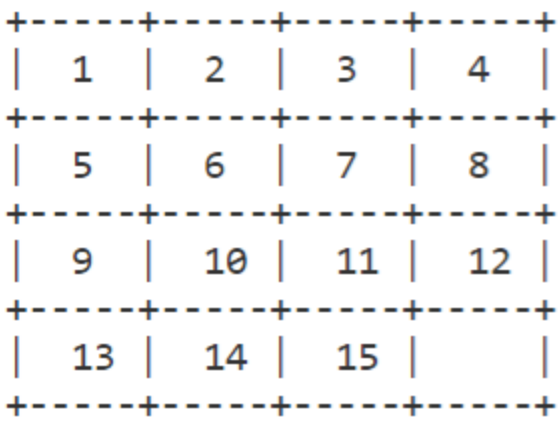
LAPORAN TUGAS KECIL III

PENYELESAIAN PERSOALAN 15-PUZZLE DENGAN ALGORITMA BRANCH AND BOUND

Laporan dibuat untuk memenuhi salah satu tugas mata kuliah

IF2211 Strategi Algoritma



Disusun oleh:

Muhammad Helmi Hibatullah

13520014

K-02

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER 2 TAHUN 2021/2022

DAFTAR ISI

Cara Kerja Program

Pendahuluan

Algoritma *Branch and Bound* adalah algoritma yang digunakan untuk menyelesaikan persoalan optimasi kombinatorial, yaitu meminimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar batasan persoalan. Mirip dengan algoritma *backtracking*, algoritma *branch and bound* melakukan pencarian solusi dengan membentuk pohon ruang status dan 'membunuh' simpul yang tidak 'mengarah' ke solusi. Pada tugas kecil ketiga ini, persoalan yang akan diselesaikan adalah permainan 15-Puzzle. 15-Puzzle adalah suatu permainan yang memiliki lima belas kotak bernomor satu hingga lima belas dalam sebuah papan dengan tinggi dan lebar masing-masing empat kotak sehingga menyisakan satu kotak kosong. Tujuan dari permainan ini adalah untuk menaruh kotak-kotak ini berdasarkan urutan angka.

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

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

Susunan Puzzle Awal (Kiri). Susunan Puzzle Tujuan (Kanan).

Cara Kerja Pengacakan 15-Puzzle

Program diawali dengan meminta masukan pengguna tentang bagaimana papan permainan akan disusun. Pengguna dapat memilih untuk menyusun papan permainan berdasarkan berkas teks yang dimasukkan atau dengan susunan acak yang dibuat program. Jika pengguna memilih untuk disusun secara acak oleh program, pengguna dapat memilih satu dari tiga tingkat kesulitan puzzle.

1. Mudah. Puzzle dapat diselesaikan paling banyak dengan sepuluh langkah dan waktu pencarian solusi relatif cepat.
2. Sedang. Puzzle dapat diselesaikan paling banyak dengan lima belas langkah dan waktu pencarian yang lebih lama dibanding tingkat kesulitan Mudah.
3. Sulit. Puzzle dapat diselesaikan paling banyak dengan dua puluh langkah dan waktu pencarian umumnya paling lambat dibandingkan kedua tingkatan sebelumnya.

Pengacakan susunan puzzle ini dilakukan dengan menggeser kotak kosong pada susunan puzzle tujuan secara acak sesuai dengan tingkat kesulitan yang dipilih pengguna. Misalkan pengguna memilih tingkat kesulitan Sedang, maka kotak kosong akan digeser sebanyak lima belas kali secara acak. Namun, penggeseran yang dilakukan pada suatu iterasi tidak akan kembali ke posisi pada iterasi sebelumnya untuk menghindari penggeseran bolak-balik. Berikut adalah penjelasan langkah per langkah.

1. Catat posisi kotak kosong saat ini, misal *A,* dan iterasi sebelumnya, misal *B*. Untuk iterasi pertama, *B* sama dengan *A*.
2. Cari semua arah penggeseran yang tidak melewati batas papan (4x4) dan taruh ke dalam sebuah list, misal *L*. Misalkan pada iterasi pertama posisi kotak kosong berada pada pojok kanan bawah. Maka, arah penggeseran yang bisa dilakukan hanya ke atas dan ke kiri.
3. Ambil secara acak arah penggeseran pada *L,* misal *C*.
4. Jika *C* sama dengan *B,* kembali ke langkah 2.
5. Tukar kotak kosong ke arah penggeseran terpilih.
6. Perbarui nilai *B* menjadi nilai *A* dan nilai *A* menjadi *C*.
7. Ulangi langkah 1-6 sebanyak tingkat kesulitan yang dipilih pengguna.

Pergeseran kotak kosong secara acak yang dilakukan pada susunan puzzle tujuan membuat puzzle yang disusun akan selalu mempunyai solusi, tidak peduli seberapa banyak pergeseran yang dilakukan. Untuk membuat susunan puzzle yang tidak mempunyai solusi, susunan puzzle yang akan diubah harus dilakukan sedikit perubahan, yaitu menukar salah satu angka sehingga tidak lagi berurut. Misalnya angka 14 dan 15 ditukar tanpa ada penggeseran seolah-olah kotak empat belas dan lima belas dicabut dari papan kemudian ditukar posisinya. Pada program yang dibuat, pengacakan susunan puzzle selalu dibuat dari puzzle tujuan sehingga pengguna akan selalu menemukan solusi jika memilih opsi ini.

Ukuran banyaknya iterasi untuk tiap kesulitan, sepuluh, lima belas, dan dua puluh, ditentukan berdasarkan pengujian waktu pencarian yang telah dilakukan sebelumnya. Pada sepuluh hingga lima belas iterasi, waktu pencarian yang didapat tidak ada yang menyentuh lebih dari lima belas detik, sedangkan pada dua puluh iterasi, waktu pencarian sangat bervariasi mulai dari di bawah sepuluh detik hingga mencapai lima menit. Jumlah iterasi ditentukan tidak melebihi dari dua puluh penggeseran karena khawatir akan terlalu lama serta akan memakan banyak memori.

Langkah-langkah Pencarian Solusi dengan Algoritma *Branch and Bound*

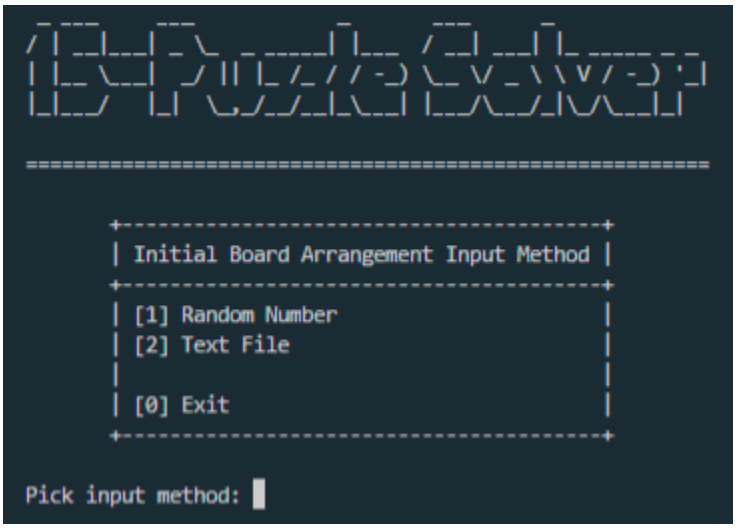
Pencarian langkah-langkah menuju solusi diawali dengan mengecek apakah susunan puzzle mula-mula bisa diselesaikan atau tidak. Jika dapat diselesaikan, buat status ruang pencarian

kemudian menghitung ongkos tiap simpul lalu memilih yang terkecil berulang-ulang hingga ongkosnya bernilai nol. Berikut adalah penjelasan langkah per langkah.

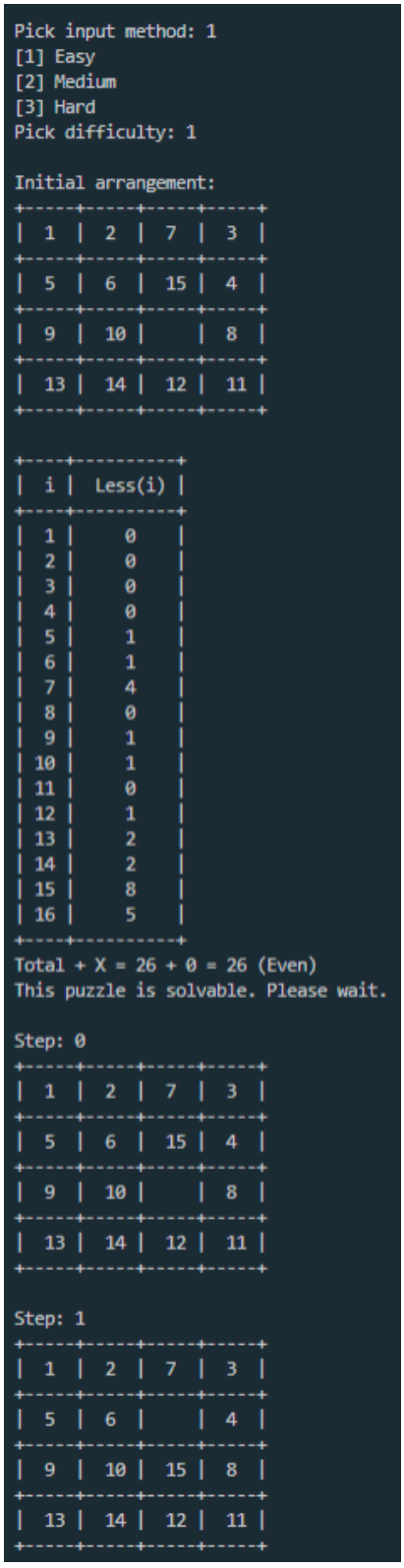
1. Lakukan pengecekan apakah susunan puzzle mula-mula bisa diselesaikan atau tidak. Susunan puzzle hanya dapat diselesaikan jika total hasil fungsi *KURANG(i)* ditambah *X* bernilai genap, dimana *i* bernilai satu sampai enam belas.
    - *KURANG(i)* adalah fungsi yang menghitung banyaknya kotak bernomor *j* sedemikian sehingga $j < i$ dan $POSISI(j) > POSISI(i)$.
    - *POSISI(i)* adalah posisi kotak bernomor *i* pada susunan puzzle yang diperiksa.
    - *X* bernilai satu jika posisi kotak kosong berada pada baris genap dan kolom ganjil atau baris ganjil dan kolom genap. Selain itu, *X* bernilai nol.
2. Jika susunan puzzle mula-mula tidak dapat diselesaikan, pencarian diberhentikan. Jika dapat diselesaikan, lanjut ke langkah 3.
3. Jadikan susunan puzzle awal sebagai simpul akar. Kemudian masukkan simpul akar ke dalam antrian Q. Antrian Q ini merupakan *priorirty queue* yang diurutkan membesar berdasarkan taksiran ongkos yang dimiliki sebuah simpul. Pada persoalan ini, taksiran ongkos sebuah simpul *p*, $\hat{c}(p)$, didapat dari panjang lintasan dari simpul akar ke *p*, $f(p)$, ditambah jumlah kotak tidak kosong pada simpul *p* yang tidak terdapat pada susunan tujuan, $g(p)$.
4. Jika Q kosong, pencarian diberhentikan.
5. Jika Q tidak kosong, ambil simpul yang memiliki ongkos terkecil, yaitu simpul pada indeks pertama karena Q sudah terurut membesar. Pada iterasi pertama, Q hanya berisi simpul akar.
6. Jika simpul tersebut adalah simpul solusi (susunannya sudah sama dengan susunan tujuan), pencarian diberhentikan.
7. Jika simpul tersebut bukan simpul solusi, maka bangkitkan semua anak-anaknya yang mungkin. Simpul anak adalah simpul yang kotak kosongnya sudah dipindahkan ke kotak sebelahnya dan tidak melewati batas papan puzzle (4x4). Jika simpul tersebut tidak memiliki anak, kembali ke langkah 4.
8. Untuk setiap anakan, hitung ongkosnya kemudian masukkan ke dalam Q.
9. Kembali ke langkah 4.

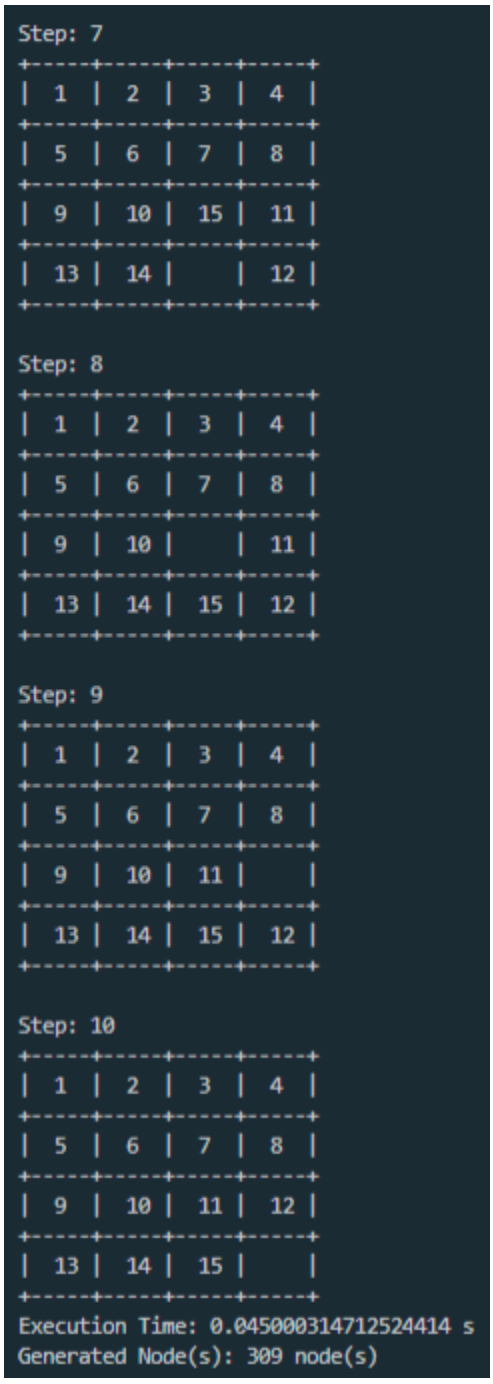
Tangkapan Layar Masukan-Luaran Program

Tampilan Memulai Program



Solvable Sepuluh Langkah





Solvable Lima Belas Langkah

```
Pick input method: 1
[1] Easy
[2] Medium
[3] Hard
Pick difficulty: 2

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


+----+----------+
|  i |  Less(i) |
+----+----------+
|  1 |     0    |
|  2 |     1    |
|  3 |     1    |
|  4 |     0    |
|  5 |     1    |
|  6 |     0    |
|  7 |     1    |
|  8 |     6    |
|  9 |     2    |
| 10 |     4    |
| 11 |     0    |
| 12 |     1    |
| 13 |     1    |
| 14 |     1    |
| 15 |     0    |
| 16 |    14    |
+----+----------+
Total + X = 33 + 1 = 34 (Even)
This puzzle is solvable. Please wait.

Step: 0
+-----+-----+-----+-----+
|  2  |     |  8  |  3  |
+-----+-----+-----+-----+
|  1  |  5  |  10 |  4  |
+-----+-----+-----+-----+
|  9  |  7  |  6  |  12 |
+-----+-----+-----+-----+
|  13 |  14 |  11 |  15 |
+-----+-----+-----+-----+

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

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

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

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

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

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

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

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

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

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

Step: 15
+-----+-----+-----+-----+
|  1  |  2  |  3  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  7  |  8  |
+-----+-----+-----+-----+
|  9  |  10 |  11 |  12 |
+-----+-----+-----+-----+
|  13 |  14 |  15 |     |
+-----+-----+-----+-----+
Execution Time: 0.08500242233276367 s
Generated Node(s): 1862 node(s)
```

Solvable Dua Puluh Langkah

```
Pick input method: 1
[1] Easy
[2] Medium
[3] Hard
Pick difficulty: 3

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

+----+----------+
| i  |  Less(i) |
+----+----------+
|  1 |     0    |
|  2 |     0    |
|  3 |     0    |
|  4 |     2    |
|  5 |     0    |
|  6 |     2    |
|  7 |     0    |
|  8 |     5    |
|  9 |     8    |
| 10 |     0    |
| 11 |     1    |
| 12 |     0    |
| 13 |     1    |
| 14 |     9    |
| 15 |     1    |
| 16 |     7    |
+----+----------+
Total + X = 36 + 0 = 36 (Even)
This puzzle is solvable. Please wait.

Step: 0
+-----+-----+-----+-----+
|  9  |  1  |  4  |  8  |
+-----+-----+-----+-----+
|  14 |  2  |  6  |  3  |
+-----+-----+-----+-----+
|     |  5  |  7  |  11 |
+-----+-----+-----+-----+
|  10 |  13 |  15 |  12 |
+-----+-----+-----+-----+

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

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

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

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

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

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

Step: 16
+-----+-----+-----+-----+
|  1  |  2  |     |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  3  |  8  |
+-----+-----+-----+-----+
|  9  |  10 |  7  |  11 |
+-----+-----+-----+-----+
|  13 |  14 |  15 |  12 |
+-----+-----+-----+-----+

Step: 17
+-----+-----+-----+-----+
|  1  |  2  |  3  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |     |  8  |
+-----+-----+-----+-----+
|  9  |  10 |  7  |  11 |
+-----+-----+-----+-----+
|  13 |  14 |  15 |  12 |
+-----+-----+-----+-----+

Step: 18
+-----+-----+-----+-----+
|  1  |  2  |  3  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  7  |  8  |
+-----+-----+-----+-----+
|  9  |  10 |     |  11 |
+-----+-----+-----+-----+
|  13 |  14 |  15 |  12 |
+-----+-----+-----+-----+

Step: 19
+-----+-----+-----+-----+
|  1  |  2  |  3  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  7  |  8  |
+-----+-----+-----+-----+
|  9  |  10 |  11 |     |
+-----+-----+-----+-----+
|  13 |  14 |  15 |  12 |
+-----+-----+-----+-----+

Step: 20
+-----+-----+-----+-----+
|  1  |  2  |  3  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  7  |  8  |
+-----+-----+-----+-----+
|  9  |  10 |  11 |  12 |
+-----+-----+-----+-----+
|  13 |  14 |  15 |     |
+-----+-----+-----+-----+
Execution Time: 0.23853635787963867 s
Generated Node(s): 9877 node(s)
```

Unsolvable

```
Pick input method: 2
Filename: unsolvable1.txt

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


+----+----------+
|  i |  Less(i) |
+----+----------+
|  1 |     0    |
|  2 |     0    |
|  3 |     1    |
|  4 |     1    |
|  5 |     0    |
|  6 |     0    |
|  7 |     3    |
|  8 |     0    |
|  9 |     0    |
| 10 |     0    |
| 11 |     3    |
| 12 |     5    |
| 13 |     0    |
| 14 |     4    |
| 15 |    11    |
| 16 |     7    |
+----+----------+
Total + X = 35 + 0 = 35 (Odd)
Sorry, this puzzle is not solvable.
```

Checklist

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

Kode Program

Kode program ditulis dalam bahasa Python dan terbagi ke dalam enam berkas kode terpisah, yaitu constant.py, Node.py, PriorityQueue.py, Solver.py, createBoard.py, dan main.py.

constant.py

Berkas kode ini hanya berisi konstanta-konstanta yang didefinisikan untuk nantinya digunakan dalam implementasi algoritma *branch and bound*.

```
N = 4
""" Board Size (NxN). """

# Order: right, bottom, left, top
VER_DIR = [ 0, 1, 0, -1 ]
""" Vertical Direction to move the empty block on the board.
+1 : Bottom
 0 : Nothing
-1 : Top
"""

# Order: right, bottom, left, top
HOR_DIR = [ 1, 0, -1, 0 ]
""" Horizontal Direction to move the empty block on the board.
+1 : Right
 0 : Nothing
-1 : Left
"""

#: Goal state
GOAL_BOARD   = [[ 1,  2,  3,  4],
                [ 5,  6,  7,  8],
                [ 9, 10, 11, 12],
                [13, 14, 15, 16]]
""" Final Board Arrangement. Block 16 is treated as an empty block. """
```

Node.py

Berkas kode ini berisi kelas Node untuk merepresentasikan sebuah simpul pada pohon. Penjelasan singkat atribut dan method sudah tercantum dalam docstring kode.

```
class Node:
    """ A class used to represent a node of a tree.

    Attributes
    ----------
    parent : Node
        The parent node
    board : list[list[int]]
        The board of the node
    empty_idx : list[int]
        The index where the empty block is located on the board
    cost : int
        The cost needed to reach this node
    level : int
        The depth level of the node

    Methods
    -------
    __lt__(other)
        For comparison purposes. Used in priority queue
    """

    def __init__(self, parent, board, empty_idx, cost, level):
        """
        Parameters
        ----------
        parent : Node
            The parent node
        board : list[list[int]]
            The board of the node
        empty_idx : list[int]
            The index where the empty block is located on the board
        cost : int
            The cost needed to reach this node
        level : int
```

```
            The depth level of the node
        """
        self.parent = parent
        self.board = board
        self.empty_idx = empty_idx
        self.cost = cost
        self.level = level

    def __lt__(self, other) -> bool:
        """ Compare this node with other node, returns True if this node
is "less" than other node. """
        return self.cost + self.level < other.cost + other.level
```

PriorityQueue.py

Berkas kode ini berisi kelas PriorityQueue untuk merepresentasikan sebuah *priority queue*. File ini memanggil modul heapq agar dapat menggunakan fungsi/prosedur heappush dan heappop untuk mengimplementasikan *push* dan *pop* pada *priority queue*. Penjelasan singkat atribut dan method sudah tercantum dalam docstring kode.

```
from heapq import heappush, heappop

class PriorityQueue:
    """ A class used to represent a priority queue.

    Attributes
    ----------
    heap : list of something
        The container for priority queue

    Methods
    -------
    push(k)
        Push value k to the heap with priority
    pop()
        Pop the first value in the heap
    empty()
        Check if the container is empty
```

```
    """

    def __init__(self):
        self.heap = []

    def push(self, k):
        """ Push value k to the heap with priority. """
        heappush(self.heap, k)

    def pop(self):
        """ Pop and return the first value in the heap. """
        return heappop(self.heap)

    def empty(self):
        """ Check if the container is empty. """
        return not self.heap
```

Solver.py

Berkas kode ini berisi kelas Solver yang merepresentasikan sebuah pemecah persoalan 15-Puzzle. Penjelasan singkat atribut dan method sudah tercantum dalam docstring kode.

```
import copy
import time
from PriorityQueue import PriorityQueue
from Node import Node
from constant import N, VER_DIR, HOR_DIR, GOAL_BOARD

class Solver:
    """ A class used to represent a solver of the 15-puzzle.

    Attributes
    ----------
    board_to_solve : list[list[int]]
        The board of 15-puzzle to be solved
    prio_queue : PriorityQueue
        The priority queue of Nodes to implement branch and bound
algorithm
```

```
    total_x : int
        The value that determines if the board is solvable or not
solvable.
    generated_nodes : int
        The counter for the generated Nodes

    Methods
    -------
    isSolvable(root_node)
        Calculate the value of total_x
    less(board_arr, i, i_idx)
        Count the number of blocks with number j such that j < i and idx_j
> idx_i
    calculateEmptyPos(root_node)
        Return 1 if the empty block is at certain position and 0 if at the
rest
    calculateCost(board)
        Calculate the cost needed for a board to go to the goal board.
    showBoard(board)
        Print the board
    showAllBoard(node)
        Print all boards recursively
    isIdxValid(idx)
        Check if idx is out of bounds
    solve()
        Perform the Branch and Bound Algorithm to solve the puzzle
    """

    def __init__(self, board_to_solve):
        """
        Parameters
        ---------
        board_to_solve : list[list[int]]
            The board of 15-puzzles to be solved
        """
        self.board_to_solve = board_to_solve
        self.prio_queue = PriorityQueue()
        self.total_x = 0
        self.generated_nodes = 0
```

```
    def isSolvable(self, root_node) -> bool:
        """ Calculate the value of total_x.

        total_x is sum of Less(i) + X where X is 1 if
        the empty blocks is at certain position
        and 0 if at the rest.

        Parameters
        ----------
        root_node : Node
            The root node of a search space tree

        Returns
        -------
        boolean
            True if total_x is even, False if odd.
        """
        # store board numbers to a list
        board_numbers = []
        for i in range(N):
            for j in range(N):
                board_numbers.append(self.board_to_solve[i][j])

        # calculate and print Less(i)
        print("+----+----------+")
        print("|  i |  Less(i) |")
        print("+----+----------+")
        total = 0
        for i in range(1, len(board_numbers) + 1):
            i_idx = 0
            for j in range(len(board_numbers)):
                if board_numbers[j] == i:
                    i_idx = j

            less_i = self.less(board_numbers, i, i_idx)
            total += less_i

            if i < 10:
                print(f"|  {i} ", end="|    ")
            else:
```

```
                print(f"| {i} ", end="|     ")

            if less_i < 10:
                print(f" {less_i}     |")
            else:
                print(f"{less_i}     |")
        print("+----+----------+")

        # determine empty value
        x = self.determineEmptyVal(root_node)
        self.total_x = total + x
        print(f"Total + X = {total} + {x} = {self.total_x}", end=" ")

        if self.total_x % 2 == 0:
            print("(Even)")
            return True
        else:
            print("(Odd)")
            return False

    def less(self, board_arr, i, i_idx) -> int:
        """ Count the number of blocks with number j such that j < i and
idx_j > idx_i.

        Parameters
        ----------
        board_arr : list[int]
            The numbers in the board that stored in a list
        i : int
            The number to check
        i_idx : int
            The index of number i in board
        """
        count = 0
        for j in range(1, i):
            # in here, k is idx_j
            for k in range(len(board_arr)):
                if board_arr[k] == j and k > i_idx:
                    count += 1
        return count
```

```
    def determineEmptyVal(self, root_node) -> int:
        """ Determine the empty block value.

        Suppose the board size is 4x4. The empty block has value according
        to its position on the board as follows: \n

        0 1 0 1\n
        1 0 1 0\n
        0 1 0 1\n
        1 0 1 0
        """
        i = root_node.empty_idx[0]
        j = root_node.empty_idx[1]
        if (i % 2 == 0 and j % 2 == 1) or (i % 2 == 1 and j % 2 == 0):
            return 1
        else:
            return 0

    def calculateCost(self, board) -> int:
        """ Calculate the cost needed for a board to go to the goal board.

        The cost is calculated by how many blocks are out of place.
        """
        cost = 0
        for i in range(N):
            for j in range(N):
                if (board[i][j] != 16 and board[i][j] !=
GOAL_BOARD[i][j]):
                    cost += 1
        return cost

    def showBoard(self, board):
        """ Print the board. """
        for i in range(N):
            print("+-----+-----+-----+-----+")
            for j in range(N):
                if board[i][j] == 16:
                    print(f"|     ", end="")
                elif board[i][j] < 10:
```

```
                    print(f"|  {board[i][j]}  ", end="")
                else:
                    print(f"|  {board[i][j]} ", end="")
            print("|")
        print("+-----+-----+-----+-----+")

    def showAllBoard(self, node):
        """ Print the board recursively. """
        if node == None:
            return
        self.showAllBoard(node.parent)
        print(f"\nStep: {node.level}")
        self.showBoard(node.board)

    def isIdxValid(self, idx) -> bool:
        """ Check if idx is out of bounds. """
        return 0 <= idx[0] < N and 0 <= idx[1] < N

    def solve(self):
        """ Perform the Branch and Bound Algorithm to solve the puzzle.
"""
        print("\nInitial arrangement:")
        self.showBoard(self.board_to_solve)
        print()

        start_time = time.time()

        # find the cost and empty block position to create the root node
        root_cost = self.calculateCost(self.board_to_solve)
        found = False
        i = 0
        while i < N and not found:
            j = 0;
            while j < N and not found:
                if self.board_to_solve[i][j] == 16:
                    found = True
                else:
                    j += 1
            if not found:
                i += 1
```

```
        root_empty_idx = [ i, j ]
        root_node = Node(None, self.board_to_solve, root_empty_idx,
root_cost, 0)

        if not self.isSolvable(root_node):
            print("Sorry, this puzzle is not solvable.")
        else:
            print("This puzzle is solvable. Please wait.")
            last_time_checked = 0

            self.prio_queue.push(root_node)
            while not self.prio_queue.empty():
                minimum_node = self.prio_queue.pop()

                # check time if it goes too long
                time_check = round(time.time() - start_time)
                if time_check != 0 and time_check != last_time_checked and
time_check % 10 == 0:
                    print(f"{time_check} seconds have passed...")
                    last_time_checked = time_check

                # if cost == 0, then the node is the soltution
                if minimum_node.cost == 0:
                    self.showAllBoard(minimum_node)
                    print(f"Execution Time: {time.time() - start_time} s")
                    print(f"Generated Node(s): {self.generated_nodes}
node(s)")
                    return

                # generate all possible child node(s)
                empty_idx_old = minimum_node.empty_idx
                for i in range(N):
                    empty_idx_new = [ empty_idx_old[0] + VER_DIR[i],
empty_idx_old[1] + HOR_DIR[i] ]

                    if self.isIdxValid(empty_idx_new):
                        self.generated_nodes += 1

                        # deep copy so that the parent board wont be
changed
```

```
                        child_board = copy.deepcopy(minimum_node.board)

                        temp =
child_board[empty_idx_old[0]][empty_idx_old[1]]
                        child_board[empty_idx_old[0]][empty_idx_old[1]] =
child_board[empty_idx_new[0]][empty_idx_new[1]]
                        child_board[empty_idx_new[0]][empty_idx_new[1]] =
temp

                        child_cost = self.calculateCost(child_board)
                        child_node = Node(minimum_node, child_board,
empty_idx_new, child_cost, minimum_node.level + 1)

                        self.prio_queue.push(child_node)
```

createBoard.py

Berkas kode ini berisi fungsi-fungsi yang diperlukan untuk menyusun papan permainan 15-Puzzle. Terdapat dua fungsi yang dibuat, yaitu from_file untuk menyusun papan permaian sesuai dengan file yang dibaca dan from_random untuk menyusun papan permainan secara acak.

```
import random
import copy
from os.path import exists
from constant import N, GOAL_BOARD, VER_DIR, HOR_DIR

def from_file() -> list[list[int]]:
    """ Read from a file then return the matrix inside it. """

    board = []
    directory = "./test/"
    filename = input("Filename: ")

    if not exists(directory + filename):
        print(f"'{directory + filename}' is not exist.")
        return []
    else:
        with open(directory + filename, 'r') as f:
```

```
            board = [[int(num) for num in line.split(' ')] for line in f
if line.strip() != "" ]

        return board


def from_random() -> list[list[int]]:
    """ Randomize the goal board according to the user choice of
difficulty then return the matrix. """

    board = copy.deepcopy(GOAL_BOARD)

    print("[1] Easy")
    print("[2] Medium")
    print("[3] Hard")
    user_choice = input("Pick difficulty: ")
    if user_choice == "1":
        difficulty = 10
    elif user_choice == "2":
        difficulty = 15
    elif user_choice == "3":
        difficulty = 20
    else:
        print("Invalid Input.")
        return []

    empty_idx = [ 3, 3 ]
    previous_idx = [ 3, 3 ]
    # the more the difficulty, the more the empty block moved
    for _ in range(difficulty):
        # find valid moves
        valid_moves = []
        for j in range(N):
            empty_idx_new = [ empty_idx[0] + VER_DIR[j], empty_idx[1] +
HOR_DIR[j] ]
            if 0 <= empty_idx_new[0] < N and 0 <= empty_idx_new[1] < N:
                valid_moves.append(empty_idx_new)

        # pick random new move, last move wont be picked again
        random_move = valid_moves[random.randint(0, len(valid_moves) - 1)]
```

```
        while random_move == previous_idx:
            random_move = valid_moves[random.randint(0, len(valid_moves) -
1)]

        # swap
        temp = board[empty_idx[0]][empty_idx[1]]
        board[empty_idx[0]][empty_idx[1]] =
board[random_move[0]][random_move[1]]
        board[random_move[0]][random_move[1]] = temp

        # update empty_idx so that there is no back and forth movement
        previous_idx = empty_idx
        empty_idx = random_move


    return board
```

main.py

Berka kode ini merupakan program utama yang nantinya akan dijalankan oleh pengguna.

```
from Solver import Solver
from createBoard import from_file, from_random

print(" _ ___      ___              _        ___       _                ")
print("/ | __|___| _ \\_  _ _____| |___  / __| ___| |_ _____ _ _ ")
print("| |__ \\___|  _/ || |_ /_ / / -_) \\__ \\/ _ \\ \\ V / -_) '_|")
print("|_|___/   |_|  \\_,_/__/__|_\\___| |___/\\___/_|\\_/\\___|_|  ")

active = True
while active:
    print()
    print("="*57)
    print()
    print("       +----------------------------------------+")
    print("       | Initial Board Arrangement Input Method |")
    print("       +----------------------------------------+")
    print("       | [1] Random Number                      |")
    print("       | [2] Text File                          |")
```

```
    print("       |                                          |")
    print("       | [0] Exit                                 |")
    print("       +------------------------------------------+")
    print()
    user_pick = input("Pick input method: ")

    if user_pick == "1" or user_pick == "2":
        if user_pick == "1":
            board = from_random()
        elif user_pick == "2":
            board = from_file()

        if board != []:
            puzzle = Solver(board)
            puzzle.solve()

    elif user_pick == "0":
        active = False
        print("Thank you!")
    else:
        print("Invalid Input.")
```

Berkas Teks Instansiasi Lima Persoalan 15-Puzzle

Berikut berkas teks untuk instansiasi lima persoalan 15-Puzzle yang terdapat dalam folder test. Perlu diingat bahwa angka 16 dianggap dan diperlakukan sebagai kotak kosong.

No	Nama File	Isi
1	solvable-easy.txt	1 2 16 4 5 6 3 7 9 10 15 8 13 14 12 11
2	solvable-medium.txt	16 2 3 4 1 14 7 8 6 5 13 11 9 15 10 12
3	solvable-hard.txt	1 2 4 7 9 16 6 3 10 5 11 12 13 14 8 15
4	unsolvable1.txt	1 3 4 15 7 2 5 12 16 6 11 14 8 9 10 13
5	unsolvable2.txt	1 2 3 4 5 6 7 8 9 10 16 11 13 14 12 15

Tautan Repository Github

Berikut tautan yang dapat diakses untuk menuju ke kode program.
https://github.com/mhelmih/Tucil3_13520014