Algoritma Runut-Balik untuk Menyelesaikan Persoalan *n*-Ratu

Faiz Ghifari Haznitrama 13515010
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
faizghifari@students.itb.ac.id

***Abstrak*—Dalam permainan catur, terdapat sebuah persoalan menarik untuk meletakkan bidak ratu pada papan catur. Jika kemudian terdapat papan catur dengan ukuran n x n, maka yang menjadi persoalan adalah bagaimana kita meletakkan sejumlah n bidak ratu pada papan catur tersebut tanpa ada ratu yang saling menjangkau ratu lainnya. Persoalan ini menjadi menarik karena solusi untuk persoalan ini sangat terbatas dan diperlukan pendekatan tertentu untuk menyelesaikannya. Salah satu pendekatan yang bisa digunakan untuk menyelesaikan persoalan ini adalah algoritma runut-balik yang berbasis pada algoritma *depth-first search*.**

***Kata Kunci*—Catur, Bidak Ratu, Runut-Balik,**

I. PENDAHULUAN

Permainan catur merupakan salah satu permainan tua yang masih eksis hingga saat ini. Kemudahan dan fleksibilitas untuk memainkannya membuatnya terus memiliki tempat special dikalangan para penggemarnya. Selain daripada kemudahan dalam memainkan catur, implementasi catur dalam kehidupan sehari-hari pun mempunyai daya tarik yang begitu besar. Mulai dari penggunaan strategi perang hingga filosofi-filosofi kehidupan yang sangat bermakna dan memberikan dampak positif bagi kehidupan

Kehadiran catur sebagai permainan yang menarik ternyata juga menimbulkan adanya persoalan-persoalan unik yang tidak bisa diselesaikan dengan cara biasa. Persoalan-persoalan ini merupakan salah satu bentuk permodelan masalah yang ada di kehidupan sehari-hari sehingga dalam bentuk penyelesaian persoalan dapat digunakan pula untuk memecahkan masalah yang memiliki model yang sama. Salah satu yang menjadi daya tarik adalah persoalan *n*-ratu.

Bidak ratu merupakan bidak yang paling menarik dan fleksibel diantara bidak-bidak catur lainnya. Ia dapat bergerak secara vertikal, horizontal, dan diagonal tanpa batasan jarak selama memang petak yang dituju tidak terhalang. Oleh karena itu, dalam sebuah papan catur berukuran n x n , akan menjadi sebuah persoalan unik untuk meletakkan sejumlah n ratu pada papan catur tanpa ada ratu yang dapat menjangkau satu sama lainnya

Salah satu metode penyelesaian untuk persoalan ini adalah algoritma runut-balik. Algoritma ini pada dasarnya merupakan lanjutan daripada algoritma *depth-first search*. Inti utama daripada algoritma ini adalah kita dapat melakukan *backtrack* atau balik ke simpul sebelumnya ketika kita mendapati sebuah simpul yang mati atau tidak menuju ke solusi.

Sangat menarik untuk melihat bagaimana implementasi algoritma runut-balik ini pada persoalan *n*-ratu. Mengingat jika memang algoritma runut-balik bisa digunakan untuk menyelesaikan persoanalan *n*-ratu ini, maka algoritma runut-balik juga dapat digunakan untuk persoalan-persoalan sejenis.

II. DASAR TEORI

A. *Algoritma Runut-Balik*

Algoritma runut-balik atau yang akrab disebut dengan *backtracking* merupakan salah satu metode pendekatan penyelesaian masalah dengan mengacu pada aturan DFS.

Pertama kali diperkenalkan oleh Derrick H. Lehmerpada tahun 1950, kemudian dikembangkan lebih lanjut oleh Robert J. Walker pada tahun 1960 untuk menjadi *backtracking* yang berbasis pada aturan *depth-first*.

Pada hakikatnya meskipun algoritma runut-balik mengikuti aturan *depth-first*, sejatinya algoritma runut-balik adalah hasil dari perbaikan yang dilakukan pada algoritma *exhaustive search*. Jika pada algoritma *exhaustive search* semua kemungkinan solusi akan diperiksa seluruhna satu persatu, pada algoritma runut-balik dilakukan optimasi sehingga seluruh rangkaian pilihan yang tidak mengarah pada solusi yang diinginkan tidak dieksplorasi lebih jauh. Sehingga kemungkinan-kemungkinan pilihan yang dieksplorasi dapat dipastikan mengarah pada solusi dan mempunyai kemungkinan untuk menemukan solusi.

Algoritma runut-balik banyak digunakan untuk menyelesaikan persoalan yang terkait dengan kecerdasan buatan, utamanya pada permainan. Berikut merupakan beberapa contoh persoalan yang dapat diselesaikan dengan algoritma runut-balik;

1. *Tic Tac Toe*;
2. permainan *maze* (sejenis labirin);
3. catur;
4. serta permainan kecerdasan buatan lainnya

Kemudian seperti yang telah disebutkan sebelumnya, pada tahun 1960 Robert J. Walker bersama dengan Golomb dan Baumert merumuskan beberapa properti umum dari algoritma runut-balik;

1. Solusi persoalan

   Solusi dapat dinyatakan sebagai vektor dengan *n-tuple*:
   $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$.
   Mungkin saja $S_1 = S_2 = \ldots = S_n$.

2. Fungsi pembangkit

   Dapat dinyatakan sebagai:
   $T(k)$
   Dengan $T(k)$ adalah sebuah fungsi untuk membangkitkan nilai dari x_k , dimana x_k adalah komponen dari vektor solusi X

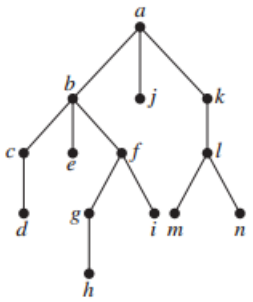
3. Fungsi pembatas

   Dapat dinyatakan sebagai:
   $B(x_1, x_2, .. , x_k)$

   B akan bernilai benar jika $(x_1, x_2, .. , x_k)$ mengarah ke solusi (masih memungkinkan untuk benar). Jika benar maka akan dibangkitkan lagi nilai untuk x_{k+1} (pembangkitan simpul dilanjutkan).

   Tetapi jika B salah, artinya $(x_1, x_2, .. , x_k)$ telah tidak memenuhi syarat untuk solusi (tidak mungkin menghasilkan solusi karena sudah salah), maka $(x_1, x_2, .. , x_k)$ akan dibuang dan tidak dilanjutkan lagi

Proses pengorganisasian solusi dengan algoritma runut-balik di organisasikan ke dalam struktur *tree* (pohon). Sebuah pohon merupakan sekumpulan simpul-simpul yang terhubung dengan busur namun tidak memiliki sirkuit (kumpulan simpul dan busur yang sirkular). Terdapat beberapa jenis pohon yang dapat digunakan, salah satu yang paling mudah adalah pohon berakar.



Gambar 2.1 Pohon Berakar
Sumber: *Discrete Mathematics and Its Applications, Kenneth H. Rosen, Page 753*

Pohon berakar yang digunakan pada algoritma runut-balik disebut sebagai pohon ruang status. Sebagai bentuk pengorganisasian ruang solusi dari persoalan. Terdapat beberapa tiga macam simpul pada pohon ini:

1. Simpul akar

   Simpul akar adalah sebuah simpul yang memiliki derajat-masuk sama dengan nol. Artinya akar merupakan permulaan dari sebuah pohon dan hanya mempunyai sisi yang keluar dari simpul. Pada gambar 2.1diibaratkan dengan simpul a.

2. Simpul daun (*goal*)

   Simpul yang mempunyai derajat nol (tidak mempunyai anak) disebut dengan daun. Merupakan simpul paling ujung pada pohon. Pada algoritma runut-balik, daun diibaratkan sebagai sebuah *goal*. Pada gambar 2.1 diibaratkan dengan simpul d, h, i, j, m, dan n.

3. Simpul dalam (*internal nodes*)

   Simpul pada pohon yang mempunyai anak dan bukan akar merupakan simpul dalam. Simpul dalam merupakan simpul yang harus ditempuh untuk mencapai sebuah *goal* (simpul daun). Pada gambar 2.1 diibaratkan dengan simpul b, c, e, f, g, k, dan l.

Tiap simpul pada pohon menyatakan sebuah kondisi atau status pada persoalan. Status ini yang kemudian akan diperiksa apakah status tersebut masih mengarah kepada solusi atau tidak. Kemudian busur-busur yang menghubungkan simpul diberi label dengan nilai-nilai x_i yang dibangkitkan.

Lintasan dari simpul akar ke daun yang

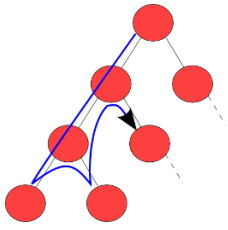
memungkinkan menyatakan sebuah solusi yang mungkin. Maka seluruh lintasan dari akar ke daun pada pohon akan membentuk ruang solusi. Algoritma runut-balik pada pohon ini dapat diibaratkan sebagai proses pencarian sebuah lintasan dari simpul akar ke simpul daun tertentu.

Prinsip dari pencarian solusi dengan algoritma runut-balik adalah dengan mencari lintasan dari simpul akar ke simpul daun dengan mengikuti aturan *depth-first*. Setiap simpul yang telah dikunjungi atau dilahirkan disimpan dan ditandai sebagai sebuah **simpul hidup**. Kemudian simpul hidup yang sedang diperluas atau diekspansi dinamakan sebagai **simpul-E**. Sekali lagi penomoran simpul mengikuti aturan DFS, artinya penomoran simpul yang dikunjungi berdasarkan urutan kelahiran simpul tersebut.

Ketika sedang melakukan perluasan dari simpul-E, tentunya lintasan yang terbentuk akan bertambah panjang. Disinilah kemudian perluasan yang dilakukan pada simpul-E diperiksa, apakah lintasan yang terbentuk mengarah kepada solusi yang diinginkan atau tidak. Jika tidak, maka fungsi pembatas akan membunuh simpul-E sehingga menjadi **simpul mati**. Simpul yang telah dinyatakan mati tidak akan diperluas lagi karena sudah dipastikan bahwa simpul tersebut tidak mengarah ke solusi.

Kemudian jika pencarian berujung pada simpul mati, maka proses pencarian solusi dilanjutkan ke simpul anak lainnya (disebut *siblings* atau saudara) lalu dilakukan perluasan dan pemeriksaan kembali seperti tahap sebelumnya. Jika tidak ada lagi simpul anak yang dapat diperluas, maka proses dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat dan menjadi simpul-E yang baru. Simpul sebelumnya yang seluruh anaknya tidak mengarah ke solusi juga dimatikan menjadi simpul mati menggunakan fungsi pembatas.

Pencarian solusi baru dihentikan ketika sebuah solusi telah ditemukan atau ketika tidak ada lagi simpul hidup yang dapat dijadikan acuan melakukan runut-balik.



Gambar 2.2 Illustrasi Runut-Balik
Sumber: http://ivandemarino.me/2010/01/22/use-backtracking-to-print-all-subsets/

Berbicara mengenai kompleksitas dari algoritma runut-balik, tentunya algoritma runut-balik memiliki kompleksitas yang lebih baik daripada algoritma *exhaustive search*. Dikarenakan pada algoritma runut-balik seluruh simpul yang tidak mengarah kepada solusi persoalan tidak diperluas lebih jauh. Kemudian pada penentuan kompleksitas algoritma runut-balik pencarian solusi juga bergantung pada keputusan pengguna. Apakah akan melakukan ekspansi terus menerus hingga tidak ada lagi simpul hidup yang dapat diperluas atau berhenti ketika solusi pertama berhasil ditemukan.

Secara umum kompleksitas algoritma runut-balik dapat didefinisikan sebagai berikut:

$$O(p(n)2^n) \text{ atau } O(q(n)n!)$$

Dengan jumlah simpul pada pohon ruang status adalah 2^n atau $n!$ dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul. Kompleksitas algoritma runut-balik diatas adalah dalam kasus terburuk.

Skema umum dari algoritma runut-balik adalah sebagai berikut:

```
procedure RunutBalik(input k:integer)
{
@variabel input k: indeks vektor solusi x[k]
keluaran solusi x = (x[1],x[2],..,x[n])
}

Algoritma
        for tiap x[k] yang belum dicoba
        sedemikian sehingga (x[k] <- T(k)) and
        B(x[1],x[2],..,x[k]) = true do
        begin
          if (x[1],x[2],..,x[n]) adalah
          lintasan dari akar ke daun then
                CetakSolusi(x)
          else
                RunutBalik(k+1)
        endfor
```

Sumber: *Strategi Algoritma, Rinaldi Munir*

Pada skema umum tersebut, terlihat bahwa terjadi pengulangan sebagai bentuk ekspansi simpul dan pencarian lintasan dari akar ke daun. Ketika memang lintasan yang didapat sudah mencapai daun dan memenuhi syarat solusi, maka kemudian solusi akan dicetak ke layar

III. PENGGUNAAN ALGORITMA RUNUT-BALIK DALAM PENCARIAN SOLUSI PERSOALAN N-RATU

A. Penerapan Algoritma Runut-Balik

Untuk persoalan kali ini, kami mencoba untuk langsung menerapkan algoritma runut-balik untuk menyelesaikan persoalan *n*-ratu dengan *n* yang digunakan adalah 5 dan pencarian solusi dihentikan ketika solusi pertama berhasil ditemukan.

Berikut merupakan skema algoritma runut-balik yang digunakan untuk menyelesaikan persoalan *n*-ratu:

```
function Tempat(input k:integer)->boolean
{
Memeriksa apakah ratu dapat ditempatkan pada kolom
x[k]
}

KAMUS LOKAL
  i : integer
  stop : boolean

ALGORITMA
  kedudukan<-true {asumsi awal x[k] dapat
ditempati}
  I <- 1
  stop <- false
  While (i<k) and (not stop) do
    if (x[i] = x[k])
       or
    (ABS(x[i]-x[k]) = ABS(i-k))
    then
       kedudukan <- false
       keluar <- true
    else
       i <- i+1
    endif
  endwhile

  return kedudukan
```

```
procedure N_RATU_R(input k:integer)
{
  Menempatkan ratu pada bari ke-k pada papan catur
  N x N tanpa melanggar aturan
  Masukan: N = jumlah ratu
  Keluaran: semua solusi x = (x[1],x[2],..,x[N])
  dicetak ke layar
}

KAMUS LOKAL
  stop : Boolean

ALGORITMA
  stop<-false
  while not stop do
    x[k] <- x[k+1]
    while (x[k] <= n) and (not Tempat(k)) do
      x[k] <- x[k+1]
    endwhile

  if (x[k] <= N) then
     if (k = N) then
       CetakSolusi(x,N)
     else
       N_RATU_R(k+1)
  else
    stop <- true
    x[k] <- 0
  endwhile
```

Sumber: *Strategi Algoritma, Rinaldi Munir*

Pada fungsi pertama yaitu Tempat, dilakukan pemeriksaan terhadap sebuah ratu apakah ia dapat menempati kolom x[k], dimana k disini bertindak sebagai parameter fungsi dengan *pass by value*. Fungsi tempat akan mengembalikan sebuah kondisi apakah ratu dapat ditempatkan di kolom x[k] atau tidak. Pada properti umum algoritma runut-balik, fungsi tempat ini berfungsi sebagai pembatas yang mengecek suatu simpul apakah simpul tersebut masih mengarah ke solusi atau tidak.

Fungsi tempat ini kemudian akan digunakan secara berulang pada prosedur N_RATU_R untuk melakukan pemeriksaan ketika penempatan ratu dilakukan. Pemanggilan fungsi tempat ini menjamin tidak ada ratu yang melanggar aturan ketika ditempatkan pada papan catur.

Kemudian masih pada prosedur N_RATU_R , ketika sebuah ratu sudah berhasil ditempatkan dengan benar, maka selanjutnya dilakukan pemeriksaan apakah memang semua ratu sudah ditempatkan pada tempat yang seharusnya atau baru sebagian saja. Jika semua sudah ditempati dengan baik (diidentifikasi dengan k = N) maka solusi akan dicetak ke layar. Jika belum, maka pada skema ini digunakan skema rekursif dengan memanggil kembali prosedur N_RATU_R dengan parameter input ditambah dengan 1 yang menandakan pemanggilan proses penempatan ratu selanjutnya. Pada prosedur N_RATU_R inilah proses runut-balik dilakukan. Ketika memang sebuah ratu tidak dapat diletakkan di kolom manapun, maka nilai x[k] dengan k adalah indeks ratu yang tidak dapat ditempatkan di kolom manapun diisi dengan nilai 0. Kemudian nilai x[k] sama dengan 0 inilah yang menjadi tanda harus dilakukannya runut-balik ke simpul hidup terdekat.

Pada program utama, yang perlu dilakukan hanyalah melakukan inisialisasi seluruh vektor solusi x = (x[1],x[2],..,x[N]) dengan 0. Lalu baru melakukan pemanggilan terhadap prosedur N_RATU_R dengan parameter input dimulai dari indeks pertama vektor solusi. Skema program utama adalah sebagai berikut:

```
program NRatuProblem

KAMUS
  I : integer

ALGORITMA
  for i<-N to n do
    x[i]<-0
  endfor

  N_RATU_R(1)
```

Sumber: *Strategi Algoritma, Rinaldi Munir*

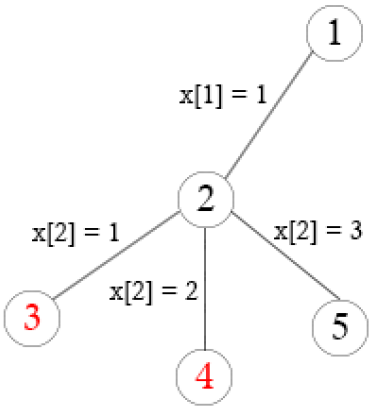
Untuk persoalan *n*-ratu dengan n = 5, maka langkah pertama yang dilakukan adalah menentukan simpul akar

yang menjadi acuan utama dalam memulai pencarian solusi. Illustrasi dari papan catur berukuran 5 x 5 sebagai tempat penempatan ratu adalah sebagai berikut:

1	2	3	4	5	
					x[1]
					x[2]
					x[3]
					x[4]
					x[5]

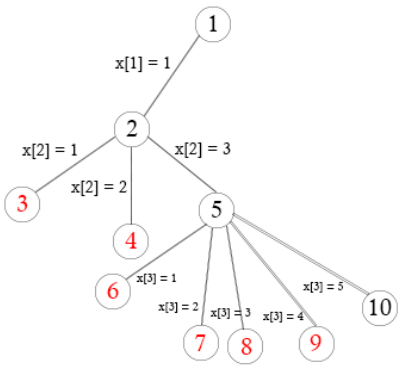
Gambar 3.1 Illustrasi Papan Catur Ukuran 5x5

Simpul akar dimulai dari penempatan x[1], yang diimplementasikan dengan pemanggilan N_RATU_R(1) pada program utama. Kemudian ratu x[1] mulai ditempatkan dari kolom ke-1, dimana tentu ini tidak melanggar fungsi pembatas. Lalu simpul tersebut menjadi simpul-E yang diperluas lebih jauh dengan berbagai kemungkinan dari penempatan ratu x[2].



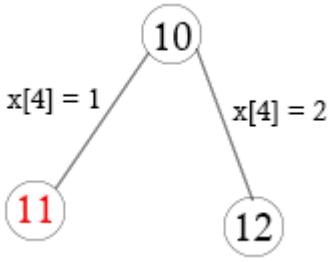
Gambar 3.2 Hasil Ekspansi Simpul ke-2 dengan Ratu x[1] pada Kolom ke-1

Terlihat dari ekspansi simpul ke-2 dimana ratu x[1] ditempatkan pada kolom ke-1, ditemukan bahwa penempatan ratu x[2] yang tidak melanggar fungsi pembatas adalah pada kolom ke-3. Maka selanjutnya simpul ke-5 menjadi simpul-E dan akan diperluas kembali kepada seluruh simpul anak dari simpul ke-5.



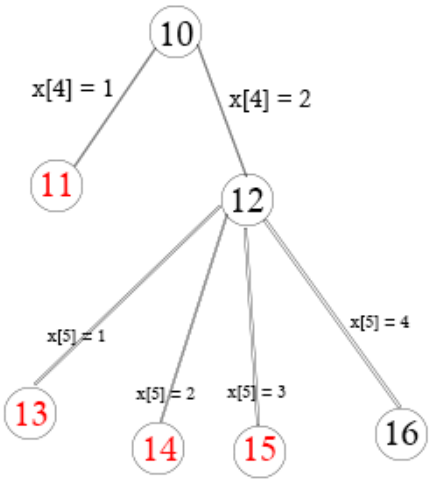
Gambar 3.3 Hasil Ekspansi Simpul ke-5 dengan Ratu x[2] pada Kolom ke-3

Dari simpul ke-5 ini yang telah menjadi simpul-E yang baru, diperluas kembali untuk penempatan ratu x[3] dan dibangkitkan seluruh kemungkinan yang masih mengarah ke solusi. Dimulai dari kolom ke-1, terlihat bahwa penempatan ratu x[3] di kolom ke-1 melanggar aturan karena se-kolom dengan ratu x[1]. Pada kolom ke-2, ratu x[3] berada pada diagonal yang sama dengan ratu x[2]. Di kolom ke-3, ratu x[3] satu kolom dengan ratu x[2]. Di kolom ke-4, sekali lagi ratu x[3] satu diagonal dengan ratu x[2]. Terakhir di kolom ke-5, ratu x[3] berhasil lolos dari fungsi pembatas sehingga simpul tersebut menjadi simpul-E baru yang akan diperluas selanjutnya.



Gambar 3.4 Hasil Ekspansi Simpul ke-10 dengan Ratu x[3] pada Kolom ke-5

Kemudian untuk pembangkitan simpul-E yang baru yaitu simpul ke-10, kembali dilakukan pengecekan ulang dimulai dari kolom ke-1 untuk penempatan ratu x[4]. Pada kolom ke-1 tentu tidak mengarah ke solusi karena berada pada satu kolom dengan ratu x[1]. Di kolom ke-2 ternyata ratu x[4] tidak melanggar fungsi pembatas. Maka dikarenakan algoritma runut-balik mengikuti aturan *depth-first*, maka simpul ke-12 dengan lintasan dari simpul 1-2-5-10-12 menjadi simpul-E yang baru dan akan diperluas kembali.



Gambar 3.5 Hasil Ekspansi Simpul ke-12 dengan Ratu x[4] pada Kolom ke-2

Masuk ke tahap penempatan terakhir yaitu dimana k = N = 5. Dikarenakan pada tahapan sebelumnya kolom yang sudah ditempati adalah kolom ke-1 , 2, 3, dan 5 maka satu-satunya kolom yang memungkinkan untuk ditempati oleh ratu x[5] adalah kolom ke-4. Tetapi sekali lagi dikarenakan algoritma runut-balik mengikuti aturan *depth-first*, untuk kolom dibawah kolom ke-4 tetap dilakukan pemeriksaan oleh fungsi pembatas. Sehingga disini kita akan memeriksa simpul ke-13, 14, dan 15 yang tentunya tidak mengarah ke solusi.

Pada penempatan ratu x[5] di kolom ke-4, ternyata berdasarkan perhitungan oleh fungsi pembatas (fungsi Tempat) tidak melanggar. Maka kita dapat meletakkan ratu x[5] pada kolom ke-4. Di simpul ke-16 ini yang disertai dengan lintasan dari akar simpul 1-2-5-10-12-16 ternyata sudah meruapakan lintasan yang lengkap dari akar menuju daun. Maka dapat disimpulkan bahwa solusi pertama dari persoalan *n*-ratu dengan n = 5 telah ditemukan. Berikut merupakan hasil papan catur 5x5 dengan penempatan solusi yang telah ditemukan:

1	2	3	4	5	
v					x[1]
x	x	v			x[2]
x	x	x	x	v	x[3]
x	v				x[4]
x	x	x	v		x[5]

Gambar 3.1 Solusi Pertama Persoalan *n*-Ratu dengan n = 5

Pada dasarnya tidak menutup kemungkinan bahwa masih terdapat solusi lain yang dapat menyelesaikan persoalan *n*-ratu tersebut. Tetapi dalam persoalan kali ini dibatasi hanya hingga solusi pertama ditemukan saja.

IV. KESIMPULAN

Algoritma runut-balik merupakan salah satu pengembangan yang sangat baik daripada algoritma *exhaustive search*. Jumlah simpul yang dibangkitkan pada algoritma runut-balik jauh lebih sedikit dibandingkan dengan algoritma *exhaustive search* yang melakukan pembangkitan pada semua simpul yang mungkin dicapai meskipun simpul tersebut sama sekali tidak mengarah ke solusi yang diinginkan.

Penerapan algoritma runut-balik pun sangat luas. Algoritma runut-balik dengan skema umum yang telah dijelaskan sebelumnya dapat dijadikan metode untuk menyelesaikan persoalan-persoalan yang berbasis pada kecerdasan buatan ataupun mempunyai pilihan-pilihan dalam setiap langkahnya. Dengan algoritma runut-balik, dipastikan setiap simpul yang dibangkitkan mempunyai kemungkinan untuk mendapatkan solusi yang diharapkan.

REFERENSI

[1] http://ivandemarino.me/2010/01/22/use-backtracking-to-print-all-subsets/ diakses pada 19 Mei 2017 pukul 03.15.

[2] Munir, Rinaldi, *Strategi Algoritma*, Bandung: Penerbit Informatika Bandung, 2009.

[3] Rosen, Kenneth H., *Discrete Mathematics and Its Applications*, New York: McGraw-Hill International, 2012, 7th ed.

[4] http://utari-21.blogspot.co.id/2012/09/algoritma-backtracking-runut-balik.html diakses pada 18 Mei 2017 pukul 13.03

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, 19 Mei 2017

Faiz Ghifari Haznitrama 13515010