Implementasi Algoritma *Backtracking* untuk Memecahkan *Puzzle The Tile Trial* pada Permainan Final Fantasy XIII-2

Michael - 13514108
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514108@std.stei.itb.ac.id

Abstract*—Permainan *puzzle* adalah permainan yang membutuhkan pemikiran yang cerdik dari pemainnya. Namun, terkadang walaupun sudah berusaha berpikir keras, pemain tidak dapat menemukan solusi dari *puzzle* yang dimainkan. Salah satu *puzzle* yang cukup sulit untuk ditemukan solusinya adalah *puzzle Tile Trial*. Penggunaan algoritma *backtracking* dapat membantu penyelesaian *puzzle Tile Trial.

***Keywords—Tile Trial, backtracking, crystal,* graf**

I. PENDAHULUAN

Permainan *puzzle* adalah permainan teka-teki di mana sang pemain harus berpikir untuk memecahkan teka-teki tersebut. Saat ini, sudah banyak sekali permainan *puzzle*, baik yang dimainkan secara fisik maupun secara digital. Permainan *puzzle* pun seringkali muncul pada permainan lain. Salah satunya adalah permainan RPG (*Role Playing Game*). Pada permainan RPG, pemain mengendalikan seorang atau beberapa orang karakter. Karakter tersebut akan berinteraksi dengan karakter lain, bertarung melawan musuh, mencari benda atau harta karun, dan lain-lain untuk terus melanjutkan cerita permainan. Terkadang, karakter yang kita kendalikan akan menemukan sebuah teka-teki, misalnya saat berada pada suatu tempat kuno, karakter harus memecahkan teka-teki supaya bisa masuk ke dalam tempat tersebut. Biasanya, *puzzle* tersebut akan memiliki beberapa *level*, dimulai dengan *level* yang mudah dan kemudian akan menjadi lebih sulit.

Salah satu permainan RPG yang karakternya sering menemukan *puzzle* di perjalanannya adalah seri permainan Final Fantasy. Salah satu puzzle yang menarik adalah *puzzle The Tile Trial* pada permainan Final Fantasy XIII-2.

II. FINAL FANTASY XIII-2

Gambar 1. Logo Final Fantasy XIII-2
(finalfantasy.wikia.com)

Final Fantasy XIII-2 adalah sebuah *video game* RPG (*Role Playing Game*) yang dikembangkan dan dipasarkan oleh perusahaan Square Enix. Permainan ini pada awalnya dirilis untuk konsol permainan PlayStation 3 dan Xbox 360 pada tahun 2011 (untuk daerah Jepang) dan tahun 2012 (untuk daerah Amerika Utara). Permainan ini kemudian dirilis sehingga dapat dimainkan pada PC (*Personal Computer*) pada tahun 2014. Final Fantasy XIII-2 adalah sekuel dari permainan Final Fantasy XIII.

Pada permainan ini, pemain akan mengendalikan 2 orang karakter, yaitu Serah Farron dan Noel Kreiss. Jalan cerita yang diikuti adalah cerita di mana kedua karakter utama berpetualang mencari kakak dari Serah yang bernama Lightning. Lightning adalah karakter utama dari permainan Final Fantasy XIII. Pada akhir cerita Final Fantasy XIII, Lightning menghilang sehingga Serah harus mencarinya. Hilangnya Lightning berkaitan dengan nasib dari dunia mereka. Lightning menghilang secara misterius, dia tidak dapat ditemukan di mana pun di dunia mereka. Ketika Serah bertemu Noel, dia mengetahui bahwa Noel bukanlah orang yang datang dari dunia Serah, melainkan dari masa depan. Noel kembali ke masa lalu untuk mengubah sejarah supaya dunia tidak kiamat. Serah yang mengetahui bahwa manusia

bisa pergi ke masa lalu dan masa depan, memutuskan untuk melakukan perjalanan melewati waktu untuk mencari kakaknya.

Pada saat Serah dan Noel mengunjungi kota Oerba pada tahun 400 AF (*After the Fall*, metode penanggalan tahun pada seri permainan Final Fantasy XIII-2), mereka menemukan beberapa *puzzle* yang harus dipecahkan untuk mendapatkan *fragment* (*fragment* adalah benda yang harus dikumpulkan pada permainan ini). Salah satu *puzzle* yang harus dipecahkan bernama *The Tile Trial*.

III. The Tile Trial

The Tile Trial, atau dikenal juga dengan nama *Vanishing Floor* adalah salah satu dari 3 *puzzle* yang harus diselesaikan pada area *Temporal Rift* di permainan Final Fantasy XIII-2. Kedua *puzzle* lainnya adalah *The Crystal Bonds* dan *The Hands of Time.*

Pada *Tile Trial*, pemain akan ditempatkan pada sebuah *area* dengan banyak ubin yang melayang. Cara menyelesaikan *puzzle* ini adalah dengan mendapatkan seluruh *crystal* yang ada pada area tersebut, sekaligus mencapai ubin finish. Pemain hanya dapat berpindah ke ubin yang bersebelahan dengan ubin yang sedang dipijak. Pemain hanya dapat berpindah 1 ubin setiap langkah dan tidak dapat bergerak secara diagonal. Pemain harus mengulang dari awal jika terjatuh.

Ada dua jenis ubin yang dapat dipijak oleh pemain. Ubin pertama berwarna merah dan ubin kedua berwarna kuning. Ubin yang berwarna merah hanya dapat dipijak sekali oleh pemain. Setelah pemain berpindah dari ubin tersebut, ubin akan menghilang. Selain itu, jika pemain berada terlalu lama di atas ubin merah, ubin juga akan menghilang dan pemain akan jatuh. Ubin yang berwarna kuning dapat dipijak dua kali oleh pemain. Setelah dipijak satu kali, ubin kuning akan berubah ubin merah.

Beberapa *puzzle* yang sulit memiliki *crystal* yang dapat berpindah tempat tiap beberapa detik. Oleh karena itu, pemain harus menyesuaikan waktu untuk bergerak dan mendapatkan *crystal*.

Gambar 2. Tile Trial sederhana 4 crystal
(youtube.com)

Gambar3. Tile Trial dengan ubin kuning
(youtube.com)

Gambar4. Tile Trial yang lebih sulit
(youtube.com)

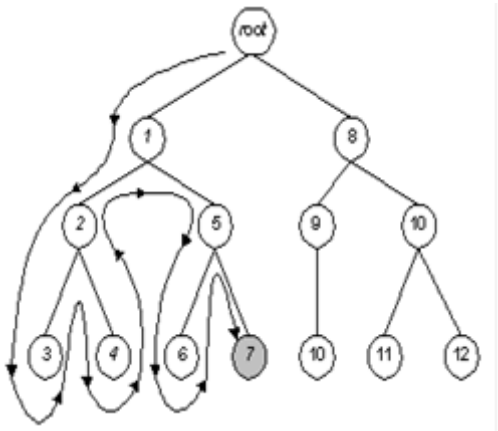
IV. Dasar Teori

A. *Depth First Search* (DFS)

DFS merupakan salah satu algoritma yang dapat digunakan untuk menelusuri graf. DFS dikenal juga sebagai algoritma traversal graf secara mendalam. Pencarian dengan DFS disebut juga preorder.

Langkah-langkah dari algoritma DFS adalah sebagai berikut :

1. Kunjungi simpul pertama, misal simpul v.
2. Kunjungi simpul yang bertetangga dengan simpul v, misal simpul w.
3. Gunakan DFS pada simpul w.
4. Jika mencapai sebuah simpul sedemikian sehingga seluruh tetangga dari simpul tersebut telah dikunjungi, *backtrack* ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai tetangga yang belum dikunjungi.
5. Pencarian selesai ketika tidak ada lagi simpul yang bisa dikunjungi yang dapat dicapai dari simpul yang sudah dikunjungi.



Gambar 5. Skema DFS
(zeromin0.blogspot.co.id)

Pada gambar 5, angka pada simpul menunjukkan urutan penelusuran graph secara DFS, yaitu dimulai dari simpul *root*, kemudian menelusuri simpul yang bertetangga dengan *root* yaitu simpul 1. Dari simpul 1, telusuri simpul yang bertetangga dengan simpul 1 yaitu simpul 2. Begitu seterusnya hingga solusi ditemukan atau tidak ada lagi simpul yang bisa dikunjungi.

Pencarian dengan DFS dapat cepat menemukan hasil jika solusi ditemukan di sebelah kiri pada pohon ruang solusi. Kelebihan dari metode DFS adalah kebutuhan memorinya sedikit. Kerugian dari metode DFS adalah tidak menjamin menemukan solusi yang optimal, walaupun DFS menjamin ditemukannya solusi jika kedalaman dari graph terbatas.

B. *Backtracking* (Runut Balik)

Algoritma *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Algoritma *backtracking* merupakan perbaikan dari algoritma *exhaustive search*. Pada *exhaustive search*, seluruh kemungkinan solusi diperiksa satu per satu, sementara pada algoritma *backtracking*, yang diperiksa hanyalah pilihan yang mengarah pada solusi. PIlihan yang tidak mengarah pada solusi dibuang.

Properti umum dari Backtracking :

1. Solusi Persoalan yang dinyatakan sebagai vektor dengan n-tuple

   X = (x1, x2, …, xn), xi ∈ Si

2. Fungsi pembangkit nilai xk yang dinyatakan sebagai T(k)
   T(k) membangkitkan nilai untuk xk yang merupakan komponen vektor solusi

3. Fungsi pembatas yang dinyatakan sebagai B(x1, x2, …, xk)

   Fungsi pembatas menentukan apakah langkah yang sudah diperoleh, yaitu (x1, x2, …, xk) mengarah pada solusi. Jika iya, maka akan dilanjutkan dengan membangkitkan xk+1. Jika tidak, maka (x1, x2, …, xk) dibuang (tidak dipertimbangkan lagi) dan melanjutkan pencarian menggunakan langkah yang lain.

Algoritma *backtracking* berbasis pada DFS, yaitu pencariannya dilakukan secara mendalam. Perbedaannya dengan DFS adalah pada backtracking jika ditemui simpul yang tidak mengarah pada solusi (melanggar fungsi pembatas) maka simpul tersebut akan dibunuh.

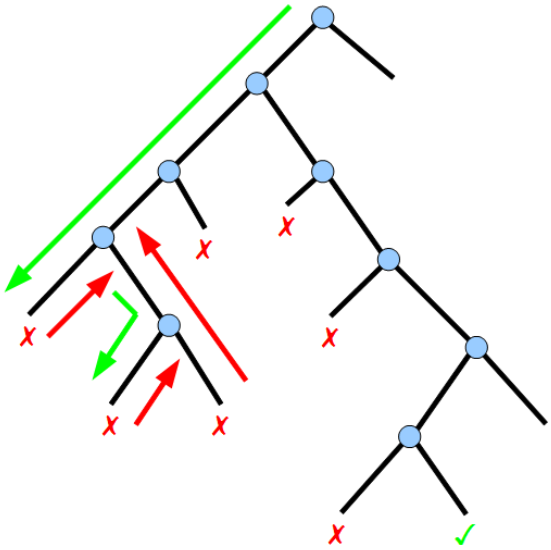
Langkah-langkah algoritma *backtracking* adalah sebagai berikut :

1. Solusi dicari degan membentuk lintasan dari akar ke daun pada pohon ruang status yang dibentuk secara dinamis. Simpul yang sudah dilahirkan disebut simpul hidup (*live node*) dan simpul yang sedang diperluas disebut simpul-E (*Expand node*).

2. Jika lintasan yang dibentuk dengan memperluas simpul-E tidak mengarah pada solusi (melanggar fungsi pembatas) maka simpul tersebut dibunuh dan menjadi simpul mati (*death node*).

3. Jika lintasan berakhir dengan simpul mati, maka pencarian akan dirunut balik sampai simpul hidup terdekat, kemudian simpul tersebut menjadi simpul-E yang baru. Pencarian dilanjutkan dengan memperluas simpul-E yang baru.

4. Pencarian berhenti jika solusi ditemukan atau tidak ada lagi simpul hidup yang dapat dijadikan simpul-E (solusi tidak ditemukan).



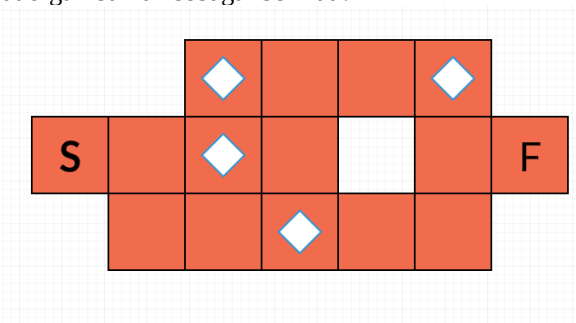
Gambar 6. Skema Backtracking
(w3.org)

Pada gambar 6, anak panah hijau menunjukkan urutan pemeriksaan simpul. Tanda x menandakan bahwa simpul tersebut merupakan simpul mati (*death node*). Anak panah berwarna merah menunjukkan alur runut balik (*backtracking*) ke simpul hidup.

V. Implementasi Algoritma Backtracking dalam Menyelesaikan Tile Trial

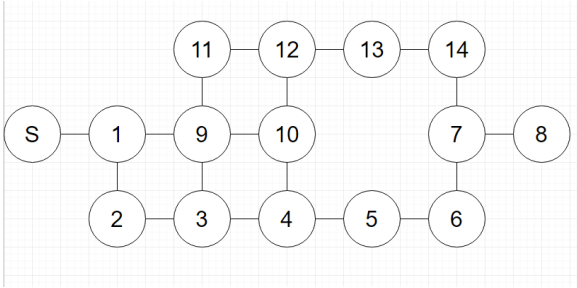
A. Representasi Graf

Puzzle Tile Trial pada gambar 2 jika dilihat dari atas, maka dapat digambarkan sebagai berikut :



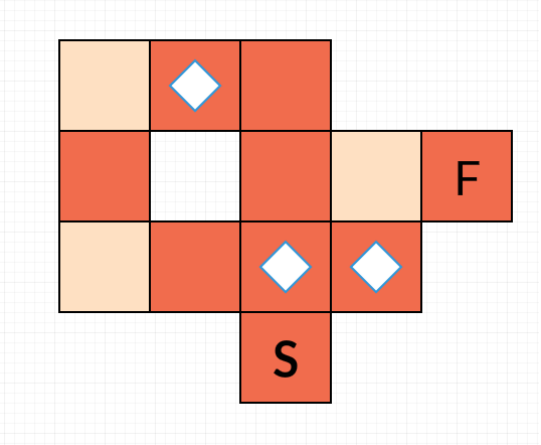
Gambar 7. *Puzzle Tile Trial* 1

Puzzle dapat direpresantasikan sebagai graf, yaitu dengan menyatakan tiap ubin sebagai simpul. Representasi graf dari gambar 7 adalah :



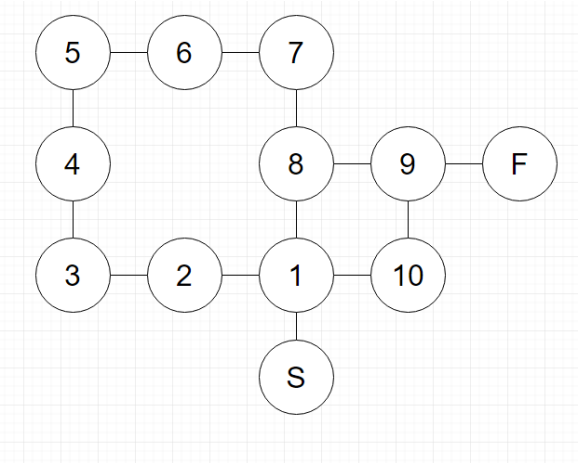
Gambar 8. Representasi Graf *Puzzle Tile Trial* 1

Sedangkan *puzzle* pada gambar 3 jika dilihat dari atas, maka dapat digambarkan sebagai berikut :



Gambar 9. Puzzle Tile Trial 2

Puzzle dapat direpresantasikan sebagai graf, yaitu :



Gambar 10. Representasi Graf *Puzzle Tile Trial* 2

Dikarenakan adanya ubin yang memiliki *crystal* dan juga ada ubin yang dapat dilewati dua kali, maka simpul dari graf harus memilki properti khusus, yaitu :

```
Deklarasi
type node  = record
  { count : integer
    {berapa kali ubin bisa dilewati}
    crystal : Boolean
    {apakah terdapat crystal pada ubin}
  }
```

Simpul 4, 9, 11, 14 pada graf di gambar x serta simpul 1, 6, dan 10 pada graf di gambar y memiliki nilai boolean crystal = true. Simpul 3, 5, dan 9 pada graf di gambar y mempunyai nilai count = 2, sedangkan simpul lainnya mempunyai nilai count = 1.

B. Algoritma Backtracking

Langkah-langkah algoritma yang digunakan untuk menyelesaikan *Tile Trial* adalah sebagai berikut :

1. Pilih simpul pertama yang bertetangga dengan simpul S, simpul tersebut dijadikan *expand node*.
2. Jika pada *expand node* crystal = true, tambahkan jumlah crystal yang diperoleh.
3. Cek apakah *expand node* adalah simpul F (finish). Jika iya, cek apakah jumlah crystal yang sudah diperoleh sama dengan jumlah crystal seharusnya, selesai. Jika tidak, *backtrack* ke simpul hidup, jadikan *expand node*.
4. Cek simpul yang bertetangga dengan *expand node*, jika count simpul tersebut > 0, jadikan simpul tersebut *expand node*. Jika simpul tetangga lebih dari 1, pilih yang memiliki nomor lebih kecil. Jika tidak ada simpul tetangga dengan count > 0, backtrack ke simpul hidup terakhir yang memiliki tetangga dengan count > 0, jadikan simpul tetangga tersebut *expand node.*

5. Panggil algoritma *backtracking* secara rekursif untuk *expand node* (ulangi langkah 2-4 untuk *expand node* baru).

C. Pseudocode Algoritma

Struktur data yang digunakan adalah sebagai berikut :

```
{ Kamus Global }
Deklarasi
type node  = record
  { count : integer
    {berapa kali ubin bisa dilewati}
    crystal : Boolean
    {apakah terdapat crystal pada ubin}
  }
const n : integer = ...
{ jumlah simpul pada graf - 1 }
graf : array[0..n, 0..n] of boolean
{ adjacency matrix }
simpul : array[0..n] of node
{ menyimpan data tentang simpul }
solusi : array[0..2*n] of integer
{ disiapkan untuk 2*n langkah }
maxcrys : integer
{ jumlah crystal yang harus didapatkan }
const start : integer = 0
{ simpul mulai }
const finish : integer = n
{ simpul finish }
```

Algoritma cetak solusi adalah sebagai berikut :

```
procedure CetakSolusi (input solusi : array
of integer, input current : integer)
{ Mencetak solusi Tile Trial
  Masukan : solusi sudah terisi dari 1
sampai dengan current
  Keluaran : solusi dicetak ke output device
}
Deklarasi
  i : integer
Algoritma
  for i<-1 to current do
    write(solusi[i])
  endfor
```

Algoritma penyelesaian *puzzle Tile Trial* secara rekursif :

```
procedure solveTileTrial (input x : array of
node, input k : integer, input jumcrys :
integer, input current : integer)
{ Mencari semua solusi menyelesaikan puzzle
  Tile Trial secara rekursif
  Masukan : x adalah array of node, yang
  memiliki informasi crystal dan count
  Masukan : k adalah nomor simpul graf
  Masukan : jumcrys adalah jumlah crystal
  yang sudah diperoleh
  Masukan : current adalah angka yang
  menunjukkan indeks solusi
}
Deklarasi

Algoritma
  x[k].count = x[k].count - 1  {simpul k
                          sudah dikunjungi}
  solusi[current] = k    {simpul k
                       dimasukkan pada solusi}
  if (x[k].crystal)
    jumcrys = jumcrys + 1
  endif
  if (a = finish)
    if (jumcrys = maxcrys)
      CetakSolusi(solusi, current)
    endif
  else
    foreach a = nomor node yang bertetangga
    dengan k do
      if (x[a].count > 0)
        solveTileTrial(x, a, jumcrys,
                          current + 1)
      endif
    endfor
    solusi[current] = 0 { jika simpul k
    sudah selesai diperiksa, solusi
    disiapkan untuk simpul lain }
  endif
```

Pada main program, setelah graf dan simpul diinisialisasi, maka untuk menyelesaikan puzzle dapat dipanggil prosedur solveTileTrial dengan cara :

```
solveTileTrial(simpul, start, 0, 1)
```

Pada graf puzzle Tile Trial 2, iterasi yang terjadi adalah sebagai berikut :

S-1-2-3-4-5-6-7-14-13-12-10-9-11

Namun karena dari simpul 11 tidak memiliki simpul bertetangga dengan count > 0, simpul 11 adalah simpul mati sehingga dilakukan backtrack ke simpul 12. Iterasi yang terjadi adalah :

S-1-2-3-4-5-6-7-14-13-12-11-9-10

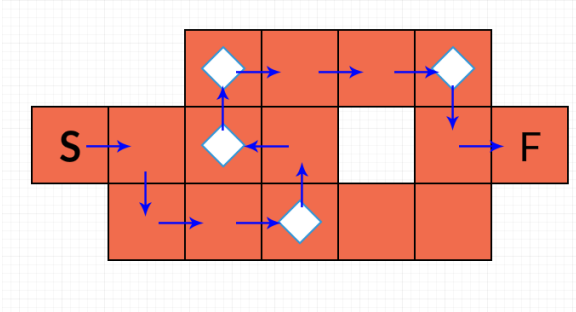
Karena simpul 10 juga simpl mati, *backtrack* ke simpul 7 :

S-1-2-3-4-5-6-7-15

Simpul 15 merupakan simpul finish, namun jumcrys tidak sama dengan maxcrys sehingga dilakukan *backtrack* lagi ke simpul 4 :

S-1-2-3-4-10-9-11-12-13-14-7-15

Simpul 15 merupakan simpul finish dan jumcrys = maxcrys, sehingga solusi tersebut akan dicetak. Selanjutnya dilakukan *backtrack* ke simpul 3 dan dicari solusi yang lain.

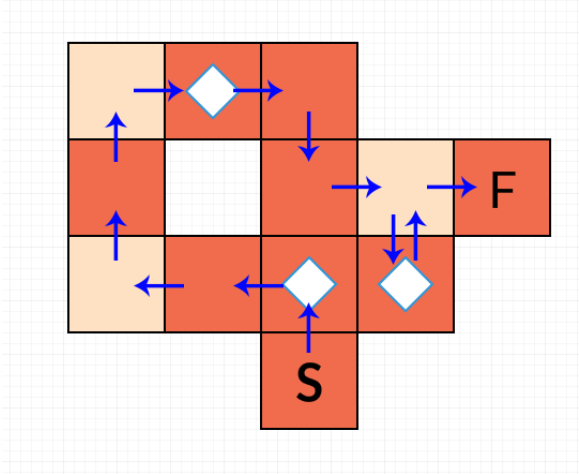


Gambar 10. Solusi Puzzle Tile Trial 1

Pada graf *puzzle Tile Trial 2*, iterasi yang terjadi adalah :

S-1-2-3-4-5-6-7-8-9-10-9-F

Simpul 9 dapat dikunjungi dua kali karena count awal simpul 9 = 2 (pada gambar x terlihat ubin yang merepresentasikan simpul 9 berwarna kuning).



Gambar 11. Solusi Puzzle Tile Trial 2

VI. KESIMPULAN

Puzzle Tile Trial yang terdapat pada permainan Final Fantasy XIII-2 dapat diselesaikan dengan menggunakan algoritma *backtracking*. *Puzzle* tersebut dapat direpresentasikan sebagai graf sederhana tak berarah. Selanjutnya, solusi untuk *puzzle* tersebut dapat ditemukan dengan mengiterasi simpul-simpul pada graf. Ketika dtemui simpul mati, dilakukan *backtrack*. Algoritma *backtracking* dijamin menghasilkan solusi jika *puzzle* yang diberikan adalah *puzzle* yang memiliki solusi.

UCAPAN TERIMA KASIH

Penulis mengucapkan terimakasih kepada Tuhan Yang Maha Esa, atas berkat-Nyaselama pengerjaan makalah ini. Mungkin makalah ini jauh dari sempurna, tapi tanpa berkat-Nya, makalah ini tidak dapat terselesaikan. Penulis juga mengucapkan terimakasih kepada yang terhormat Bapak Dr. Ir. Rinaldi Munir, M.T. dan Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. atas pengajaran dan bimbingan yang diberikan pada mata kuliah Strategi Algoritma, sehingga makalah ini bisa terselesaikan dengan baik.

REFERENCES

[1] Munir, Rinaldi. 2009. *Diktat Kuliah IF2211 Strategi Algoritma*. Bandung : Penerbit Informatika.
[2] Johnston, Nathaniel. 2012. *The Complexity of the Puzzles of Final Fantasy XIII-2*.
[3] http://zeromin0.blogspot.co.id/2011/07/analisa-algoritma-depth-first-search.html , diakses pada tanggal 7 Mei 2016 pukul 9.10
[4] http://utari-21.blogspot.co.id/2012/09/algoritma-backtracking-runut-balik.html , diakses pada tanggal 7 Mei 2016 pukul 9.20
[5] https://www.w3.org/2011/Talks/01-14-steven-phenotype/ , diakses pada tanggal 7 Mei 2016 pukul 9.28
[6] https://www.youtube.com/watch?v=V4r5xMeOUlw , diakses pada tanggal 7 Mei 2016 pukul 9.41
[7] https://www.youtube.com/watch?v=EOD1pwK17Nc , diakses pada tanggal 7 Mei 2016 pukul 9.46
[8] http://finalfantasy.wikia.com/wiki/Final_Fantasy_XIII-2 , diakses pada tanggal 7 Mei 2016 pukul 9.57.

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, 7 Mei 2016

Michael – 13514108