Penerapan Algoritma Backtracking dalam Permainan Futoshiki Puzzle

Julio Savigny, 13513084
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
jsavigny@students.itb.ac.id

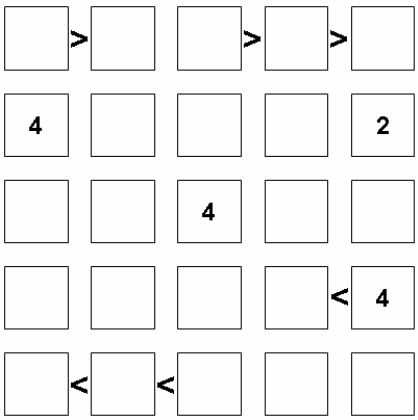
***Abstract*—Backtracking adalah suatu algoritma yang pada dasarnya adalah perbaikan dari exhaustive search. Pada *Backtracking*, sekuens kemungkinan yang dipilih adalah sekuens yang memenuhi constraint tertentu. Penerapan Algoritma *Backtracking* ada banyak, terutama pada pemecahan masalah permainan puzzle seperti n-Queen, Knight's Tour, Sudoku, dll. Salah satu penerapannya adalah pada permainan Futoshiki Puzzle.**

***Index Terms*—Backtracking, Futoshiki, Runut-balik, Solusi, DFS**

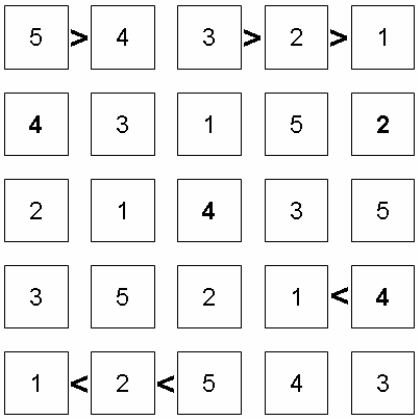
I. PENDAHULUAN

1.1 Futoshiki

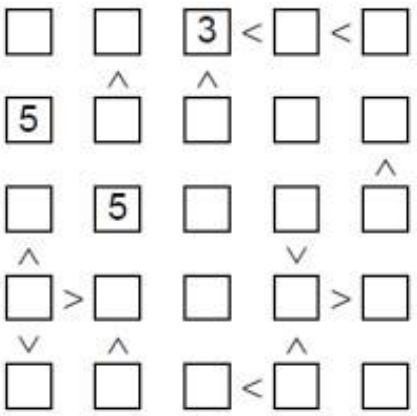
Futoshiki (不等式) adalah adalah permainan teka-teki logika yang berasal dari Jepang. Arti dari kata Futoshiki sendiri adalah "Ketidaksamaan". Teka-teki dimainkan pada grid persegi, misalnya 5 x 5 . Tujuannya adalah untuk menempatkan angka 1 sampai 5 (atau dimensi lainnya) sehingga setiap baris dan kolom berisi masing-masing angka 1 sampai 5 . Ada angka yang sudah diisi di grid pada awal permainan, sebagai panduan. Selain itu, ada beberapa tanda ketidaksamaan (logical constraint) (< atau >) diberikan di antara beberapa kotak, sehingga nilai yang berada dalam kotak harus memenuhi constraint tersebut.



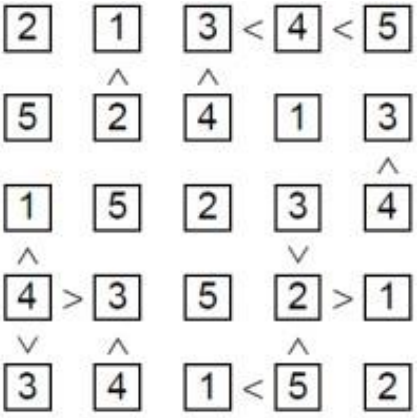
Gambar 1.1 Ilustrasi Keadaan Awal Papan Futoshiki



Gambar 1.2 Ilustrasi Keadaan Solusi Papan Futoshiki



Gambar 1.3 Ilustrasi Keadaan Awal Papan Futoshiki



Gambar 1.4 Ilustrasi Keadaan Solusi Papan Futoshiki

II. DASAR TEORI

2.1. DFS

Depth-First Search adalah sebuah cara untuk mentraverse sebuah graf. Prinsip dari algoritma DFS simpel : maju telusuri (secara mendalam) selama masih memungkinkan, jika tidak memungkinkan lakukan *backtrack*.

- Traversal dimulai dari simpul v.
- Algoritma:

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.2. Algoritma *Backtracking*

Istilah *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950 dan terdapat juga tokoh-tokoh seperti R.J.Walker, Golomb, Baumert yang menyajikan uraian umum tentang *backtracking*.

Algoritma *backtracking* adalah algoritma pencarian solusi yang berbasis pada *DFS*. Algoritma *backtracking* banyak diterapkan untuk program games :

1. Permainan *tic-tac-toe*
2. Menemukan jalan keluar dari sebuah *maze*
3. Catur termasuk didalamnya permainan dari *knight's tour* ini.

Algoritma *backtracking* merupakan perbaikan dari algoritma *brute-force*. Pada *brute-force* semua kemungkinan solusi dieksplorasi satu per satu sedangkan pada backtracking hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan kembali.

2.2.1. Properti Umum Algoritma *Backtracking*

1. Solusi Persoalan

Solusi dinyatakan dalam bentuk vektor dengan *tuple*: $X = (x_1,x_2,x_3,...,x_n)$, $x_i \in S_i$
Mungkin terjadi $S_1 = S_2 = ... = S_n$
Contoh : $S_i = \{0,1\}$, x_i=0 atau 1

2. Fungsi Pembangkit

Fungsi pembangkit nilai x_k dinyatakan dalam predikat *T(k)* dimana *T(k)* membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

3. Fungsi Pembatas

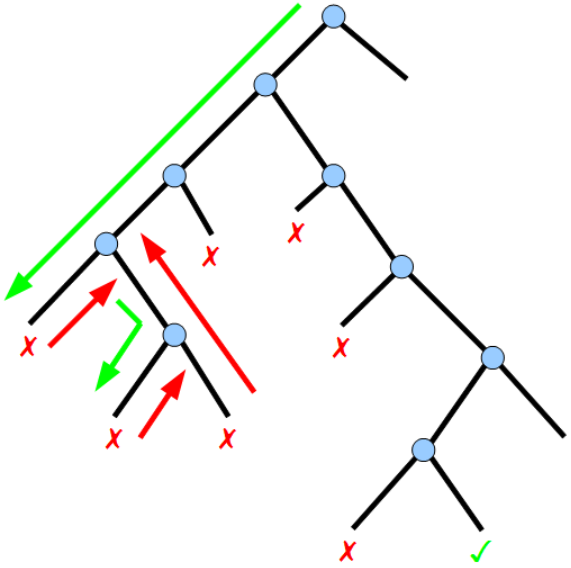
Dinyatakan dalam predikat : $B(x_1,x_2,..,x_k)$.

B bernilai *true* jika $(x_1,x_2,..,x_k)$ mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk x_k+1 dilanjutkan, tetapi jika *false*, maka $(x_1,x_2,..,x_k)$ dibuang.

2.2.2. Prinsip Pencarian Solusi dengan Metode *Backtracking*

- Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan *depht-first order* (DFS).
- Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*)
- Simpul-simpul yang sedang diperluas dinamakan **simpul-E** (*Expand node*)
- Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.
- Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi **simpul mati** (*dead node*)
- Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas**.
- Simpul yang sudah mati tidak pernah diperluas lagi.
- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul aras diatasnya.
- Lalu, teruskan dengan membangkitkan simpul anak lainnya.
- Selanjutnya simpul ini menjadi simpul-E yang baru.
- Pencarian dihentikan bila kita telah sampai pada *goal node*.

Berikut merupakan gambar ilustrasi algoritma *backtrack* :



Gambar 2.1 Ilustrasi Algoritma *Backtrack*

2.2.3. Kompleksitas Waktu Algoritma *Backtracking*

- Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.
- Jika jumlah simpul dalam pohon ruang status adalah 2^n atau n!, maka untuk kasus terburuk, algoritma *backtrack* membutuhkan waktu dalam :

  1. $O(p(n)2^n)$ atau

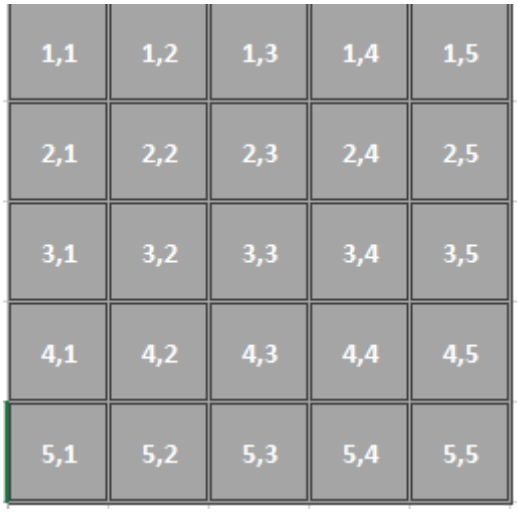
  2. $O(q(n)n!)$

- Dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi simpul.

III. PENERAPAN ALGORITMA

3.1 Representasi Matrix

Pertama, berikan indeks untuk merepresentasikan papan Futoshiki



Gambar 3.1 Index Papan

Angka yang ada di dalam matriks merupakan indeks posisi kotak, indeks dimulai dari ujung kiri-atas yaitu (1,1) sampai ujung kanan-bawah yaitu (5,5). Hal ini akan bermanfaat untuk proses implementasi selanjutnya.

3.2 Definisi Properti

Definisi Properti Algoritma Backtracking untuk persoalan Futoshiki adalah

1. **Solusi Persoalan**

   Solusi Persoalan pada Futoshiki 5x5 adalah

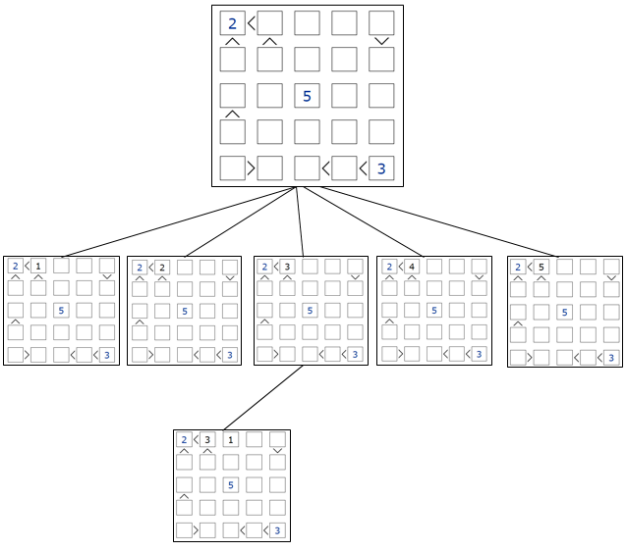
   X = (x11,x12,x13,x14,x15
   x21,x22,x23,x24,x25
   x31,x32,x33,x34,x35
   x41,x42,x43,x44,x45
   x51,x52,x53,x54,x55)

   dimana xij $\in$ {1,2,3,4,5}
2. **Fungsi Pembangkit**
3. **Fungsi Pembatas**

   Fungsi Pembatas pada persoalan ini adalah :
   a. Nilai xij tidak boleh sama dengan nilai yang sudah ada di baris i dan kolom j.
   b. Nilai xij harus memenuhi constraint logic yang ada (jika board(i,j) tersebut memiliki constraint logic).

3.3 Pohon Solusi

Setiap simpul pada pohon ruang solusi adalah status papan pada setiap waktunya.



Gambar 3.2 Contoh Ruang Status Futoshiki

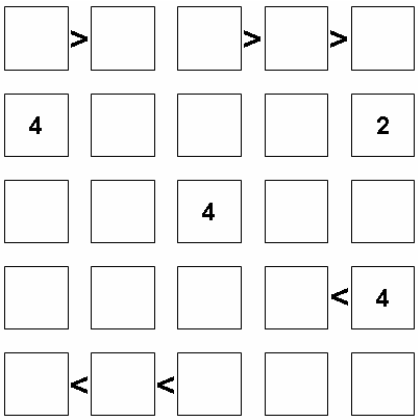
Kedalaman dari pohon tersebut akan ditentukan oleh banyaknya tempat kosong (yang akan diisi). Misalkan, pada Futoshiki 5x5 pada Gambar 1.1 ada 21 blok kosong, maka kedalaman Pohon Solusi, jika solusi ditemukan, adalah 21.

Untuk mempermudah implementasi, inisialisasi lah setiap blok kosong dengan 0.

3.4 Pencarian Solusi

Secara umum, pencarian solusi persoalan Puzzle Futoshiki dengan algoritma backtracking adalah sebagai berikut :

1. Tentukan simpul akar, yaitu status awal papan.
2. Cari blok kosong, iterasi dimulai dari baris pertama, dengan kolom pertama – terakhir, lalu ke baris kedua, dst.
3. Setelah blok kosong ditemukan, coba isi blok tersebut dengan angka 1. Dengan mengisi blok tersebut, simpul baru akan dihidupkan.
4. Cek apakah simpul tersebut melanggar Fungsi Pembatas:
    a. Jika tidak, lanjutkan ke blok kosong selanjutnya, hidupkan simpul baru dengan mengisi blok tersebut.
    b. Jika ya, backtrack ke simpul sebelumnya.
5. Cek apakah simpul masih bisa di ekspan dengan memasukkan angka yang belum dimasukkan, jika tidak bisa menghidupkan simpul apapun, backtrack ke simpul sebelumnya.
6. Proses akan dilakukan secara terus menerus hingga solusi ditemukan atau tidak ada simpul hidup lagi.



Gambar 3.3 Permasalahan Futoshiki

Anggap kita akan menyelesaikan persoalan Futoshiki yang ada di Gambar 3.3. Dengan simpul akar S0 adalah status board yang ada di Gambar 3,3.

Pertama, expand dari S0, bangkitkan blok ke 1,1. Ada 5 value yang mungkin dapat mengisi blok tersebut yaitu 1,2,3,4,5. Pertama, kita bangkitkan 1, maka akan terbentuk simpul baru S1 yang merupakan anak dari S0. Lalu cek apakah S1 melanggar fungsi pembatas:

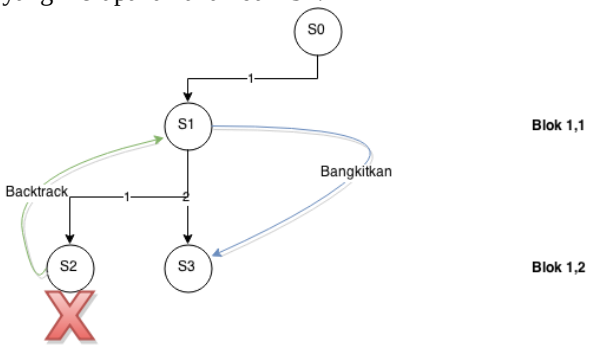
- Apakah 1 ada di baris 1? Tidak.
- Apakah 1 ada di kolom 1? Tidak
- Ada kah logical constraint? Ya
    - Apakah 1 lebih besar dari 0 (blok kosong 1,2)? Ya

Ternyata, simpul tersebut tidak melanggar fungsi pembatas, maka pencarian diteruskan melalui simpul tersebut (karena berbasis DFS).

Setelah blok 1,1 diisi oleh 1, maka langkah selanjutnya adalah untuk pindah ke blok 1,2. Bangkitkanlah blok 1,2 dengan 1, terbentuk S2 yang merupakan anak dari S1, lalu cek apakah S2 tersebut melanggar fungsi pembatas :

- Apakah 1 ada di baris 1? Ya

Ternyata, S2 melanggar fungsi pembatas, maka S2 dibunuh, dan kita kembali (backtrack) ke S1. Dari S1, kita coba bangkitkan blok 1,2 dengan 2, maka terbentuk S3 yang merupakan anak dari S1.



Gambar 3.4 Ilustrasi Backtrack dari S2, pembangkitan S3.

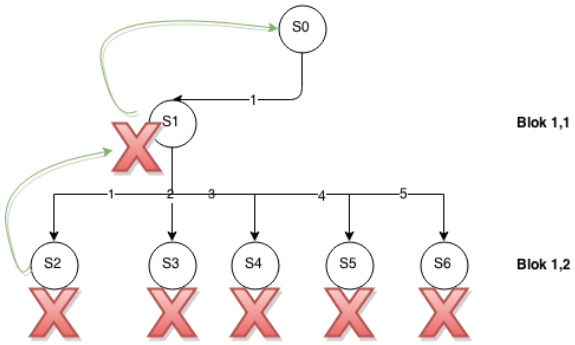
Lalu, cek apakah S3 melanggar fungsi pembatas:

- Apakah 2 ada di baris 1? Tidak.
- Apakah 2 ada di kolom 2? Tidak.
- Apakah ada logical constraint di blok 1,2? Ya.

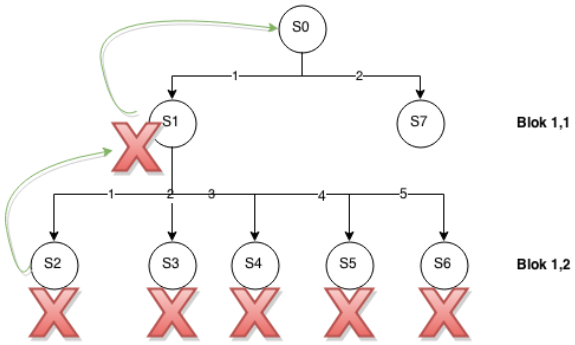
- Apakah 2 lebih kecil dari 1 (blok 1,1)? Tidak.

Ternyata, S3 juga melanggar fungsi pembatas, oleh karena itu, S3 dibunuh, dan kita kembali ke S1.

Dari sini kita bisa meneruskan mencari dengan algoritma seperti diatas, dan akan menemukan bahwa blok 1,2 tidak akan bisa diisi oleh apapun dari 1-5, selama blok 1,1 berisi 1, yang artinya semua anak dari S1 mati, oleh karena itu solusi tidak dapat ditemukan di S1, maka S1 pun harus dibunuh, lalu kita akan backtrack ke S0, dan menghidupkan simpul selanjutnya (S7) dengan membangkitkan 2 di blok 1,1.



Gambar 3.5 Ilustrasi Backtrack



Gambar 3.6 Ilustrasi Pembangkitan Simpul S7 dengan mengisi 2 di blok 1,1.

Setelah membangkitkan S7, Cek apakah S7 melanggar Fungsi Pembatas:

- Apakah 2 ada di baris 1? Tidak.
- Apakah 2 ada di kolom 1? Tidak.
- Apakah ada Logical constraint di blok 1,1? Ya.
  - Apakah 2 lebih besar dari 0 (blok kosong 1,2)? Ya.

Karena tidak melanggar fungsi pembatas, maka kita akan pindah ke blok kosong selanjutnya yaitu 1,2.

Hidupkan simpul baru S8 dengan membangkitkan 1 di blok 1,2. Cek apakah S8 melanggar Fungsi Pembatas :

- Apakah 1 ada di baris 1? Tidak.
- Apakah 1 ada di kolom 2? Tidak.
- Apakah ada Logical Constraint di blok 1,2? Ya.
  - Apakah 1 lebih kecil dari 2 (blok 1,1)? Ya.

Karena tidak melanggar fungsi pembatas, maka kita akan pindah ke blok kosong selanjutnya yaitu 1,3.

Hidupkan simpul baru S9 dengan membangkitkan 1 di blok 1,3. Cek apakah S9 melanggar Fungsi Pembatas :

- Apakah 1 ada di baris 1? Ya.

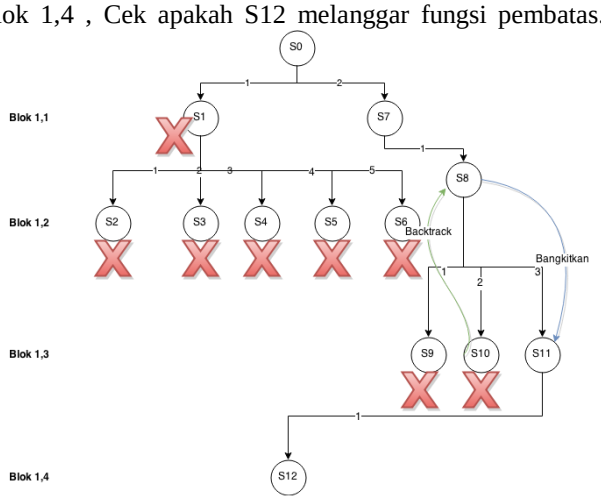
Karena S9 melanggar, maka bunuh S9, dan lakukan backtrack ke S8. Hidupkan Simpul baru S10 dengan membangkitkan 2 di blok 1,3. Cek apakah S10 melanggar Fungsi Pembatas:

- Apakah 2 ada di baris 1? Ya.

Karena S10 melanggar, maka bunuh S10, dan lakukan backtrack ke S8. Hidupkan Simpul baru S11 dengan membangkitkan 3 di blok 1,3. Cek apakah S11 melanggar Fungsi Pembatas:

- Apakah 3 ada di baris 1? Tidak.
- Apakah 3 ada di kolom 3? Tidak.
- Apakah ada logical constraint? Ya.
  - Apakah 3 lebih besar dari 0 (blok kosong 1,4)? Ya.

Karena tidak melanggar fungsi pembatas, maka kita akan pindah ke blok kosong selanjutnya yaitu 1,4. Hidupkan simpul baru S12 dengan membangkitkan 1 di blok 1,4 , Cek apakah S12 melanggar fungsi pembatas.



Gambar 3.7 Pohon Status so far..

Proses diatas akan terus dilakukan sampai bertemu dengan satu dari dua kemungkinan yang ada :

Solusi ditemukan, dengan

X = (x11,x12,x13,x14,x15
x21,x22,x23,x24,x25
x31,x32,x33,x34,x35
x41,x42,x43,x44,x45
x51,x52,x53,x54,x55)

dimana xij ∈ {1,2,3,4,5}

dan dengan kedalaman pohon 21, atau

a. Solusi tidak ditemukan, yaitu tidak ada simpul yang bisa dihidupkan, dan tidak dapat melakukan backtrack.

V. KESIMPULAN DAN SARAN

Permasalahan Puzzle Futoshiki dapat diselesaikan dengan algoritma *Backtracking*.

Algoritma *Backtracking* atau Runut-balik merupakan algoritma yang cukup mangkus untuk mendapatkan solusi dari suatu permasalahan, karena algoritma ini tidak memeriksa semua kemungkinan seperti di *Exhaustive search*, namun hanya memeriksa kemungkinan yang mengarah kepada solusi.

Dari Algoritma diatas, banyak yang bisa dikembangkan lagi untuk membuat algoritma tersebut lebih mangkus, misalnya dengan mengisi blok yang memiliki constraint lebih besar ('>') dengan angka yang bukan angka 1, karena 1 tidak mungkin lebih besar dari apapun (1-5), atau juga dengan menggabungkan algoritma greedy.

VI. ACKNOWLEDGMENT

Penulis mengucapkan puji syukur kepada Tuhan Allah SWT yang telah memberikan penulis kekuatan dan kemampuan berpikir untuk dapat menulis makalah ini. Penulis juga mengucapkan terimakasih kepada kedua Orang Tua penulis yang telah menyediakan tempat berteduh juga alat untuk menulis makalah ini. Tidak lupa penulis mengucapkan terimakasih kepada Dosen pengampu matakuliah IF2211 Strategi Algoritma yaitu Dr. Nur Ulfa Maulidevi, S.T, M.Sc dan Dr. Ir. Rinaldi Munir, MT. yang telah memberikan ilmu yang bermanfaat kepada penulis.

REFERENSI

[1] Munir, Rinaldi, Diktat Kuliah Strategi Algoritma, Program Studi Teknik Informatika, STEI, ITB.
[2] Slide presentasi, Algoritma Runut-balik (Backtracking)
[3] Spesifikasi Tugas Kecil 1 IF2211 Strategi Algoritma Semester II Tahun 2013/2014, Program Futoshiki dengan Algoritma *Brute Force*
[4] http://www.brainbashers.com/showfutoshiki.asp, diakses 4 Mei 2015.
[5] http://www.algolist.net/Algorithms/Graph/Undirected/Depth-first_search, diakses 4 Mei 2015

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 4 Mei 2015

Julio Savigny, 13513084