LAPORAN TUGAS KECIL 3

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

IF2211 Strategi Algoritma

Oleh:

Farhan Hafiz 13520027

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

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

Algoritma Branch and Bound merupakan lanjutan dari algoritma Breadh-First Search yang memperhatikan persoalan optimisasi yaitu meminimalkan atau memaksimalkan suatu fungsi objektif , yang tidak melanggar Batasan constraints) persoalan. Algoritma B&B juga menerapkan pemangkasan ” pada jalur yang dianggap tidak lagi mengarah pada solusi. Kriteria pemangkasan secara umum, Nilai simpul tidak lebih baik dari nilai terbaik sejauh ini, Simpul tidak merepresentasikan solusi yang ‘feasible’ karena ada batasan yang dilanggar, Solusi pada simpul tersebut hanya terdiri atas satu titik.

Cara kerja dari algoritma branch and bound secara umum adalah sebagai berikut, misal terdapat sebuah antrian dan sebuah simpulawal yang akan dilakukan pencarian ke sebuah simpulsolusi maka

1.Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (goal simpul), maka solusi telah ditemukan. Stop.

2.Jika Q kosong, tidak ada solusi. Stop.

3.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 secara sembarang.

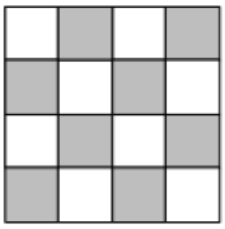
4.Jika simpul i adalah simpul solusi, berarti solusi sudah ditemukan, stop. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika i tidak mempunyai anak, kembali ke langkah 2.

5.Untuk setiap anak j dari simpul i, hitung ĉ(j), danmasukkan semua anak-anak tersebut ke dalam Q. 6.Kembali ke langkah 2.

15-Puzzle merupakanpermainan dimana terdapat persegi berukuran 4x4 satuan ubin.. Didalam persegi tersebut akan terdapat 15 petak yang terisi ubin yang tersusun secara acak. Pemain harus menyelesaikan puzzle tersebut sehingga seluruh ubin pada puzzle terurut menaikdari sebelah kiri-atas ke kanan bawah. Pemain menata pola tersebutdengan memanfaatkan petakkosong yang digunakan untuk mengubah posisi ubin lainnya.

Langkah-langkah Algoritma *Branch and Bound* dalam Penyelasaian 15-Puzzle :

1. Pastikan persoalan dapat diselesaikan dengan menghitung fungsi $\sum_{i=1}^{16} KURANG(i) + X$ bernilai genap.
   Nilai X bernilai 1 jika berada di pada ubin yang diarsir seperti di bawah ini



KURANG(i) bisa didapat dari banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > i , POSISI(i) = posisi ubin bernomor i pada susunan yang diperiksa. Apabila fungsi tersebut bernilai ganjil, maka persoalan tidak dapat diselesaikan, sebaliknya jika fungsi tersebut bernilai genap, dapat ke Langkah selanjutnya

2. Menghitung jumlah cost dari simpul hidup.
   Cost yang dihitung pada tiap simpul hidup meruapakan taksiran yang didapat dengan cara sebagai berikut

   $$\hat{c}(P) = f(P) + \hat{g}(P)$$

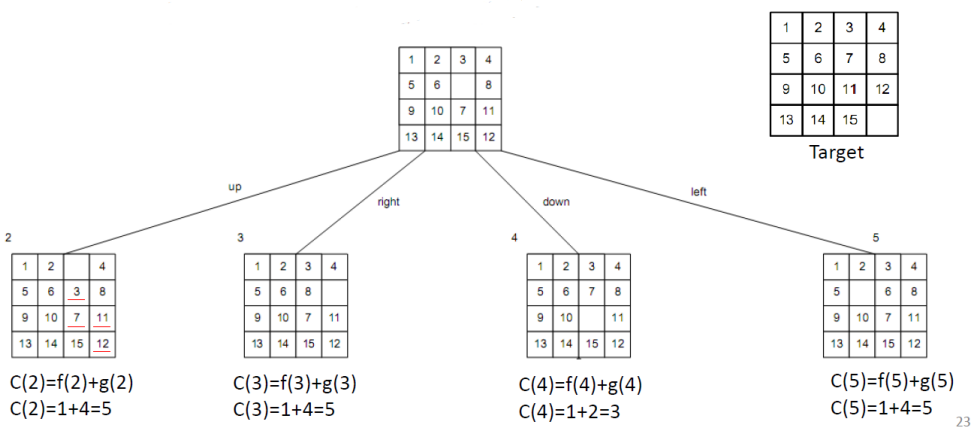
   $f(P)$ = adalah panjang lintasan dari simpul akar ke P

   $\hat{g}(P)$ = taksiran panjang lintasan terpendek dari P ke simpul solusi pada upapohon yang akarnya P.

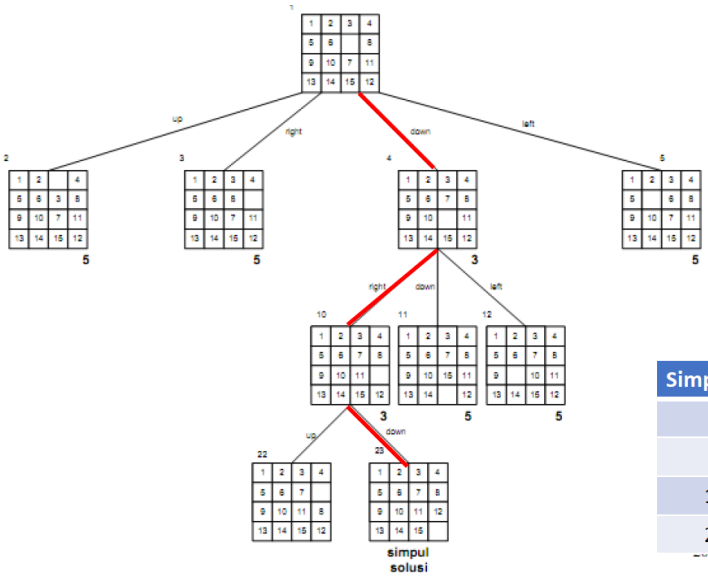
   Dimana

   $\hat{g}(P)$ = jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir

   Ilustrasi :



3. Ambil tiap simpul dengan cost paling kecil dan lakukan seterusnya hingga mencapai sususan seperti target



Simpul-E	Simpul Hidup
1	4,2,3,5
4	10,2,3,5,11,12
10	23,2,3,5,11,12,22
23	Solusi ketemu

Kode Program

Source Program dibuat dalam Bahasa python dengan nama ‘FifteenPuzzleSolver.py’ pada file ini terdapat beberapa bagian source code antara lain

Bagian Pembuatan Class

Terdapat beberapa class yang dibangun pada program ini untuk memudahkan penyelesaian persoalan yang diberikan

Class PriorityQueue :

```python
class PriorityQueue :
    # Konstruktor : Inisialisasi Priority Queue
    def __init__(self):
        self.heap = []

    # Melakukan Push : Menambahkan elemen num ke PQ
    def push(self, num):
        heappush(self.heap, num)

    # Melakukan Pop : Menghapus elemen pertama dari PQ
    def pop(self):
        return heappop(self.heap)

    # Mengecek isEmpty : bernilai true jika kosong, false jika sebaliknya
    def isEmpty(self):
        if not self.heap:
            return True
        else :
            return False
```

Class PriorityQueue digunakan untuk melakukan push dan pop dari simpul-simpul yang ada yang terah terisi secara berurutan sesuai cost-nya.

Class Node :

```
class node :
    def __init__(self, parent, matrix, empty_tile, cost, level):
        self.parent = parent

        self.matrix = matrix

        self.empty_tile = empty_tile

        self.cost = cost

        self.level = level

    def __lt__ (self, next):
        return self.cost < next.cost
```

Class node digunakan untuk menyimpan beberapa atribut seperti matriks, letak ubin yang kosong, cost yang digunakan, serta kedalaman simpul.

Class Table :

```
class Table:
    def __init__(self,root):

        # code for creating table
        for i in range(total_rows):
            for j in range(total_columns):
                if(lst[i][j] == 16):
                    self.e = Entry(root, width=3, fg='white', bg='white',
                                   font=('Arial',16,'bold'))

                    self.e.grid(row=i, column=j)
                    self.e.insert(END, lst[i][j])
                elif(lst[i][j] != 16):
                    self.e = Entry(root, width=3, fg='white', bg='blue',
                                   font=('Arial',16,'bold'))

                    self.e.grid(row=i, column=j)
                    self.e.insert(END, lst[i][j])
```

Class table digunakan untuk menampilkan penyelesaian dalam bentuk GUI

Bagian Mengecek Apakah Persoalan dapat diselesaikan

```python
# Mengecek apakah puzzle dapat diselesaikan
def isReachable (initial_matrix, empty_tile) -> bool :
    countX = 0
    countLess = 0

    # Menghitung fungsi Kurang(i)
    for i in range(4):
        for j in range(4):
            m = i
            n = j
            if(m==i):
                while(n<4):
                    if(initial_matrix[m][n]<initial_matrix[i][j]):
                        countLess += 1
                    n+=1
                m+=1
            while(m<4):
                n = 0
                while(n<4):
                    if(initial_matrix[m][n]<initial_matrix[i][j]):
                        countLess += 1
                    n+=1
                m+=1

    # Menghitung nilai X sesuai letak ubin kosong
    if ((empty_tile[0]+empty_tile[1]) %2 == 1):
        countX += 1
    print("\nNilai dari fungsi Kurang(i) = ", countLess)
    countTotal = countX + countLess
    print("Nilai dari fungsi Kurang(i) + X = ", countTotal, "\n")
    if (countTotal % 2 == 0):
        return True
    else :
        return False
```

Fungsi isReachable ini berguna untuk menghitung nilai dari fungsi $\sum_{i=1}^{16} KURANG(i) + X$. Jika bernilai genap maka persoalan dapat diselesaikan, jika sebaliknya, maka tidak dapat diselesaikan.

Bagian Menghitung cost

```python
# Menghitung banyaknya jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir
def countCost (initial_matrix, final_matrix) -> int :
    count = 0
    for i in range(4):
        for j in range(4):
            if ((initial_matrix[i][j]) and (initial_matrix[i][j] != final_matrix[i][j])):
                count += 1

    return count
```

Fungsi countCost akan menghasilkan jumlah cost yang diperlukan dari initial_matrix menuju target final_matrix.

Bagian Penyelesaian

```
def newNode (parent, matrix, empty_tile, new_empty_tile, level, final_matrix) -> node:
    # Menyalin matrix awal ke matrix baru
    new_matrix = copy.deepcopy(matrix)

    x1 = empty_tile[0]
    y1 = empty_tile[1]
    x2 = new_empty_tile[0]
    y2 = new_empty_tile[1]

    # Menukar antara empty_tile dan new_empty_tile setelah melakukan pergerakan
    new_matrix[x1][y1], new_matrix[x2][y2] =  new_matrix[x2][y2], new_matrix[x1][y1]

    # Menghitung banyaknya jumlah ubin yang berbeda dengan susunan akhir
    new_cost = countCost(new_matrix, final_matrix)

    new_node = node(parent, new_matrix, new_empty_tile, new_cost, level)
    return new_node
```

fungsi newNode akan menghasilkan node baru sesuai input initial_matrix, empty_tile, dan pergerakan yang diinginkan (atas,kanan,bawah, atau kiri) sesuai cost yang paling kecil.

```
# Menyelesaikan 15-Puzzle
def Solve(initial_matrix, empty_tile, final_matrix):

    # Inisialisasi Priority Queue
    pq = PriorityQueue()

    # Menghitung banyaknya jumlah ubin yang berbeda dengan susunan akhir
    current_cost = countCost(initial_matrix, final_matrix)

    # Membuat node baru berupa root dari node awal
    root_node = node(None, initial_matrix, empty_tile, current_cost, 0)

    # Melakukan push root_node sebelumnya ke dalam Priority Queue
    pq.push(root_node)
    visited_node = []
    visited_node.append(root_node)

    while not pq.isEmpty():
        is_visited = False
        min_node = pq.pop()

        # Jika cost = 0, artiny tidak ada lagi ubin yang berbeda dengan susunan akhir
        if min_node.cost == 0 :
            print("Jumlah simpul yang dibangkitkan = ", min_node.level+1, "\n")
            return min_node

        for i in range(4):
            new_tile = [min_node.empty_tile[0] + rows[i],
                        min_node.empty_tile[1] + cols[i],]

            if (isValid(new_tile[0], new_tile[1])):
                child = newNode(min_node, min_node.matrix, min_node.empty_tile, new_tile, min_node.level + 1, final_matrix)
                pq.push(child)
```

Fungsi Solve akan menyelesaikan persoalan dan menghasilkan simpul-simpul yang dibangkitkan hingga mencapai target final_matrix.

Bagian Memberikan Input

```
# Masukan matriks yang dibangkitkan secara acak
def generate_random():
    initial_matrix = np.arange(1, 17)
    np.random.shuffle(initial_matrix)
    initial_matrix = np.reshape(initial_matrix,(4,4))
    return initial_matrix
```

Fungsi generate_random akan dipanggil jika pengguna ingin melakukan input dengan integer secara acak.

```
# Masukan matriks yang dibangkitkan sesuai input pengguna
def input_user(input):
    with open(input, 'r') as f:
        initial_matrix = [[int(num) for num in line.split(',')] for line in f]
    return initial_matrix
```

Fungsi input_user akan dipanggil jika pengguna ingin melakukan input dengan menuliskan nama file

Bagian Menghasilkan Output

```
# Mengeluarkan output berupa matriks
def printMatrix(mat):

    for i in range(4):
        for j in range(4):
            if (mat[i][j] > 0 and mat[i][j] <= 9) :
                print("%d " % (mat[i][j]) + " ", end = " ")
            elif (mat[i][j] == 16) :
                 print("-  ", end = " ")
            else :
                print("%d " % (mat[i][j]), end = " ")
        print()

# Mengeluarkan semua path matriks dari solusi
def printPath(root_node):

    if root_node == None:
        return

    printPath(root_node.parent)
    printMatrix(root_node.matrix)
    print()
```

Fungsi di atas digunakan untuk mengeluarkan output berupa matriks dari simpul-simpul yang dibangkitkan dan melakukan print pada console yang digunakan.

```
# Mengeluarkan output berupa GUI dari solusi
def printPathGUI (root_node):
    global total_rows, total_columns, lst
    if root_node == None:
        return

    printPathGUI(root_node.parent)
    lst = root_node.matrix
    total_rows = len(lst)
    total_columns = len(lst[0])
    root = Tk()
    t = Table(root)
    root.mainloop()
```

Fungsi di atas digunakan untuk mengeluarkan output dalam bentuk GUI

Bagian Main Program

```
# Main Program
print("_____________________________")
print("|        15 Puzzle Solver       |")
print("|_____________________________|")
print("Pilih masukkan yang diinginkan : ")
print("1. Acak ")
print("2. Nama File")

# Memilih ingin masukkan berupa acak atau file
choice = int(input("Masukkan yang diinginkan (Pilih 1/2) = "))
# Jika memilih acak
if (choice == 1):
    initial_matrix = generate_random()
    empty_tile = find_empty_tile(initial_matrix)
    start_time = time.time()
    reach = isReachable(initial_matrix, empty_tile)
    if (reach == False):
        print("Waktu Eksekusi = %s detik\n" % (time.time() - start_time))
        printMatrix(initial_matrix)
        print("\nPersoalan diatas tidak dapat diselesaikan")
    else :
        min_node = Solve(initial_matrix, empty_tile, final_matrix)
        print("Waktu Eksekusi = %s detik\n" % (time.time() - start_time))
        printPath(min_node)
        is_GUI = input("\nApakah ingin menampilkan GUI dari solusi (ya/tidak) = ")
        if (is_GUI == "ya"):
            printPathGUI(min_node)
# Jika memilih nama file
elif (choice == 2):
    file_input = input("Masukkan nama file = ")
    initial_matrix = input_user(file_input)
    empty_tile = find_empty_tile(initial_matrix)
    start_time = time.time()
    reach = isReachable(initial_matrix, empty_tile)
    if (reach == False):
        print("Waktu Eksekusi = %s detik\n" % (time.time() - start_time))
        printMatrix(initial_matrix)
        print("\n Persoalan diatas tidak dapat diselesaikan")
    else :
        min_node = Solve(initial_matrix, empty_tile, final_matrix)
        printPath(min_node)
        print("Waktu Eksekusi = %s detik\n" % (time.time() - start_time))
        is_GUI = input("\nApakah ingin menampilkan GUI dari solusi (ya/tidak) = ")
        if (is_GUI == "ya"):
            printPathGUI(min_node)
else :
    print("Input salah")
```

Bagian di atas merupakan main program yang memiliki alur antara lain : Pengguna memilih apakah ingin menggunakan matriks secara acak atau melalui file, Program akan mengecek apakah persoalan dapat diselesaikan. Apabila tidak dapat diselesaikan dikeluarkan pesan. Apabila dapat diselesaikan, program akan mencetak waktu eksekusi, nilai fungsi Kurang(i), Jumlah simpul yang dibangkitkan, dan matriks-matriks dari simpul solusi. Program juga dapat mengeluarkan dalam bentuk GUI apabila diinginkan oleh pengguna.

Test Case

Catatan : 16 merupakan letak dimana ubin kosong

- **tes_gagal1.txt**

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

- **tes_gagal2.txt**

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

- **tes_berhasil1.txt**

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

- **tes_berhasil2.txt**

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

- **tes_berhasil3.txt**

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

Screenshot Input-Output Program

- **File tes_gagal1.txt**

  Input :

```
_______________________________
|        15 Puzzle Solver       |
|_______________________________|
Pilih masukkan yang diinginkan :
1. Acak
2. Nama File
Masukkan yang diinginkan (Pilih 1/2) = 2
Masukkan nama file = tes_gagal1.txt
```

  Output:

```
Nilai dari fungsi Kurang(i) =  37
Nilai dari fungsi Kurang(i) + X =  37

Waktu Eksekusi = 0.0013566017150878906 detik

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

 Persoalan diatas tidak dapat diselesaikan
```

- **File tes_gagal2.txt**

Input :

```
 _______________________________
|        15 Puzzle Solver       |
|_______________________________|
Pilih masukkan yang diinginkan :
1. Acak
2. Nama File
Masukkan yang diinginkan (Pilih 1/2) = 2
Masukkan nama file = tes_gagal2.txt
```

Output:

```
Nilai dari fungsi Kurang(i) =  73
Nilai dari fungsi Kurang(i) + X =  73

Waktu Eksekusi = 0.002008676528930664 detik

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

 Persoalan diatas tidak dapat diselesaikan
```

- **File tes_berhasil1.txt**

Input :

```
_______________________________
|        15 Puzzle Solver       |
|_______________________________|
Pilih masukkan yang diinginkan :
1. Acak
2. Nama File
Masukkan yang diinginkan (Pilih 1/2) = 2
Masukkan nama file = tes_berhasil1.txt
```

Output:

```
Nilai dari fungsi Kurang(i) =  15
Nilai dari fungsi Kurang(i) + X =  16

Jumlah simpul yang dibangkitkan =  4

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

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

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

Waktu Eksekusi = 0.012998104095458984 detik
```

- **File tes_berhasil2.txt**

Input :

```
_______________________________
|        15 Puzzle Solver       |
|_______________________________|
Pilih masukkan yang diinginkan :
1. Acak
2. Nama File
Masukkan yang diinginkan (Pilih 1/2) = 2
Masukkan nama file = tes_berhasil2.txt
```

Output:

```
Nilai dari fungsi Kurang(i) =  23
Nilai dari fungsi Kurang(i) + X =  24

Jumlah simpul yang dibangkitkan =  8

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

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

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

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

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

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

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

Waktu Eksekusi = 0.038954973220825195 detik
```

- **File tes_berhasil3.txt**

Input :

```
_______________________________
|        15 Puzzle Solver       |
|_______________________________|
Pilih masukkan yang diinginkan :
1. Acak
2. Nama File
Masukkan yang diinginkan (Pilih 1/2) = 2
Masukkan nama file = tes_berhasil3.txt
```

Output:

```
Jumlah simpul yang dibangkitkan =  14

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Waktu Eksekusi = 0.036002397537231445 detik
```

- **Masukkan secara acak**

  Input :

```
 _______________________________
|        15 Puzzle Solver       |
|_______________________________|
Pilih masukkan yang diinginkan :
1. Acak
2. Nama File
Masukkan yang diinginkan (Pilih 1/2) = 1
```

  Output :

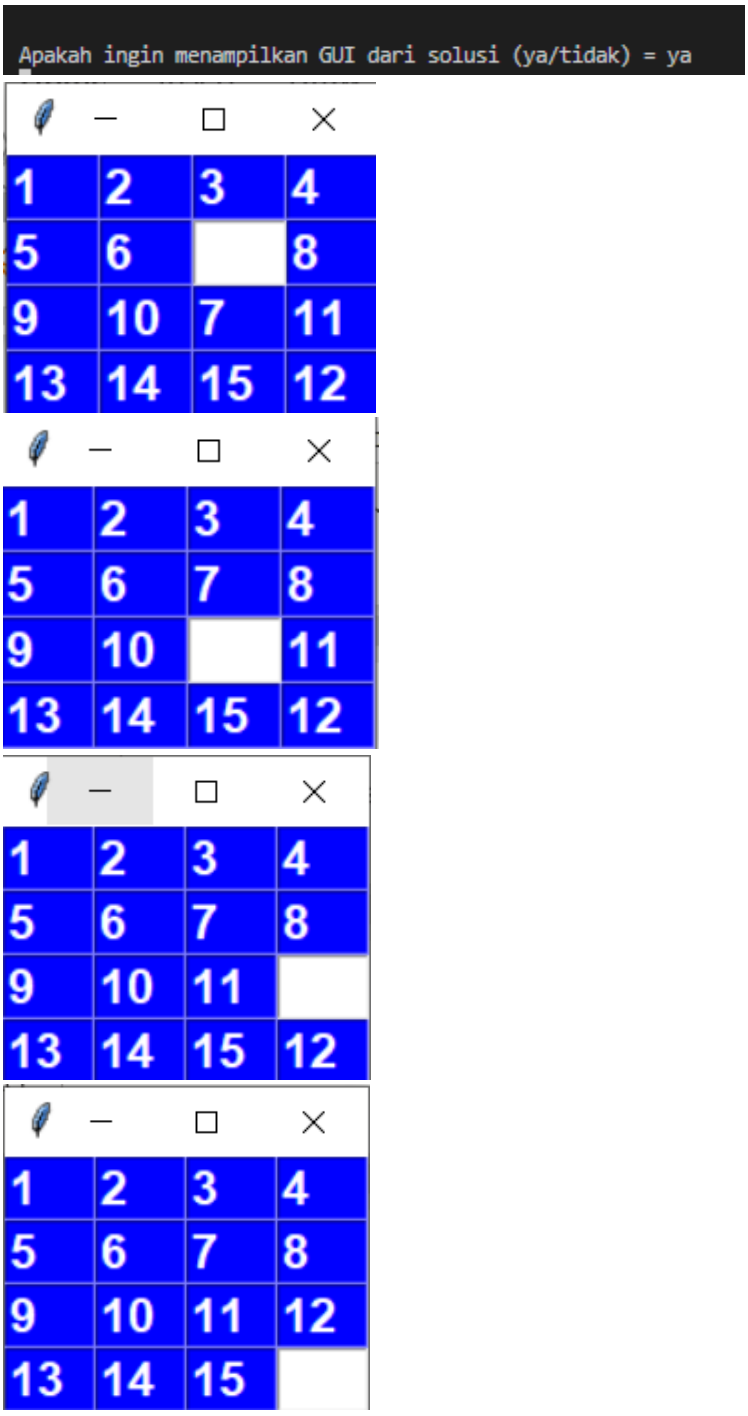
```
Nilai dari fungsi Kurang(i) =  59
Nilai dari fungsi Kurang(i) + X =  59

Waktu Eksekusi = 0.0010173320770263672 detik

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

Persoalan diatas tidak dapat diselesaikan
```

- **Contoh GUI (untuk file tes_berhasil1.txt) -Bonus-**



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	

Link Github Program

https://github.com/Fnhafiz/Tucil3_13520027.git

Drive Program (Alternative Link)

https://drive.google.com/drive/folders/1ZQNG6RTh0NSQ-NiSvFAbLJVeyQKWyxif?usp=sharing