Penyelesaian *n-Queens Problem* dengan Metode *Recursive Backtracking*

Rifo Ahmad Genadi 13516111
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516111@std.stei.itb.ac.id

***Abstrak*— Catur adalah permainan yang tak lekang oleh waktu. Bagaimana tidak? Permainan ini sudah ada sejak abad ke-6 Masehi dan masih banyak dimainkan hingga saat ini. Kini, kita dapat menjumpai permainan papan ini dalam bentuk fisiknya maupun dalam bentuk digitalnya. Dalam aturan main aslinya, permainan ini dimainkan oleh dua orang, tujuan dari setiap pemain adalah 'membunuh' bidak raja milik lawannya. Selain aturan main aslinya, dapat kita jumpai berbagai teka-teki yang berdasarkan pada permainan catur ini, salah satunya adalah *n-queen problem.* Singkat cerita, kita harus meletakkan n buah ratu pada papan catur berukuran n x n, sehingga tidak ada dua ratu yang saling menyerang. Makalah ini akan menjelaskan bagaimana cara mendapatkan solusi tersebut dengan metode *recursive backtracking*.**

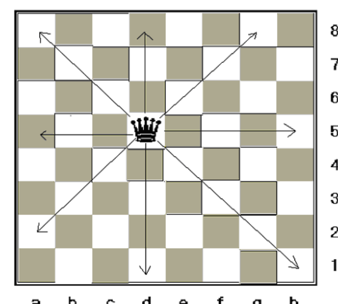
***Kata kunci*— *n-queens problem,* rekursif, *backtrack*, algoritma, teka-teki catur.**

I. PENDAHULUAN

Teka-teki N-Queens menantang kita untuk menempatkan n buah ratu pada sebuah papan catur berukuran n x n sedemikian rupa sehingga tidak ada ratu yang saling menyerang. Teka-teki ini pertama kali dicetuskan oleh tahun 1848, dengan kasus yang lebih khusus, yaitu *Eight Queens Puzzle,* oleh seorang pemain catur dari Jerman bernama M. Bezzel.pada bukunya yang berjudul *Berliner Schachzeitung*.. [5]

Gambar 1 : Papan catur berukuran 8x8
(Sumber : https://images.chesscomfiles.com/uploads/v1/landing/259.69d5ae42.300x300o.ab57f1eacf3a.jpg diakses pada 3 Desember 2017, pukul 18:32 GMT+7)

Lalu mengapa teka-teki-nya adalah n-queens? Bukan n-rooks, n-kings, n-bishops, atau yang lainnya?
Seperti yang kita tahu, bidak ratu adalah bidak yang 'terkuat pada permainan catur, hal ini dikarenakan gerakannya yang lebih fleksibel dibandingkan dengan jenis bidak lainnya. Ia dapat bergerak secara horizontal seperti benteng, dan juga bergerak secara diagonal seperti menteri.



Gambar 2 : Gerakan bidak ratu
(Sumber : http://www.chesscourse.com/images/beginner_2_5.gif diakses pada 3 Desember 2017, pukul 19:32 GMT+7)

Franz Nauck merupakan orang yang pertama kali menyelesaikan teka-teki ini, ia menemukan 92 solusi dari teka-teki ini pada tahun 1850, sekaligus mengembangkan permasalah ini menjadi lebih umum, yaitu *n-queens problem*.

Sejak 1850 pula, banyak matematikawan yang mempelajari teka-teki ini, baik *the eight queens problem,* maupun versi lebih umumnya, *n-queen problems,* termasuk Carl Freidrich Gauss, dan Edsger Dijkstra. Berbagai algoritma komputer juga dikembangkan untuk menyelesaikan teka-teki ini, mulai dari yang paling naif : metode *brute-force* (mencoba semua ${}_{64}C_8$ kemungkinan), menggunakan operasi *bitwise,* pendekatan-pendekatan dengan *constraint programming, logic programming,* atau *genetic programming*. Lalu, yang akan dibahas dalam masalah ini, yaitu dengan memanfaatkan algoritma rekursif dan teknik *backtracking*.

II. LANDASAN TEORI

2.1 Kombinatorika

A. Permutasi

Sebuah permutasi adalah penyusunan objek-objek yang ada pada suatu himpunan. Urutuan penyusunan r buah elemen dari sebuah himpunan dengan n buah elemen dinamakan permutasi-r, dilambangkan dengan P(n,r). [1]

Contoh :
Misalkan S = {1,2,3}. Susunan 3,1,2 adalah permutasi dari S. Sedangkan susunan 3,2 adalah permutasi-2 dari S.

Jika n dan r adalah bilangan bulat, dan 0 <= r <= n. Maka :

$$P(n, r) = \frac{n!}{(n-r)!}$$

B. Kombinasi

Kombinasi adalah bentuk khusus dari permutasi. Pada kombinasi, urutan kemunculan dari elemen diabaikan. Maka, sederhananya kombinasi-r dari elemen-elemen sebuah himpunan adalah *subset* dari himpunan dengan elemen sebanyak r [1].

Banyaknya kombinasi-r sebuah himpunan dengan n buah elemen, dengan n adalah sebuah bilangan bulat tidak negatif dan r adalah bilangan bulat, dan 0 <= r <= n, adalah :

$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

2.2 Rekursif

Definisi rekursi adalah pendefinisian suatu objek pada suatu himpunan dalam terminologi elemen lainnya pada himpunan tersebut [3].

Pada computer science, suatu metode memiliki sifat rekursif apabila ia dapat didefinisikan oleh dua bagian berikut :
1. Basis
2. Rekurens atau sekumpulan aturan yang mereduksi seluruh kasus lainnya menuju basis.

Ada pula lelucon pada kamus berikut yang merupakan 'definisi' dari rekursif [4]

Rekursi
Lihat bagian 'Rekursi'.

Barisan Fibonacci merupakan contoh klasik dari rekursi :

Fib(0) = 0 (Basis).
Fib(1) = 1 (Basis)
Fib (n) = Fib(n-1) + Fib(n-2), untuk semua bilangan bulat > 1 (definisi rekursif).

Pada *computer science*, suatu algoritma dikatakan rekursif apabila ia menyelesaikan masalah dengan cara mereduksinya masalah tersebut dengan masalah yang sama, namun dengan masukan yang lebih kecil. [1]

Selain di matematika dan ilmu komputer, rekursi juga diterapkan pada seni, contohnya pada Boneka Matryoshka dan lukisan *Stefaneschi Tryptych* karya Giotto.

Gambar 3 : Boneka Matryoshka
(Sumber : http://russian-crafts.com/images/nesting-dolls/nesting-doll-first.jpg, diakses pada 3 Desember pukul 22:14 GMT +7)

2.3 *Backtracking*

Backtracking adalah algoritma umum untuk mencari sebagian atau seluruh solusi dari suatu persoalan, khususnya persoalan yang solusinya harus memenuhi kondisi-kondisi tertentu, dengan cara memeriksa setiap kandidat solusi, dan mengabaikan setiap kandidat parsial ("*backtrack*") setelah kandidat tersebut ditetapkan tidak dapat memenuhi kondisi yang memenuhi solusi. Sederhananya, ia merupakan versi lebih baik dari pendekatan secara *brute force*. [6][7]

Teknik backtracking dapat diterapkan untuk menyelesaikan teka-teki seperti sudoku, scrabble, aritematika verbal, dan banyak lagi. Ia juga merupakan teknik yang efektif untuk melakukan parsing, menyelesaikan knapsack problem, dan persoalan optimisasi kombinatorial lainnya.

2.4 Kompleksitas Algoritma

A. Algoritma

Algoritma merupakan sekumpulan urutan intruksi untuk melakukan suatu perhitungan atau menyelesaikan persoalan. [1]

Suatu algoritma yang baik ditentukan oleh berapa jumlah waktu dan ruang memori yang dibutuhkan untuk menjalankannya. Kebutuhan waktu sebuah algoritma biasanya dihitung dalam satuan detik, mikrodetik, dan sebagainya, sedangkan ruang memori yang dibutuhkannya diukur dalam satuan *byte, kilobyte,* dan sebagainya.

Suatu algoritma dikatakan efisien atau mangkus apabila ia dapat meminimalkan waktu eksekusi serta ruang memorinya [2] .

B. Kompleksitas Algoritma

Kompleksitas algoritma merupakan besaran yang dipakai untuk menentukan tingkat keefektifan suatu algoritma, terlepas dari mesin dan jenis *compiler* yang digunakan.

Terdapat dua macam kompleksitas algoritma, yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu, T(n) merupakan banyaknya eksekusi perhitungan saat menjalankan algoritma sebagai fungsi dari banyaknya masukan *n*. Kompleksitas ruang, S(n) adalah jumlah memori yang digunakan oleh struktur data yang digunakan untuk algoritma sebagai fungsi dari banyaknya masukan *n* [2].

Kompleksitas waktu sendiri dibedakan menjadi tiga macam, yaitu :

- $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk.
- $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik.
- $T_{avg}(n)$: kompleksitas waktui untuk kasus rata-rata.

C. Kompleksitas Waktu Asimtotik

Biasanya, kita tidak perlu mengetahui kompleksitas waktu yang presisi, tapi kita cukup tau gambaran kasar kebutuhan waktu dari suatu algoritma, dan seberapa cepat fungsi tersebut tumbuh.

Perbedaan kinerja suatu algoritma baru terlihat saat n yang menjadi masukannya sangat besar, misalnya kita bandingkan kedua algoritma dengan kompleksitas waktu sebagai berikut :

$T_1(n) = n^2 + 3n$
$T_2(n) = n + 1$

n	$T_1(n)$	$T_2(n)$
10	130	11
100	10.300	101
1000	1.003.000	1001
10.000	100.030.000	10.001

Tabel 1 : Perbandingan kompleksitas waktu algoritma

Berdasarkan tabel tersebut, dapat kita lihat perbedaan kedua algoritma tersebut sangatlah signifikan. Selain itu, dapat kita lihat bahwa $T_1(n)$ tumbuh bersamaan dengan fungsi n^2, sehingga suku-suku lainnya yang tidak mendominasi dapat kita abaikan.

Maka dapat kita katakana bahwa $T_1(n)$ berorde n^2 dan dapat dituliskan

$$T(n) = O(n^2)$$

Notasi O tersebut dinamakan notasi *Big-O*, yang definisi formalnya adalah sebagai berikut :

T(n) = O(f(n)), artinya T(n) berorde paling besar f(n), bila terdapat konstanta C dan n_0 sedemikian sehingga :

$$T(n) <= C.f(n)$$

$$\text{untuk } n >= n0$$

Maknanya, apabila sebuah algoritma mempunnyai waktu asimtotik O(f(n)), jika n dibuat semakin besar, maka waktu yang dibutuhkannya tidak akan melebihi konstanta C dikalikan f(n). Perlu dicatat bahwa C serta n_0 yang dipakai tidaklah unik, ada banyak kemungkinan yang memenuhinya.

Berikut pengelompokkan algoritma berdasarkan notasi *Big-O :*

Algoritma Polinomial :

- O(1), berarti waktu eksekusi algoritma adalah tetap, tidak bergantung pada banyaknya masukan. Walaupun *n* nya dijadikan dua kali semula, waktu eksekusinya hanyalah bergantung pada konstanta saja.

- O(log n), artinya laju pertumbuhannya tumbuh bersamaan dengan fungsi logaritmik. Fungsi log n baru meningkat menjadi dua kali semula saat *n* nya dinaikkan menjadi n^2 kali semua.

- O(n), artinya waktu eksekusi algoritmanya tumbuh secara lanjar, bila *n* dinaikkan menjadi dua kali semula, maka waktu pelaksanaannya pun naik menjadi dua kali semula.

- O(n log n), waktu pelaksanaannya lebih lambat O(n), namun masih lebih cepat dibandingkan $O(n^2)$, Bila n = 1000, maka n log n kurang lebih 20.000. Sedangkan bila n dijadikan 2 kali semula, maka n log n pun sedikit lebih kecil dari 2 kali semulanya.

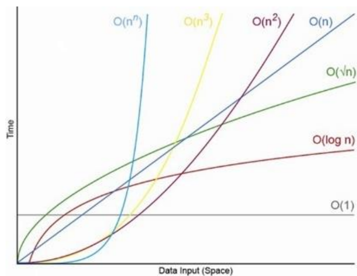
- $O(n^2)$, pertumbuhan waktu pelaksanaannya meningkat secara kuadratik, apabila inputnya dua kali semula, maka waktu pelaksanaannya menjadi empat kali semula. Biasanya pada algoritmanya terdapat dua pengulangan bersarang (*nested loops).*

- $O(n^3)$, pertumbuhan waktu pelaksanaannya meningkat secara kubik, seperti algoritma kuadratik, biasanya pada algoritmanya terdapat tiga buah pengulangan bersarang. Apabila masukannya dijadikan dua kali semula, maka waktu eksekusinya menjadi delapan kali semula.

Algoritma eksponensial

- $O(2^n)$, algoritma eksponensial pertumbuhannya jauh lebih cepat daripada algoritma polynomial, biasanya algoritma *brute force* ('nguli') memiliki kompleksitas ini (walaupun biasanya idenya lebih mudah ditemukan). Apabila *n* dua kali semula, waktu eksekusinya menjadi kuadrat kali semula.

- O(n!), algoritma dengan kompleksitas factorial memproses setiap masukan dan menghubungkannya dengan n-1 masukan lainnya. Apabila *n* dijadikan dua kali lipat, maka waktu eksekusinya menjadi faktorial dari 2n.



Gambar 4 : Laju pertumbuhan fungsi kompleksitas waktu
(Sumber : https://camo.githubusercontent.com/874181d7b840a494fe94a11cc13c1fad5d372217/68747470733a2f2f6170656c6261756d2e66696c65732e776f726470726573732e636f6d2f323031312f31302f796161636f766170656c6261756d6269676f706c6f742e6a7067, diakses pada 4 Desember pukul 01:21 GMT+7)

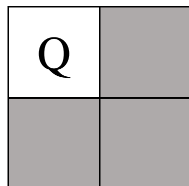
III. Analisis dan Pembahasan

A. Batasan masalah

Sekilas, dapat kita lihat bahwa permasalahan ini tidak memiliki untuk semua *n* (di mana *n* adalah bilangan bulat positif). Pengecualiannya adalah saat n = 2 dan n = 3.

Cukup mudah untuk membuktikan bahwa mustahil menemukan solusi saat n = 2 dan n = 3.

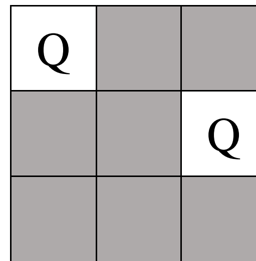
Kita lihat untuk n = 2, maka ukuran papannya adalah 2 x 2. Misal kita taruh satu bidak ratu di petak sembarang, maka seperti yang terlihat pada gambar 5, ratu pertama yang kita taruh itu dapat menjangkau setiap petak lainnya pada peta.

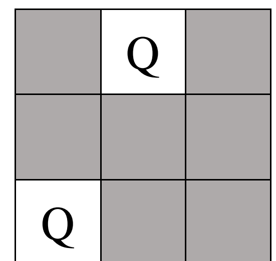


Gambar 5 : *n-queens problem* dengan n = 2

Sedangkan, untuk n = 3, pertama kita taruh bidak ratu pertama kita di suatu kolom. Maka, untuk kolom selanjutnya, hanya ada satu petak yang dapat ditempati oleh bidak ratu kedua. Kemudian, bidak ratu kedua itu akan ‘menutup’ satu petak yang tersisa pada kolom terakhir, sehingga tidak ada ruang lagi bagi bidak ratu yang ketiga.

Bahkan, ada kasus di mana tidak ada lagi petak yang tersisa pada suatu kolom setelah diletakkan satu buah bidak ratu.





Gambar 6 : *n-queens problem* dengan n = 3.

Sedangkan, untuk n = 1, permasalahan ini dapat diselesaikan, namun hanya terdapat satu solusi unik saja. Lalu, untuk n = 4, n = 5, dan selanjutnya, dapat ditemukan setidaknya satu solusi.

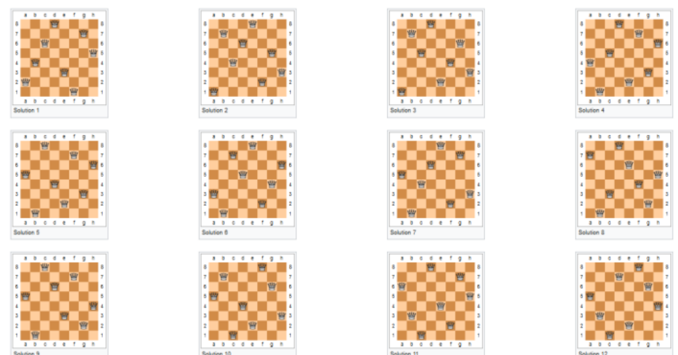
Maka, yang akan dibahas pada makalah ini adalah *n-queens problem* dengan n >= 4. Lalu, yang akan dibahas adalah permasalahan *n-queens* dengan papan berbentuk standar (matriks), bukan bentuk lainnya seperti toroida (pembahasan *n-queens problem* dengan papan berbentuk toroida dapat ditemukan di literatur lain).

B. *Eight-Queens Problem*

Mula-mula, kita tinjau versi klasik dari permasalahan ini, yaitu menggunakan ukuran asli papan catur (dimensi 8 x 8), berarti n = 8.Teka-teki ini pertama kali dipecahkan oleh Franz Nauck, ia menemukan 92 solusi dari permasalahan ini. Carl Freidrich Gauss pun menyatakan bahwa memang ada 92 solusi untuk teka-teki ini.

Tetapi, dari 92 solusi itu, apabila kita menghitung solusi yang hanya merupakan simetri dari solusi lainnya sebagai satu, maka hanya terdapat 12 buah solusi dasar saja.

Pada setiap solusi dasar umumnya terdapat delapan buah varian solusi, yaitu dengan memutarnya 90°, 180°, atau 270^0, kemudian merefleksikan keempat varian tersebut terhadap bagian tengah papan. Lalu, dari dua belas buah solusi dasar tersebut, hanya ada tepat satu buah solusi yang mana bila dirotasikan 180° ia sama seperti semula. Maka, terdapat 11 x 8 + 1 x 4 = 92 buah solusi.



Gambar 7 : 12 Solusi dasar *eight-queens problem*
(Sumber : https://en.wikipedia.org/wiki/Eight_queens_puzzle)

C. Algoritma Penyelesaian

Solusi paling mudah yang terpikir (dan yang paling naif) untuk menyelesaikan *eight-queens problem* adalah mendaftar seluruh kombinasi yang mungkin, yaitu memilih 8 petak dari 8 x 8 = 64 petak yang ada pada papan catur, kemudian memeriksa apakah 8 bidak ratu dapat diletakkan tanpa posisi tersebut tanpa masalah . Maka, akan ada ${}_{64}C_8$ (sekitar 4 triliun) kemungkinan. Algoritma *brute-force* tersebut masih memungkinkan untuk papan berukuran 8 x 8, tetapi saat n >= 20, ia akan menjadi persoalan *intractable*, karena 20! = 2.433×10^{18}.

Lalu, perlu kita sadari bahwa setiap kolom hanya boleh ada tepat satu buah bidak ratu. Maka, kini hanya ada 8^8 (sekitar 17 juta) kemungkinan. Jauh lebih baik dari algoritma sebelumnya, tetapi mari kita tinjau lebih lanjut lagi.

Kita tahu bahwa tidak boleh ada dua ratu yang menempati baris *atau* kolom yang sama. Maka, sekarang kemungkinan yang tersisa adalah 8! (sekitar 40 ribu) buah saja.

Kemudian, ada satu batasan lagi : tidak boleh ada dua ratu pada garis diagonal yang sama. Misalkan ratu A berada di posisi (i, j) dan ratu B berada di posisi (k, l). Maka, abs(i – k) tidak boleh sama dengan abs(j-l).

$$\text{Abs}(i-k) \text{ != } \text{abs}(j-l) \quad [8]$$

Algoritma *recursive backtracking* untuk menyelesaikan *eight-queens problem* menempatkan ratu satu-per-satu pada kolom 0 sampai 7, dan mengecek seluruh kondisi di atas. Cara yang sama juga berlaku bagi versi yang lebih umum : *n-queens problem*.

Berikut *pseudo-code* dari algoritma *recursive backtracking* yang digunakan :

1) Mulai dari kolom paling kiri
2) Jika semua ratu telah ditempatkan, kembalikan true
3) Periksa seluruh baris pada kolom yang saat ini ditinjau. Lakukan hal berikut untuk setiap baris yang diperiksa :
   a) Jika ratu dapat dengan aman pada baris ini, maka tandai [baris,kolom] sebagai bagian dari solusi, dan periksa secara rekursif bahwa menempatkan ratu pada petak ini merupakan solusi yang *valid*.
   b) Jika penempatan ratu di [baris, kolom] merupakan solusi yang valid, kembalikan true.
   c) Jika penempatan ratu bukan merupakan solusi yang valid, maka hapus tanda [baris, kolom] yang dibuat pada langkah a (*backtrack*) dan kembali ke langkah a dengan baris yang lain.
4) Jika seluruh baris telah dicoba dan tidak ditemukan solusi yang *valid*, maka kembalikan *false* untuk memicu *backtrackking*.

Berikut implementasi dari *pseudo-code* tersebut dalam bahasa C [9]

```
#define N 4
#include<stdio.h>

/* mencetak solusi ke layar */
void printSolution(int board[N][N])
{
    static int k = 1;
    printf("%d-\n",k++);
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
    printf("\n");
}

/* Fungsi bantuan untuk memeriksa apakah ratu dapat
diletakkan pada [row][col]. */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* memeriksa bagian kiri baris saat ini */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* memeriksa bagian kiri diagonal atas */
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;

    /* memeriksa bagian kiri diagonal bawah */
    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

/* Fungsi rekursif untuk mencari solusi */
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: jika semua ratu sudah diletakkan,
kembalikan true */
    if (col == N )
    {
        printSolution(board);
        return true;
    }

    /* Periksa kolom ini */
    for (int i = 0; i < N; i++)
    {
        /* periksa apakah ratu dapat diletakkan di
        board[i][col] */
        if ( isSafe(board, i, col) )
        {
            /* letakkan ratu pada board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            solveNQUtil(board, col + 1) ;

            // below commented statement is replaced
            // by above one
            /* if ( solveNQUtil(board, col + 1) )
                return true;*/

           /* jika gagal, hapus ratu dari
       board[i][col] */
            board[i][col] = 0; // BACKTRACK
        }
    }
```

```
    /* jika ratu tidak dapat diletakkan di baris mana
pun, kembalikan false */
    return false;
}

void solveNQ()
{
    int board[N][N] = { {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
    };

    if (solveNQUtil(board, 0))
    {
        printf("Solusi tidak ada");
        return ;
    }

    return ;
}

int main()
{
    solveNQ();
    return 0;
}
```

D. Kompleksitas Algoritma *Recursive Backtracking* untuk menyelesaikan *n-queens* problem

Sesuai dengan batasan-batasan yang telah dibahas pada bagian C, banyaknya kemungkinan yang perlu diperiksa adalah sebanyak n!. Maka, kompleksitasnya adalah O(n!).

V. Kesimpulan

Ada banyak teka-teki 'berbasis' permainan catur. Salah satu yang teka-teki yang menantang adalah *n-queen problems* ini. Algoritma *recursive backtracking* di atas digunakan untuk menampilkan seluruh solusi *n-queens problem* sesuai dengan *n* yang masukan. Dapat kita lihat kompleksitas algoritma tersebut termasuk kompleksitas eksponensial, maka eksekusinya akan jauh lebih lama untuk *n* yang lebih besar.

Sebenarnya, untuk mencari satu solusi saja, ada cara lain, yaitu dengan mencari solusi eksplisit. Selain itu, secara matematis, ada cara-cara lain untuk mendapatkan solusinya, ada yang menggunakan metode determinan matriks, pendekatan dengan teori graf, dan lain-lain.

VI. Ucapan Terima Kasih

Pertama-tama, penulis mengucapkan syukur kepada Tuhan Yang Maha atas segala nikmat yang telah diberika sehingga penulis dapat menyelesaikan makalah ini. Penulis juga ingin mengucapkan terima kasih kepada dosen mata kuliah Matematika Diskrit, bapak Rinaldi Munir, ibu Harlili, dan khususnya kepada bapak Judhi Santoso yang telah mengajar kelas K03 selama satu semester ini. Penulis juga ingin mengucapkan terima kasih kepada keluarga penulis yang membuat motivasi penulis selalu terjaga. Terakhir, penulis ingin mengucapkan terima kasih kepada seluruh pihak yang telah berkontribusi dalam pengerjaan makalah ini, baik secara langsung maupun tidak langsung.

Referensi

[1] Rosen, K.H., *Discrete Mathematics and Its Application*, New York: McGraw-Hill, 2012, edisi ketujuh.
[2] Munir, Rinaldi. *Matematika Diskrit*, Bandung: Informatika, 2012, edisi kelima
[3] Azcel, *An Introduction to Inductive Definitions*, North-Holland, 1977
[4] http://catb.org/~esr/jargon/html/R/recursion.html, diakses tanggal 3 Desember 2017, pukul 19:36 GMT+7.
[5] E. J. Hoffman et al., "Construction for the Solutions of the m Queens Problem". Mathematics Magazine, Vol. XX (1969), pp. 66–72.
[6] Donald E. Knuth (1968). The Art of Computer Programming. Addison-Wesley.
[7] Gurari, Eitan (1999). "CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms". Archived from the original on 17 March 2007
[8] Halim, Steven, *Competitive Programming 3*, Singapore, 2013, 3rd edition.
[9] http://www.geeksforgeeks.org/printing-solutions-n-queen-problem/ diakses tanggal 4 Desember 2017 pukul 04:26 GMT+7.

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, 3 Desember 2017

Rifo Ahmad Genadi 13516111