Pembentukan Pohon Pencarian Solusi dalam Persoalan N-Ratu (The N-Queens Problem)

Pradipta Yuwono – NIM 13506103
Prodi Teknik Informatika, Sekolah Teknik Elektro dan Informatika,
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung, email : if16103@students.if.itb.ac.id

Abstract *- Teori pohon adalah salah satu teori matematika diskrit yang sering diaplikasikan dalam bidang ilmu komputer. Salah satu bentuk pengaplikasian teori pohon adalah pada pembentukan pohon pencarian solusi pada algoritma runut-balik untuk memecahkan berbagai persoalan, seperti persoalan N-Ratu salah satunya. Dalam makalah ini penulis akan membahas proses pembentukan pohon pencarian solusi pada algoritma runut balik yang digunakan untuk memecahkan persoalan N-Ratu.*

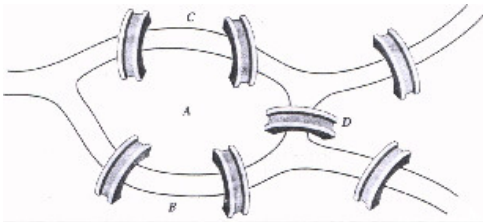
Kata Kunci*: graf, pohon, runut-balik, brute force, N-Ratu.*

1. Pendahuluan Graf dan Pohon

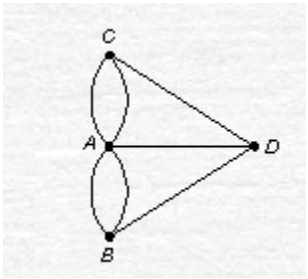
1.1 Teori Graf

Pada pembahasan tentang pohon pencarian solusi sebaiknya kita memahami dahulu konsep pohon dan konsep graf. Graf adalah representasi dari objek-objek dan hubungan antar objek-objek tersebut. Definisi teknis dari Graf adalah, kumpulan himpunan simpul yang tidak boleh kosong beserta kumpulan himpunan sisi yang menghubungkan sepasang simpul-simpul tersebut. Simpul direpresentasikan sebagai noktah, bulatan, atau titik sedangkan sisi direpresentasikan sebagai garis yang menghubungkan antar simpul-simpul tersebut.

Masalah jembatan Konigsberg adalah masalah yang pertama kali menggunakan konsep graf. Dahulu, di kota Konigsberg terdapat sungai yang mengalir mengitari pulau dan bercabang menjadi dua buah anak sungai. Tiap-tiap daratan yang dipisahkan oleh sungai tersebut dihubungkan oleh beberapa jembatan. Masalah ini dapat diilustrasikan dengan gambar di bawah ini.



Masalah yang diajukan pada jembatan Konigsberg ini adalah, apakah mungkin kita melalui ketujuh jembatan tersebut masing-masing tepat satu kali dan kembali lagi ke tempat semula? Permasalahan ini berhasil dipecahkan oleh seorang ahli matematika Swiss bernama Euler yang merepresentasikan jembatan Koenigsberg menjadi sebuah graf, dimana simpul merupakan representasi daratan dan sisi merupakan representasi dari jembatan, seperti di bawah ini.



Dalam solusinya, Euler mengatakan bahwa tidak mungkin kita melalui ketujuh jembatan itu masing-masing sekali dan kembali lagi ke tempat asal keberangkatan jika derajat setiap simpul tidak seluruhnya genap. Derajat adalah jumlah garis yang bersisian dengan suatu simpul. Karena tidak semua simpul berderajat genap, maka tidak mungkin melakukan perjalanan yang berupa sirkuit pada graf tersebut.

Pada graf, lintasan didefinisikan sebagai kumpulan beberapa simpul-simpul dimana terdapat satu simpul sebagai simpul awal dan satu simpul sebagai simpul akhir dan diantara simpul awal dan akhir itu terdapat himpunan sisi

dan simpul yang menghubungkan simpul awal dan simpul akhir. Sebagai contoh, pada graf jembatan Koenigsberg di atas C – A – B – D merupakan sebuah lintasan.

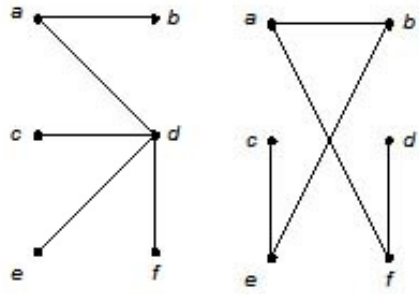
Bentuk khusus dari lintasan adalah sirkuit atau siklus. Sirkuit adalah sebuah lintasan yang berawal dan berakhir pada simpul yang sama. Pada graf jembatan Koenigsberg diatas, C – A – B – D – C merupakan sebuah sirkuit.

Graf yang sisi-sisinya tidak mempunyai orientasi arah disebut graf tak-berarah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Sebaliknya jika graf yang setiap sisinya diberikan orientasi arahnya disebut sebagai graf berarah.

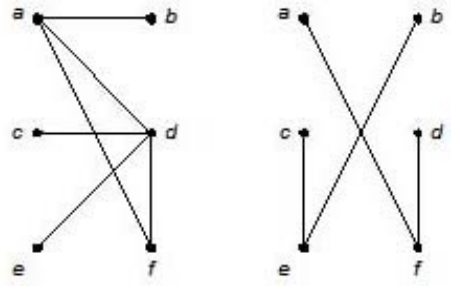
Dua buah simpul pada graf dikatakan terhubung jika terdapat lintasan antara kedua simpul tersebut. Jika dua buah simpul terhubung maka pasti suatu simpul dapat dicapai dari simpul lain. Jika setiap pasang simpul di dalam graf terhubung, maka graf tersebut kita katakan graf terhubung. Sebaliknya jika ada sebuah simpul saja yang tidak terhubung dengan simpul manapun, maka graf tersebut dikatakan graf tak-terhubung.

1.2 Teori Pohon

Pohon adalah bentuk khusus dari graf. Definisi dari pohon adalah: graf tak-berarah terhubung yang tidak mengandung sirkuit. Berdasarkan definisi tersebut, maka dua graf di bawah ini termasuk pohon.



Sedangkan dua graf di bawah ini bukanlah pohon. Graf di sebelah kiri bukan pohon karena mengandung sirkuit A – D – F – A, dan graf di sebelah kanan bukan pohon karena tidak terhubung (titik persilangan pada graf tersebut bukanlah simpul).



Jika graf G adalah sebuah pohon, maka graf G memiliki sifat-sifat seperti di bawah ini:

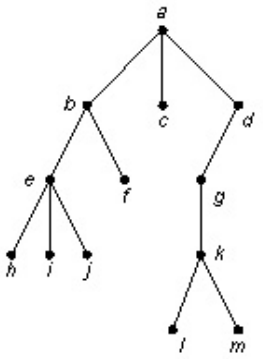
1. Setiap pasang simpul di dalam *G* terhubung dengan lintasan tunggal.
2. *G* terhubung dan memiliki $m = n - 1$ buah sisi.
3. *G* tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
4. *G* tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
5. *G* terhubung dan semua sisinya adalah jembatan.

Pada pohon, setiap simpul umumnya disebut dengn nama node untuk membedakannya dengan istilah yang digunakan pada graf.

1.3 Pohon Berakar

Pohon seperti digambarkan di atas akan sulit digunakan pada pohon pencarian solusi, karena pohon pencarian solusi membutuhkan satu buah node yang akan digunakan sebagai node awal. Oleh karena itu pohon pencarian solusi menggunakan bentuk khusus dari pohon yang disebut pohon berakar.

Pohon berakar adalah pohon yang sebuah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah. Akar mempunyai derajat masuk sama dengan nol dan simpul-simpul lainnya berderajat masuk sama dengan satu. Simpul yang mempunyai derajat keluar sama dengan nol disebut daun. Simpul yang mempunyai derajat keluar tidak sama dengan nol disebut simpul dalam. Setiap simpul di pohon dapat dicapai dari akar dengan sebuah lintasan unik. Sebagai perjanjian, arah sisi di dalam pohon dapat dibuang. Karena setiap simpul di pohon harus dicapai dari akar, maka lintasan di dalam pohon berakar selalu dari atas ke bawah. Contoh pohon berakar dapat dilihat pada gambar di bawah ini.



Pada pohon berakar terdapat beberapa terminologi yang sering digunakan dipandang dari node yang sedang digunakan. Istilah terminologi yang terpenting adalah parent dan child. Parent adalah simpul yg berada di atas node yang sedang digunakan, sedangkan child merupakan simpul yang terletak di bawah node yang sedang digunakan. Sebagai contoh pada gambar diatas, jika sedang menggunakan simpul b, simpul itu memiliki parent simpul a, dan memiliki 2 child yaitu simpul e dan simpul f. Simpul yang tidak memiliki parent dinamakan akar sedangkan simpul yang tidak memiliki child dinamakan daun. Pada gambar di atas, akar adalah simpul a, sedangkan simpul h, i, j, f, c, l, m adalah simpul daun.

2. Algoritma Runut-Balik

2.1 Definisi Algoritma Runut-Balik

Algoritma runut-balik adalah sebuah algoritma yang digunakan untuk menemukan semua atau beberapa solusi dari beberapa masalah komputasi, yang secara incemental membangun kandidat-kandidat baru untuk solusi yang diberikan, dan meninggalkan masing-masing bagian kandidat (runut-balik) segera setelah diketahui bahwa kandidat tersebut tidak mungkin diselesaikan menjadi solusi yang valid. Istilah runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950 .

Algoritma runut-balik hanya dapat diaplikasikan pada masalah yang mengenal konsep "solusi kandidat parsial" dan yang dapat dilakukan pengecekan dengan cepat apakah masalah tersebut dapat diselesaikan menjadi solusi yang valid atau tidak. Algoritma runut-balik sendiri adalah algoritma yang berbasis pada DFS (*Depth First Search*) untuk mencari solusi persoalan yang paling mangkus. Jika algoritma runur balik dapat diaplikasikan, algoritma ini sering menjadi jauh lebih cepat ketimbang mengenumerasi secara *brute force* seluruh kandidat yang ada, karena algoritma ini dapat mengeliminasi banyak kandidat yang tidak memungkinkan dengan tes tunggal.

Algoritma runut balik bergantung pada "*black box procedures*" yang diberikan oleh user , yang mendefinisikan masalah yang akan diselesaikan, sifat dasar dari kandidat parsial, dan bagaimana mereka diperluas menjadi kandidat lengkap. Bagaimanapun algoritma ini adalah sebuah *metaheuristic* ketimbang algoritma yang spesifik. Walaupun, tidak seperti banyak *metaheuristic* lainnya, algoritma ini menjamin dapat menemukan semua solusi sebuah masalah dalam jangka waktu yang terbatas.

Dalam menerapkan metode runut balik, properti-properti berikut harus didefinisikan:

1. Solusi persoalan.
   Solusi dinyatakan sebagai vektor dengan *n-tuple*:

   $$X = (x_1, x_2, \ldots, x_n),$$

   $x_i \in$ himpunan berhingga S_i. Mungkin saja $S_1 = S_2 = \ldots = S_n$.

   Contoh: $S_i = \{0, 1\}$,
   $x_i = 0$ atau 1

2. Fungsi pembangkit nilai x_k, dinyatakan sebagai:

   $$T(k)$$

   $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

3. Fungsi pembatas / fungsi kriteria, dinyatakan sebagai:

   $$B(x_1, x_2, \ldots, x_k)$$

   Fungsi pembatas menentukan apakah $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak, maka $(x_1, x_2, \ldots, x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

2.2 Prinsip Pencarian Solusi Pada Algoritma Runut Balik

Langkah-langkah pencarian solusi pada pohon ruang status yang dibangun secara dinamis adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan pencarian mendalam (DFS). Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*). Simpul hidup yang *sedang* diperluas dinamakan **simpul-E** (*Expand-node*).

2. Tiap 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** (*dead node*). Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas** (*bounding function*). Simpul yang sudah mati tidak akan pernah diperluas lagi.

3. 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 orangtua yang akan menjadi simpul-E yang baru.

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

Dibawah ini adalah skema umum algoritma runut-balik dalam versi rekursif dan versi iteratif. Skema versi rekursif adalah skema yang lebih tepat karena algoritma runut-balik lebih alami dinyatakan dalam bentuk rekursi.

(a) Pseudocode versi rekursif

```
procedure RunutBalikR(input k:integer)

{Mencari semua solusi persoalan dengan
metode runut-balik; skema rekursif
 Masukan: k, yaitu indeks komponen 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
        if (x[1], x[2], ... ,x[k]) adalah
        lintasan dari akar ke daun
        then
                CetakSolusi(x)
        endif
        RunutBalikR(k+1) { tentukan nilai
        untuk x[k+1]}
endfor
```

(b) Pseudocode versi iteratif

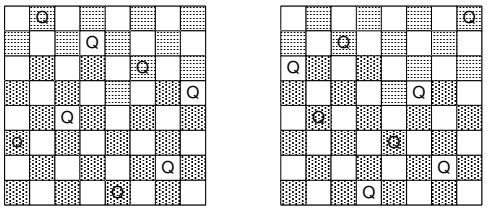
```
procedure RunutBalikI(input n:integer)

{Mencari semua solusi persoalan dengan
metode runut-balik; skema iteratif.
 Masukan: n, yaitu panjang vektor solusi
 Keluaran: solusi x = (x[1], x[2], …,
x[n])
}
Delarasi:
   k : integer

Algoritma:
k←1
while k > 0 do
        if (x[k] belum dicoba sedemikian
        sehingga x[k]←T(k)) and
        (B(x[1], x[2], ... ,x[k])= true)
        then
                if (x[1],x[2],...,x[k])
                adalah lintasan dari akar
                ke daun
                then
                        CetakSolusi(x)
                endif
                k←k+1  {indeks anggota
                tupple berikutnya}
        else    {x[1], x[2], …, x[k] tidak
        mengarah ke simpul solusi }
                k←k-1  {runut-balik ke
                anggota tupple sebelumnya}
        endif
endwhile{ k = 0 }
```

Kasus terburuk untuk algoritma runut-balik akan terjadi jika jumlah simpul dalam pohon ruang status adalah 2^n atau n! yang memiliki kompleksitas waktu masing-masing O(p(n)2n) dan O(q(n)n!), dengan p(n) dan q(n) adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

3. Persoalan N-Ratu

3.1 Definisi Persoalan N-Ratu

Contoh klasik yang sering digunakan dalam pemberian contoh penggunaan algoritma runut-balik adalah persoalan N-Ratu (The N-Queens Problem) yang meminta penempatan N buah ratu pada sebuah papan berukuran N x N (biasanya menggunakan papan catur) sehingga tidak ada ratu yang saling menyerang satu dengan lainnya, atau dengan kata lain tidak ada dua buah ratu yang terletak pada satu baris yang sama, satu kolom yang sama, atau satu diagonal yang sama. Gambar dibawah ini adalah dua contoh solusi pada persoalan 8 – Ratu:



3.2 Penyelesaian Persoalan N-Ratu Menggunakan Algoritma Brute Force

Persoalan N-Ratu dapat diselesaikan dengan menggunakan algoritma brute force standar maupun menggunakan algoritma runut-balik yang merupakan perbaikan dari algoritma brute force. Dalam penggunaan algoritma brute force sendiri ada tiga ide yang dapat digunakan dalam penyelesaian persoalan tersebut. Berikut ini akan dijabarkan ketiga macam algoritma-algoritma brute force tersebut.

a) Brute Force 1

Ide pada algoritma ini adalah dengan mencoba semua kemungkinan solusi penempatan ratu pada papan dengan cara menaruh seluruh ratu pada setiap petak secara sembarang, lalu diperiksa apakah telah menghasilkan sebuah solusi. Seandainya algoritma ini digunakan untuk menyelesaikan persoalan 8-ratu dan kita hitung peluangnya akan diketahui bahwa algoritma ini jelas sangat tidak mangkus, karena kita harus memeriksa sebanyak $C(64, 8) = 4.426.165.368$ kemungkinan solusi

b) Algoritma Brute Force II

Algoritma ini berdasarkan pada ide bahwa salah satu persyaratan dalam persoalan N-Ratu adalah tidak boleh ada 2 ratu atau lebih yang terletak pada baris yang sama. Sehingga dari pernyataan ini dapat dibuat langkah lebih maju dengan meletakkan masing-masing ratu hanya pada baris yang berbeda. Di bawah ini diberikan pseucode yang akan menjelaskan penyelesaian persoalan 8-Ratu dengan algoritma ini:

```
procedure Ratu1
{Mencari semua solusi penempatan delapan
ratu pada petak-petak papan catur yang
berukuran 8 x 8 }
Deklarasi
  i1, i2, 13, 14, i5, i6, i7, i8 :
integer

Algoritma:
  for i1←1 to 8 do
   for i2←1 to 8 do
     for i3←1 to 8 do
        for i4←1 to 8 do
           for i5←1 to 8 do
             for i6←1 to 8 do
                for i7←1 to 8 do
                   for i1←1 to 8 do
                        if Solusi(i1, i2,
                        i3, i4, i5, i6,
                        i7, i8) then
                        write(i1, i2, i3,
                        i4, i5, i6, i7,
                        i8)
                        endif
                   endfor
              endfor
            endfor
          endfor
        endfor
     endfor
   endfor
  endfor
 endfor
```

Dengan menggunakan algoritma brute force yang kedua ini, jumlah kemungkinan solusinya telah berkurang pesat menjadi $8^8 =$ 16.777.216.

c) Algoritma Brute Force 3

Menurut algoritma ini pada setiap baris kita hanya perlu menempatkan ratu pada setiap kolom yang berbeda. Seandainya persoalan tersebut adalah persoalan 8-Ratu, maka papan yang digunakan memiliki delapan buah kolom, dan setiap ratu ditempatkan pada kolom 1, 2, .., 8 pada baris yang berbeda-beda. Kita cukup menyatakan solusinya dengan menuliskan nomor kolom-kolom papan catur tempat sebuah ratu diletakkan. Misalnya solusi tersebut dinyatakan dalam vektor 8-tuple:

$$X = (x_1 , x_2 , \ldots , x_8)$$

Kita dapat menghitung kemungkinan penyelesaian persoalan ini dengan mengamati bahwa vektor solusi merupakan permutasi bilangan 1 sampai 8. Jumlah permutasi bilangan

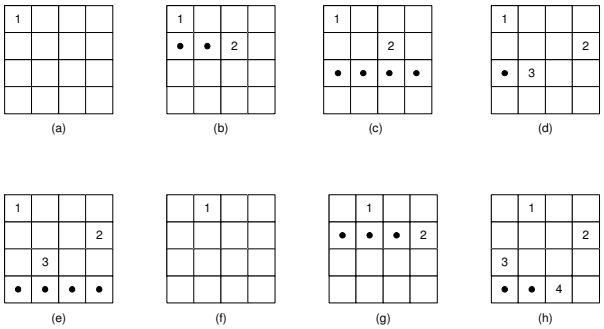
1 sampai 8 adalah P(1, 8) = 8! = 40.320 buah. Di bawah ini adalah pseudocode yang akan menjelskan algoritma ini.

```
procedure Ratu2
{Mencari semua solusi penempatan delapan
ratu pada petak-petak papan catur yang
berukuran 8 x 8 }
Deklarasi
  X : vektor_solusi
  n,i : integer

Algoritma:
n←40320{ Jumlah permutasi (1, 2, …, 8) }
i←1
repeat
        X←Permutasi(8)    { Bangkitan
        permutasi (1, 2, …, 8) }

        { periksa apakah X merupakan
        solusi }
        if Solusi(X) then
                TulisSolusi(X)
        endif

        i←i+1    { ulangi untuk permutasi
        berikutnya }
until i > n
```

3.3 Penyelesaian Persoalan N-Ratu Menggunakan Algoritma Runut Balik

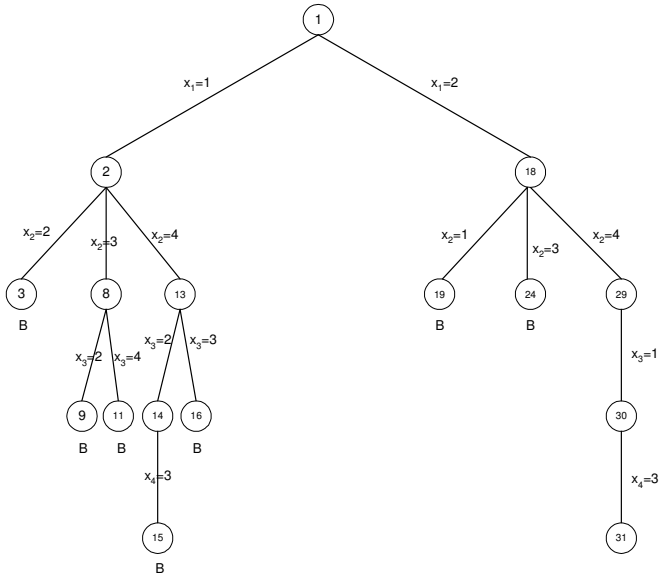
Algoritma runut-balik dapat memperbaiki efisiensi algoritma brute force. Ruang solusinya adalah semua permutasi dari angka-angka 1,2, ... , N. Organisasi dari ruang solusi adalah sebuah pohon ruang status. Setiap permutasi dari 1, 2, ..., N dinyatakan dengan lintasan dari akar ke daun. Sisi-sisi pada pohon diberi label nilai x_i.

Untuk menyederhanakan persoalan, disini hanya akan diilustrasikan proses penyelesaian persoalan 4-Ratu. Ruang solusi adalah semua permutasi bilangan 1 sampai 4. Ruang solusi ini diorganisasikan menjadi pohon ruang status statis. Pohon dinomori dari atas ke bawah sesuai dengan aturan pencarian DFS. Ganbar berikut memperlihatkan langkah-langkah penempatan 4 ratu dengan algoritma runut balik. Bulatan titik menyatakan percobaan menempatkan ratu dan ditolak karena segaris dengan ratu lainnya.



Pada langkah (c) pada gambar diatas, keempat kolom telah dicoba tetapi gagal menempatkan ratu berikutnya pada papan catur. Proses runut-balik dilakukan dengan menggeser ratu nomor dua ke kolom empat. Langkah (d) memperlihatkan solusi pertama yang ditemukan.

Di bawah ini diperlihatkan pohon ruang yang dibentuk secara dinamis selama proses pencarian solusi pertama. Solusi pertama adalah lintasan dari simpul 1 sampai simpul 31, yaitu X = (2, 4, 1, 3). Disini nomor simpul sengaja dipertahankan sama dengan nomor simpul pada pohon statisnya. Huruf B di bwah simpul menyatakan bahwa simpul tersebut dibunuh selama pencarian



Berikut ini akan diberi penjelasan skema algoritma runut-balik secara iteratif dan rekursif.

(a) Versi Iteratif

Kita bayangkan papan catur sebagai sebuah matriks berukuran N x N. Dua buah ratu terletak pada baris yang sama berarti i = k. Dua buah ratu terletak pada kolom yang sama, berarti j = 1. Dua buah ratu terletak pada diagonal yang sama, berarti

↘i-j=k-l atau ↙i+j=k+l
⇔ i-k=j-l atau k-i=j-l
⇔ |j-l| = |i-k|

Pada kedua versi yang digunakan, fungsi TEMPAT mengembalikan nilai true jika ratu dapat ditempatkan pada kolom ke-k (dinyatakan dengan x[k]). Fungsi ini memeriksa apakah x[k] berbeda dengan x[1], x[2], .., x[k-1] dan juga memeriksa apakah ada dua ratu ditempatkan pada kolom yang sama atau diagonal yang sama. Pemeriksaan ratu pada baris yang sama tidak perlu dilakukan karena setiap ratu sudah pasti ditempatkan pada baris yang berbeda.

Dibawah ini adalah pseudocode skema algoritma pada persoalan N-Ratu secara iteratif

```
procedure N_RATU_I(input N:integer)
{ Mencetak semua solusi penempatan N buah ratu pada
  petak papan catur N x N tanpa melanggar kendala; versi iteratif

  Masukan: N = jumlah ratu
  Keluaran: semua solusi x = (x[1], x[2], …, x[N]) dicetak ke layar.
}

Deklarasi
  k : integer

Algoritma:
k←1 {mulai pada baris catur ke-1}
x[1]←0 {inisialisasi kolom dengan 0}
while k > 0 do
        x[k]←x[k]+1{pindahkan ratu ke kolom berikutnya}
        while (x[k] ≤ N) and (not TEMPAT(k)) do
                {periksa apakah ratu dapat ditempatkan pada kolom x[k]}
                x[k]:=x[k] + 1
        endwhile
        {x[k] > n or TEMPAT(k) }

        if x[k]≤ n then   { kolom penempatan ratu ditemukan }
                if k=N then    { apakah solusi sudah lengkap?}
                        CetakSolusi(x,N)
                        { cetak solosi}
                else
                        k←k+1  {pergi ke baris berikutnya}
                        x[k]←0
                        {inisialisasi kolom dengan 0}
                endif
        else
                k←k-1  { runut-balik ke baris sebelumnya}
        endif
endwhile
{ k = 0 }
```

Di bawah ini adalah pseudocode fungsi TEMPAT yang akan digunakan oleh kedua versi algoritma runut-balik

```
procedure N_RATU_I(input N:integer)
{ Mencetak semua solusi penempatan N buah ratu pada
  petak papan catur N x N tanpa melanggar kendala; versi iteratif

  Masukan: N = jumlah ratu
  Keluaran: semua solusi x = (x[1], x[2], …, x[N]) dicetak ke layar.
}

Deklarasi
  k : integer

Algoritma:
k←1{mulai pada baris catur ke-1}
x[1]←0 {inisialisasi kolom dengan 0}
while k > 0 do
        x[k]←x[k]+1{pindahkan ratu ke kolom berikutnya}
        while (x[k] ≤ N) and (not TEMPAT(k)) do
                {periksa apakah ratu dapat ditempatkan pada kolom x[k]}
                x[k]:=x[k] + 1
        endwhile
        {x[k] > n or TEMPAT(k) }

        if x[k]≤ n then   { kolom penempatan ratu ditemukan }
                if k=N then    { apakah solusi sudah lengkap?}
                        CetakSolusi(x,N)
                        { cetak solosi}
                else
                        k←k+1{pergi ke baris berikutnya}
                        x[k]←0
                        {inisialisasi kolom dengan 0}
                endif
        else
                k←k-1  { runut-balik ke baris sebelumnya}
        endif
```

```
endwhile
{ k = 0 }
```

(b) Versi Rekursif

Algoritma:

1. Mula-mula inisialisasi x[1], x[2], ..., x[N] dengan 0

```
for i←N to n do
        x[i] ← 0
endfor
```

2. Kemudian panggil prosedur N_RATU_R yang pseucodenya seperti di bawah ini

```
procedure N_RATU_I(input N:integer)
{ Mencetak semua solusi penempatan N buah
ratu pada
  petak papan catur N x N tanpa melanggar
kendala; versi iteratif

  Masukan: N = jumlah ratu
  Keluaran: semua solusi x = (x[1], x[2],
…, x[N]) dicetak ke layar.
}

Deklarasi
  k : integer

Algoritma:
k←1{mulai pada baris catur ke-1}
x[1]←0 {inisialisasi kolom dengan 0}
while k > 0 do
        x[k]←x[k]+1    {pindahkan ratu ke
        kolom berikutnya}
        while (x[k] ≤ N) and (not
        TEMPAT(k)) do
                {periksa apakah ratu dapat
                ditempatkan pada kolom
                x[k]}
                x[k]:=x[k] + 1
        endwhile
        {x[k] > n or TEMPAT(k) }

        if x[k]≤ n then    { kolom
        penempatan ratu ditemukan }
                if k=N then     { apakah
                solusi sudah lengkap?}
                        CetakSolusi(x,N)
                        { cetak solosi}
                else
                        k←k+1{pergi ke
                        baris berikutnya}
                        x[k]←0
                        {inisialisasi
                        kolom dengan 0}
                endif
        else
                k←k-1  { runut-balik ke
                baris sebelumnya}
        endif
endwhile
{ k = 0 }
```

4. Kesimpulan

Dari pembahasan makalah ini dapat disimpulkan bahwa algoritma runut-balik mampu menambah efisiensi waktu penyelesaian sebuah persoalan secara signifikan jika dibandingkan dengan menggunakan algoritma brute force standar. Walaupun begitu tidak seperti algoritma brute-force, algoritma runut-balik ini hanya dapat diterapkan pada persoalan yang terbatas dan memiliki sifat-sifat tertentu saja.

Daftar Referensi

1. Munir , Rinaldi. 2004, Bahan Kuliah IF2151 Matematika Diskrit, Departemen Teknik Informatika, Institut Teknologi Bandung
2. Munir , Rinaldi. 2004, Bahan Kuliah IF2251 Strategi Algoritmik, Departemen Teknik Informatika, Institut Teknologi Bandung
3. Wikipedia, (2007). Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Backtracking Tanggal akses : 29 Desember 2008 pukul 04.00 GMT +7.