Graf Hamilton pada Permainan *Knight's Tour* dan Pemecahan dengan Algoritma *Divide-and-Conquer* dan *Bactrack*

Muhammad Farhan Kemal – 13513085
Program studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
farhankemal@students.itb.ac.id

Abstrak*—Permainan catur sudah eksis sangat lama dan sejak dahulu kala. Permainan strategi ini dahulunya dibuat untuk memenuhi hasrat manusia yang terus berpikir. Dari eksistensinya yang sudah lama tersebut dan latar belakang manusia sebagia makluk yang berakal pikiran, catur kini bukan hanya sebagai sebuah permainan sederhana. Catur kini sudah dipertandingkan dan dimodifikasi ke dalam bentuk-bentuk aturan permainan lain yang sangat banyak variannya. Salah satunya dalam bentuk permainan *knight's tour* atau perjalanan kuda. Inti dari permainan ini adalah bagaimana seorang pemain dengan sebuah bidak catur kuda mampu melewati semua kotak dalam papan catur dengan hanya melewati setiap kotak hanya satu kali. Permainan yang awal mula dimainkan pada sekitaran tahun 1700-an ini sangat mendapat perhatian masyarakat Eropa saat itu tentang bagaimana solusi dari permainan terebut. Tercatat Leonhard Euler yang pertama kali menyelesaikan permainan ini secara matematis di tahun 1759. Saat ini telah banyak dikembangkan cara menyelesaikan permainan *knight's tour* ini, bukan hanya pada papan catur standar (8x8), tetapi juga dengan berbagai macam papan, bahkan dengan papan dengan ukuran mxn sekalipun. salah satu cara penyelesaian permainan ini adalah dengan cara algoritma *divide-and-conquer.

Keywords*— knight's tour, graf Hamilton, algoritma *divide-and-conquer.

I. PENDAHULUAN

Catur, sebuah permainan strategi satu lawan satu adalah sebuah permainan klasik yang diyakini berasal dari India dan ditemukan pada abad ke-6 dikenal sebagai *Chaturanga* yang berarti empat divisi terpisah. Dahulu di India catur dimainkan dengan 4 orang pemain dan mengambil basis di setiap sisi.

Gambar 1. Satu game permainan catur (id.wikipedia.org)

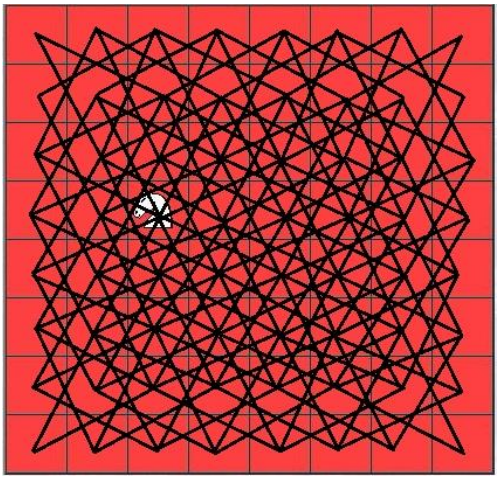
Namun sekarang dalam permainan catur modern, pemain direduksi menjadi satu lawan satu dengan setiap pemain memiliki identitas warna pada bidak masing-masing pemain. Aturan permainannya adalah masing masing-pemain diberikan 6 macam bidak catur yang bisa digerakkan pada papan kotak-kotak hitam-putih berukuran 8x8 kotak. Masing masing bidak memiliki langkah yang unik. Permainan berakhir ketika salah satu dari pemain kehilangan bidak rajanya atau kedua pemain tidak mungkin membunuh bidak king lawan satu sama lain atau yang lebih dikenal sebagai keadaan remis.

Dalam sejarah catur bangsa Eropa telah banyak mengembangkan permainan catur ini, antara lain dengan membuat papan caturnya berwarna hitam dan putih. Ini terjadi kira-kira abad-10. Sebelumnya, kotak-kotak itu berwarna sama. Malah sering orang membuat arena permainan catur ini di atas pasir atau di mana saja yang bisa diberi garis. Dari Eropa ini juga dibuat peraturan bahwa pion boleh maju dua kotak pada langkah pertama dan menteri (ratu) boleh bergerak lebih leluasa baik maju ke depan maupun diagonal.

Perlahan catur mengalami perkembangan. Dari nama, bentuk, serta peraturan permainannya. Kesemuanya itu mewakili simbol perubahan peradaban. Salah satu bentuk perkembangan tersebut adalah bentuk permainannya yang semakin bervariasi dan lebih membutuhkan logika berpikir. Salah satu Bentuk permainan yang bervariasi ini adalah permainan *Knight's Tour*. Permainan ini pada dasarnya adalah seorang pemain diberikan satu buah bidak kuda dan pemain tersebut harus menggerakkan bidak tersebut ke semua kotak di papan dengan hanya singgah di setiap kotak hanya satu kali.

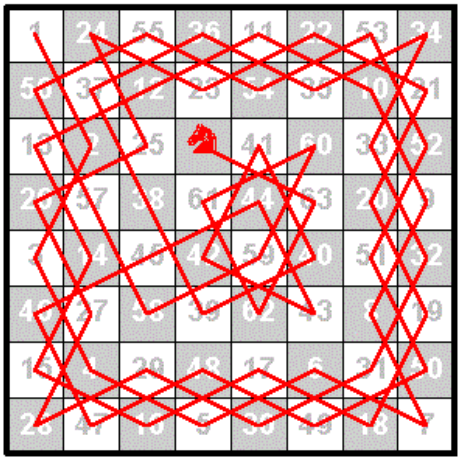
II. KNIGHT'S TOUR

Permainan catur telah banyak menghasilkan modifikasi permainan, salah satunya adalah permainan *Knight's Tour* atau yang dalam bahasa Indonesia dikenal sebagai Perjalanan Kuda. Permainan ini dimainkan oleh satu orang pemain dengan menggunakan sebuah bidak kuda. Bidak kuda bisa dibilang bidak dengan langkah paling unik disbanding bidak-bidak lainnya. Secara umum bidak pada permainan catur hanya mampu bergerak secara horizontal, vertical, atau secara diagonal. Namun tidak pada bidak kudak. Bidak ini bergerak membentuk huruf "L" dengan perbandingan alas L dan tinggi L adalah 2:3.



Gambar 2. Visualisasi segala kemungkinan pergerakan kuda di papan catur (koleksi pribadi)

Aturan dari permainan ini sangat sederhana. Seorang pemain harus menggerakkan bidak kudanya ke semua kotak pada papan dengan menyinggahi setiap kotak hanya boleh satu kali saja. Ada dua macam kondisi terpenuhi oleh pemain ketika berhasil menyelesaikan permainan ini, yaitu *Closed Tour* dan *Open Tour*. *Closed tour* kondisi ketika kotak awal pergerakan kuda adalah kotak yang sama ketika ia telah menyinggahi semua kotak. *Open tour* adalah ketikan kotak awal pergerakan tidak sama dengan kotak akhir.

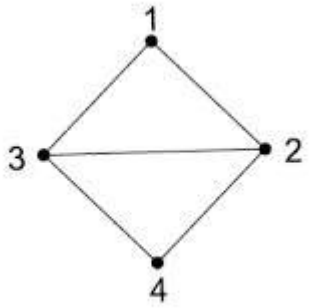


Gambar 3. Salah satu contoh open tour (Layton.wikia.com)

Permainan ini berkembang di Eropa sekitar abad ke-18. Permainan ini banyak menarik perhatian para ilmuwan saat itu karena tingkat kesulitannya yang tinggi dan perlu pendalaman logika serta penyelesaian secara matematis yang terstruktur. Pada tahun 1759, seorang matematikawan asal Swiss, Leonhard Euler. Setelah teori graf ditemukan oleh Sir William Rowan Hamilton, pergerakan bidak kuda dalam papan catur digambarkan oleh sebuah graf dengan setiap kotak pada papan digambarkan oleh sebuah simpul dan perpindahan kuda dari satu kotak ke kotak lain digambarkan oleh sebuah sisi.

III. GRAF

Graf adalah himpunan tidak kosong yang beranggotakan himpunan tidak kosong yang berisi simpul-simpul (*vertices*) dan himpunan yang berisi sisi yang menghubungkan simpul-simpul. Dari definisi di atas, suatu graf dapat hanya berbentuk sebuah simpul tanpa ada sisi, atau banyak simpul tanpa dihubungkan oleh sisi namun graf tidak dapat dikatan sebuah graf tanpa ada simpul sama sekali

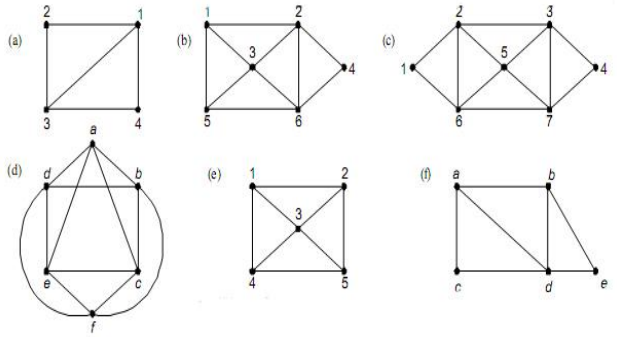


Gambar 4. Sebuah graf sederhana (koleksi pribadi)

III. A. GRAF EULER

Graf Euler adalah graf yang melalui masing-masing sisi graf tepat satu kali dan kembali ke simpul asal, gaf ini

dinamakan juga sebagai sirkuit Euler. Lintasan Euler adalah lintasan yang melalui masing-masing sisi graf tepat satu kali namun tidak kembali ke simpul asal. Lintasan Euler juga dapat disebut graf semi-Euler.



Gambar 5. (a) dan (b) grafsemi-Euler, (c) dan (d) graf Euler , (e) dan (f) bukan graf semi-Euler atau graf Euler

Suatu graf tidak berarah dikatakan memiliki lintasan Euler jika dan hanya jika terhubung dan memiliki dua buah simpul berderajat ganjil atau tidak ada simpul berderajat ganjil sama sekali. Graf tidak berarah *G* adalah graf Euler (memiliki sirkuit Euler) jika dan hanya jika setiap simpul berderajat genap.

III. B. GRAF HAMILTON

Graf Hamilton hampir sama dengan graf Euler. Namun perbedaannya terletak pada graf Hamilton melewati satu simpul tepat satu kali. Jadi, lintasan Hamilton adalah lintasan yang melalui yiap simpul pada graf tepat satu kali. Sedangkan sirkuit Hamilon adalah lintasan Hamilton yang kembali ke simpul awal.

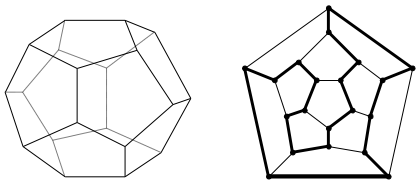
Untuk menentukan apakah suatu graf merupakan graf hamilton atau tidak, digunakan teorema Ore. Bunyi Teorema Ore adalah sebagai berikut:

"misal G(V,E) adalah graf dengan $|V| \geq 3$, dan setiap pasangan simpul tak terhubung *u* dan *v* pada G

$\deg(u) + \deg(v) \geq M$,

untuk M bilangan integer. Jika M sama dengan |V| maka G adalah graf Hamilton."
Beberapa teorema lain terkait graf Hamilton yaitu:

TEOREMA "Setiap gaf lengkap adalah graf hamilton"
TEOREMA "Didalam graf lengkap G dengan n simpul ($n \geq 3$), terdapat (n-1)!/2 buah sirkuit Hamilton."



Gambar 6. Dodecahedron Hamilton, Dan graf yang mengandung sirkuit Hamilton (Diktat Matematika Diskrit Rinaldi Munir)

Berdasarkan pengertian dari lintasan dan sirkuit Hamilton di atas, maka dapat kita artikan bahwa *Closed Tour* dari permainan *knight's tour* adalah sebuah sirkuit Hamilton dan *Open Tour* merupakan lintasan Hamilton.

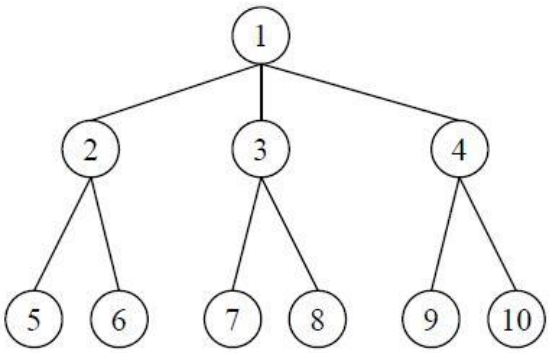
Sebuah lintasan Hamilton dapat dibentuk dari beberapa simpul menggunakan beberapa macam algoritma, diantaranya algoritma *backtracking* dan algoritma *divide-and-conquer*.

IV. ALGORITMA *BACKTRACKING*

Secara umum kita dapat menentukan langkah yang akan di ambil dengan mencoba segala kemungkinan solusi atau yang dikenal sebagai *brute-force.* Namun jika mencoba semua kemungkinan solusi, misalnya kita menggunakan papan catur 8x8 kotak, maka total kemungkinan solusi adalah sebanyak n!, dengan n=64 atau sama dengan 1.2688693×10^{89}. Tentu cara ini bukanlah cara yang efisien untuk menyelesaikan persoalan kita. Di dalam (64!) kemungkinan tersebut, sebenarnya banyak kemungkinan yang tidak kita butuhkan atau tidak mungkin dicapai saat kita berada dalam suatu kondisi. Untuk itu kita perlu membuang kemungkinan yang sedang tidak kita butuhkan tersebut. Salah satu algoritma yang bersifat seperti itu adalah algoritma *backtracking*.

Algoritma backtracking mempunyai prinsip dasar yang sama seperti brute-force yaitu mencoba segala kemungkinan solusi. Perbedaan utamanya adalah pada ide dasarnya, semua solusi dibuat dalam bentuk pohon solusi (pohon ini tentunya berbentuk abstrak) dan algoritma akan menelusuri pohon tersebut secara DFS (depth field search) sampai ditemukan solusi yang layak.

Misalkan sebuah persoalan divisualisasikan dalam sebuah pohon seprti pada pohon di bawah:



Misalkan pohon diatas menggambarkan solusi dari suatu permasalahan. Untuk mecapai solusi (5), maka jalan yang ditempuh adalah (1,2,5), demikian juga dengan solusi solusi yang lain. Algoritma backtrack akan memeriksa mulai dari solusi yang pertama yaitu solusi (5). Jika ternyata solusi (5) bukan solusi yang layak maka algoritma akan melanjutkan ke solusi (6). Jalan yang ditempuh ke solusi (5) adalah (1,2,5) dan jalan untuk ke solusi (6) adalah (1,2,6). Kedua solusi ini memiliki jalan awal yang sama yaitu (1,2). Jadi daripada memeriksa ulang dari (1) kemudian (2) maka hasil (1,2) disimpan dan langsung memeriksa solusi (6). Pada pohon yang lebih rumit, cara ini akan jauh lebih efisien daripada brute-force.

Pada beberapa kasus, hasil perhitungan sebelumnya harus disimpan, sedangkan pada kasus yang lainnya tidak perlu.

Pada persoalan *Knight's Tour* papan NxN misalkan kotak awal dari pergerakan bidak kuda adalah akar dari pohon keputusan langkah bidak tersebut. Secara umum persyaratan penelusuran yang disetujui adalah ketika dari suatu node ke node lain tidak ada siklus yang terbentuk. Jika pada perjalanan pohon terbentuk sebuah siklus artinya kita telah melewati suatu kotak yang sudah pernah kita lewati sebelumnya. Persyaratan ini terdapat pengecualian jika terjadi sisi dari daun menuju akar dan tinggi pohon sudah mencapai N. dalam sebuah pseudo code digambarkan sebagai berikut:

```
type chess_board is array
      (1..n,1..n) of integer;
procedure knight (board : in out
      chess_board;
      x,y,move : in out integer;
      ok : in out Boolean) is
      w, z : integer;

begin
      if move = n^2+1 then
            ok := ( (x,y) =
            (1,1) );
      elsif board(x,y) /= 0 then
            ok := false;
      else
            board(x,y) := move;
            loop
            (w,z) := Next position
                  from (x,y);
            knight(board, w, z,
                  move+1, ok );
            exit when (ok or No
            moves remain);
            end loop;
            if not ok then
                  board
                        ( x,y ) :=
                        0;
                  Backtracking
            end if;
      end if;
end knight;
```

Atau dalam bahasa C, dapat diimplementasikan sebagai berikut:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

typedef unsigned char cell;
int dx[] = { -2, -2, -1, 1, 2,  2,  1, -1 };
int dy[] = { -1,  1,  2, 2, 1, -1, -2, -2 };

void init_board(int w, int h, cell **a, cell **b)
{
	int i, j, k, x, y, p = w + 4, q = h + 4;
	/* b is board; a is board with 2 rows
padded at each side */
	a[0] = (cell*)(a + q);
	b[0] = a[0] + 2;

	for (i = 1; i < q; i++) {
		a[i] = a[i-1] + p;
		b[i] = a[i] + 2;
	}

	memset(a[0], 255, p * q);
	for (i = 0; i < h; i++) {
		for (j = 0; j < w; j++) {
			for (k = 0; k < 8; k++) {
				x = j + dx[k], y
= i + dy[k];
				if (b[i+2][j] ==
255) b[i+2][j] = 0;
				b[i+2][j] += x
>= 0 && x < w && y >= 0 && y < h;
			}
		}
	}
}

#define E "\033["
int walk_board(int w, int h, int x, int y, cell **b)
{
	int i, nx, ny, least;
	int steps = 0;
	printf(E"H"E"J"E"%d;%dH"E"32m[]"E"
m", y + 1, 1 + 2 * x);

	while (1) {
		/* occupy cell */
		b[y][x] = 255;
		/* reduce all neighbors' neighbor
count */
		for (i = 0; i < 8; i++)
			b[ y + dy[i] ][ x + dx[i]
]--;

		/* find neighbor with lowest
neighbor count */
	least = 255;
		for (i = 0; i < 8; i++) {
if (b[ y + dy[i] ][ x + dx[i] ] < least) {
				nx = x +
dx[i];
				ny = y +
dy[i];
				least =
b[ny][nx];
			}
		}

		if (least > 7) {
			printf(E"%dH", h
+ 2);
			return steps == w *
h - 1;
		}

		if (steps++)
printf(E"%d;%dH[]", y + 1, 1 + 2 * x);
		x = nx, y = ny;

	printf(E"%d;%dH"E"31m[]"E"m", y
+ 1, 1 + 2 * x);
		fflush(stdout);
		usleep(120000);
	}
}

int solve(int w, int h)
{
	int x = 0, y = 0;
	cell **a, **b;
	a = malloc((w + 4) * (h + 4) +
sizeof(cell*) * (h + 4));
	b = malloc((h + 4) * sizeof(cell*));

	while (1) {
		init_board(w, h, a, b);
		if (walk_board(w, h, x, y, b
+ 2)) {

	printf("Success!\n");
			return 1;
		}
		if (++x >= w) x = 0, y++;
		if (y >= h) {
			printf("Failed to
find a solution\n");
			return 0;
		}
		printf("Any key to try next
start position");
		getchar();
	}
}

int main(int c, char **v)
```

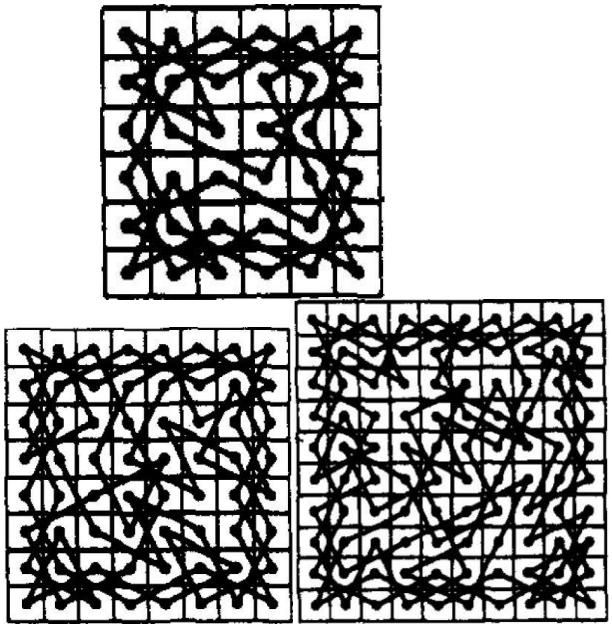
```
{
        int w, h;
        if (c < 2 || (w = atoi(v[1])) <= 0) w
= 8;
        if (c < 3 || (h = atoi(v[2])) <= 0) h
= w;
        solve(w, h);

        return 0;
}
```

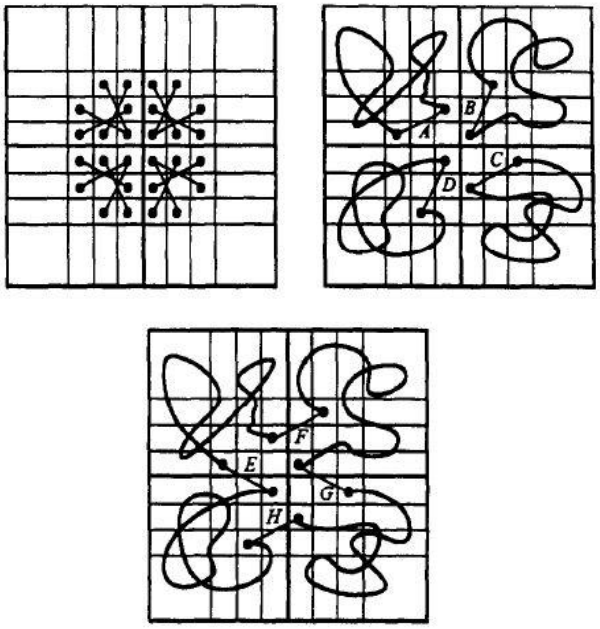
V. ALGORITMA *DIVIDE-AND-CONQUER*

Algoritma *divide-and-conquer* adalah jenis algoritma yang menyelesaikan permasalahan dengan cara membagi permasalahan tersebut ke dalam bentuk yang lebih kecil dan lebih mudah untuk diselesaikan. Algoritma ini sangat membantu untuk memecahkan persoalan *Knight's Tour* yang menggunakan papan lebih besar dari 10x10 dengan N genap.

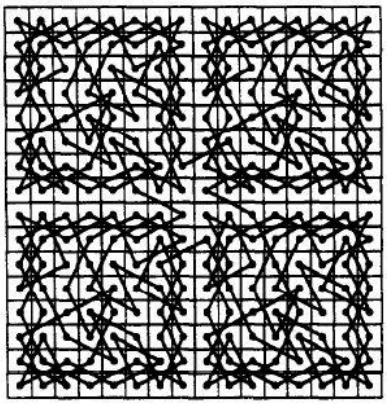
Penyelesaiannya adalah dengan cara membagi papan ke dalam 4 buah kuadran. Basis persoalan yang digunakan adalah papan 6x6, 8x8, 10x10 yang dapat dicari dengan mudah dengan menggunakan algoritma *backtracking.* Saat sudah dilakukan backtracking, terdapat cara menyambungkan kuadran-kuadran tersebut yang divisualisasikan ke dalam gambar berikut:



Gambar 7 basis yang digunakan dalam algoritma, papn 6x6, 8x8, 10x10 (Ian Parbery, Discrete Applied Mathematics)



Gambar 8, cara menghubungkan basis (Ian Parbery, Discrete Applied Mathematics)



Gambar 9 contoh penyelesaian pada pana 16x169(Ian Parbery, Discrete Applied Mathematics)

VI. KESIMPULAN

Banyak contoh graf Hamilton yang bisa kita temui dalam kehidupan antara lain penerapan dalam permainan *Knight's Tour* atau pada persoalan tukan pos yang harus melakukan pekerjaannya seefisien mungkin dengan tidak menyinggahi tempat yang sama berkali-kali.

Langkah-langkah dalam membentuk graf Hamilton pada permainan *knight's tour* dapat dibuat lebih mudah dengan menggunakan algoritma *backtracking*. *Backtracking* terbukti lebih mangkus daripada algoritma lain dalam menyelesaikan *knight's tour* karena algoritma ini membuang segala kemungkinan yang bukan merupakan solusi dari permainan. Berbeda dengan

faktorial yang akan melakukan pengecekan terhadap setia kemungkinan yang ada. Untuk menyelesaikan persoalan graf Hamilton yang sangat kompleks atau yang sangat besar maka kita perlu memecahnya menjadi beberapa bagian agar mudah untuk diselesaikan.

VII. REFERNCES

[1] Parberry, Ian. *"Discrete Applied Mathematics"* . NH Elsiever, Texas : 1997.

[2]. Munir, Rinaldi, "Matematika Diskrit", Informatika, Bandung: 2010.

[3] http://rosettacode.org/wiki/Knight%27s_tour#Haskell diakses pada tanggal 10 Desember 2014

[4] http://www.academia.edu/4904020/Backtracking diakses pada tanggal 10 Desember 2014

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, 10 Desember 2013

Muhammad Farhan Kemal