**LAPORAN TUGAS KECIL**

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

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

Disusun oleh:

**Rania Dwi Fadhilah (K1) 13520142**



**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2022**

## DAFTAR ISI

[**I. PENJELASAN ALGORITMA PROGRAM 1**](#_Toc99578662)

[**II. HASIL PERCOBAAN 2**](#_Toc99578663)

[**2.1 INTERAKSI I/O 2**](#_Toc99578664)

[**2.2 UJI COBA TXT 4**](#_Toc99578665)

[**2.3 UJI COBA MANUAL 18**](#_Toc99578666)

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

## I. PENJELASAN ALGORITMA PROGRAM

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

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

*cost* (harga jalur),

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

letak basis,

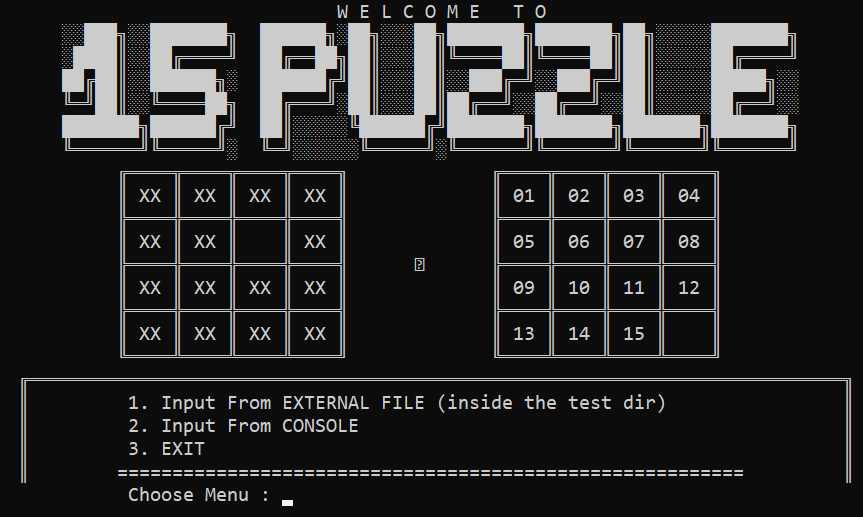
bentuk atas, bawah, kanan, dan kirinya.

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

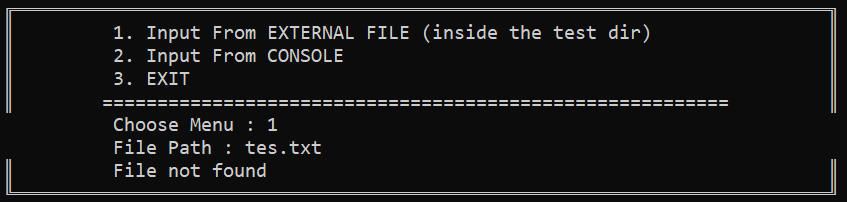
## II. HASIL PERCOBAAN

## 2.1 INTERAKSI I/O

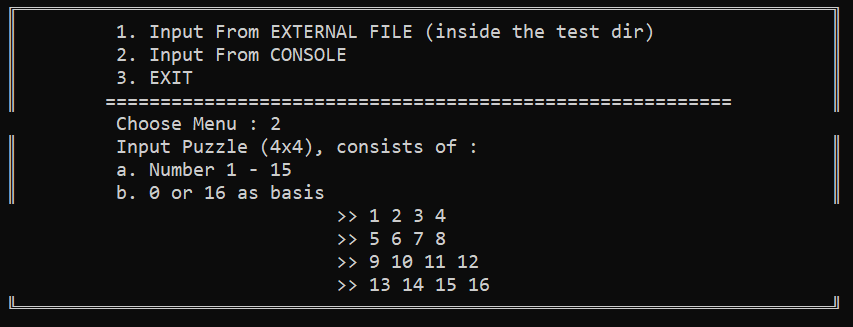
* 1. Interaksi *Input* Menu Awal



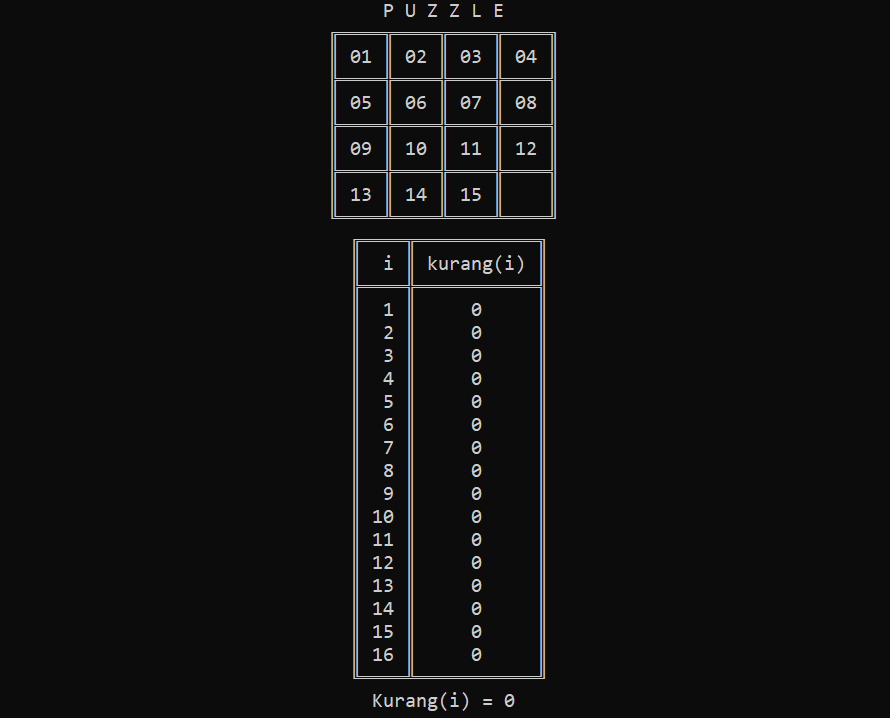
* 1. Interaksi *Input* Menu 1 *(External File)*



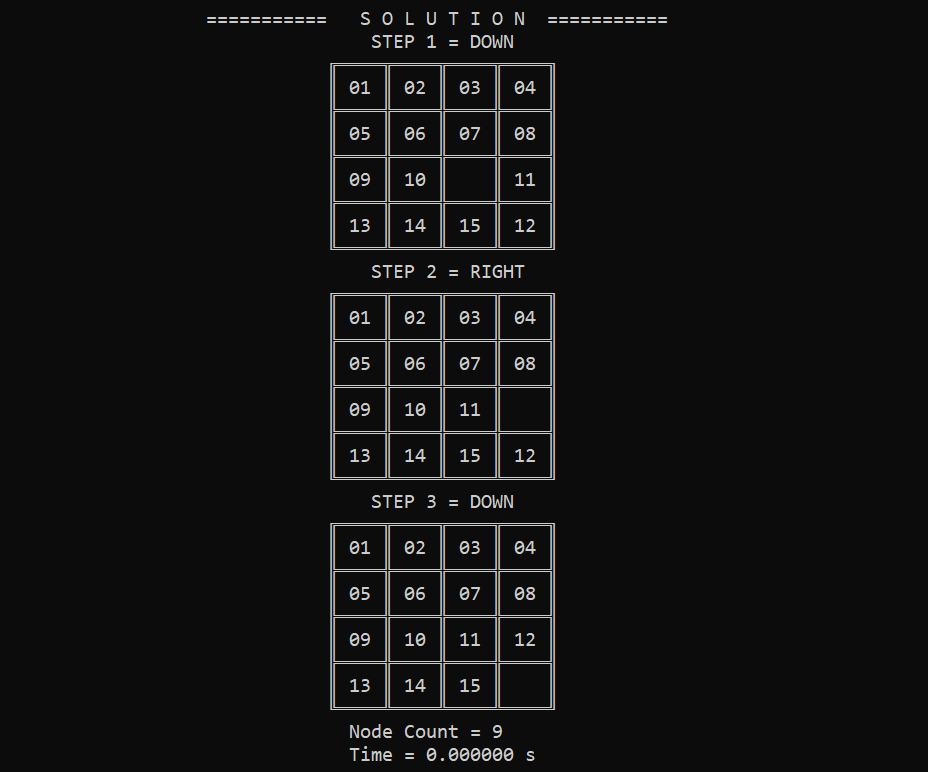
* 1. Interaksi *Input* Menu 2 (*Console)*

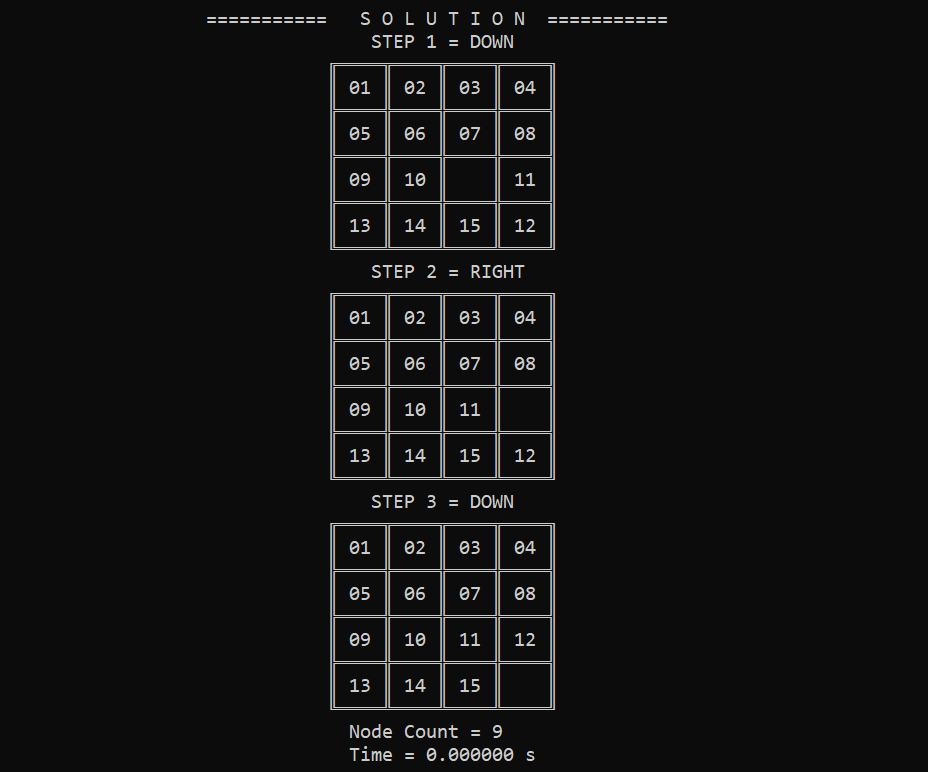


* 1. Interaksi *Output* Bagian Kurang(i)



* 1. Interaksi *Output* Bagian Solusi





## 2.2 UJI COBA TXT

*Percobaan 1*

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

*Percobaan 2*

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

*Percobaan 3*

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

*Percobaan 4*

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

*Percobaan 5*

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

*Percobaan 6*

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

## 2.3 UJI COBA MANUAL

*Percobaan 7*

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

*Percobaan 8*

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

## LAMPIRAN

1. Repository Github :

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

2. Checklist :

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