Algoritma Penyelesaian *Knight's Tour Problem*

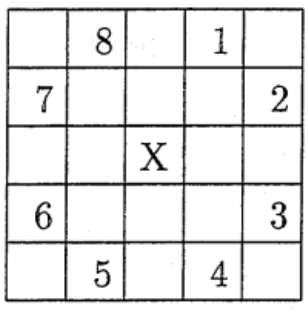
Rifo Ahmad Genadi/13516111
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13516111@std.stei.itb.ac.id

***Abstrak*— *The Knight's Tour Problem*, atau Permasalahan Perjalanan Kuda merupakan teka-teki catur yang meminta kita untuk menentukan sekumpulan langkah yang mungkin ditempuh oleh bidak kuda sehingga ia mengunjungi setiap petak pada papan catur tepat satu kali. Pada makalah ini, penulis membahas mengenai beberapa algoritma yang telah diusulkan untuk menyelesaikan Permasalahan Perjalanan Kuda, serta membandingkan hasil eksekusinya.**

Kata Kunci—Knight's tour problem, lintasan hamilton, backtracking, Warnsdorff, heuristic, greedy, teka-teki catur

I. PENDAHULUAN

The Knight's Tour Problem merupakan permasalahan kombinatorika dengan sejarah yang panjang, banyak matematikawan terkenal yang telah mencoba menyelesaikan teka-teki ini [2]. Teka-teki ini mulai dipelajari secara formal oleh Euler pada tahun 1759, menggunakan papan catur standar berukuran 8 x 8 [1.9]. Seiring waktu, teka-teki ini juga 'melahirkan' beberapa versi berbeda, kesemuanya meminta kita menyelesaikan suatu 'tugas' dengan menggerakkan bidak kuda.



Gambar 1. Langkah yang dapat diambil bidak kuda

Versi dasarnya, kita diminta untuk menentukan banyak langkah paling sedikit yang dapat diambil oleh kuda untuk mencapai suatu petak, diberikan sebuah petak awal. Pada versi lainnya, diberikan papan catur berukuran n x m, kita diminta mencari rute langkah yang dapat ditempuh kuda sehingga ia mengunjungi seluruh petak yang ada pada papan catur, versi inilah yang selanjutnya akan dibahas pada makalah ini.

Rute langkah tersebut kita namakan lintasan kuda. Suatu lintasan kuda dikatakan *tertutup* apabila petak terakhir yang dikunjungi dapat dikunjungi dari petak pertama dengan aturan gerak bidak kuda, dan dikatakan *terbuka* jika sebaliknya.

Lintasan kuda pada papan catur berukuran m x n kita gambarkan dengan sebuah matriks seperti berikut (untuk m = n = 8) :

1	16	27	22	3	18	47	56
26	23	2	17	46	57	4	19
15	28	25	62	21	48	55	58
24	35	30	45	60	63	20	5
29	14	61	34	49	44	59	54
36	31	38	41	64	53	6	9
13	40	33	50	11	8	43	52
32	37	12	39	42	51	10	7

Gambar 2. Lintasan kuda untuk papan catur 8 x 8

Setiap petak diberi label angka berdasarkan urutan langkah yang diambil oleh bidak kuda.

Pencarian lintasan perjalanan kuda merupakan kasus khusus dari pencarian lintasan hamilton pada graf generik, yang termasuk NP-hard problem. [4]

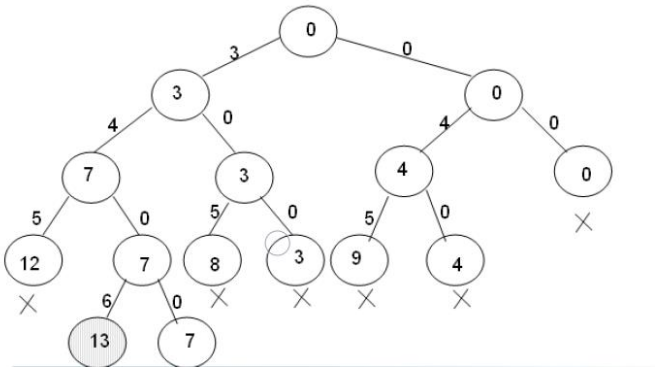
Rute perjalanan kuda dapat ditemukan untuk papan catur berukuran m > 4 dan n > 4. Telah diketahui pula algoritma pencarian lintasan kuda tersebut. Makalah ini akan berfokus pada algoritma *backtracking* serta algoritma 'heuristik' yang dikembangkan H.C Warnsdorff, yang hingga kini diketahui sebagai algoritma optimal untuk permasalahan ini.

II. ALGORITMA RUNUT-BALIK

Runut-balik (Backtracking) merupakan metode terstuktur dan sistematis dalam pencarian solusi dengan cara mengambil keputusan satu per satu hingga ditemukan solusi yang valid. Backtracking pertama kali diperkenalkan oleh D.H. Lehmer

pada tahun 1950. Kemudian diperjelas dan digeneralisasikan oleh R.J. Walker, Golomb, dan Baumert. Runut-balik berbasis pada DFS, ia merupakan perbaikan dari algoritma *brute-force*, sedemikian rupa sehingga kita hanya mempertimbangkan pencarian yang mengarah ke solusi saja, tidak harus memeriksa seluruh kemungkinan. Runut-balik lebih alami dinyatakan dalam algoritma rekursif. Kadang-kadang disebutkan pula bahwa runut-balik merupakan bentuk tipikal dari algoritma rekursif. [Diktat pak Rin]

Implementasi dari runut-balik digambarkan dengan cara menkontruksi sebuah pohon berdasarkan keputusan yang diambil, yang dikenal sebagai pohon ruang-status. Akar dari pohon tersebut merepresentasikan status awal seblum pencarian dimulai. Simpul pada aras pertama merepresentasikan keputusan yang diambil untuk komponen pertama untuk solusi, simpul pada aras kedua merepresentasikan keutusan yang diambil untuk komponen kedua solusi, dan seterusnya. [Levitin]



Gambar 3. Pohon Ruang-Status pada Metode Runut-Balik

Apabila suatu diketahui bahwa suatu simpul tidak mengarah ke solusi, maka simpul tersebut akan dimatikan. Karenanya, banyaknya pencarian akan berkurang secara signifikan, dan waktu pencarian solusi pun akan jadi lebih cepat.

Metode runut-balik memiliki beberapa properti umum :

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

$X = (x_1, x_2, ..., x_n)$
$x_i \in$ himpunan berhingga S_i.

xi merepresentasikan keputusan yang diambil pada langkah ke i, di mana x merupakan anggota dari himpunan S_i, yaitu seluruh kemungkinan langkah yang dapat diambil untuk langkah ke i. Himpunan langkah yang mungkin dapat saja sama untuk setiap langkahnya ($S_1 = S_2 = ... = S_n$).

2. Fungsi pembangkit nilai x_i
Fungsi pembangkit dinyatakan sebagai

T(k).

Fungsi tersebut membangitkan nilai x yang merupakan komponen dari vektor solusi pada setiap langkahnya.

3. Fungsi Pembatas
Fungsi pembatas dinyatakan sebagai

$B(x_{1, } x_{2, \dots , } x_{k)}$

Fungsi pembatas akan menentukan apakah solusi parsial yang dibangun oleh $(x_{1, } x_{2, \dots , } x_{k)}$ mengarah ke solusi. Jika ia mengarah ke solusi, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak, maka $(x_{1, } x_{2, \dots , } x_{k)}$ dibuang dan tidak akan dipertimbangkan lagi dalam pencarian solusi.

Pencarian solusi dengan metode runut-balik memiliki prinsip sebagai berikut :

- Solusi dicari dengan membentuk lintasan dari akan ke daun. Aturan yang dipakai adalah metode pencarian mendalam (DFS). Simpul yang telah dilahirkan dinamakan simpul hidup. Simpul hidung yang sedang diperluas dinamakan simpul-E. Simpul dinomori dari atas ke bawah sesuai urutan kelahirannya, sesuai dengan prinsip DFS.

- Setiap 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. Fungsi yang digunakan untuk membunuh simpul-E adalah fungsi pembatas. Simpul yang sudah mati tidak akan pernah diperluas lagi.

- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.

- Pencarian dihentikan bila telah ditemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

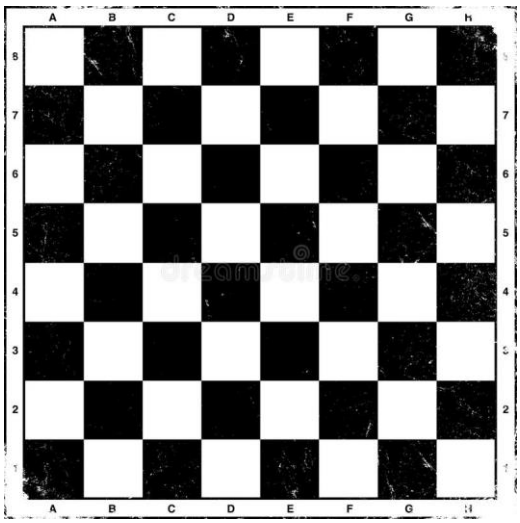
Kompleksitas algoritma untuk algoritma runut-balik mirip dengan algoritma *brute-force*, yakni untuk kasus terburuknya ia membutuhkan waktu dalam $O(p(n) 2^n)$ atau $O(q(n) n!)$, dengan p(n) dan q(n) adalah polinom berderajat n yang menyatakan waktu komputasi setiap simpul.

III. LINTASAN HAMILTON DAN BATASAN MASALAH PERMASALAHAN PERJALANAN KUDA

Lintasan Hamilton (dinamakan berdasarkan matematikan terkenal dari Irlandia, William Rowan Hamilton), adalah sekuens dari simpul-simpul v1, v2, ..., vn yang mengandung setiap simpul tepat satu kali, berlaku (vi, vi+1) £ E. Bila (vn, v1) juga terdapat di E, maka ia merupakan sirkuit hamilton. [3]

Euler, orang pertama yang menekuni Knight's Tour Problem, pernah mendemonstrasikan bahwa perjalanan kuda ini dengan papan catur standar. Ia juga yang menyajikan uraian umum bagi persoalan ini dengan menunjukkan bahwa ada ukuran papan lainnya yang dapat memiliki rute perjalanan kuda. Lalu, apakah papan catur dengan ukuran m x n, di mana m, n £ bilangan positif selalu memiliki lintasan kuda atau sirkuit kuda?

Jawabannya adalah tidak. Pertama, perhatikan bahwa papan catur memiliki dua warna, di mana setiap petak yang bersebelahan memiliki warna yang berbeda. Artinya bidak kuda, dengan gerakannya, diharuskan untuk bergerak ke petak tujuan dengan warna yang beda dengan petak asalnya. Maka, jika jumlah petaknya ganjil, tidak mungkin papan catur tersebut memiliki sirkuit kuda, karena ada warna yang lebih banyak dari warna lain. Tetapi hal ini belum menghilangkan kemungkinan adanya lintasan kuda.



Gambar 4. Papan Catur Standar

(sumber : https://thumbs.dreamstime.com/b/vintage-chess-board-vector-16184578.jpg, diakses tanggal 14 Mei 2018)

Suatu papan catur memiliki lintasan kuda apabila min(m,n) >= 5. Untuk lebih jelasnya, berikut dua teorema yang dituliskan oleh Paul Cull dan Jeffery de Curtins [3] :

Teorema 1 : Suatu papan catur berukuran n x m, nm genap, dan min(n, m) >= 5, memiliki sirkuit kuda

Teorema 2 : Setiap papan catur berukuran n x m, dan min(n,m) >= 5, memiliki lintasan

Selain kedua teorema itu, terdapat empat buah lemma juga yang mendukung pembuktian kedua teorema tersebut. Pun, pada makalah ini, kasus uji yang hendak digunakan memenuhi kedua teorema tersebut.

IV. ALGORITMA RUNUT-BALIK UNTUK PENYELESAIAN PERMASALAHAN PERJALANAN KUDA

Pertama, kita bahas terlebih dahulu algoritma naif, brute-force, yang pertama kali terpikirkan untuk menyelesaikan persoalanan Knight's tour ini. Algoritma naifnya adalah : membangkitkan seluruh lintasan yang mungkin satu per satu, kemudian memeriksa apakah lintasan tersebut memenuhi batasan-batasan yang berlaku.

Cara tersebut jelas tidaklah efisien. Pada papan berukuran 8 x 8, terdapat tepat 26.534.728.821.064 buah rute tertutup berarah (artinya, dua rute yang menempuh lintasan yang sama dengan arah terbalik dianggap berbeda). Untuk rute tertutup tak berarah, jumlahnya setengah dari angka tersebut. Dengan angka tersebut, tidak mungkin kita menerapkan algoritma naif tadi.

Maka, terdapat algoritma runut-balik yang membuat pencarian solusi yang lebih 'cepat' untuk persoalan Knight's tour. Runut-balik bekerja secara langkah-per-langkah untuk menyelesaikan masalah.

Sesuai dengan prinsipnya, algoritma runut-balik pada permasalahan perjalanan kuda memunkingkan bidak kuda untuk berjalan sejauh mungkin sampai ia menemukan jalan buntu, kemudian ia 'mundur' beberapa langkah dan mencoba jalan lain.

Berikut algoritma runut-balik untuk persoalan Knight's tour :

- Jika seluruh petak telah dikunjungi, maka cetak solusi
- Jika tidak,
    - Tambahkan satu langkah selanjutnya ke dalam vektor solusi dan secara rekursif, periksa apabila langkah tersebut mengarah ke solusi. (Bidak catur memiliki 8 gerakan yang mungkin. Kita memilih salah satu dari 8 gerakan tersebut pada tiap langkahnya).
    - Apabila langkah yang diambil tidak mengarah ke solusi, matikan langkah ini dari vektor solusi dan coba pilihan gerakan lainnya.
    - Jika tidak ada satu pun pilihan yang mengarah ke solusi, kembalikan false (mengembalikan false akan menghapus langkah sebelumnya lalu program akan melakukan runut-balik pada langkah tersebut, dan apabila false dikembalikan oleh pemanggilan awal, maka tidak ada solusi untuk kasus tersebut)

Maka berikut *pseudo-code* beserta property dari algoritma tersebut :

```
function isSafe(int x, int y, int sol[N][N]) -> boolean
/* fungsi pembantu yang memeriksa apakah i,j
merupakan indeks yang valid pada papan berukuran
N*N */

procedure printSolution(int sol[N][N])
/* fungsi pembantu untuk mencetak matriks solusi
sol[N][N] */

function solveKT() -> boolean
/* fungsi ini menyelesaikan persoalan Perjalanan
Kuda (the Knight Tour problem) dengan metode
runut-balik. Fungsi ini memanfaatkan fungsi
solveKTUtil() untuk menyelesaikan persoalan
tersebut. Jika tidak ada lintasan yang
memungkinkan, ia mengembalikan nilai false,
sebaliknya, ia akan mengembalikan true dan
mencetak lintasan kuda apabila terdapat solusi yang
mungkin.
Persoalan ini mungkin saja memiliki banyak solusi,
tetapi fungsi ini hanya mencetak salah satu solusi
saja. */
Deklarasi :
        sol : array[N][N] of integer
        xMove[8] = array[8] of integer = {  2, 1, -1,
-2, -2, -1,  1,  2 }
        yMove[8] = array[8] of integer = {  1, 2,  2,
1, -1, -2, -2, -1 };

Algoritma :
begin
        /* Inisialisasi seluruh isi matriks solusi
dengan -1 */
        /* Bidak kuda mulai melangkah dari petak
pojok kiri atas */
        sol[0][0] = 0
        /* Eksplorasi 'seluruh' rute perjalanan yang
mungkin */
        if (solveKTUtil(0, 0, 1, sol, xMove,
yMove) == false) then
        beginif
                output("Solusi tidak ada")
                return false
        endif
        else
                printSolution(sol)

        return true
end
```

```
function solveKTUtil(int x, int y, int movei, int
sol[N][N],
             int xMove[N], int yMove[N]) ->
integer
/* fungsi utilitas untuk menyelesaikan Knight
Tour Problem */
Deklarasi :
        int k, next_x, next_y : integer

Algoritma :
begin
        if (movei == N*N)
                return true

        /* coba seluruh gerakan yang mungkin
dilakukan dari koordinat saat ini (x, y) */
        for (k = 0; k < 8; k++)
  beginfor
     next_x = x + xMove[k];
     next_y = y + yMove[k];
     if (isSafe(next_x, next_y, sol))
     beginif
       sol[next_x][next_y] = movei;
       if (solveKTUtil(next_x, next_y, movei+1,
sol,
                xMove, yMove) == true)
          return true;
       else
          sol[next_x][next_y] = -1;//
backtracking
     endif
  endfor
  return false;
end
```

V. ALGORITMA WARNSDORFF

sebuah algoritma yang mencari lintasan tanpa melakukan runut-balik dengan cara menghitung nilai heuristik untuk 'penerus' langkah dari posisi saat ini. Maksud dari penerus di sini yakni petak-petak yang belum dikunjungi dan dapat dicapai dari posisi saat ini dalam satu langkah. Petak penerus dengan nilai heuristik tertinggi adalah petak yang memiliki penerus paling sedikit. Dengan cara ini, petak yang terisolasi cenderung untuk dikunjungi terlebih dahulu. Waktu yang dibutuhkan untuk menjalankan algoritma ini tumbuh secara linier dengan jumlah petak pada papan, sayangnya implementasi dari algoritma ini mencapai titik buntu pada papan catur berukuran lebih dari 76 x 76, walaupun ia berjalan dengan baik pada papan dengan ukuran yang lebih kecil. [wolfram]

Warnsdorff sendiri mencetuskan sebuah aturan, yang dikenal sebagai Warnsdorff's Rule [4]. Untuk mengkonstruksi sebuah lintasan kuda pada papan berukuran m kali m, maka :

Petak ke (n+1) pada lintasan (langkah yang diambil selanjutnya) merupakan petak yang :

1. Bertetanggaan/bersampingan dengan petak ke n.
2. Belum dikunjungi (belum ada pada lintasan solusi), dan
3. Memiliki petak tetangga yang belum dikunjungi paling sedikit

Apabila ada lebih dari satu petak yang memenuhi ketiga kondisi di atas, maka pilih secara acak, jika tidak ada yang memenuhi, maka hentikan pencarian.

Berikut cara kerja dari Algoritma Warnsdorff,

Misalkan :

- Posisi Q dapat diakses dari posisi P jika P dapat mencapai Q dalam satu langkah kuda, dan Q belum dikunjungi.
- Aksesabilitas posisi P adalah banyaknya posisi yang dapat dicapai dari P.

Algoritma :

- Set P sebagai posisi awal kuda pada papan, misal dari [0,0].
- Tandai papan pada posisi P dengan angka '1'
- Lakukan hal berikut untuk langkah ke 2 hingga langkah terakhir (banyaknya petak pada papan) :
    - Misalkan S merupakan himpunan posisi yang dapat diakses dari P.
    - Ubah P menjadi posisi pada S dengan aksesabilitas terkecil
    - Tandai P dengan nomor langkah saat ini
- Kembalikan papan (matriks solusi) yang telah ditandai, setiap petak pada papan memiliki angka sesuai urutan kunjungan kuda.

Berikut *pseudo-code* nya :

```
/* Algoritma Warnsdorff untuk penyelesaian
persoalan Knight Tour */

/* Kamus global */
cx = array of integer[N] = {1,1,2,2,-1,-1,-2,-2}
cy = array of integer[N] = {2,-2,1,-1,2,-2,1,-1}

function limits(int x, int y) -> boolean
/* memeriksa apakah petak pada [x,y] masih berada
di dalam papan catur berukuran 8 x 8 */

function isempty(int a[], int x, int y) -> boolean
/* memeriksa apakah petak belum dikunjungi dan
tidak berada di luar batas */

function getDegree(int a[], int x, int y) -> integer
/* Mengembalikan nilai heuristik, yaitu banyaknya
petak kosong yang bertetanggaan dengan (x, y) */

function nextMove(int a[], int *x, int *y) ->
boolean
/* Memilih langkah selanjutnya sesuai dengan
aturan heuristik Warnsdorff, mengembalikan
false apabila tidak ada langkah yang
memungkinkan */
Deklarasi :
min_deg_idx = integer = -1
c  = integer
min_deg = integer = N + 1
nx, ny = integer

Algoritma :
  /* periksa setiap petak yang dapat dicapai
oleh kuda dari (x, y)
   // Jika petak selanjutnya tidak ditemukan
   if (min_deg_idx == -1)
      return false;

   // menyimpan koordinat petak tujuan
   nx = *x + cx[min_deg_idx];
   ny = *y + cy[min_deg_idx];

   // tandai petak tujuan
   a[ny*N + nx] = a[(*y)*N + (*x)]+1;

   // perbarui posisi bidak
   *x = nx;
   *y = ny;

   return true;

function neighbour(int x, int y, int xx, int yy) ->
boolean
/* memeriksa petak-petak yang bertetanggan
dengan (x, y). Apabila bidak kuda dapat
menempuh petak awal dalam satu langkah,
maka hentikan pencarian */

procedure findClosedTour()
/* Menghasilkan lintasan kuda dengan
memanfaatkan algoritma heuristik Warnsdorff.
Mengembalikan false jika tidak memungkinkan
*/
```

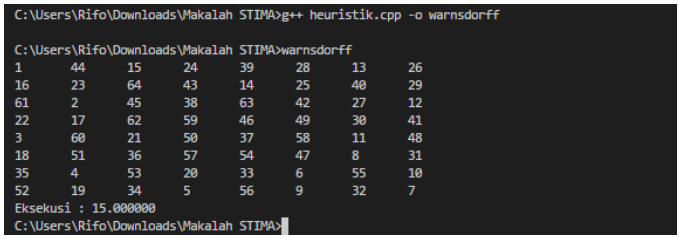
VI. KOMPARASI ALGORITMA RUNUT-BALIK DENGAN ALGORITMA WARNSDORFF

Penulis melakukan pengujian dengan menjalankan masing-masing algoritma, kemudian mencatat waktu eksekusinya. Ukuran papan yang digunakan sebagai kasus uji adalah 8 x 8 dan 10 x 10.

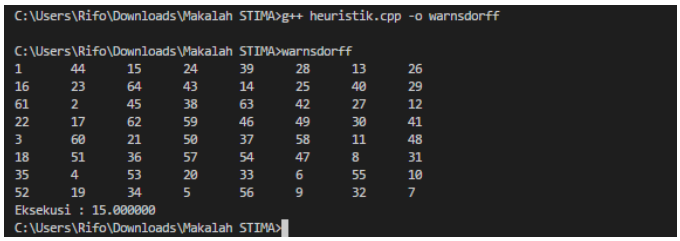
Berikut hasil yang didapat berdasarkan hasil pengujian

Ukuran papan	*Backtracking*	Warnsdorff
8 x 8	1000 ms	15 ms
10 x 10	>= 10000 ms	355 ms

Pada ukuran 10 x 10, hingga 10 detik, algoritma backtracking belum juga berhasil menemukan solusi. Berdasarkan data tersebut, kita dapati bahwa algoritma heuristic Warnsdorff jauh lebih cepat dibandingkan algoritma *backtracking*.



Gambar 5. Hasil pengujian algoritma Warnsdorff



Gambar 6. Hasil pengujian algoritma *Backtracking*

VII. KESIMPULAN

Algoritma Warnsdorff, menurut pengamatan penulis mirip dengan greedy best first search atau A* search, yakni memanfaatkan perhitungan heuristik untuk memutuskan langkah berikutnya.

Berdasarkan hasil pengujian, didapat bahwa algoritma Warnsdorff jauh lebih mangkus dan sangkil dibandingkan algoritma backtracking. Bahkan kompleksitas waktunya cenderung linear dengan banyaknya petak pada papan catur.

UCAPAN TERIMA KASIH

Penulis senantiasa bersyukur kepada Allah swt, yang atas rahmatnya, penulis dapat menyelesaikan makalah ini. Tentunya penulis juga mengucapkan terima kasih kepada pak Rinaldi Munir selaku dosen pengajar mata kuliah strategi algoritma, serta kawan-kawan dari prodi Informatika angkatan 2016 yang baik langsung maupun tidak langsung telah membantu dalam penyusunan makalah ini

REFERENSI

[1] Parberry I., “An efficient algorithm for Knight’s Tour Problem”, in Discrete Applied Mathematics, pp. 251-260, 1997

[2] Lin S and Wei C. , “Optimal algorithms for constructing knight’s tours on arbitrary n×m chessboards”, in Discrete Applied Mathematics, pp. 219-232, 2005

[3] Cull P. and Curtins J.D., “Knight’s Tour Revisited”, June 1978

[4] Squirrel D., “A Warnsdorff-Rule Algorithm for Knight's Tour on Square Chessboards”, August 1996

[5] Munir, Rinaldi, 2009, Diktat Kuliah IF2211 Strategi Algoritma, Bandung: Penerbit Informatika.

[6] Levitin, Anany, “Introduction to The Design & Analysis of Algorithms”, 3rd ed. , Villanova University, Penssy Ivania : 2012, Pearson.

[7] Weisstein, Eric W. "Knight Graph." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/KnightGraph.html, diakses pada 14 Mei 2018

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, 14 Mei 2018

Rifo Ahmad Genadi
13516111