**LAPORAN TUGAS KECIL 3**

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

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

Disusun oleh:

**Saul Sayers (K1)**  **13520094**



**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2021**

1. **Algoritma *Branch and Bound***

Algoritma *Branch and Bound* merupakan algoritma yang digunakan untuk memecahkan persoalan optimasi dengan cara melakukan percabangan (branching) dari node awal menjadi node - node anak dan melakukan pembatasan (bounding) dari node - node tersebut untuk mengarah ke solusi. Proses branching dan bounding tersebut dilakukan berulang kali hingga ditemukan solusi yang optimal. Proses pencarian dengan algoritma ini menggunakan pembentukan pohon ruang status dinamis dengan permasalahan awal menjadi node akar dan bercabang menjadi node - node anak lainnya hingga node solusi. Berbeda dengan Algoritma BFS ataupun DFS, urutan pemeriksaan node dalam algoritma Branch and Bound dipilih berdasarkan node yang memiliki cost terkecil (*Least Cost Search*). Cost tersebut memiliki makna estimasi ongkos termurah lintasan sebuah node yang dihitung secara herustik. Secara umum, berikut adalah persamaan untuk menentukan cost dari sebuah node dalam algoritma *Branch and Bound* :

Dimana :

Dalam Tugas Kecil ini, diimplementasikan algoritma *Branch and Bound* untuk menyelesaikan persoalan 15-Puzzle. Sebelum menerapkan algoritma *Branch and Bound*, program akan menerima sebuah puzzle terlebih dahulu baik melalui file eksternal ataupun mengenerate sendiri secara random. Apabila melalui file eksternal, ubin blank diwakili dengan angka 0 ataupun 16. Dalam program ini, state matriks akan disimpan dalam sebuah node yang direpresentasikan oleh sebuah class yang menyimpan atribut cost, matriks (numpy), serta array yang berisi pergerakan untuk menuju node tersebut. Kemudian, program akan menentukan terlebih dahulu apakah puzzle / matriks yang didapatkan solvable atau tidak. Proses penentuan solvablenya dilakukan dengan metode berikut :

Dimana :

Program akan mencetak matriks awal dari puzzle serta memberikan luaran tiap nilai dari kurang(i), X, dan status. Apabila nilai status dari puzzle awal adalah ganjil, maka puzzle tersebut merupakan unsolvable dan program akan langsung mencetak pesan unsolvable. Apabila statusnya genap, maka program akan langsung menerapkan algoritma *Branch and Bound* untuk membangkitkan simpul anak dan menghitung costnya hingga ditemukan simpul solusi. Dalam kasus 15-Puzzle Solver, perhitungan cost dilakukan dengan nilai dan sebagai berikut:

Langkah kerja implementasi algoritma *Branch and Bound* dalam program 15-Puzzle Solver ini adalah sebagai berikut :

1. Memasukkan node puzzle awal dan costnya ke dalam sebuah PriorityQueue. Jika puzzle matriks dari awal sudah solved, maka pencarian berhenti karena sudah merupakan solusi. Apabila belum solved, lanjut ke langkah 2.
2. Pilih sebuah node dalam PriorityQueue yang memiliki nilai cost terkecil. Jika terdapat beberapa node dengan nilai cost yang memenuhi, maka ambil salah satu yang memiliki kedalaman terpendek.
3. Jika node yang dipilih merupakan node solusi, maka solusi sudah ditemukan dan pencarian berhenti.
4. Jika node yang dipilih bukan node solusi, maka bangkitkan semua child nodes dari node tersebut. Kemudian, hitung cost untuk tiap child node tersebut dan masukkan ke dalam PriorityQueue.
5. Ulangi dari langkah 2.

Dengan catatan : Karena sudah diperiksa dari awal apakah puzzle solvable atau tidak, maka diasumsikan bahwa node solusi sudah pasti akan ditemukan dalam proses branching sehingga tidak perlu mengatasi sampai PriorityQueue kosong. Kemudian, setelah program menemukan solusi node dari puzzle tersebut, program akan menampilkan luaran berupa banyaknya simpul yang dibangkitkan, banyaknya step yang dibutuhkan untuk mencapai solusi, progress perubahan state matriks dari awal hingga mencapai solusi per langkahnya, serta waktu eksekusi program.

1. ***Source Program***

Source code program ditulis dalam bahasa pemrograman Python dengan menggunakan library numpy dan pyfiglet. Source code program terbungkus menjadi 2 file utama, yakni solver.py dan main.py

1. **solver.py** : Merupakan file library untuk algoritma *Branch and Bound* dalam implementasi 15-Puzzle Solver yang telah saya buat. Note bahwa penjelasan tiap fungsi sudah disertakan pada source code dalam bentuk komentar

|  |
| --- |
| # NAMA  : SAUL SAYERS  # NIM   : 13520094  # KELAS : K-01 STRATEGI ALGORITMA  # MERUPAKAN FILE ALGORITMA PUZZLESOLVER UNTUK TUCIL 3 STRATEGI ALGORITMA  import numpy as np  import os.path  from queue import PriorityQueue  import random  class Node:      """ This class is what we use to represent the          nodes in this program.          Attributes:              - prev (array of string) : list of the              moves to reach this node.              - matrix (numpy 2d-array) : the matrix              of integers where we store the puzzle.              - cost (int) : the cost of the node.      """      def \_\_init\_\_(self,prev,cost):          """ Constructor of the Node Class          Args:              - prev (array of string) : list of the              moves to reach this node.              - cost (int) : the cost of the node.          """          self.prev = prev          self.matrix = np.arange(16).reshape((4,4))          self.cost = cost      def generateMatrix(self) :          """ this method is used to randomly              generate the matrix of the node.          """          numList = [i for i in range (16)]          random.shuffle(numList)          self.matrix = np.array(numList).reshape((4,4))      def readFile(self):          """ This method is used to fill the matrix              from an external txt file.          """          numList = []          while True:  # Looping until filename exists              filename = input("Input filename here (dengan .txt): ")              path = "test/" + filename              if (os.path.isfile(path)):                  break              else :                  print("Filename doesnt exist! Please re-input filename.")          file = open(path)          for i in range(4) :              numList.extend([int(number) for number in file.readline().split()])          for i in range(len(numList)):              if numList[i] == 16: # Handle if blank as 16 then change to 0                  numList[i] = 0          self.matrix = np.array(numList).reshape((4,4))      def printMatrix(self):          """ This method is used to print              the matrix of the node.          """          print("---------------------")          for arr in self.matrix :              for angka in arr :                  print("|",end="")                  if angka == 0 :                      print("   ",end=" ")                  elif angka < 10 :                      print(" ",angka,end=" ")                  else :                      print("",angka,end=" ")              print("|")              print("---------------------")      def locateBlank(self):          """ This method is used to get              the X of the matrix.          Returns:              int : the X of the matrix          """          result = np.where(self.matrix == 0)          return (result[0][0] + result[1][0])%2      def kurang\_i(self):          """ This method is used to get              the kurang(i) of the matrix          Returns:              array of integers : each index i of the              array refers to the value of kurang(i).          """          temp = self.matrix          temp = temp.flatten()          arr = [0 for i in range (16)]          for i in range(len(temp)-1):              count = 0 # Count untuk mendapatkan kurang\_i untuk tiap i              for j in range (i+1, len(temp)):                  if (temp[j] < temp[i] and temp[j] != 0) or (temp[i] == 0):                      count += 1              arr[temp[i]] = count          return arr      def isSolved(self):          """ This method is used to check whether              the matrix is in final state or not.          Returns:              boolean : True if matrix is solved,              False if otherwise.          """          list = [i for i in range (1,16)]          list.append(0)          list = np.array(list).reshape((4,4))          return (self.matrix==list).all()      def countSyarat(self):          """ This method is used to count the condition              which is the sum of kurang(i) plus X.          Returns:              integer : the sum of kurang(i) + X          """          return sum(self.kurang\_i()) + self.locateBlank()      def countCost(self):          """ This method is used to count              the cost of the matrix.          Returns:              int : the cost of the matrix          """          temp = self.matrix.flatten()          count = 0          for i in range(16) :              if (temp[i] != (i+1) and temp[i] != 0):                  count += 1          return count      def isSolvable(self) :          """ This method is used to check whether              the puzzle is solveable or not.          Returns:              Boolean : If the condition is even then              solvable so return True, False if otherwise          """          if (self.countSyarat())%2 == 0:              return True          else:              return False      def moveBlankLeft(self) :          """ This method is used to return a              matrix where the blank is moved left          Returns:              numpy 2d-array : the matrix              of integers where we store the puzzle.          """          result = np.where(self.matrix == 0)          x = result[0][0]          y = result[1][0]          hasil = self.matrix.copy()          hasil[x,y-1] = self.matrix[x,y]          hasil[x,y] = self.matrix[x,y-1]            return hasil        def moveBlankRight(self) :          """ This method is used to return a              matrix where the blank is moved right          Returns:              numpy 2d-array : the matrix              of integers where we store the puzzle.          """          result = np.where(self.matrix == 0)          x = result[0][0]          y = result[1][0]          hasil = self.matrix.copy()          hasil[x,y+1] = self.matrix[x,y]          hasil[x,y] = self.matrix[x,y+1]          return hasil        def moveBlankUp(self) :          """ This method is used to return a              matrix where the blank is moved up          Returns:              numpy 2d-array : the matrix              of integers where we store the puzzle.          """          result = np.where(self.matrix == 0)          x = result[0][0]          y = result[1][0]          hasil = self.matrix.copy()          hasil[x-1,y] = self.matrix[x,y]          hasil[x,y] = self.matrix[x-1,y]          return hasil        def moveBlankDown(self) :          """ This method is used to return a              matrix where the blank is moved down          Returns:              numpy 2d-array : the matrix              of integers where we store the puzzle.          """          result = np.where(self.matrix == 0)          x = result[0][0]          y = result[1][0]          hasil = self.matrix.copy()          hasil[x+1,y] = self.matrix[x,y]          hasil[x,y] = self.matrix[x+1,y]          return hasil      def \_\_lt\_\_(self, other):          """ Function overloading of the node              for the lower than operator. Set to              True so the newest node will be checked last.          Args:              other (\_type\_): \_description\_          Returns:              \_type\_: \_description\_          """          return False  class Solver:      """ This class is what we use to contain          the nodes and solve the puzzle          Attributes:          - checked (array): an array of nodes          where the nodes have been checked before          - queue (prioqueue) : a prioqueue of nodes that          haven't been checked and use the cost as the priority          - mapMatrix (dictionary) : a hashmap to check whether          the node has already been added or not          - startMatrix (node) : the start of the puzzle or          the root of the nodes.          - solusi (node) : the node that is the solution          to the puzzle.      """      def \_\_init\_\_(self):          """ The constructor for the Solver.          """          self.checked = []          self.queue = PriorityQueue()          self.mapMatrix = {}          self.startMatrix = Node(["-"],0)          self.solusi = Node(["-"],0)      def bangkitkanSimpul(self):          """ The looping part of the solver.              Here, we continue to get the child              nodes of the node most prioritized              in the priority queue.              For each child node we raise, we put them              into the priorityqueue with it's cost as              the priority.          """          while True:              # Checking the current node.              node = self.queue.get()[1]              self.checked.append(node)              result = np.where(node.matrix == 0)              x = result[0][0]              y = result[1][0]              currMove = node.prev              if node.isSolved(): # If current node is solution, stop.                  self.solusi = node                  break              # If blank is not at the top, then get the child              # node where the blank moves up.              if (x != 0 and node.prev[len(node.prev)-1] != "DOWN"):                  newNode = Node(currMove + ["UP"] , len(node.prev))                  newNode.matrix = node.moveBlankUp()                  newNode.cost += newNode.countCost()                  if newNode.matrix.tobytes() not in self.mapMatrix.keys() :                      self.mapMatrix[newNode.matrix.tobytes()] = True                      self.queue.put((newNode.cost, newNode))                      if newNode.isSolved():                          self.solusi = newNode # If child node is solution, stop.                          break              # If blank is not at the buttom, then get the child              # node where the blank moves down.              if (x != 3 and node.prev[len(node.prev)-1] != "UP"):                  newNode = Node(currMove + ["DOWN"],len(node.prev))                  newNode.matrix = node.moveBlankDown()                  newNode.cost += newNode.countCost()                  if newNode.matrix.tobytes() not in self.mapMatrix.keys() :                      self.mapMatrix[newNode.matrix.tobytes()] = True                      self.queue.put((newNode.cost, newNode))                      if newNode.isSolved():                          self.solusi = newNode # If child node is solution, stop.                          break              # If blank is not at the most left, then get the child              # node where the blank moves left              if (y != 0 and node.prev[len(node.prev)-1] != "RIGHT"):                  newNode = Node(currMove + ["LEFT"],len(node.prev))                  newNode.matrix = node.moveBlankLeft()                  newNode.cost += newNode.countCost()                  if newNode.matrix.tobytes() not in self.mapMatrix.keys() :                      self.mapMatrix[newNode.matrix.tobytes()] = True                      self.queue.put((newNode.cost, newNode))                      if newNode.isSolved():                          self.solusi = newNode # If child node is solution, stop.                          break              # If blank is not at the most right, then get the child              # node where the blank moves right              if (y != 3 and node.prev[len(node.prev)-1] != "LEFT"):                  newNode = Node(currMove + ["RIGHT"],len(node.prev))                  newNode.matrix = node.moveBlankRight()                  newNode.cost += newNode.countCost()                  if newNode.matrix.tobytes() not in self.mapMatrix.keys() :                      self.mapMatrix[newNode.matrix.tobytes()] = True                      self.queue.put((newNode.cost, newNode))                      if newNode.isSolved():                          self.solusi = newNode # If child node is solution, stop.                          break      def cetakSolusi(self):          """ This method is used to print the amount of nodes raised,              the amount of steps needed to get the solution,              and print a matrix for each of the step.          """          print("Solusi sudah ditemukan!\n")          print("Banyaknya simpul yang dibangkitkan:", self.queue.qsize() + len(self.checked))          print("Banyak steps:", len(self.solusi.prev) - 1 ) # the amount of steps          temp = self.startMatrix          for i in range (1,len(self.solusi.prev)):              print("Step ke-" + str(i) +": ")              if (self.solusi.prev[i] == "RIGHT") :                  print("Command : Move blank right")                  temp.matrix = temp.moveBlankRight()              if (self.solusi.prev[i] == "UP") :                  print("Command : Move blank up")                  temp.matrix = temp.moveBlankUp()              if (self.solusi.prev[i] == "LEFT") :                  print("Command : Move blank left")                  temp.matrix = temp.moveBlankLeft()              if (self.solusi.prev[i] == "DOWN") :                  print("Command : Move blank down")                  temp.matrix = temp.moveBlankDown()              temp.printMatrix()              print()          print("Banyaknya simpul yang dibangkitkan:", self.queue.qsize() + len(self.checked))      def solve(self):          """ This method is used to initiate the solving process              of the puzzle. This method will print the startmatrix,              the kurang(i), and the X of the startMatrix,              then determine whether the puzzle is solvable or not.              If the startMatrix is unsolvable, then the method prints so.              otherwise, the method will continue to the bangkitkanSimpul method.          """          print("\nMatriks awalnya: ")          self.startMatrix.printMatrix()          print()          print("Mencari tiap kurang(i): ")          arr = self.startMatrix.kurang\_i()          for i in range (1,16):              if i < 10 :                  print("Kurang("+str(i) +")  =", arr[i])              else :                  print("Kurang("+str(i) +") =", arr[i])          print("Kurang(16) =",arr[0])          print("Total sigma(i) =", sum(arr))          print("X =", self.startMatrix.locateBlank())          print("\nHasil Sigma kurang(i) + X:", self.startMatrix.countSyarat())          if (not self.startMatrix.isSolvable()):              print("Syarat bernilai ganjil, maka puzzle unsolvable.")          else:              print("Syarat bernilai genap, maka puzzle solvable.\n)  print(Sedang proses pencarian solusi... ")              self.startMatrix.cost = self.startMatrix.countCost()              self.queue.put((self.startMatrix.cost, self.startMatrix))              self.mapMatrix[self.startMatrix.matrix.tobytes()] = True              self.bangkitkanSimpul() |

1. **main.py** : Merupakan file yang dijalankan untuk mendapatkan solusi dari 15-Puzzle yang tersedia.

|  |
| --- |
| # NAMA  : SAUL SAYERS  # NIM   : 13520094  # KELAS : K-01 STRATEGI ALGORITMA  # MERUPAKAN FILE MAIN PUZZLESOLVER UNTUK TUCIL 3 STRATEGI ALGORITMA  import pyfiglet  import solver as s  import time  print("--------------------------------------------")  print(pyfiglet.figlet\_format("15-Puzzle"))  print("Welcome to Saul's 15-Puzzle Solver Program")  print("--------------------------------------------")  while (True) :      puzzlesolver = s.Solver()      print("Cara generate puzzle:")      print("1. Dari txt file")      print("2. Random puzzle generator")      choice = 0      while (choice != 1 and choice != 2):          try:              choice = int(input("Enter command here (Masukkan angka 1 atau 2) : "))          except :              print("Input salah, silahkan coba lagi!")      if choice == 1 :          puzzlesolver.startMatrix.readFile()      else :          puzzlesolver.startMatrix.generateMatrix()        waktuawal = time.time()      puzzlesolver.solve()      if puzzlesolver.startMatrix.isSolvable():          puzzlesolver.cetakSolusi()      waktuakhir = time.time()      print("Waktu eksekusi:", waktuakhir-waktuawal, "sekon")      print("Apakah anda ingin solve puzzle lain? (ketik y/n)")      lanjut = input("Enter choice here [defaultnya y] : ")      if lanjut == "n":          break      print()    print("Terimakasih telah menggunakan program saya :D") |

1. **Screenshots *input* dan *output***

Terdapat 8 testcase yang diujikan menggunakan program ini :

1. **pptkuliah.txt**

Testcase ini merupakan testcase puzzle yang didapatkan dari slide materi perkuliahan.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 3 | 4 | 15 |
| 2 |  | 5 | 12 |
| 7 | 6 | 11 | 14 |
| 8 | 9 | 10 | 13 |

Sigma kurang(i) + X : 37

Status : unsolvable

Waktu eksekusi : 0,02 sekon

Banyak simpul dibangkitkan : 0

Banyaknya steps : 0

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks

Description automatically generated

Membandingkan dengan yang ada pada slide perkuliahan :

Sebuah gambar berisi teks

Description automatically generated

Tabel kurang(i) dan hasil statusnya terlihat sama, maka luaran output program terbukti benar untuk kasus ini.

1. **unsolvable.txt**

Testcase ini digunakan untuk mencoba puzzle matrix yang tidak bisa disolve

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | 4 | 6 | 3 |
| 1 | 10 | 12 | 11 |
|  | 15 | 2 | 13 |
| 5 | 8 | 7 | 14 |

Sigma kurang(i) + X : 47

Status : unsolvable

Waktu eksekusi : 0.017 sekon

Banyak simpul dibangkitkan : 0

Banyaknya steps : 0

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks

Description automatically generated

1. **easy1.txt**

Testcase ini digunakan untuk mencoba puzzle yang sangat mudah diselesaikan

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 |  | 10 | 12 |
| 13 | 14 | 11 | 15 |

Sigma kurang(i) + X : 10

Status : solvable

Waktu eksekusi : 0.03 sekon

Banyak simpul dibangkitkan : 10

Banyaknya steps : 3

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks, plakat, cuplikan layar

Description automatically generated

1. **easy2.txt**

Testcase ini digunakan untuk mencoba puzzle yang cukup mudah diselesaikan

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 6 | 2 | 3 |
| 5 | 10 | 7 | 4 |
| 9 | 14 | 12 | 8 |
| 13 |  | 11 | 15 |

Sigma kurang(i) + X : 20

Status : solvable

Waktu eksekusi : 0.077 sekon

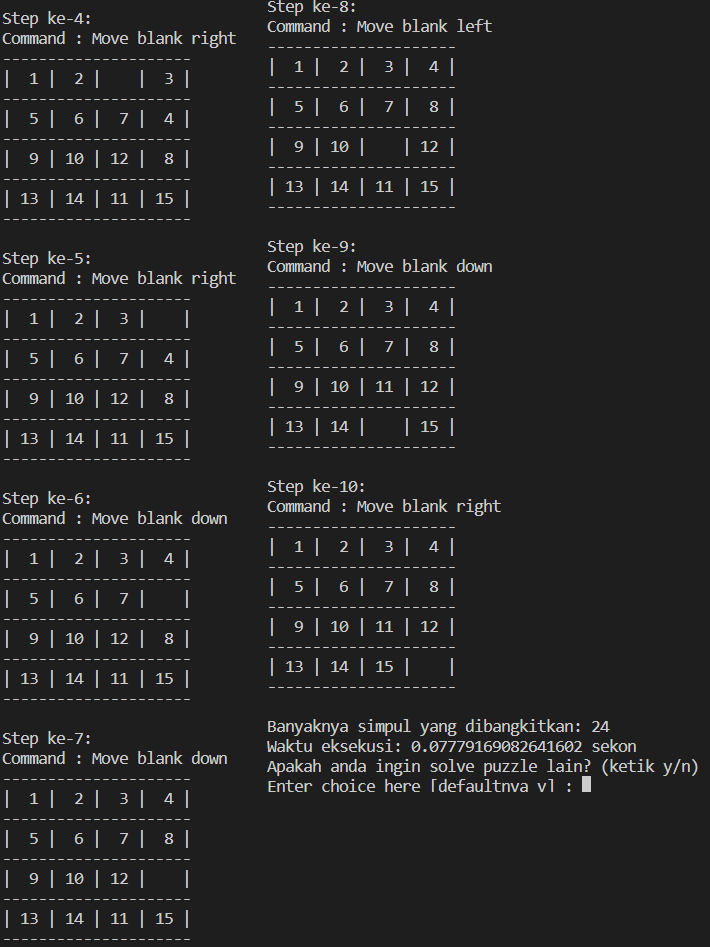
Banyak simpul dibangkitkan : 24

Banyaknya steps : 10

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks, plakat, cuplikan layar

Description automatically generated



1. **medium1.txt**

Testcase ini digunakan untuk mencoba puzzle dengan difficulty sedang

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 7 | 6 | 4 |
| 1 | 3 | 11 | 8 |
| 5 |  | 9 | 12 |
| 13 | 10 | 14 | 15 |

Sigma kurang(i) + X : 26

Status : solvable

Waktu eksekusi : 0.117 sekon

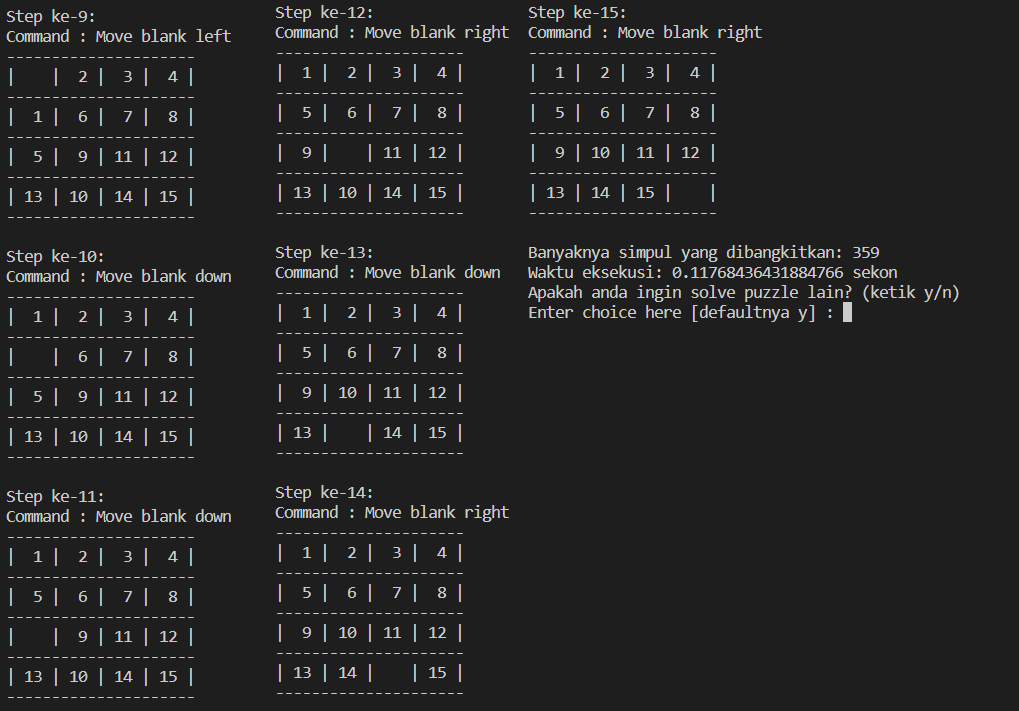
Banyak simpul dibangkitkan : 359

Banyaknya steps : 15

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks, plakat, papan skor

Description automatically generated



1. **medium2.txt**

Testcase ini digunakan untuk mencoba puzzle dengan difficulty sedang

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 5 | 7 | 3 |
| 9 | 1 | 4 |  |
| 10 | 11 | 6 | 15 |
| 13 | 14 | 12 | 8 |

Sigma kurang(i) + X : 34

Status : solvable

Waktu eksekusi : 0.26 sekon

Banyak simpul dibangkitkan : 2579

Banyaknya steps : 20

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks, plakat, papan skor

Description automatically generated

Sebuah gambar berisi teks, papan skor, plakat

Description automatically generated

Sebuah gambar berisi teks

Description automatically generated

1. **hard1.txt**

Testcase ini digunakan untuk mencoba puzzle dengan difficulty sulit

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | 11 | 2 | 4 |
| 9 | 1 | 3 | 7 |
| 13 | 10 | 6 | 8 |
|  | 14 | 15 | 12 |

Sigma kurang(i) + X : 34

Status : solvable

Waktu eksekusi : 0.6 sekon

Banyak simpul dibangkitkan : 10091

Banyaknya steps : 21

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks

Description automatically generated

Sebuah gambar berisi teks, plakat, papan skor

Description automatically generated

Sebuah gambar berisi teks

Description automatically generated

1. **hard2.txt**

Testcase ini digunakan untuk mencoba puzzle dengan difficulty sulit

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 5 | 2 | 3 |
| 1 | 8 | 4 | 12 |
| 14 | 10 | 9 | 7 |
| 13 |  | 11 | 15 |

Sigma kurang(i) + X : 28

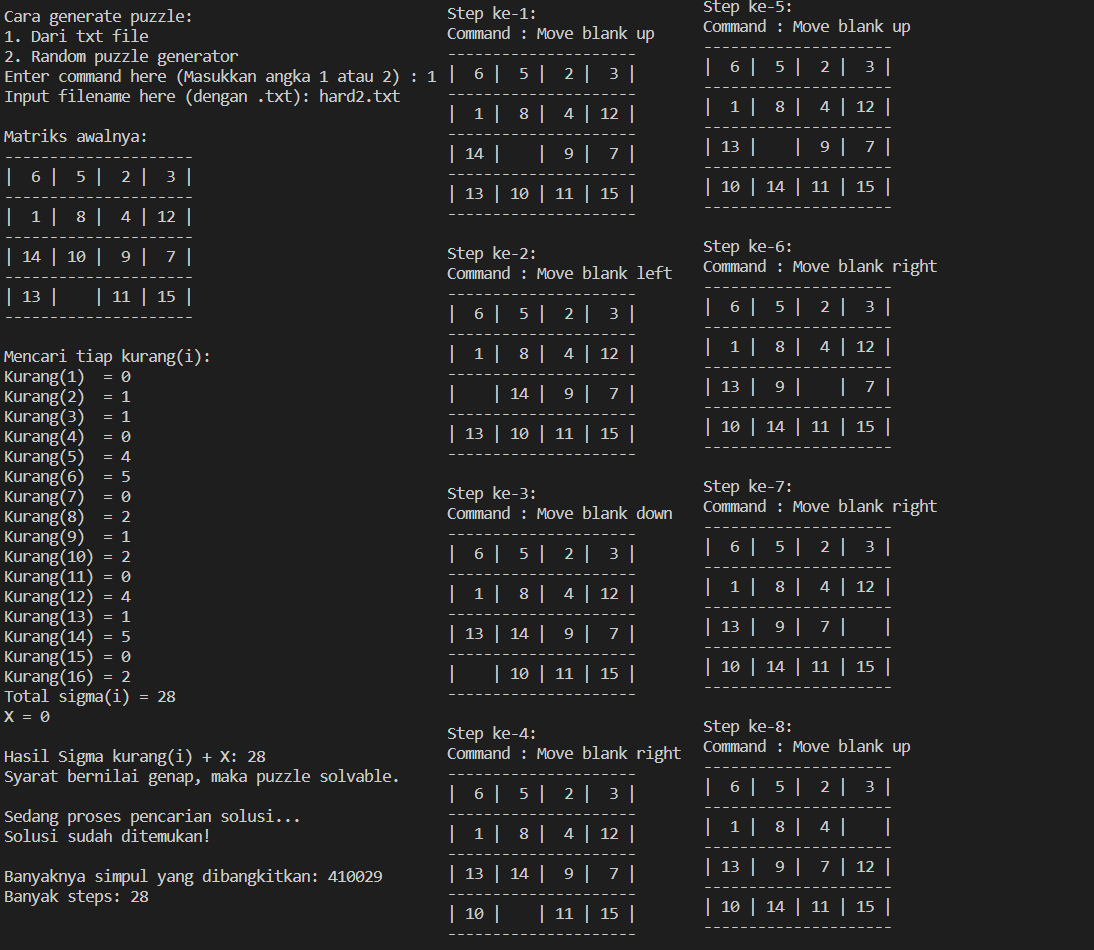
Status : solvable

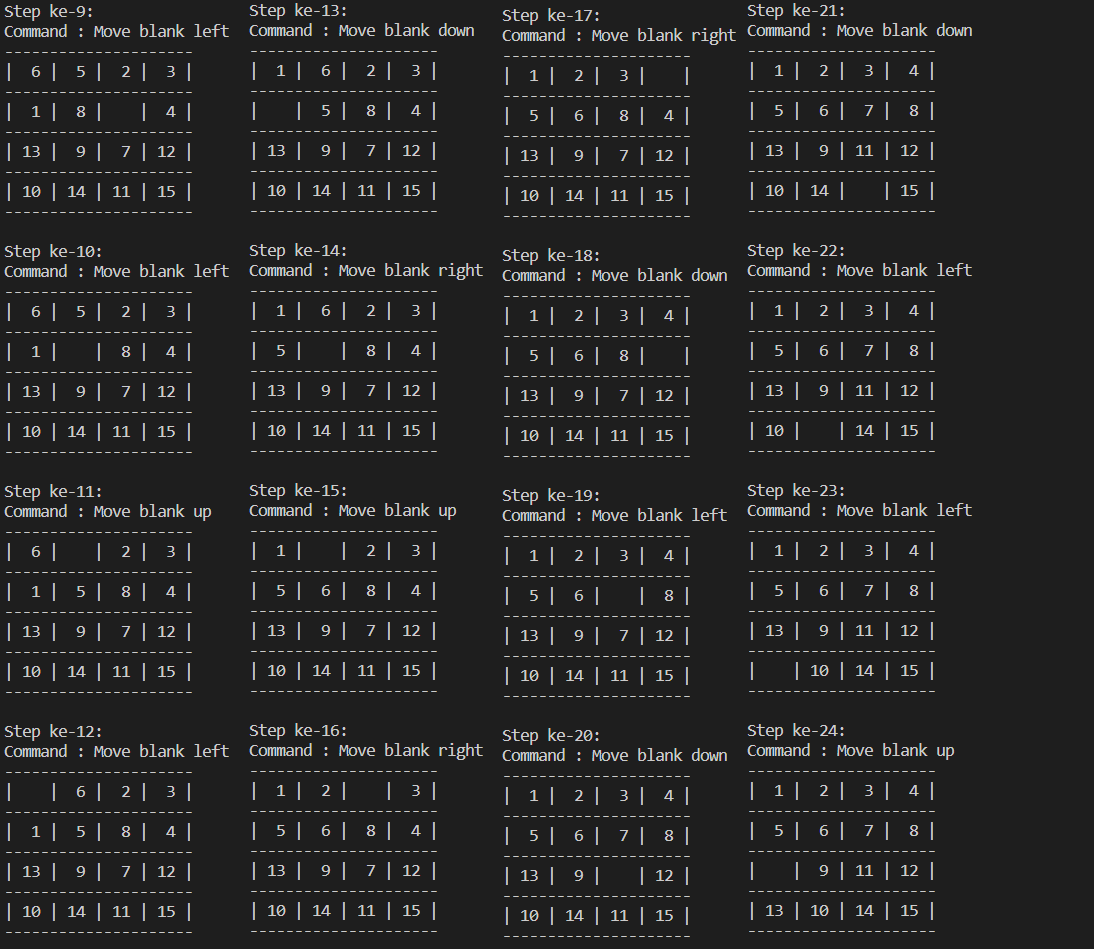
Waktu eksekusi : 19,6 sekon

Banyak simpul dibangkitkan : 410029

Banyaknya steps : 28

Berikut adalah screenshots hasil eksekusinya :





Sebuah gambar berisi teks, papan skor, plakat

Description automatically generated

1. **hard3.txt**

Testcase ini digunakan untuk mencoba puzzle dengan difficulty sulit

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | 2 | 8 | 10 |
| 1 | 11 | 6 | 4 |
| 7 | 9 |  | 3 |
| 13 | 14 | 15 | 12 |

Sigma kurang(i) + X : 34

Status : solvable

Waktu eksekusi : 87,8 sekon

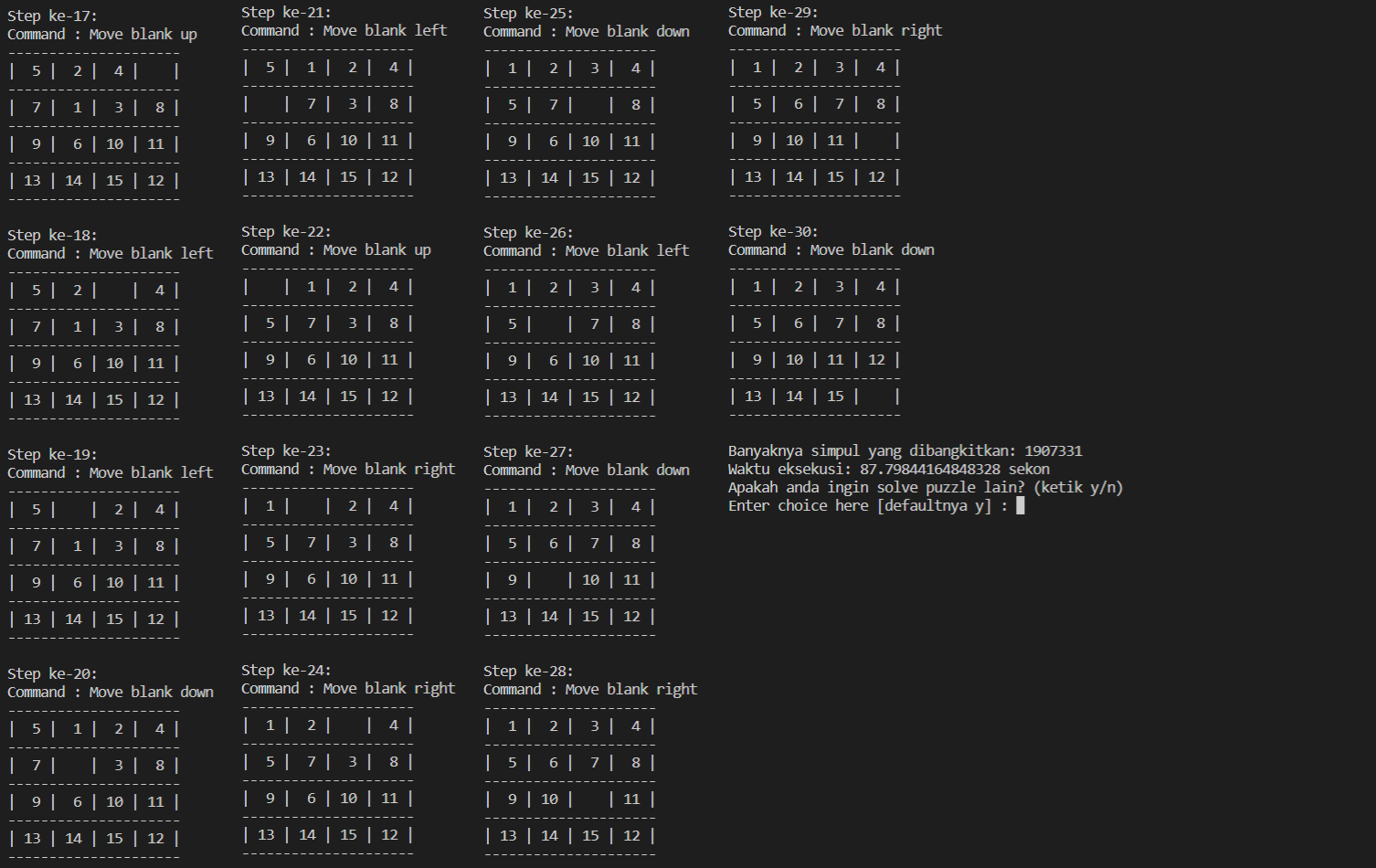
Banyak simpul dibangkitkan : 1907331

Banyaknya steps : 30

Berikut adalah screenshots hasil eksekusinya :

Sebuah gambar berisi teks, plakat, papan skor

Description automatically generated



1. **Link to Repository (Drive Source Code)**

<https://github.com/saulsayerz/Tucil3_13520094>

1. **Tabel Checklist**

Sebuah gambar berisi meja

Description automatically generated