

Laporan Tugas Kecil  
IF2211 Strategi Algoritma



Dibuat oleh:

13520026 Muhammad Fajar Ramadhan

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung  
Jl. Ganesha 10, Bandung 40132

## DAFTAR ISI

Algoritma <i>Branch and Bound</i> .....	3
Test Case .....	4

## Algoritma *Branch and Bound*

*Branch and Bound* merupakan salah satu algoritma dalam menyelesaikan permasalahan komputasi. Algoritma ini biasa digunakan untuk persoalan optimasi yaitu meminimalkan atau memaksimalkan suatu fungsi objektif. Dalam algoritma *Branch and Bound* terdapat beberapa hal yang harus diperhatikan

1. Setiap simpul diberi sebuah nilai cost, yaitu nilai taksiran lintasan termurah ke simpul status tujuan yang melalui sebuah simpul status
2. Simpul berikutnya yang akan di-expand tidak lagi berdasarkan urutan pembangkitannya, tetapi berdasarkan cost tergantung dari permasalahan minimasi atau optimasi

Dalam menyelesaikan permasalahan *15-Puzzle Problem* ini strategi algoritma yang diterapkan adalah sebagai berikut.

1. Cek apakah simpul akhir atau goal state dapat dicapai dari status awal dengan menggunakan fungsi kurang.
2. Fungsi kurang adalah banyaknya ubin bernomor  $j$  sedemikian sehingga  $j < i$  dan  $POSISI(j) > POSISI(i)$ .  $POSISI(i)$  = posisi ubin bernomor  $i$  pada susunan yang diperiksa. Status tujuan hanya dapat dicapai jika jumlah nilai fungsi kurang ditambah letak kotak kosong bergantung pada arsip adalah genap
3. Jika puzzle dapat diselesaikan bangkitkan simpulawal root dengan state awal dan hitung costnya. Cost pada algoritma ini adalah  $c(i) = f(i) + g(i)$ . Dengan  $f(i)$  ongkos mencapai simpul  $i$  dari akar dan  $g(i)$  adalah ongkos mencapai simpul tujuan dari simpul  $i$
4. Perhitungan  $g(i)$  adalah jumlah ubin tidak kosong yang tidak sesuai dengan susunan akhir
5. Bangkitkan simpul anak berikutnya dengan penukaran kotak kosong tergantung Gerakan yang dilakukan. Hitung cost pada node baru dan masukkan kedalam list simpul hidup
6. Bangkitkan simpul hidup dengan cost paling kecil dan lakukan hal yang sama hingga ditemukan goal state

## Test Case

- Masukkan

```
~~~~~
|  Welcome to the 15-puzzle solver!  |
~~~~~
|                                     |
|                                     |
-----
Choose your input:
1. Random Generator
2. From Text File
```

- Solvable 1

```
Input your test file name (ex: test1.txt) : Solvable1.txt
-----

Your initial matrix:
#---#---#---#---#
| 1 | 2 | 3 | 4 |
#---#---#---#---#
| 5 | 6 |   | 8 |
#---#---#---#---#
| 9 | 10 | 7 | 11 |
#---#---#---#---#
| 13 | 14 | 15 | 12 |
#---#---#---#---#
```

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

Nilai fungsi kurang: 16

```
Your initial matrix is solvable!
Here is the steps to solve it:
#---#---#---#---#
| 1 | 2 | 3 | 4 |
#---#---#---#---#
| 5 | 6 |   | 8 |
#---#---#---#---#
| 9 | 10 | 7 | 11 |
#---#---#---#---#
| 13 | 14 | 15 | 12 |
#---#---#---#---#
|
|
V
#---#---#---#---#
| 1 | 2 | 3 | 4 |
#---#---#---#---#
| 5 | 6 | 7 | 8 |
#---#---#---#---#
| 9 | 10 |   | 11 |
#---#---#---#---#
| 13 | 14 | 15 | 12 |
#---#---#---#---#
```

```

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

```

```

|
|

```

V

```

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

```

Elapsed Time for solving: 0.0169985294342041

Total Node: 10

- Unsolvable1

Your initial matrix:

```

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

```

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

Nilai fungsi kurang: 63

Your initial matrix is not solvable!

## Kode Program

```
- Solver.py
- # Library algoritma
-
- import copy
- from heapq import heappop, heappush
-
- def Reachable(arr):
-     costCount = 0
-     dict =
- {1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,16:0
- }
-     for i in range(len(arr)):
-         for j in range(len(arr[0])):
-             costG = 0
-             k = i
-             l = j
-             while(k<len(arr)):
-                 while(l<len(arr[0])):
-                     if(arr[i][j] > arr[k][l]):
-                         costG += 1
-                     l += 1
```

```

-         l = 0
-         k += 1
-         dict[arr[i][j]] = costG
-         costCount += costG
-     x,y = findBlankCoor(arr)
-     if(((x+1)%2!=0) and ((y+1)%2==0)):
-         costCount += 1
-     elif(((x+1)%2==0) and ((y+1)%2!=0)):
-         costCount += 1
-     return costCount,dict
-
- def calculateCost(mat, final):
-     count = 0
-     for i in range(len(mat)):
-         for j in range(len(mat)):
-             if ((mat[i][j] != final[i][j])):
-                 count += 1
-
-     return count
-
- class PrioQueue:
-     def __init__(self):
-         self.pq = []
-
-     def isEmpty(self):
-         if not self.pq:
-             return True
-         else:
-             return False
-
-     def enqueue(self,item):
-         heappush(self.pq, item)
-
-     def dequeue(self):
-         return heappop(self.pq)
-
- def isSafe(x, y):
-     return x >= 0 and x < 4 and y >= 0 and y < 4
-
- def findBlankCoor(matrix):
-     for i in range(len(matrix)):
-         for j in range(len(matrix[0])):
-             if(matrix[i][j]==16):
-                 return (i,j)
-
- class node:
-     totalNode = 0
-     def __init__(self,parent,matrix,cost,depth):

```

```

-         self.parent = parent
-         self.matrix = matrix
-         self.cost = cost
-         self.depth = depth
-         node.totalNode += 1
-
-     def __lt__(self, nxt):
-         return (self.cost + self.depth) < (nxt.cost + nxt.depth)
-
- def newNode(parent, matrix, final, move):
-     newMatrix = copy.deepcopy(matrix)
-     blankCoor = findBlankCoor(matrix)
-     x = blankCoor[0] + move[0]
-     y = blankCoor[1] + move[1]
-     newMatrix[x][y], newMatrix[blankCoor[0]][blankCoor[1]] =
-     newMatrix[blankCoor[0]][blankCoor[1]], newMatrix[x][y]
-
-     if (parent.parent is not None):
-         if (parent.parent.matrix == newMatrix):
-             return None
-         else:
-             cost = calculateCost(newMatrix, final)
-             depth = parent.depth + 1
-             return node(parent, newMatrix, cost, depth)
-     else:
-         cost = calculateCost(newMatrix, final)
-         depth = parent.depth + 1
-         return node(parent, newMatrix, cost, depth)
-
- def solvePuzzle(initial, final):
-     move = [[1,0],[0,1],[-1,0],[0,-1]]
-     solved = False
-
-     pq = PriorityQueue()
-     cost = calculateCost(initial, final)
-     root = node(None, initial, cost, 0)
-     pq.enqueue(root)
-
-     while (not pq.isEmpty() and not solved):
-         current = pq.dequeue()
-         if (current.cost == 0):
-             solved = True
-         else:
-             for i in range(len(move)):
-                 temp = findBlankCoor(current.matrix)
-                 x = temp[0] + move[i][0]
-                 y = temp[1] + move[i][1]
-                 if (isSafe(x, y)):

```



```

-         tempNode =
-         newNode(current,current.matrix,final,move[i])
-             if(tempNode is not None):
-                 pq.enqueue(tempNode)
-
-         return current
-
- def printMatrix(matrix):
-     for i in range(len(matrix)):
-         print("#----#----#----#----#")
-         for j in range(len(matrix[0])):
-             if(matrix[i][j]==16):
-                 print("|    ",end="")
-             elif(matrix[i][j]>9):
-                 print("| "+str(matrix[i][j]),end="")
-             else:
-                 print("| " + str(matrix[i][j]) + " ",end="")
-             if(j==3):
-                 print(" |",end="")
-             else:
-                 print(" ",end="")
-         print()
-     print("#----#----#----#----#")
-
- def printPath(node):
-     if(node.parent is not None):
-         printPath(node.parent)
-         print("\t |")
-         print("\t |")
-         print("\t V")
-     printMatrix(node.matrix)
-
- def printKurangFunc(initial):
-     flag,kurangDict = Reachable(initial)
-     print("-----")
-     print("|      i      |      kurang(i)      |")
-     print("-----")
-     for i in range(1,17):
-         if(i<10):
-             if(kurangDict[i]>9):
-                 print("|      "+str(i)+"      |      "+str(kurangDict[i])
- )+"      |")
-             else:
-                 print("|      "+str(i)+"      |      "+str(kurangDict[i])
- )+"      |")
-         else:
-             if(kurangDict[i]>9):
-                 print("|      "+str(i)+"      |      "+str(kurangDict[i])
- "+"      |")

```

```

-         else:
-             print("|      "+str(i)+"      |      "+str(kurangDict[i])
+"      |")
-
-     print("-----")
-     print("Nilai fungsi kurang: ",flag)
-

```

- Main.py

```

- import Solver as Sv
- import os
- import random
- import time
-
- if __name__ == "__main__":
-     print("~~~~~")
-     print("|   Welcome to the 15-puzzle solver!   |")
-     print("~~~~~")
-     print("|                                     |")
-     print("|                                     |")
-     print("-----")
-     print("Choose your input:")
-     initialMatrix = []
-     finalMatrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
-     n = int(input("1. Random Generator\n2. From Text File\n"))
-     if(n==1):
-         randomlist2 = random.sample(range(1, 5), 4)
-         randomlist3 = random.sample(range(5, 9), 4)
-         randomlist = random.sample(range(9, 17), 8)
-         initialMatrix.append(randomlist2)
-         initialMatrix.append(randomlist3)
-         k = 0
-         for i in range(2):
-             temp = []
-             for j in range(4):
-                 temp.append(randomlist[k])
-                 k+=1
-             initialMatrix.append(temp)
-     else:
-         tempMatrix = []
-         print("Ket: file harus berada dalam folder test")
-         fileName = input("Input your test file name (ex: test1.txt) :
")

```

```

-     path = r'../test/'
-     with open(path+fileName, 'r') as f:
-         for line in f.readlines():
-             tempMatrix.append(line.split(' '))
-
-     for i in range(len(tempMatrix)):
-         temp = []
-         for j in range(len(tempMatrix[0])):
-             temp.append(int(tempMatrix[i][j]))
-         initialMatrix.append(temp)
-
-     print("-----")
-     print("\nYour initial matrix: ")
-     Sv.printMatrix(initialMatrix)
-     print("\n")
-     Sv.printKurangFunc(initialMatrix)
-     value,_ = Sv.Reachable(initialMatrix)
-     if(value%2==0):
-         print("\nYour initial matrix is solvable!")
-         print("Here is the steps to solve it:")
-         start = time.time()
-         solvedPuzzle = Sv.solvePuzzle(initialMatrix, finalMatrix)
-         Sv.printPath(solvedPuzzle)
-         end = time.time()
-         print("\nElapsed Time for solving: ", end-start)
-         print("\nTotal Node: ",solvedPuzzle.totalNode)
-     else:
-         print("\nYour initial matrix is not solvable!")

```

## Berkas Teks

Pada folder test

- Solvable : Solvable1.txt, Solvable2.txt, Solvable3.txt
- Unsolvble: Unsolvble1.txt, Unsolvble2.txt

## Referensi

- <https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/>
- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf>

## Link Drive

- [https://github.com/fajarm15/Tucil3\\_13520026](https://github.com/fajarm15/Tucil3_13520026)

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

