Penerapan Algoritma BFS dan DFS pada Numeron

Fransiskus Xaverius Christian/13511016
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511016@std.stei.itb.ac.id

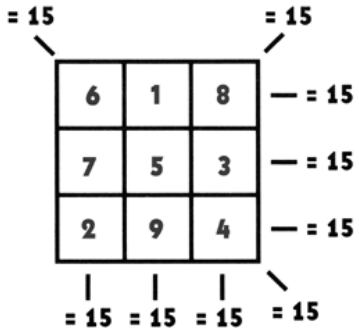
***Abstraksi*—Numeron adalah sebuah permainan(teka-teki) matematika yang dirancang untuk berbagai kalangan, mulai dari anak-anak, remaja bahkan orang tua. Perkembangan teknologi yang sangat pesat, membuat Numeron menjadi salah satu permainan yang digemari. Hal ini, mengakibatkan munculnya kebutuhan pemanfaatan teknologi informasi agar dapat menemukan solusi yang tepat untuk berbagai teka-teki ini ketika mengalami kesulitan. Karena solusi yang dicari dalam bentuk lintasan tertentu, maka digunakanlah algoritma pencarian lintasan yaitu algoritma BFS dan DFS. Pencarian lintasan-lintasan ini dilakukan berulangkali sampai seluruh ruang solusi untuk menyelesaikan teka-teki ini ditemukan.**

***Kata kunci*—DFS, BFS, lintasan.**

I. PENDAHULUAN

Matematika telah ada sejak ribuan tahun yang lalu. Ilmu ini sudah sangat membantu kita dalam berbagai aspek di kehidupan manusia dari dulu, sampai sekarang. Perkembangan ilmu yang sangat pesat, membuat masalah timbul dari berbagai ruang persoalan yang ada pada ilmu ini. Banyak persoalan matematika rumit yang hanya dapat dipecahkan oleh para ahli, sehingga kurang dimengerti oleh orang awam. Hal ini yang membuat sebagian orang, baik anak-anak, remaja, maupun dewasa kurang menyukai matematika. Orang-orang menganggap matematika itu hal yang sulit, rumit dan kompleks. Tetapi, ada juga persoalan matematika yang tidak terlalu rumit, tetapi memiliki tantangan tersendiri bagi orang awam untuk menyelesaikannya. Dan ada beberapa di antaranya sengaja dirancang dengan kesulitan tertentu agar menjadi tantangan untuk dipecahkan bahkan untuk orang awam. Hal ini sering disebut dengan teka-teki matematika. Di mana diberikan sebuah persoalan matematika, dengan ruang lingkup tertentu untuk memecahkan masalahnya. Ada beberapa teka-teki matematika yang sudah terkenal, yang sering kita temui, seperti, kotak ajaib dan Sudoku.

Permainan kotak ajaib adalah permainan di mana kita menyusun angka-angka yang awalnya berurutan, menjadi komposisi tertentu di dalam sebuah persegi agar ketika dijumlahkan setiap elemen pada baris, kolom serta diagonalnya menghasilkan jumlah tertentu yang sama. Permainan Sudoku adalah permainan matematika dengan kotak berukuran 9x9 di mana ada beberapa kotak yang sudah terisi dengan angka. Tugas pemain adalah mengisi angka-angka lain(1-9) di mana setiap baris, kolom dan daerah(kotak 3x3) harus mengandung angka 1-9 masing-masing tepat satu kali.



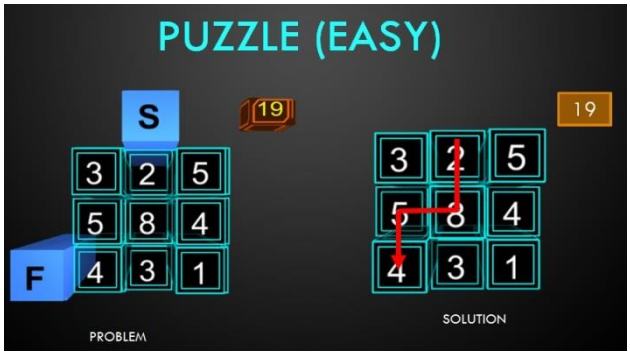
Gambar 1 Contoh permainan kotak ajaib pada persegi 3x3

II. NUMERON

Numeron adalah sebuah teka-teki matematika berbentuk persegi yang tersusun dari kumpulan angka. Pemain akan diminta untuk menemukan lintasan yang diawali dari kotak tertentu, dan harus berujung pada kotak yang telah ditentukan juga. Hal ini harus dilakukan dengan syarat, total angka yang dilalui sebagai lintasan pada persegi tersebut, jika dijumlahkan harus sama dengan nilai tertentu yang telah ditentukan sejak awal teka-teki dimulai.

Numeron merupakan sebuah permainan yang dibuat oleh dua orang Mahasiswa Teknik Informatika yaitu Ahmad Fauzan (13510004) dan Martha Monica (13510080) dalam perlombaan *Hackathon 2013*, dan berhasil meraih 2 penghargaan, yaitu juara ke-2 dalam kategori *Microsoft Challenge* dan juara ke-3 dalam *Kii Challenge*. Mereka membuat Numeron dengan sasaran orang-orang yang menganggap matematika adalah musuh, tetapi menyukai tantangan. Karena itulah, maka Numeron dibuat. Karena matematika bukanlah musuh, melainkan tantangan yang menyenangkan.

Dalam teka-teki Numeron, didesain memiliki 3 buah solusi untuk setiap teka-tekinya. Sehingga memungkinkan jika pemain yang satu dengan yang lain berhasil menyelesaikan teka-teki tersebut melalui lintasan yang berbeda. Hal ini juga sengaja dirancang agar setiap golongan pemain(baik muda maupun dewasa) dapat menentukan tingkat kesulitannya sendiri dalam menentukan lintasan solusinya.



Gambar 2. Contoh salah satu teka-teki Numeron

Dapat kita lihat, pada gambar di atas, ditunjukkan tampilan Numeron ketika dimainkan. Misalkan permainan kali ini dibuat dengan ukuran kotak 3x3, harus dimulai dari kotak kedua, dan berakhir pada kotak ketujuh, dengan jumlah seluruh lintasan bernilai 19. Solusi yang dapat diambil dalam kasus ini dari kotak kedua, menuju kotak kelima, lau keempat, sampai akhirnya ketujuh, dan tepat berjumlah 19 seperti pada gambar di atas. Kurang lebih, ini cara menyelesaikan teka-teki matematika Numeron.

III. DASAR TEORI

A. BREADTH-FIRST SEARCH(BFS)

Breadth-First Search (BFS) adalah salah satu strategi pencarian pada suatu graf yang terdiri dari dua proses utama, yaitu mengunjungi serta mengecek suatu simpul dari graf dan mengunjungi lagi setiap simpul tetangga dari simpul yang sedang dicek sekarang. Penelusuran pada BFS dimulai dari simpul akar/awal, kemudian pengecekan dilakukan untuk setiap simpul tetangga yang belum dikunjungi, begitu seterusnya sampai semua simpul sudah dikunjungi atau objektif dari penelusuran sudah dicapai, misal mencari suatu simpul dengan kondisi tertentu. Algoritma ini menggunakan struktur data queue/antrian untuk menyimpan simpul sementara yang akan dikunjungi dan sudah dikunjungi.

Untuk lebih mempermudah, ilustrasikan ide dari *Breadth-First Search* sebagai berikut:
suatu gelombang dikirimkan dari sumber, yaitu simpul akar/awal. Gelombang tersebut akan mengenai semua simpul yang berjarak satu sisi dari simpul akar. Dari sini, gelombang itu akan mengenai semua simpul yang berjarak duat sisi dari simpul akar yang bertetangga dengan semua simpul yang sebelumnya sudah dikenai, dan begitu seterusnya. Untuk mempertahankan gelombang di bagian depan dari proses, ilustrasi ini menggunakan antrian dengan aturan FIFO (*First-In-First-Out*), di mana suatu simpul akan berada di antrian apabila gelombang sudah mengenainya tetapi belum keluar dari simpul tersebut. Berikut adalah penjelasan mengenai strategi secara umum dari *Breadth-First Search*:

Setiap simpul pada graf memiliki salah satu warna atau state sebagai berikut:
1. Belum ditemukan
2. Ditemukan tetapi belum dikunjungi
3. Sudah dikunjungi

Kondisi dari suatu simpul, u, ditandai dengan satu warna khusus:
1. warna[u] = putih – untuk kondisi pertama
2. warna[u] = abu-abu – untuk kondisi kedua
3. warna[u] = hitam – untuk kondisi ketiga

Algoritma BFS ini mengimplementasikan pohon *breadth-first search* dengan simpul awal/sumber adalah s, sebagai akar. Simpul ayah yang mendahului suatu simpul adalah simpul yang menghasilkan simpul anak yang membuat simpul anak tersebut ditemukan pertama kali. Untuk setiap simpul, v, simpul ayah dari v diletakkan di variabel π[v]. Variabel lain, d[v] memiliki jumlah sisi yang dibutuhkan untuk mencapai simpul v dari simpul s (akar). Algoritma ini menggunakan antrian FIFO, Q, untuk menyimpan simpul yang berwarna abuabu.
Berikut adalah algoritma untuk mengimplementasikan BFS seperti yang sudah dijelaskan di atas:

```
BFS(V, E, s)
for each u in V − {s}//untuk setiap simpul u di V[G] kecuali s.
    do warna[u] ← putih
       d[u] ← ∞
       π[u] ← NULL
warna[s] ← abu-abu  // simpul akar ditemukan
d[s] ← 0  // inisialisasi
π[s] ← NULL    // inisialisasi
Q ← {}  // bersihkan antrian Q
ENQUEUE(Q, s)
while Q not-empty
    do u ← DEQUEUE(Q)  // u = head[Q]
       for each v bertetangga to u//iterasi simpul bertetangga
          do if warna[v] ← putih   // belum ada putih
             then warna[v] ← abu-abu
                  d[v] ← d[u] + 1
                  π[v] ← u
                  ENQUEUE(Q, v)
       DEQUEUE(Q)
warna[u] ← hitam
```

B. DEPTH-FIRST SEARCH(DFS)

Seperti algoritma *Breadth-First Search*, algoritma *Depth-First Search* menelusuri komponen-komponen yang saling terhubung pada graf dan mendefinisikan suatu pohon merentang. Ide dasar dari DFS adalah sebagai berikut: algoritma ini secara bertahap akan mengeksplorasi setiap sisi.

Penelusuran dimulai dari simpul akar, segera setelah sebuah simpul ditemukan dari simpul akar, algoritma ini akan mulai mengeksplorasi dari situ, begitu seterusnya sampai buntu, setelah tidak ada jalan, proses akan melakukan backtrack untuk kembali ke simpul di atasnya dan proses lagi ke bawah, begitu seterusnya (Tidak seperti BFS yang menaruh simpul-simpul di dalam antrian agar dapat dikunjungi belakangan).

Berikut adalah penjelasan mengenai strategi secara umum dari *Depth-First Search*:
Pilih simpul akar s dari graf, dan tandai "dikunjungi". Simpul s ini menjadi simpul u yang saat ini dikunjungi. Lalu, lakukan penelusuran pada graf untuk menemukan suatu sisi (u,v) dari simpul u. Apabila sisi (u,v) merujuk kepada simpul v yang sudah dikunjungi, maka lakukan *backtrack* kembali ke simpul u. Tetapi, jika sisi (u,v) merujuk ke simpul yang belum dikunjungi, maka tandai simpul tersebut sudah dikunjungi dan jadikan simpul v sebagai simpul saat ini, dan ulangi langkah-langkah di atas. Suatu saat proses ini akan menemui jalan buntu, yaitu keadaan di mana setiap sisi dari simpul u saat ini merujuk kepada simpul yang sudah dikunjungi. Untuk keluar dari situasi *deadlock* ini, proses harus berjalan mundur melalui sisi yang membawa ke simpul u saat ini dan menuju ke simpul v yang sebelumnya sudah dikunjungi. Simpul v ini kemudian dijadikan simpul u saat ini dan lakukan proses seperti di atas untuk sisi-sisi yang masih belum diproses. Apabila algoritma ini sudah melakukan proses mundur atau *backtrack* sampai ke simpul akar s, maka telah terbentuk pohon DFS dengan semua simpul sudah ditelusuri dari simpul akar s.

Seperti BFS, setiap simpul dari pohon pada DFS berada dalam salah satu kondisi di bawah ini:

1. Belum ditemukan
2. Sudah ditemukan (belum semua simpul di bawah simpul ini ditelusuri dari simpul ini)
3. Selesai (semua simpul di bawah simpul ini sudah ditelusuri)

Kondisi atau state dari simpul disimpan dalam satu variabel warna seperti di bawah ini:

1. warna[u] = putih – untuk kondisi satu
2. warna[u] = abu-abu – untuk kondisi dua
3. warna[u] = hitam – untuk kondisi tiga

Seperti BFS, *Depth-First Search* menggunakan π[v] untuk mencatat simpul ayah dari simpul v. π[v] = NULL jika dan hanya jika simpul v adalah simpul akar.

Berikut adalah algoritma untuk mengimplementasikan DFS seperti yang sudah dijelaskan di atas:

```
DFS (V, E)
for each simpul u in V[G]
    do warna[u] ← putih
       π[u] ← NULL
for each simpul u in V[G]
    do if warna[u] ← putih
       then DFS-Visit(u) //membuat pohon DFS baru dari
                          simpul u
```

```
DFS-Visit(u)
warna[u] ← abu-abu  // u ditemukan
for each simpul v yang bertetanggadengan u  //telusuri
                                            sisi (u, v)
    do if warna[v] ← putih
       then π[v] ← u
            DFS-Visit(v)
warna[u] ← hitam
```

IV. IMPLEMENTASI BFS DAN DFS PADA NUMERON

Pencarian lintasan solusi pada Numeron dilakukan dengan menggunakan algoritma BFS dan DFS menggunakan Bahasa C++.

A. Pencarian Lintasan Solusi Numeron Menggunakan Algoritma BFS

Algoritma BFS dipilih karena dapat menemukan seluruh ruang solusi yang memungkinkan secara merata, dari berbagai tingkat kedalaman. Pencarian dilakukan dengan syarat sebagai berikut :

a. Prioritas gerak
Pergerakan yang dilakukan dilihat dengan urutan prioritas *Right*(*R*), *Down*(*D*), *Left*(*L*), lalu *Up*(*U*).

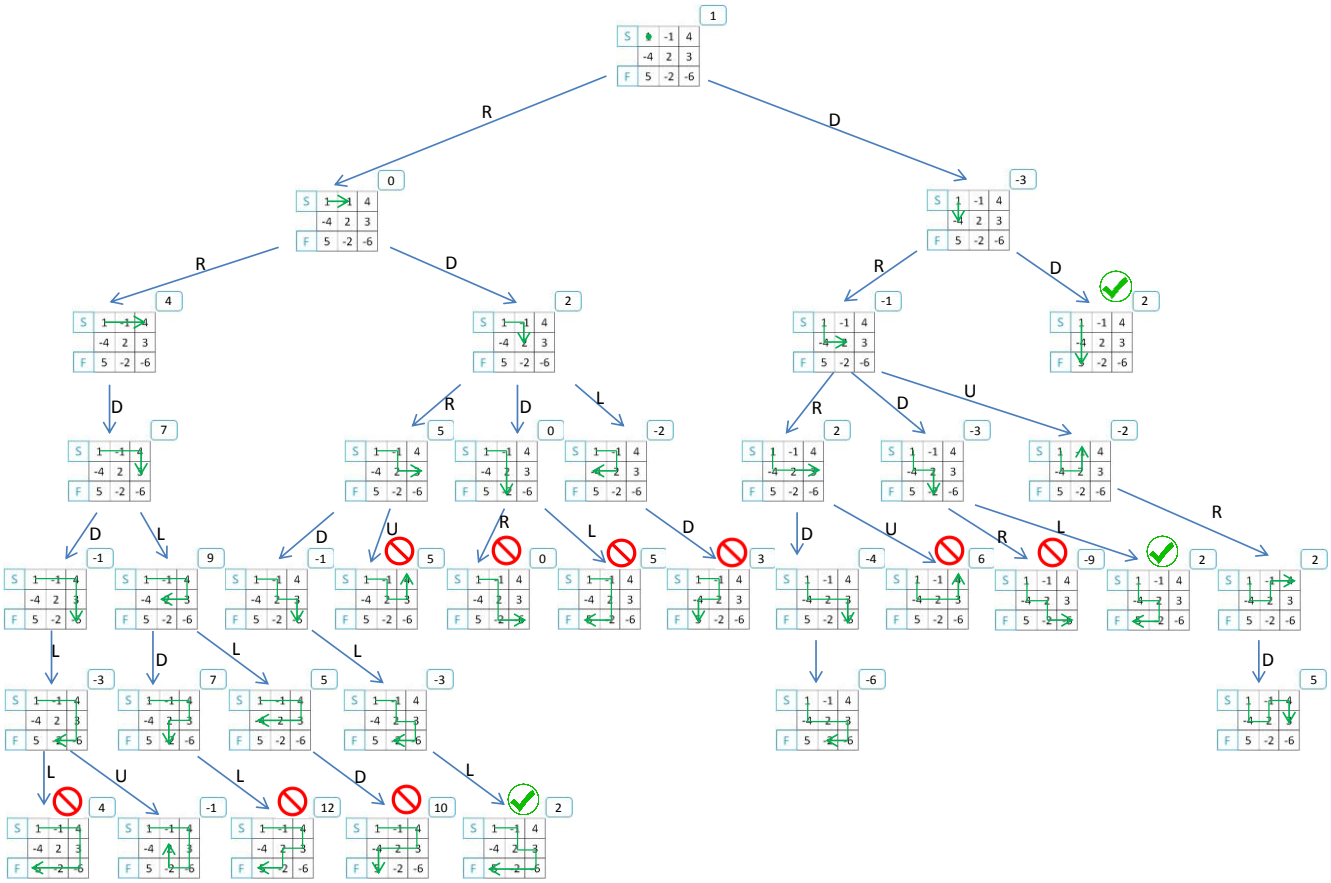
b. Status lintasan
Status lintasan solusi setiap teka-teki di antrian juga di cek, apakah lintasan tersebut sudah pernah dilalui atau belum. Jika belum, lintasan tersebut akan dicek, anak lintasan yang mungkin ke mana saja, lalu ditambahkan pada antrian baru lintasan itu akan diproses. Jika sudah, maka lintasan yang menunggu di antrian akan diambil untuk diproses.

c. Batas pergerakan
Setiap arah gerak akan dicek apakah arah tersebut memungkinkan atau tidak. Jika pada arah tersebut sudah buntu, atau telah mencapai titik finish tetapi belum memiliki total nilai yang sesuai dengan yang ditentukan, maka kemungkinan lintasan tersebut akan dianggap gagal, dan akan dilanjutkan dengan antrian lintasan selanjutnya.

Berikut dilampirkan *pseudo-code* Numeron BFS Solver :

```
Numeron BFS Solver

Antrian.tambahkan(mulai);
Map <Puzzle,boolean> Dikunjungi;
Integer batasnilai;
NumeronBFS(){
  while(Antrian tidak kosong) do{
     CurrentPuzzle ← Antrian.ambil();
```



Gambar 3 Pohon Ruang Status Numeron BFS Solver

```
    If(belum Dikunjungi(CurrentPuzzle){
      Dikunjungi.tambahkan(CurrentPuzzle,true);
       If(count(CurrentPuzzle)=batasnilai and
       LastMove=finish)
            Solusi.tambahkan(CurrentPuzzle);
       else if(LastMove=finish)//do nothing;
       else
           CekFeasibleMove(CurrentPuzzle);
            //mengecek Current Puzzle dapat
               bergerak ke arah mana, dan
               menambahkan pada Antrian
               seluruh kemungkinan anak pada
               simpul tersebut.
    }
  }
}
```

B. Pencarian Lintasan Solusi Numeron Menggunakan Algoritma DFS

Algoritma DFS dipilih karena dapat menemukan seluruh ruang solusi bisa ditemukan di mana saja. Oleh karena itu, Pencarian dilakukan dengan syarat sebagai berikut :

a. Prioritas gerak

Pergerakan yang dilakukan sama seperti BFS, dilihat dengan urutan prioritas *Right*(*R*), *Down*(*D*), *Left*(*L*), lalu *Up*(*U*).

b. Status lintasan

Status lintasan solusi setiap teka-teki di *stack* juga di cek, apakah lintasan tersebut sudah pernah dilalui atau belum. Jika belum, lintasan tersebut akan dicek, apakah jumlahnya sudah sama dengan nilai batas yang ditentukan atau belum. Jika belum, cari satu pergerakan yang mungkin lalu ditambahkan pada *Stack*. Lalu lanjutkan pencarian.

c. Batas pergerakan

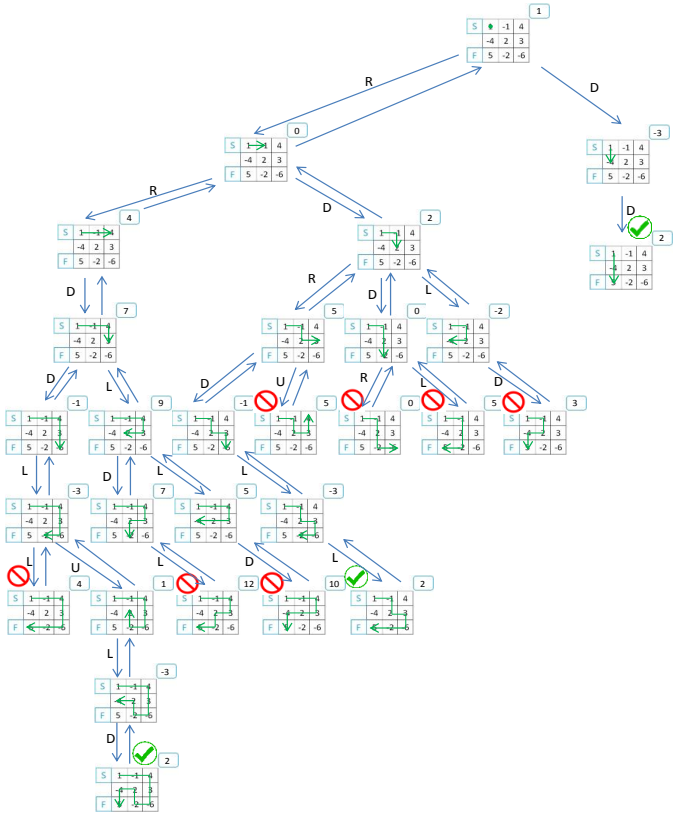
Setiap arah gerak akan dicek apakah arah tersebut memungkinkan atau tidak. Jika pada arah tersebut sudah buntu, atau telah mencapai titik finish tetapi belum memiliki total nilai yang sesuai dengan yang ditentukan, maka kemungkinan lintasan tersebut akan dianggap gagal, dan akan dilanjutkan dengan antrian lintasan selanjutnya.

Berikut dilampirkan *pseudo-code* Numeron DFS Solver :

Numeron DFS Solver

```
Stack.tambahkan(mulai);
Map <Puzzle,boolean> Dikunjungi;
Integer batasnilai;
NumeronDFS(){
while(Stack tidak kosong) do{
   CurrentPuzzle ← Stack.ambil();
```

```
    If(belum Dikunjungi(CurrentPuzzle){
        Dikunjungi.tambahkan(CurrentPuzzle,true);
        If(count(CurrentPuzzle)=batasnilai and
        LastMove=finish)
            Solusi.tambahkan(CurrentPuzzle);
        else if(LastMove=finish)//do nothing;
        else
            CekNextFeasibleMove(CurrentPuzzle);
            //jika ada, tambahkan satu next move
              pada Stack
      }
    }
}
```



Gambar 4. Pohon Ruang Status Numeron DFS Solver

V. PENGUJIAN DAN PEMBAHASAN

Penerapan konsep ini juga didukung dengan implementasi program pada Bahasa C++.

Gambar 5. Persoalan Numeron 3x3

Berikut hasil eksekusi program untuk persoalan Numeron di atas dengan menggunakan algoritma BFS dan DFS:

```
Batas nilai yang harus dicapai : 2

Start : 0

Finish : 7

Bentuk teka-teki Numeron :
----------
1 ! -1 ! 4
----------
-4 ! 2 ! 3
----------
5 ! -2 ! -6
----------

Solusi Numeron dengan menggunakan Algoritma BFS :
Solusi 1 :
Down-Down.
Solusi 2 :
Down-Right-Down-Left.
Solusi 3 :
Right-Down-Right-Down-Left-Left.
```

Gambar 6. Hasil eksekusi menggunakan BFS

Diperoleh 3 solusi dari persoalan Numeron di atas menggunakan algoritma BFS. Selanjutnya dilampirkan hasil menggunakan algoritma DFS :

```
Batas nilai yang harus dicapai : 2

Start : 0

Finish : 7

Bentuk teka-teki Numeron :
----------
1 ! -1 ! 4
----------
-4 ! 2 ! 3
----------
5 ! -2 ! -6
----------

Solusi Numeron dengan menggunakan Algoritma DFS :
Solusi 1 :
Right-Right-Down-Down-Left-Up-Left-Down.
Solusi 2 :
Right-Down-Right-Down-Left-Left.
Solusi 3 :
Down-Down.
```

Gambar 7. Hasil Eksekusi menggunakan DFS

Ditemukan 3 solusi dari persoalan di atas menggunakan algoritma DFS. Kedua algoritma ini cukup efektif dalam mencari lintasan solusi dalam persoalan Numeron kali ini. Perbedaan waktu eksekusi di antara kedua algoritma ini, tidak signifikan, karena pencarian solusi pada permasalahan ini mencari beberapa lintasan solusi sekaligus sehingga rata-rata waktunya kurang lebih sama.

VI. PENGAKUAN

Akhirnya Tugas *paper* mata kuliah Strategi Algoritma pada semester 5 ini, dapat diselesaikan dengan baik. Tetapi semua pengalaman, pelajaran, ilmu pada Kuliah Strategi Algoritma ini tidak akan diperoleh tanpa bimbingan kedua dosen luar biasa yang mau membimbing saya sampai menjadi lebih mengerti, dan mencoba mengeksplorasi strategi algoritma dengan lebih baik lagi. Saya mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir dan Ibu Dr. Masayu Leylia Khodra yang mengajarkan kami banyak perlajaran, dan pengalaman eksplorasi yang cukup banyak dalam mata kuliah ini. Semoga, apa yang saya pelajari pada mata kuliah ini, kelak akan berguna dan bermanfaat bagi kehidupan saya dan sekitar.

DAFTAR PUSTAKA

[1] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm Diakses tanggal 19 Desember 2013 pk.10.00

[2] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/depthSearch.htm Diakses tanggal 19 Desember 2013 pk.11.00

[3] Munir, Rinaldi, Diktat Kuliah Strategi Algoritma, Penerbit Informatika : Bandung, 2009

[4] http://www.brainbashers.com/logicpuzzles.asp Diakses tanggal 15 Desember 2013 pk 21.00

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, 20 Desember 2013

Fransiskus Xaverius Christian / 13511016