**LAPORAN TUGAS KECIL 3**

**PENYELESAIAN PERSOALAN 15-PUZZLE DENGAN ALGORITMA BRANCH AND BOUND**

**IF2211 Strategi Algoritma**

**­­**

Oleh:

**Daffa Pratama Putra 13518033**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2020**

# **BAB I PENYELESAIAN 15-PUZZLE ALGORITMA *BRANCH AND BOUND***

Algoritma *Branch and Bound* merupakan salah satu strategi algoritma yang digunakan untuk menyelesaikan masalah optimasi. Algoritma ini menggunakan pohon ruang status untuk melakukan pencarian solusi terbaik. Perjalanan dalam pencarian status solusi menggunakan usaha yang paling optimal. Salah satu penerapan algoritma ini adalah dalam menyelesaikan persoalan 15-Puzzle.

15-Puzzle adalah salah satu permainan dengan menggunakan papan yang berisikan angka 1 sampai 15 dalam 16 bagian ubin. Terdapat satu ubin kosong yang dapat digerakkan ke atas, bawah, kiri, dan kanan untuk menggeser ubin lainnya. Tujuan yang dicapai dari permainan ini adalah menyusun angka 1 sampai 15 terurut dari atas ke bawah dengan cara menggeser ubin kosong.

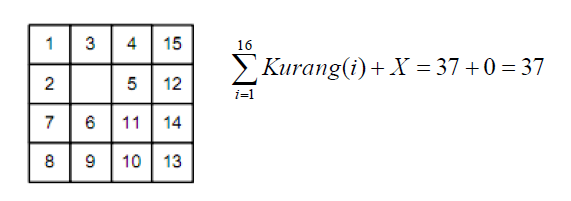
A picture containing light, drawing

Description automatically generated

Gambar 1. Goal permainan 15-Puzzle

Untuk menyelsaikan permainan 15-Puzzle yang memiliki susunan acak, terlebih dahulu perlu dilakukan pengecekan apaah susunan awal dapat mencapai susunan *goal* seperti pada gambar 1 di atas. Terdapat 16! kemungkinan susunan awal puzzle atau kira-kira sebanyak 20,9 x 1012. Dari kemungkinan susunan awal puzzle tersebut, hanya ada setengahnya yang dapat mencapai susunan *goal*. Pengecekan dilakukan dengan menghitung inversi dari susunan awal puzzle.

Dengan menggunakan teorema bahwa status tujuan hanya dapat dicapai dari status awal jika nilai dari adalah genap. Kurang(i) yang dimaksud adalah inversi susunan puzzle, yakni A[i] > A[j] tetapi i < j, sedangkan X adalah letak ubin kosong pada susunan awal. X bernilai 1 saat berada pada ubin dengan koordinat i+j adalah ganjil dan bernilai 0 pada ubin dengan koordinatn i+j adalah genap. Jika teorema di atas terpenuhi, maka susunan ubin tersebut solveable, begitu juga sebaliknya. Apabila susunan ubin solveable, maka akan dilakukan proses pencarian simpul tujuan dari simpul utama. Contoh penerapan teorema di atas adalah sebagai berikut



Gambar 2. Kasus ubin yang not solveable

Proses pencarian dilakukan dengan algoritma *Branch & Bound*. Algoritma ini menggunakan prinsip *least cost search* pada pencarian simpul anak pada tiap langkahnya. *Least cost search* digunakan untuk menentukan simpul anak mana yang akan ditelusuri selanjutnya. Tiap pembangkitan simpul anak, akan dihitung nilai *cost*nya dan dimasukkan ke dalam antrian pemrosesan simpul hidup. *Queue* yang digunakan adalah *Priority Queue* dengan prioritas utama adalah simpul anak dengan nilai *cost* terkecil. Dengan demikian simpul anak yang dipilih yang memiliki *cost* terkecil dari simpul anak yang lain pada antrian pemrosesan. *Cost* yang digunakan adalah sebagai berikut :

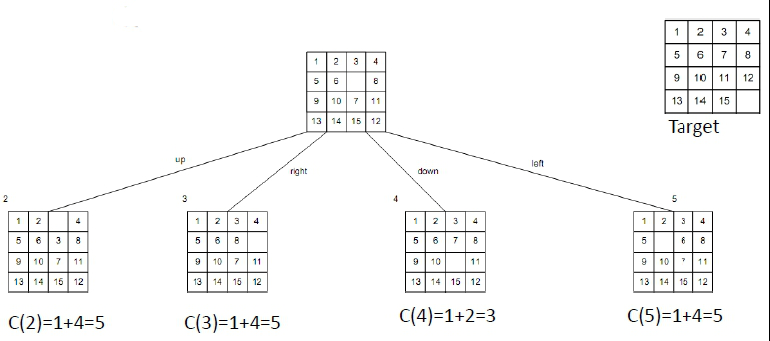
c(i) = f(i) + g(i)

c(i) = *cost* untuk simpul ke i

f(i) = *cost* untuk mencapai simpul ke i dari simpul akar (*depth*)

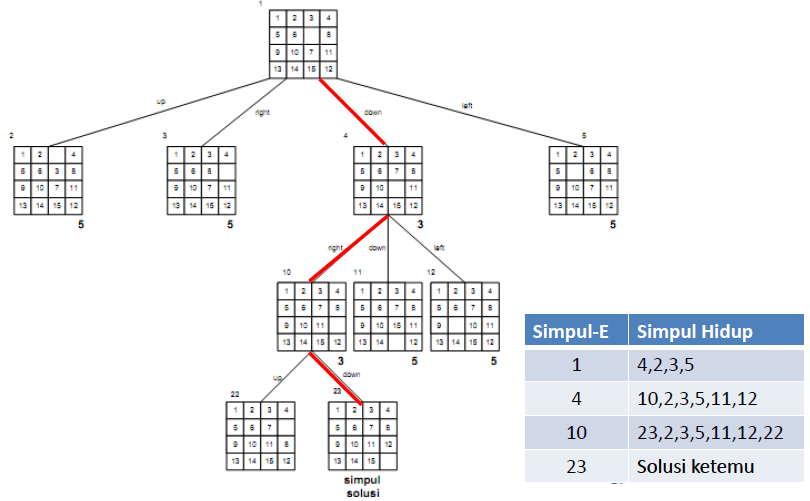
g(i) = *cost* untuk mencapai simpul tujuan dari simpul ke i. Dalam kasus ini adalah jumlah susunan ubin pada simpul anak yang tidak terdapat pada susunan akhir

Seperti contoh adalah susunan ubin berikut :



Gambar 3. Pohon ruang status

dari susunan awal seperti di atas, maka akan dipilih susunan berikutnya adalah simpul 4, yakni dengan *cost* terkecil. Jika pencarian tersbut dilanjutkan, maka akan terbentuk pohon seperti gambar dibawah.

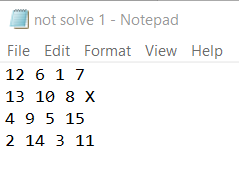


Gambar 4. Pohon ruang status (lanjutan)

# **BAB II PENGUJIAN PROGRAM**

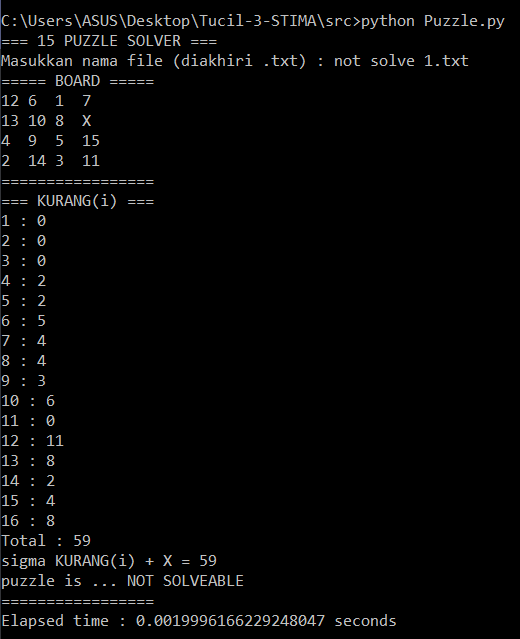
1. Uji Kasus *Not Solveable* 1

Berikut adalah contoh susunan puzzle yang tidak dapat diselesaikan :



Gambar 5. Susunan puzzle dari file “not solve 1.txt”

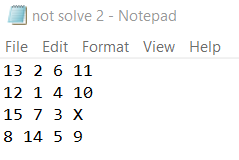
Keluaran program :



Gambar 6. Susunan puzzle dari file “not solve 1.txt”

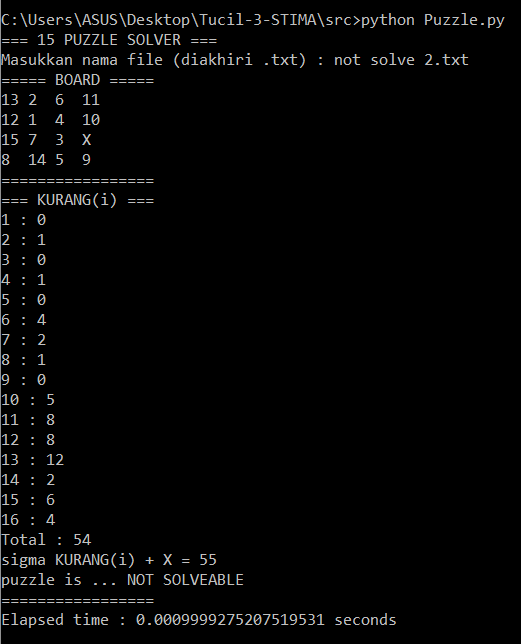
1. Uji Kasus *Not Solveable* 2

Berikut adalah contoh susunan puzzle yang tidak dapat diselesaikan :



Gambar 7. Susunan puzzle dari file “not solve 2.txt”

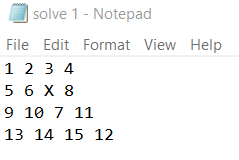
Keluaran program :



Gambar 8. Susunan puzzle dari file “not solve 2.txt”

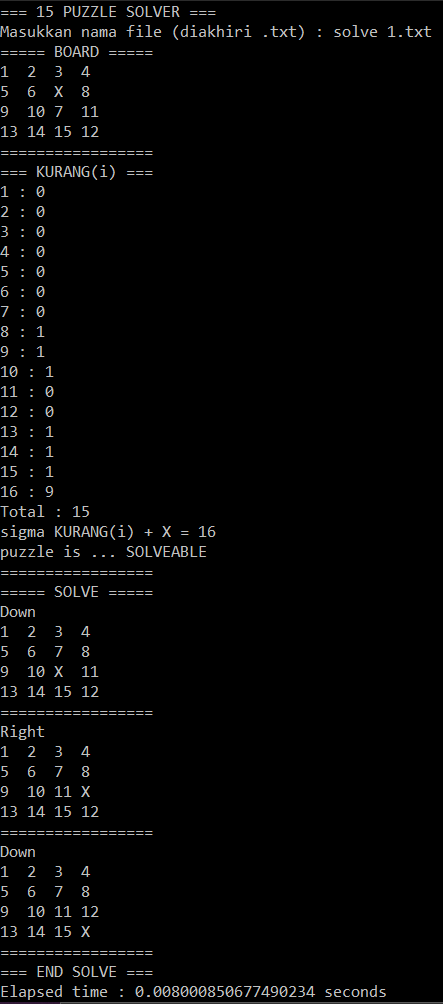
1. Uji Kasus *Solveable* 1

Berikut adalah contoh susunan puzzle yang tidak dapat diselesaikan :



Gambar 9. Susunan puzzle dari file “solve 1.txt”

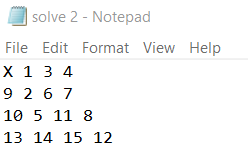
Keluaran program :



Gambar 10. Susunan puzzle dari file “not solve 2.txt”

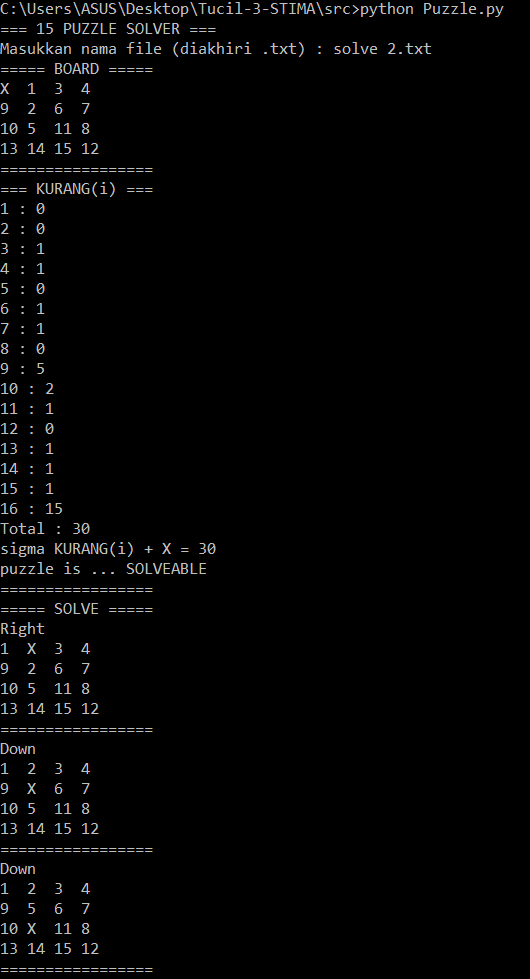
1. Uji Kasus *Solveable* 2

Berikut adalah contoh susunan puzzle yang tidak dapat diselesaikan :

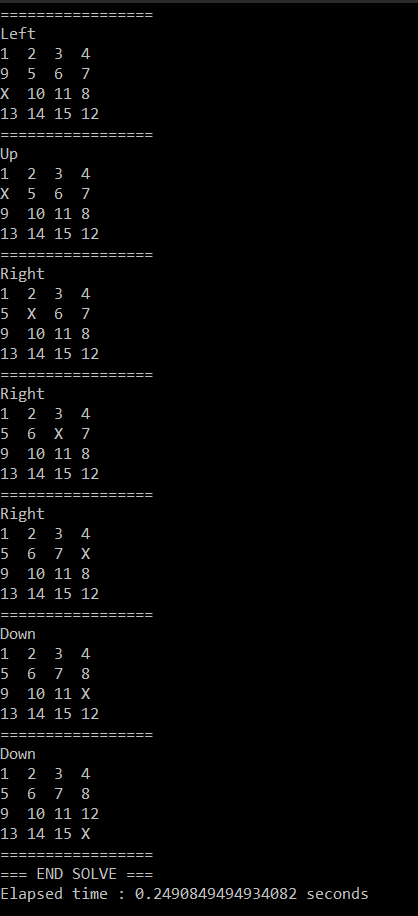


Gambar 11. Susunan puzzle dari file “solve 2.txt”

Keluaran program :



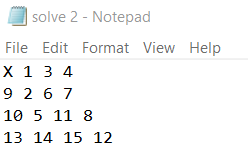
Gambar 12. Susunan puzzle dari file “solve 2.txt”



Gambar 13. Susunan puzzle dari file “solve 2.txt” (lanjutan)

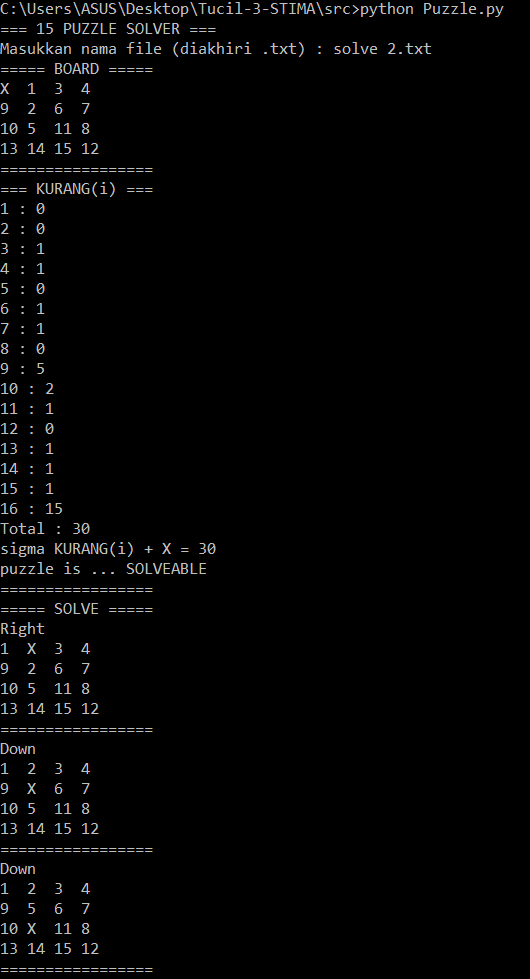
1. Uji Kasus *Solveable* 3

Berikut adalah contoh susunan puzzle yang tidak dapat diselesaikan :

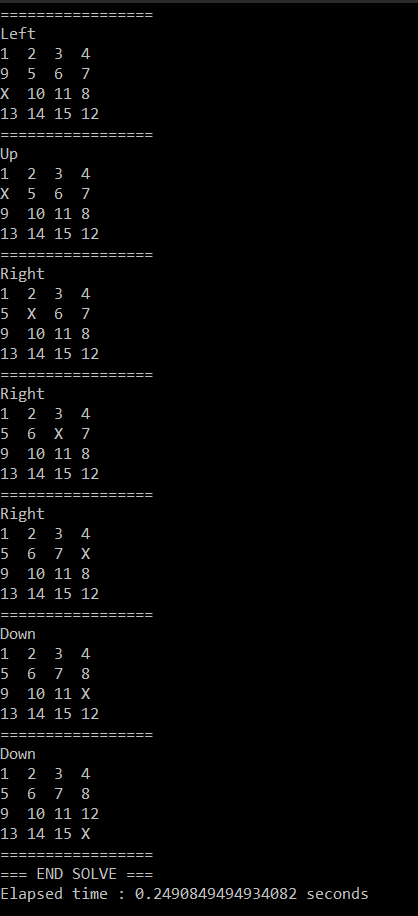


Gambar 11. Susunan puzzle dari file “solve 2.txt”

Keluaran program :



Gambar 12. Susunan puzzle dari file “solve 2.txt”



Gambar 13. Susunan puzzle dari file “solve 2.txt” (lanjutan)

Penilaian Asisten

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