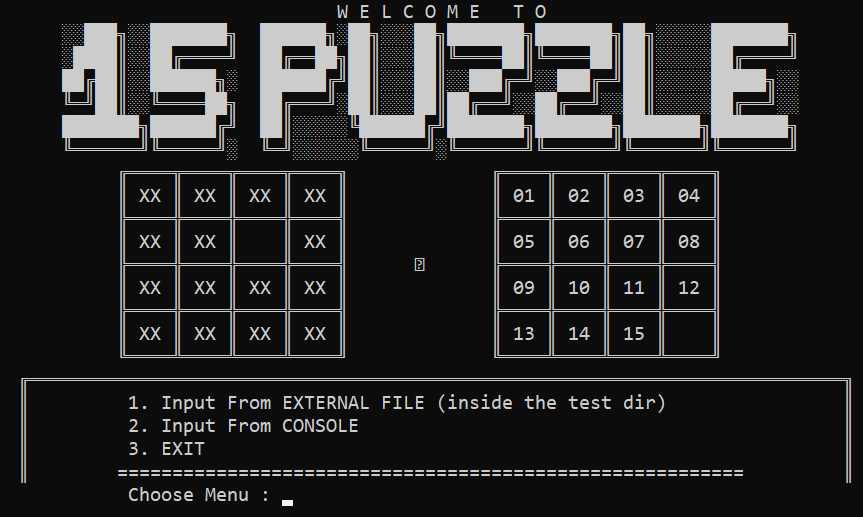
b. main.py

|  |
| --- |
| import puzzle  def Mainmenu(first) :      if first == True :          print("  ╔══════════════════════════════════════════════════════════════════════════╗")      print("  ║         1. Input From EXTERNAL FILE (inside the test dir)                ║")      print("  ║         2. Input From CONSOLE                                            ║")      print("  ║         3. EXIT                                                          ║")      print("  ║        =========================================================         ║")      try :          op = int(input("            Choose Menu : "))          if (op == 1) :              filename = input("            File Path : ")              Puz15 = puzzle.readPuzzleFromFile(".\\test\\" + filename)          elif (op == 2) :              Puz15 = puzzle.readPuzzleFromConsole()          elif (op == 3) :              Puz15 = None          print("  ╚══════════════════════════════════════════════════════════════════════════╝")          return op, Puz15      except :          print("  ║                   T R Y  A G A I N ! (invalid input)                     ║")          print("  ║        =========================================================         ║")          return Mainmenu(False)  print("                               W E L C O M E   T O                  ")  print("      ░░███╗░░███████╗  ██████╗░██╗░░░██╗███████╗███████╗██╗░░░░░███████╗")  print("      ░████║░░██╔════╝  ██╔══██╗██║░░░██║╚════██║╚════██║██║░░░░░██╔════╝")  print("      ██╔██║░░██████╗░  ██████╔╝██║░░░██║░░███╔═╝░░███╔═╝██║░░░░░█████╗░░")  print("      ╚═╝██║░░╚════██╗  ██╔═══╝░██║░░░██║██╔══╝░░██╔══╝░░██║░░░░░██╔══╝░░")  print("      ███████╗██████╔╝  ██║░░░░░╚██████╔╝███████╗███████╗███████╗███████╗")  print("      ╚══════╝╚═════╝░  ╚═╝░░░░░░╚═════╝░╚══════╝╚══════╝╚══════╝╚══════╝")  print("           ╔════╦════╦════╦════╗             ╔════╦════╦════╦════╗")  print("           ║ XX ║ XX ║ XX ║ XX ║             ║ 01 ║ 02 ║ 03 ║ 04 ║")  print("           ╠════╬════╬════╬════╣             ╠════╬════╬════╬════╣")  print("           ║ XX ║ XX ║    ║ XX ║             ║ 05 ║ 06 ║ 07 ║ 08 ║")  print("           ╠════╬════╬════╬════╣      ➜      ╠════╬════╬════╬════╣")  print("           ║ XX ║ XX ║ XX ║ XX ║             ║ 09 ║ 10 ║ 11 ║ 12 ║")  print("           ╠════╬════╬════╬════╣             ╠════╬════╬════╬════╣")  print("           ║ XX ║ XX ║ XX ║ XX ║             ║ 13 ║ 14 ║ 15 ║    ║")  print("           ╚════╩════╩════╩════╝             ╚════╩════╩════╩════╝")  op, Puz15 = Mainmenu(True)  while (op!= 3) :      if (Puz15 != None) :          print()          print("                                   P U Z Z L E")          puzzle.printPuzzle(Puz15.mtx)          count= puzzle.Reachable(Puz15)          print("                                  Kurang(i) = %d" % count)          if (count % 2 == 0) :              print("\n                   ===========   S O L U T I O N  ===========")              Result, time, simpul = puzzle.solvePuzzle(Puz15)              print("                                Node Count = %d" % simpul)              print("                                Time = %f s" % (time))          else :              print("                            Puzzle can not be solved !!")      op, Puz15 = Mainmenu(True) |

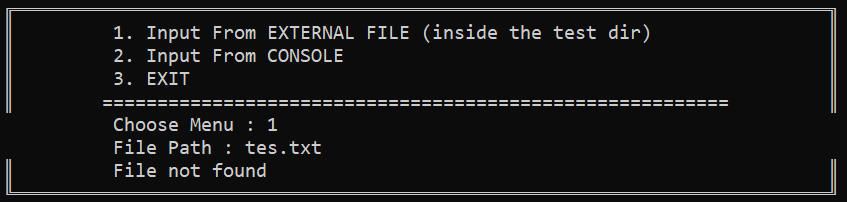
## II. HASIL PERCOBAAN

## 2.1 INTERAKSI I/O

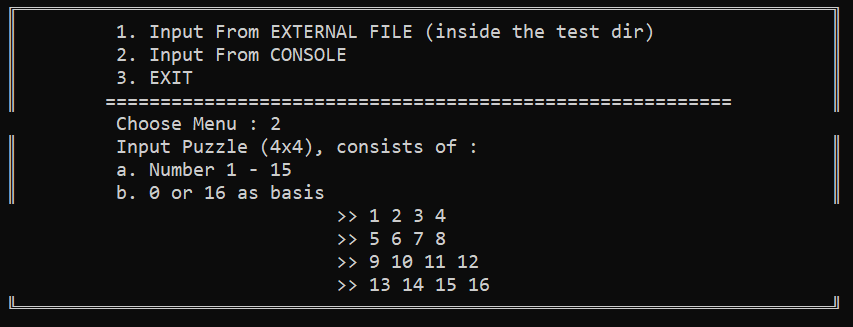
* 1. Interaksi *Input* Menu Awal



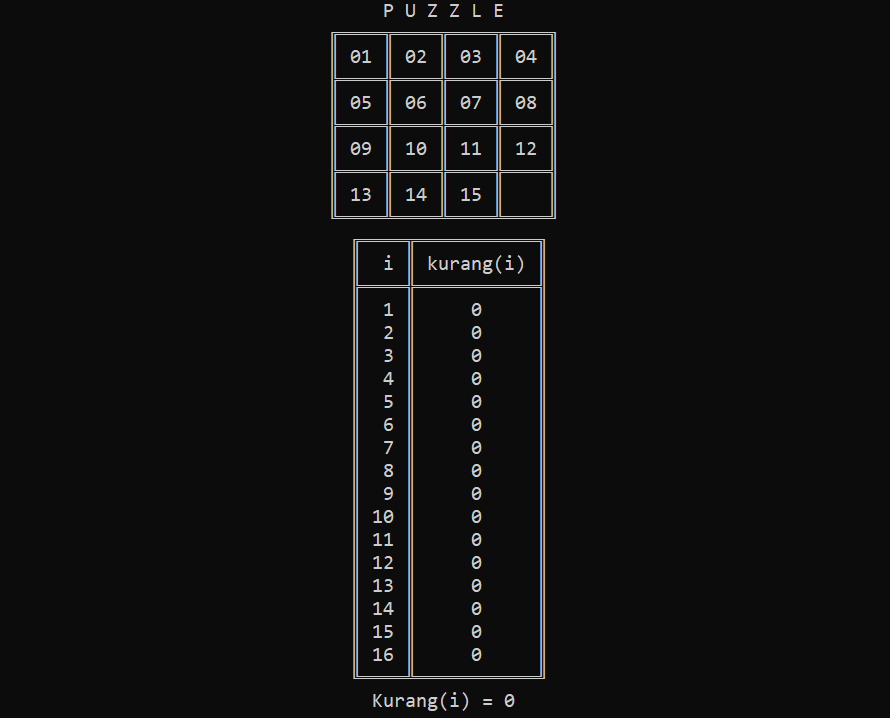
* 1. Interaksi *Input* Menu 1 *(External File)*



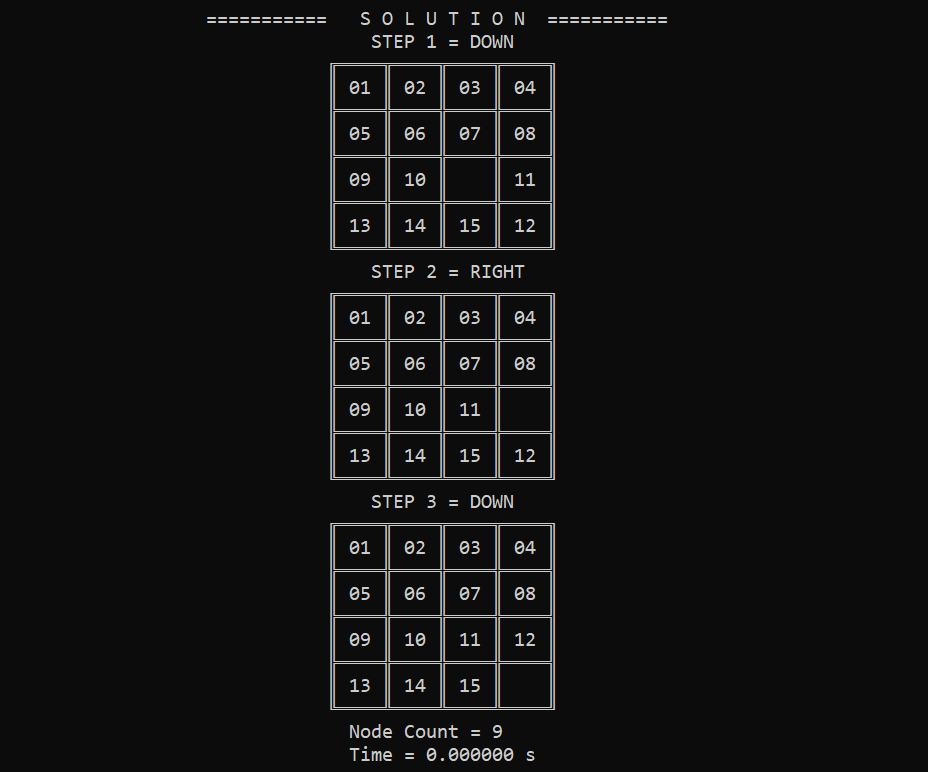
* 1. Interaksi *Input* Menu 2 (*Console)*

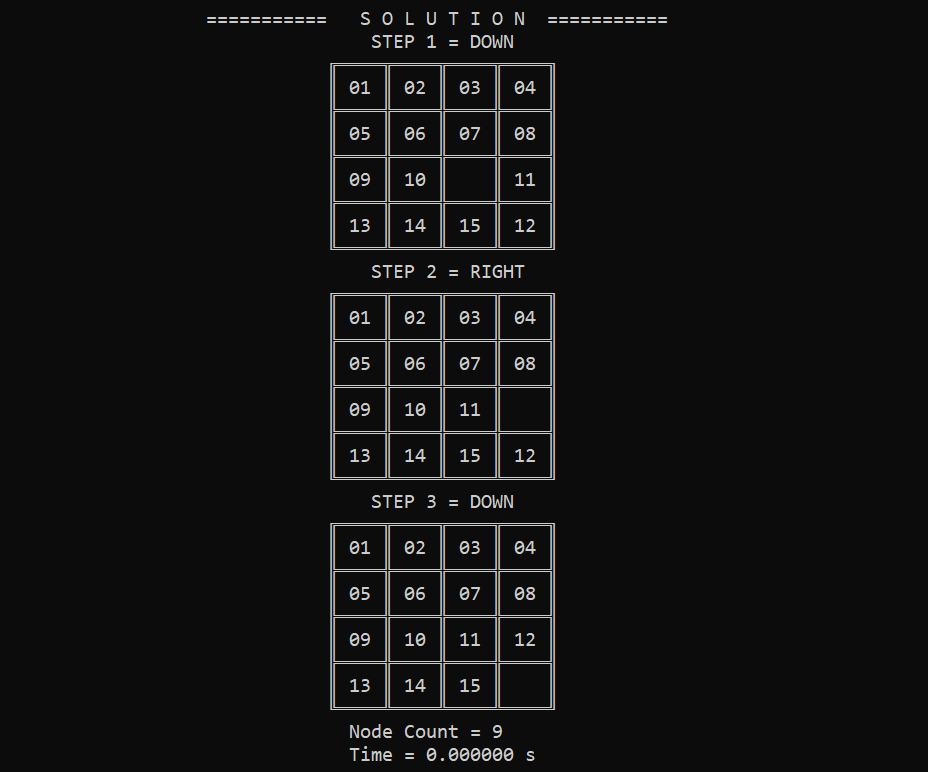


* 1. Interaksi *Output* Bagian Kurang(i)



* 1. Interaksi *Output* Bagian Solusi





## 2.2 UJI COBA TXT

*Percobaan 1*

|  |  |
| --- | --- |
| **Nama File** | tc1easy.txt |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

*Percobaan 2*

|  |  |
| --- | --- |
| **Nama File** | tc2medium.txt |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

*Percobaan 3*

|  |  |
| --- | --- |
| **Nama File** | tc3intermediate.txt |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

*Percobaan 4*

|  |  |
| --- | --- |
| **Nama File** | tc4expert.txt |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

*Percobaan 5*

|  |  |
| --- | --- |
| **Nama File** | tc5unsolved.txt |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

*Percobaan 6*

|  |  |
| --- | --- |
| **Nama File** | tc6error.txt |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

## 2.3 UJI COBA MANUAL

*Percobaan 7*

|  |  |
| --- | --- |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

*Percobaan 8*

|  |  |
| --- | --- |
| **Bentuk Awal** |  |
| **Hasil** | |
|  | |

## LAMPIRAN

1. Repository Github :

<https://github.com/raniadf/15puzzle>

2. Berkas Teks :

|  |  |
| --- | --- |
| tc1easy | 1 2 3 4  5 6 16 8  9 10 7 11  13 14 15 12 |
| tc2medium | 1 2 3 4  5 6 7 16  9 10 12 8  11 13 14 15 |
| tc3intermediate | 2 3 4 11  1 5 10 8  9 6 12 15  13 14 16 7 |
| tc4expert | 1 2 3 4  5 6 11 15  9 14 13 10  16 7 8 12 |
| tc5unsolved | 1 3 4 15  2 0 5 12  7 6 11 14  8 9 10 13 |

3. Checklist :

|  |  |  |
| --- | --- | --- |
| **Poin** | **Ya** | **Tidak** |
| 1. Program berhasil dikompilasi | **✓** |  |
| 1. Program berhasil *running* | **✓** |  |
| 1. Program dapat menerima input dan menuliskan output | **✓** |  |
| 1. Luaran sudah benar untuk semua data uji | **✓** |  |
| 1. Bonus dibuat |  | **✓** |