Penerapan Algoritma Runut Balik pada Pathuku Games

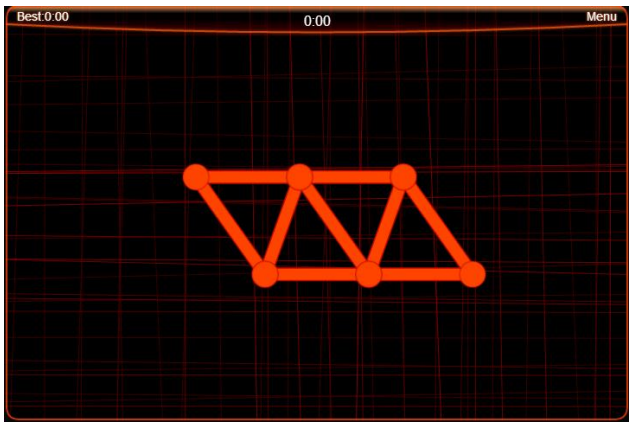
Junita Sinambela 13512023
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
junita@students.itb.ac.id

***Abstrak*—Pathuku (pada https://www.*pathuku*.com) merupakan sebuah games berbasis web yang dikembangkan pada HTML5, sistem operasi Android, iPhone, maupun Windows Phone. Pathuku berbentuk sebuah bidang dengan beberapa titik yang dihubungkan dengan beberapa garis (sisi). Tujuan akhir permainan ini adalah mencapai kondisi di mana semua sisi harus pernah dilalui maksimal satu kali. Pada level-level tertentu, ada chaser yang akan mengejar melalui *path* yang telah kita ambil sebelumnya, dan ketika chaser berhasil mengejar, maka permainan berakhir atau sering disebut *game over*. Untuk mendapatkan solusi yang sesuai dengan syarat permainan dalam waktu yang singkat, kita memerlukan strategi yang mangkus.**

***Kata Kunci*— Algoritma runut-balik, algoritma *backtracking*, *pathuku*, tabel pencarian, pohon keputusan, simpul, *path*.**

I. PENDAHULUAN

Pathuku merupakan sebuah *game* berbasis web. Game ini dapat dimainkan pada *web browser* yang mendukung HTML5. Permainan ini dimainkan dengan cara menghubungkan satu titik dengan titik yang lain dengan syarat semua sisi (garis yang menghubungkan satu simpul dengan simpul yang lain di bidang permainan) harus pernah dihubungkan.



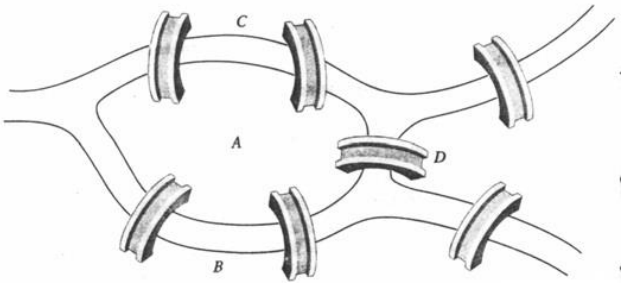
Gambar 1 Pathuku sebelum permainan

Pathuku dimainkan dengan mengklik titik yang akan menjadi tujuan kita. Titik tujuan haruslah berhubungan langsung dengan titik asal permainan. Titik asal permainan dipilih oleh pemain dan ketika semua sisi sudah dilalui tepat sekali, maka permainan berakhir dan game otomatis menghitung lama pemain menyelesaikan permainan tersebut.

II. TEORI DASAR

A. Jembatan Königsberg

Jembatan Königsberg adalah salah satu masalah yang sering dijadikan sebagai salah satu pertanyaan paling terkenal dalam menjelaskan teori graf. Pada tahun 1736, seorang ilmuwan bernama Leonard Euler mengajukan pertanyaan : bisakah melalui jembatan tepat sekali dan kembali lagi ke tempat semula?



Gambar 2 Jembatan Königsberg
Sumber :
http://arifcahyadiblog.blogspot.com/2010_08_06_archive.html

Pada dasarnya, semua simpul pada graf harus berderajat genap untuk dapat membentuk sebuah *path* yang melalui tepat sekali masing-masing sisi dan kembali ke simpul awal. Derajat sebuah simpul adalah banyaknya sisi yang berhubungan langsung dengan simpul tersebut.

*B. Algoritma Runut-Balik (*Backtracking*)*

Algoritma runut balik merupakan salah satu fase algoritma *Depth-First Search* (DFS). Pada algoritma DFS, pengecekan dilakukan menurut prioritas. Misalkan dalam pencarian simpul yang akan dilalui dari sebuah simpul awal pada sebuah graf, jika prioritas pencarian adalah simpul dengan nomor terkecil, maka pengecekan akan dimulai dari simpul dengan nomor terkecil. Jika simpul dengan nomor terkecil tidak dapat menghasilkan solusi yang diharapkan, maka pengecekan dilakukan pada simpul pada nomor berikutnya secara menaik, sampai solusi didapatkan.

Algoritma DFS sendiri merupakan algoritma yang dibentuk berdasarkan pohon keputusan yang dibentuk dari sekumpulan kemungkinan solusi. Pada pengambilan

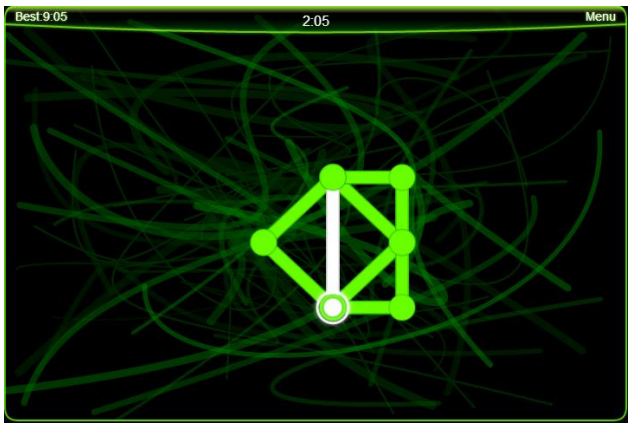
Algoritma runut-balik merupakan algoritma yang dikembangkan karena terkadang kita tidak memiliki informasi yang cukup untuk mengetahui apa yang akan dipilih dari sekumpulan cara yang mungkin dilakukan pada sebuah masalah. Setiap keputusan yang diambil akan mengarah kepada sekumpulan pilihan baru. Jika tidak ada pilihan yang memungkinkan tercapainya solusi, maka kita akan kembali ke keputusan sebelumnya dan mengganti keputusan untuk memperoleh solusi.

Tahapan-tahapan pada algoritma runut-balik juga menyerupai tahapan-tahapan pada algoritma *brute-force*. Akan tetapi, algoritma runut-balik lebih mengoptimalkan keputusan yang diambil. Apabila sebuah keputusan dirasa tidak mengarah kepada solusi, keputusan kemudian dibatalkan dan ia akan mencari alternatif lain yang memungkinkan solusi tercapai. Pencarian akan dihentikan jika sebuah solusi sudah tercapai.

III. PENCARIAN SOLUSI

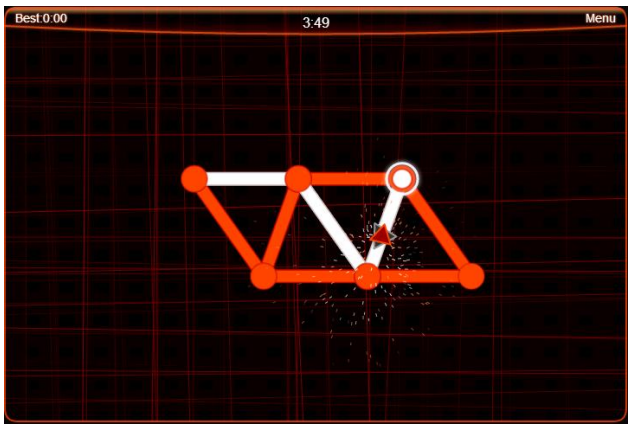
A. Peletakan Simpul Awal

Pada permainan ini, simpul awal diletakkan di sebuah sembarang simpul. Akan tetapi, untuk mendapatkan solusi, kita perlu mencermati apakah simpul hanya terhubung dengan dua simpul atau tidak. Jika simpul hanya terhubung dengan 2 sisi dan simpul-simpul yang berhubungan dengan simpul awal membentuk segitiga dengan simpul awal. Peletakan simpul ini menandakan dimulainya pengecekan dan pengambilan keputusan untuk mengambil *path* yang sesuai agar kondisi solusi dapat tercapai. Sewaktu keputusan pertama diambil, penghitungan lama pengambilan keputusan dalam mencari solusi oleh pemain juga dimulai.



Gambar 3 Pengambilan sisi pertama pada graf

Pada level-level yang lebih kompleks, chaser akan mengejar pemain melalui *path* yang telah diambil secara berurutan. Jika chaser mencapai simpul terakhir dari *path* solusi yang diambil oleh pemain, permainan akan berakir dan pemain dinyatakan kalah.

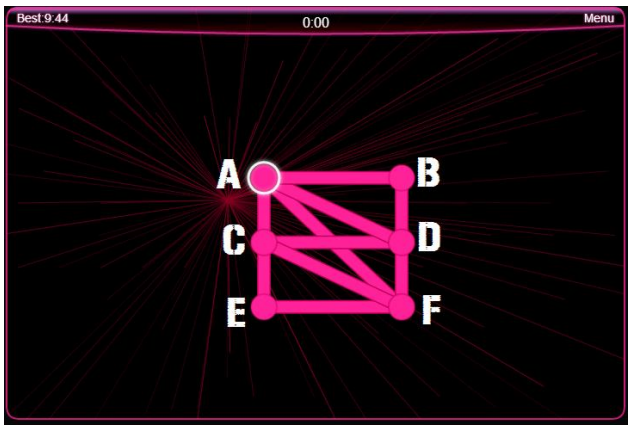


Gambar 4 Chaser sedang menuju simpul terakhir *path*

B. Pengambilan Keputusan

Dalam menentukan keputusan, kita harus mendeteksi apakah sisi yang akan diambil dapat mencapai solusi. Pencarian solusi dilakukan dengan mendeteksi simpul yang dapat dituju.

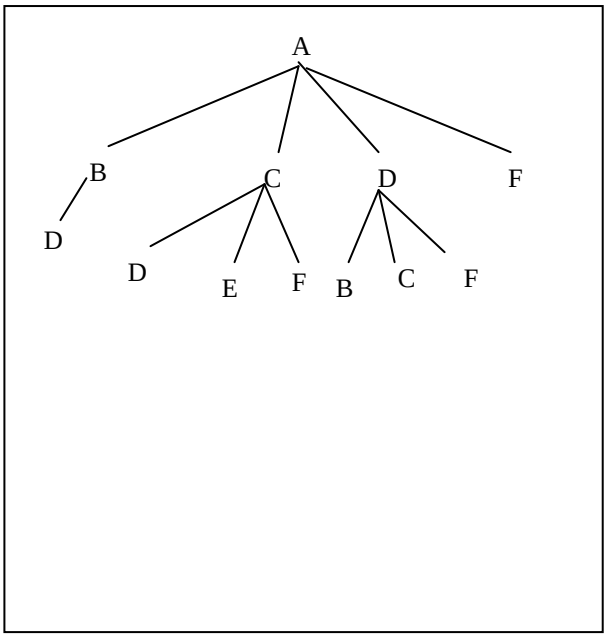
Misalkan kita akan mengambil *path* solusi pada bidang permainan seperti pada gambar 5.



Gambar 5 Bidang permainan yang ingin dicari solusinya

Pada bidang permainan di atas, simpul awal yang kita ambil adalah simpul A. Pengambilan simpul pertama, seperti sudah dijelaskan di bagian A, dilakukan secara acak saja. Dari simpul A, kita memutuskan untuk mengambil sisi yang memenuhi syarat permainan. Pada kasus ini, misalkan kita mengambil sisi AB sehingga state kita berpindah dari A ke B.

Pada tahap pengambilan keputusan selanjutnya, kita dapat memilih sisi selain sisi AB. Dalam mengambil keputusan, kita dapat menerapkan beberapa cara pengoptimalan pengambilan keputusan. Misalkan, kita mendeteksi apakah sisi yang sedang dicek membentuk *path* yang sirkular, kecuali jika sisi yang sedang dicek adalah satu-satunya sisi yang tersisa. Jika ya, pengecekan dilakukan pada sisi lain yang memungkinkan. Jika tidak, kita memasuki "*dead end*" yang menyebabkan game over dan kita sudah pasti kalah.



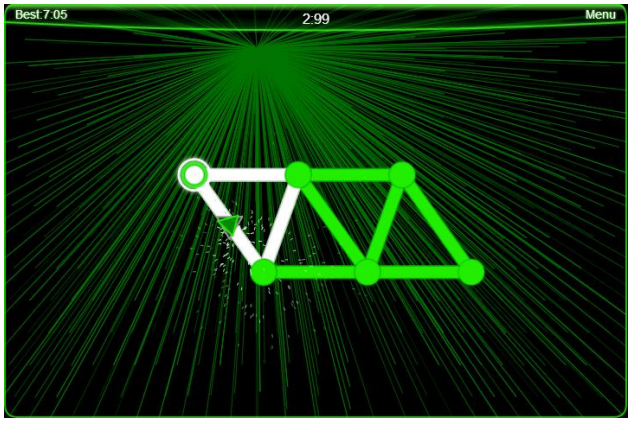
Gambar 6 Sisi yang mungkin diambil dari simpul A

Misalkan pada tahap selanjutnya kita mengambil sisi BD, sehingga simpul terakhir yang dikunjungi adalah D. Kita dapat memilih sisi yang menuju A, C, atau F.

C. Pengoptimalan Pengambilan Keputusan

Jika hal ini dilupakan, maka pengambilan keputusan tidak akan optimal lagi. Pengoptimalan dilakukan dengan mempertimbangkan kemungkinan terjadinya kondisi yang dapat menyebabkan *game over*. Dalam hal ini, ada beberapa hal yang dapat menyebabkan game over.

1. Keputusan yang diambil membentuk *path* sirkular, sementara masih ada sisi yang belum diambil dan tidak berhubungan langsung dengan simpul terakhir keputusan. Hal ini menyebabkan tidak ada lagi jalan keluar bagi pemain untuk mencapai solusi, sehingga permainan berakhir dan pemain dinyatakan kalah.



Gambar 7 Contoh pengambilan keputusan yang salah dan menyebabkan *game over*

2. Pengambilan simpul awal yang salah. Seperti telah dijelaskan di bagian A, jika simpul awal yang kita ambil hanya terhubung dengan 2 simpul dan keputusan paling awal yang kita ambil membentuk *path* sirkular, maka dapat dipastikan bahwa pemain akan kalah. Misalkan pada gambar 1, terdapat beberapa simpul yang membentuk sirkular dan dapat mengakibatkan solusi tidak tercapai (pemain tidak bisa bergerak ke manapun karena sisi- sisi yang terhubung secara langsung dengan simpul terakhir yang membentuk *path* keputusan sudah pterdapat di dalam *path* keputusan). Akibatnya, permainan berakhir dan pemain dinyatakan kalah.

3. Dalam mengambil keputusan, hendaknya simpul-simpul yang memiliki derajat yang bernilai genap yang diutamakan.

Dalam menentukan keputusan, beberapa hal di atas dapat mengoptimalkan pencarian *path* solusi sehingga waktu yang dipakai dalam mencari solusi dapat diminimalkan.

Terkadang, beberapa hal di atas terlalu abstrak dan terkesan tidak bisa mewakili cara pencarian solusi yang optimal. Akan tetapi, beberapa hal di atas dapat meningkatkan pengoptimalan pengambilan keputusan.

D. Penerapan Teka-Teki Jembatan Königsberg dalam Pengambilan Keputusan

Permasalahan jembatan Königsberg menyimpulkan bahwa jembatan-jembatan tersebut tidak dapat dilalui tepat satu kali dan kembali ke tempat awal perpindahan. Pada *pathuku*, kita dapat menerapkan pencarian solusi berdasarkan derajat simpul terlebih dahulu. Pada kasus ini, kita dapat mengambil simpul yang berderajat genap sebagai simpul awal permainan.

Pada gambar 6, kita dapat melihat contoh pohon keputusan yang dapat dibentuk dari graf yang ada. Pohon keputusan ini belum mewakili solusi, karena pohon keputusan tersebut masih dibentuk sesuai kemungkinan-kemungkian sisi yang dapat dilalui melalui simpul terakhir yang menjadi *path* keputusan yang diambil.

Pencarian solusi pertama kali dilakukan pada simpul A. Ketika simpul A ditetapkan menjadi simpul awal, kita mengecek sisi-sisi yang berhubungan langsung dengan simpul A. Misalkan kita mengambil keputusan pada sisi AB. Secara otomatis, simpul terakhir yang menjadi *path* solusi yang telah kita bentuk adalah B. Dari simpul B, kita menentukan sisi yang menjadi *path* solusi kita selanjutnya.

Karena sisi yang berhubungan langsung dengan simpul B dan belum dijadikan sebagai *path* solusi hanyalah sisi BD, maka kita mengambil sisi BD sebagai *path* solusi. Kita telah berpindah ke sisi D, dan selanjutnya kita memilih antara sisi DA, DC, atau DF. Misalkan kita mengambil sisi DF (kita memutuskan pengecekan secara berurut ke bawah), sehingga posisi kita terakhir adalah di simpul F. pada simpul F, sisi yang belum pernah dilalui ada tiga, yaitu FA, FC, dan FE. Misalkan kita mengambil sisi FE sebagai solusi kita, maka posisi terakgir kita terdapat pada simpul E.

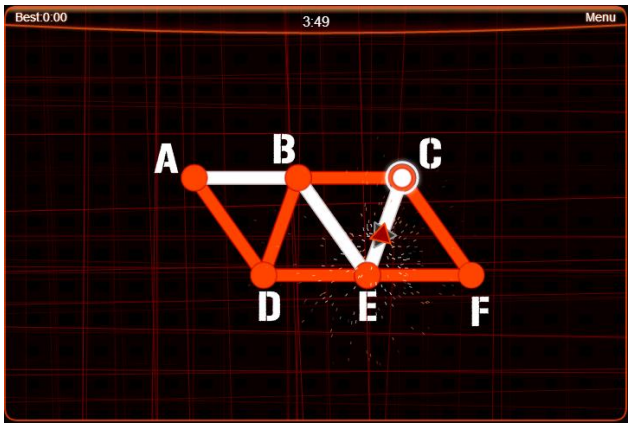
Karena sisi yang berhubungan langsung dengan simpul E dan belum dijadikan sebagai *path* solusi hanyalah sisi BD, maka kita mengambil sisi EC sebagai *path* solusi. Agar simpul-simpul bagian terbawah tidak perlu dilalui lagi, maka kita kemudian mengambil sisi CF sebagai *path* solusi kita.

Satu-satunya sisi yang terhubung dengan F dan belum dilalui adalah sisi FA, maka kita mengambil sisi FA sebagai solusi dan kita menuju simpul A. Demikian juga halnya dengan simpul A, sisi yang belum dilalui dan terhubung langsung dengan simpul A adalah sisi AC dan AD, sehingga kita harus memilih sala satu di antaranya. Pada graf, kita mengecek apakah sisi AC atau AD akan membentuk *path* sirkular dan tertutup (tidak ada jalan keluar) bagi pemain. Karena kedua sisi tidak mempunyai kemungkinan terjadinya *game over*, maka kita dapat memilih salah satu diantaranya. Misalkan kita mengambil sisi AC sebagai keputusan kita.

Karena sisi yang berhubungan langsung dengan simpul C dan belum dijadikan sebagai *path* solusi juga hanyalah sisi CD, maka kita mengambil sisi CD sebagai *path* solusi. Demikian juga halnya dengan simpul D, sisi yang tersisa hanyalah sisi DA, sehingga kita menuju simpul A lagi. Karena pada simpul A kita tidak menemukan sisi yang belum pernah dilalui, kita mengecek apakah semua sisi sudah termasuk pada *path* solusi.

Path solusi yang terbentuk :
A – B – D – F – E – C – F – A – C – D – A

Misalkan lagi untuk gambar di bawah ini.



Kita dapat memutuskan simpul awal yang menjadi *path* awal kita. Perhatikan simpul A, B, dan D! Ketiga simpul ini dapat membentuk *path* yang sirkular yang dapat mengakibatkan tidak tercapainya kondisi solusi. Maka sebaiknya kita tidak mengambil simpul A (karena simpul D berderajat ganjil). Jika kita mengambil *path* A – B – E – C – F – E – D – B – C sebagai *path* solusi kita, maka sisi AD tidak pernah dilalui dan simpul C tidak terhubung langsung ke sisi itu, sementara simpul C tidak memiliki sisi yang belum pernah dilalui lagi, sehingga permainan berakhir dan pemain dinyatakan kalah.

Kita dapat mengambil simpul B sebagai simpul awal kita. Selanjutnya kita mengecek dan memutuskan apakah simpul A, D, E, atau C yang menjadi simpul yang kita kunjungi. Misalkan saja kita mengambil sisi A, sehingga sisi yang kemudian menjadi *path* solusi kita adalah AD. D memiliki sisi DE dan DB yang belum pernah dilalui.

Pada pengecekan apakah sisi yang sedang kita cek akan membentuk sebuath *path* sirkuler tertutup, kita mengecek satu per satu. Sisi DE tidak akan membentuk sirkular tertutpu dengan *path* solusi, sedangkan sisi BD

membentuk *path* sirkuler, namun tidak tertutup (karena adanya sisi BE dan BC yang belum pernah dilalui) sehingga kedua sisi dapat dilalui. Misalkan saja kitamengambil sisi DB.

Dari simpul B, kita mengambil sisi BC. Pengambilan sisi ini mengakibatkan kita harus memutuskan kembali apakah sisi CF ataupun CE yang akan menjadi *path* solusi saat ini. Karena CE dapat mengakibatkan *path* sirkuler tertutup pada simpul – simpul C, E, dan F, kita mengambil sisi CF sebagai solusi kita kali ini. Pengambilan sisi ini mengakibatkan satu-satunya pilihan yang tersedia pada simpul F adalah FE. Dari simpul E, kita harus memutuskan apakah simpul C, B, atau D yang akan menjadi *path* solusi kita. Ketiga simpul ini dapat menyebabkan *path* solusi tertutup, sementara sisi yang tertinggal hanyalah sisi – sisi yang terhubung secara langsung dengan simpul C, sehingga solusi tidak tercapai. Kita dapat merunut-balik segala solusi ini tahap per tahap, dimulai terurut dari simpul terakhir solusi sampai simpul awal.

Path solusi yang diambil di atas

IV. KESALAHAN UMUM

Pada pencarian solusi, terkadang pemain tidak mendeteksi dengan jeli apakah keputusan yang diambil memungkinkan kondisi solusi atau tidak. Hal ini terjadi karena terkadang pemain berpacu dengan waktu dan tidak ingin tertangkap oleh chaser. Dalam pergerakannya, chaser telah ditetapkan pada kecepatan tertentu dan mulai bergerak pada suatu waktu tertentu. Jika kecepatan rata-rata pengambilan keputusan lebih besar daripada kecepatan chaser, maka dapat dipastikan pemain akan kalah.

Selain itu, algoritma runut-balik juga sering diartikan sebagai algoritma *brute-force*. Pengambilan keputusan yang terkesan "menurut abjad" atau menurut urutan simpul secara horizontal ataupun vertikal dapat memperlambat pengambilan keputusan. Hal ini menyebabkan pengambilan keputusan menjadi tidak efektif.

Pengambilan simpul awal juga menjadi salah satu penyebab pengambilan keputusan menjadi tidak efektif (atau bahkan dapat menyebabkan *zero solution*).

V. KESIMPULAN

Algoritma runut-balik dapat diterapkan pada permainan *pathuku*. Akan tetapi, dalam mengoptimalkan pencarian solusi, kita perlu menerapkan beberapa hal yang dirasa perlu dalam mengoptimalkan pencarian keputusan.

Pada pencarian solusi, terkadang pemain tidak mendeteksi dengan jeli apakah keputusan yang diambil memungkinkan kondisi solusi atau tidak. Hal ini terjadi karena terkadang pemain berpacu dengan waktu dan tidak ingin tertangkap oleh chaser. Dalam pergerakannya, chaser telah ditetapkan pada kecepatan tertentu dan mulai bergerak pada suatu waktu tertentu. Jika kecepatan rata-rata pengambilan keputusan lebih besar daripada kecepatan chaser, maka dapat dipastikan pemain akan kalah.

Selain itu, algoritma runut-balik juga sering diartikan sebagai algoritma *brute-force*. Pengambilan keputusan yang terkesan "menurut abjad" atau menurut urutan simpul secara horizontal ataupun vertikal dapat memperlambat pengambilan keputusan. Hal ini menyebabkan pengambilan keputusan menjadi tidak efektif.

Pengambilan simpul awal juga menjadi salah satu penyebab pengambilan keputusan menjadi tidak efektif (atau bahkan dapat menyebabkan *zero solution*).

VI. SANWACANA

Penulis mengucapkan rasa syukur dan terima kasih kepada Tuhan yang memampukan untuk menyelesaikan penulisan makalah ini. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir dan Dr. Masayu Leyla Khodra yang mendukung penulisan makalah ini. Penulis juga berterima kasih kepada teman-teman Teknik Informatika angkatan 2012 atas kesediaannya dalam mendukung baik dari segi pengetahuan maupun semangat. Semoga makalah ini dapat menambah pengetahuan pembaca dan menambah wawasan bagi pembaca, khususnya pembaca yang bergerak dalam bidang keinformatikaan.

REFERENSI

[1] http://majalah1000guru.net/2013/08/jembatan-konigsberg/
[2] Munir, Rinaldi, *Diktat Kuliah Strategi Algoritma* 3rd, Bandung: Institut Teknologi Bandung , 2009.
[3] http://arifcahyadiblog.blogspot.com/2010_08_06_archive.html
[4] Munir, Rinaldi, *Diktat Kuliah Matematika Diskrit*, Bandung: Institut Teknologi Bandung , 2006.

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, 19 Mei 2014

Junita Sinambela 13512023

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, 19 Mei 2014

Junita Sinambela 13512023