PENERAPAN ALGORITMA RUNUT-BALIK DALAM PENCARIAN SOLUSI TEKA-TEKI *BATTLESHIP*

Abraham Ranardo Sumarsono

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jln. Ganesha 10 Bandung
e-mail: if17056@students.if.itb.ac.id

ABSTRAK

Teka-teki *Battleship* adalah salah satu teka-teki yang membutuhkan logika sebagai dasar penyelesaiannya. Teka-teki ini direpresentasikan dengan kisi yang berisi sekumpulan kotak-kotak persegi dengan angka-angka di pinggir kanan dan bawah kisi. Tujuan dari teka-teki ini adalah menebak letak posisi kapal-kapal yang tersembunyi di dalam teka-teki *Battleship* tersebut. Satu-satunya petunjuk adalah angka-angka di pinggir kisi yang menunjukkan jumlah bagian kapal yang tersembunyi di dalam setiap kolom ataupun baris yang bersangkutan serta beberapa bagian kapal ataupun bagian air pada posisi tertentu yang bervariasi pun dapat ditunjukkan dari awal teka-teki.
Setiap teka-teki *Battleship* ini memiliki solusi yang unik. Sehingga, teka-teki ini pun dapat diselesaikan dengan mudah dengan bantuan algoritma. Di dalam makalah ini akan dibahas bagaimana cara menemukan solusi dari teka-teki *Battleship* dengan menggunakan penerapan algoritma Runut-Balik, yang berbasis pada algoritma *Depth First Search* (DFS).

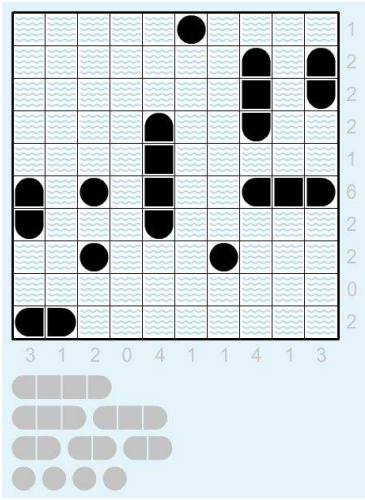
Kata kunci: Teka-teki *Battleship*, Algoritma Runut-Balik, Algoritma *Depth First Search*, DFS.

1. PENDAHULUAN

Teka-teki *Battlehip* atau sering juga disebut *Bimaru* ataupun *Solitaire Battleships*, adalah sebuah teka-teki logika yang didasarkan pada permainan tebak *Battleship*. Teka-teki *Battleship* ini diciptakan di Argentina oleh Jaime Poniachik dan dipublikasikan untuk pertama kalinya pada tahun 1982 di majalah "Humor & Juegos". *Battleship* mendapatkan popularitasnya setelah debut internasionalnya pada kompetisi teka-teki dunia yang pertama (*World Puzzle Championship*) di kota New York pada tahun 1992.

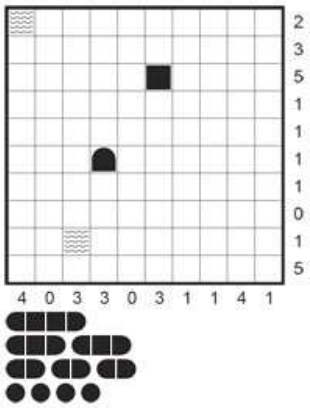
Teka-teki *Battleship* dimainkan pada sebidang kisi dari kotak-kotak persegi yang menyembunyikan kapal-kapal yang bermacam-macam ukurannya. Angka yang berada di pinggir kanan menandakan ada berapa banyaknya bagian kapal yang tersembunyi dalam satu baris tersebut. Sedangkan, angka yang berada di pinggir bawah menandakan ada berapa banyaknya bagian kapal yang tersembunyi di dalam satu kolom tersebut.

Gambar di bawah ini merupakan salah satu contoh teka-teki *Battleship* yang telah terpecahkan. Pada bagian bawah dari gambar, terdapat daftar kapal-kapal yang harus ditemukan di dalam kotak-kotak persegi tersebut.



Gambar 1. Teka-teki *Battleship*

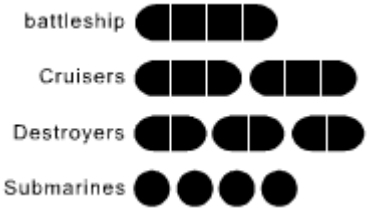
Teka-teki *Battleship* pada umumnya berisi 10x10 kotak persegi yang berisi armada kapal yang tersembunyi. Informasi yang ada hanyalah angka-angka yang menyatakan berapa banyak bagian kapal yang tersembunyi di masing-masing baris dan kolom. Beberapa bagian kapal pun dapat diberikan pada awal permainan di tempat yang bermacam-macam di kotak-kotak persegi. Tujuan permainan adalah untuk menemukan letak semua armada kapal di kotak-kotak persegi.



Gambar 2. Contoh Keadaan Awal Teka-teki *Battleship*

Setiap teka-teki *Battleship* merepresentasikan sebuah laut dengan armada kapal-kapal yang tersembunyi, yang mungkin saja memiliki orientasi peletakan horizontal ataupun vertikal di dalam kotak-kotak persegi. Teka-teki *Battleship* juga memiliki ketentuan lainnya, yaitu tidak ada kapal yang menyentuh satu sama lainnya dan juga kapal tidak dapat diletakan secara diagonal. Angka-angka yang berada di kanan dan bawah kisi menunjukkan seberapa banyak kotak pada baris dan kolom yang terkait yang memiliki oleh bagian kapal.

Teka-teki *Battleship* dimainkan dengan berbagai ukuran dan juga dengan berbagai armada yang memiliki kapal-kapal yang masing-masingnya berbeda ukuran. Sebagai contoh, armada kapal yang digunakan pada umumnya di teka-teki *Classic Battleship* 10x10 terdiri dari satu *battleship*, dua *cruisers*, tiga *destroyers*, dan empat *submarines*.



Gambar 3. Armada Kapal Teka-teki *Battleship*

Submarines terdiri dari sebuah bagian kapal, *destroyers* memiliki 2 bagian kapal, *cruisers* memiliki 3 bagian kapal, dan *battleship* terdiri dari 4 bagian kapal. Semua kotak persegi yang tersisa di kisi merupakan bagian air.

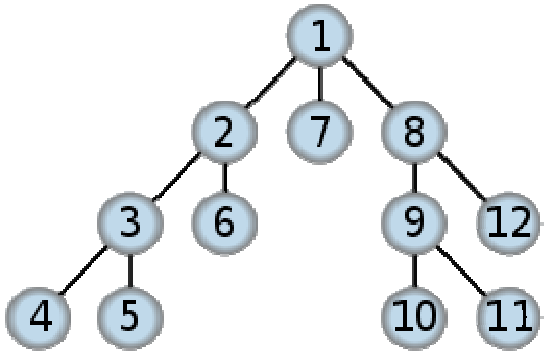
2. DASAR ALGORITMA

Pencarian solusi dari teka-teki *Battleship* ini didapatkan dengan menggunakan algoritma yang sudah ada. Algoritma yang dapat membantu di dalam kasus ini adalah algoritma Runut-Balik yang merupakan basis dari algoritma *Depth First Search* (DFS).

2.1 ALGORITMA *DEPTH FIRST SEARCH*

Depth First Search (DFS) adalah sebuah algoritma untuk mencari sebuah elemen di dalam sebuah pohon, struktur pohon, atau graf dimana proses yang terjadi merupakan proses pencarian secara mendalam.

Proses pencarian dilakukan pada satu simpul dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka akan dilakukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang dinginkan) dan pencarian dilanjutkan pada simpul sebelah kanan. Simpul yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka dilakukan proses *backtracking* kembali dan pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan, maka tidak diperlukan proses *backtracking*.



Gambar 4. Pohon Urutan Solusi DFS

Jika setiap simpul membangkitkan b buah simpul baru, dan batas maksimum kedalaman pohon ruang status adalah m, maka algoritma DFS ini memiliki komplesitas waktu $O(b^m)$. Komplesitas ruang yang dimiliki algoritma DFS adalah O(bm), karena kita hanya perlu menyimpan satu buah lintasan tunggal dari akar hingga daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.

Berikut ini adalah contoh *pseudocode* untuk algoritma *Depth First Search* dalam notasi algoritmik.

```
Procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma
pencarian DFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi ditulis ke
layar
}
Deklarasi
       w : integer
```

```
Algoritma:
        write(v)
        dikunjungi[v] ← true
        for w ← l to n do
              if A[v,w] = l then
            {simpul v dan simpul w bertetangga }
                      if not dikunjungi[w] then
                            DFS(w)
                      endif
              endif
        endfor
```

2.2 ALGORITMA RUNUT-BALIK

Runut-balik adalah algoritma yang berbasis pada algoritma DFS untuk mencari solusi persoalan secara lebih optimal. Algoritma ini secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Prinsip kerja dari algoritma ini pemangkasan simpul-simpul yang tidak mengarah ke solusi. Sehingga, setiap simpul yang tidak memenuhi suatu fungsi pembatas, tidak akan diproses di algoritma ini.

Di dalam algoritma ini, terdapat beberapa fungsi dan variabel yang digunakan di dalam pemrosesan solusi, yaitu ruang solusi, fungsi pembangkit, serta fungsi pembatas. Ruang solusi persoalan dinyatakan sebagai vektor dengan *n-tuple*. Fungsi pembangkit untuk membangkitkan nilai untuk nilai elemen dengan indeks tertentu yang merupakan komponen vektor solusi. Fungsi pembatas digunakan untuk menentukan apakah suatu simpul dapat mengarahkan ke solusi atau tidak.

Ruang solusi diatur sehingga menjadi sebuah struktur pohon seperti pada algoritma DFS. Setiap simpul dari pohon tersebut menyatakan status persoalan, sedangkan sisi/cabang dilabeli dengan nilai-nilai komponen persoalan. Lintasan dari akar pohon hingga daun menyatakan solusi yang mungkin. Pengaturan pohon ruang solusi ini disebut juga sebagai pohon ruang status.

Solusi dicari dengan cara membentuk lintasan dari akar hingga ke daun. Aturan pembentukan yang digunakan mengikuti aturan DFS. Simpul-simpul yang sudah dibentuk dinamakan simpul hidup. Simpul hidup yang sedang dikembangkan dinamakan simpul-E. Setiap kali simpul-E dikembangkan, lintasan yang dibangun pun bertambah panjang. Jika lintasan tersebut tidak mengarah ke solusi, maka simpul-E tersebut ditinggalkan sehingga menjadi simpul mati. Simpul yang sudah mati tidak akan pernah diperluas lagi.

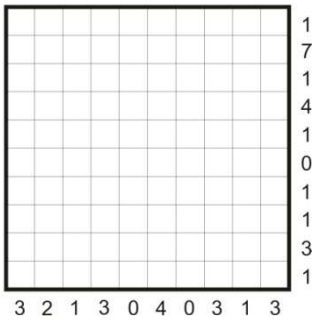
Apabila pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian pun diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka dapat dilakukan runut-balik ke simpul orangtua yang terdekat dan masih merupakan simpul hidup. Simpul orangtua ini pun yang akan menadi simpul-E yang baru. Proses berhenti apabila telah ditemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

Berikut ini adalah contoh *pseudocode* untuk algoritma Runut-Balik dengan menggunakan skema rekursif di dalam notasi algoritmik.

```
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
```

3. SOLUSI TEKA-TEKI *BATTLESHIP*

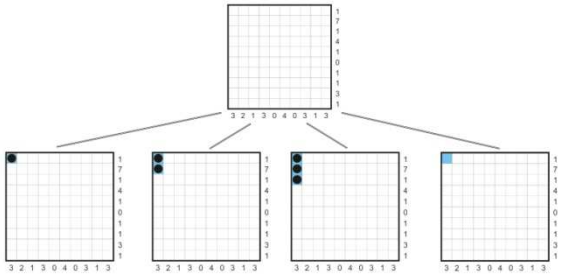
Solusi dari teka-teki *Battleship* dapatditemukan dengan berbasiskan algoritma Runut-Balik. Fungsi pembangkit pada kasus ini adalah semua kemungkinan bagian kapal baik secara vertikal maupun horizontal serta bagian air. Fungsi pembatas yang digunakan adalah perbandingan jumlah bagian kapal pada suatu kolom dan baris dengan angka pembanding yang berada di pinggir kanan (untuk baris) dan bawah (untuk kolom). Selain itu, fungsi pembatas lainnya adalah pengecekan ada atau tidaknya bagian kapal di sekitar lokasi penempatan bagian kapal. Apabila ada, maka lokasi tersebut tidak dapat ditempatkan bagian kapal, sehingga dapat ditempatkan bagian air.



Gambar 5. Contoh Kasus Teka-Teki *Battleship*

Algoritma pencarian solusi teka-teki ini dapat dimulai dengan mengisi kotak pada baris pertama dan kolom pertama dengan bagian kapal yang memenuhi fungsi pembatas. Misalnya, pada gambar 5, pada baris pertama serta kolom pertama, komponen yang dapat dibangkitkan

adalah komponen yang memenuhi angka pinggir kanan baris tersebut yatu satu dan pinggir bawah kolom tersebut yaitu tiga. Komponen yang memenuhi kriteria tersebut adalah bagian air, kapal *submarine* (1 bagian), kapal *destroyer* (2 bagian) secara vertikal, dan kapal *cruiser* (3 bagian) secara vertikal.Sehingga kita dapat memperoleh pohon ruang status awal seperti pada gambar 6.



Gambar 6. Pohon Ruang Status Awal

Lalu, kita ambil simpul pertama, yang merupakan kapal *submarine*, untuk kemudian dijadikan simpul-E. Dari status tersebut, kita dapat kembangkan lagi dengan mengisi kotak pada baris yang sama dan kolom selanjutnya. Hal ini dilakukan terus menerus, hingga mencapai kolom terakhir pada baris tersebut. Di sini setelah kotak tersebut diisi dengan komponen yang memenuhi fungsi pembatas, dilakukan proses pengecekan apakah jumlah bagian kapal pada baris tersebut sudah sama dengan angka pinggir kanan pada baris yang bersangkutan. Apabila tidak sama, simpul dengan status tersebut dimatikan, lalu dilakukan proses runut-balik menuju simpul orangtua di atasnya yang masih hidup. Simpul orangtua tersebut pun kemudian menjadi simpul-E. Begitu seterusnya hingga mencapai kolom terakhir.

Pada baris terakhir, setelah proses pengisian pada kotak yang sedang ditunjuk, dilakukan pengecekan apakah bagian kapal pada kolom yang ditunjuk sudah sesuai dengan angka pinggir bawah pada kolom yang bersangkutan. Apabila sudah sama, maka proses yang sama dilakukan di kolom selanjutnya. Namun apabila belum memenuhi, maka dilakukan proses runut-balik seperti sebelumnya hingga dapat memenuhi angka pinggir bawah pada kolom yang bersangkutan.

Berikut ini adalah *pseudocode* algoritma pencarian solusi teka-teki *Battleship* dalam notasi algoritmik.

```
procedure findsolution(input x,y: integer)
{
mencari solusi dari teka-teki Battleship
dengan mengisi Matriks pada kolom x dan baris y
dengan semua kemungkinan penempatan bagian kapal
yang mungkin
}
Algoritma:
   if Matriks[x][y] kosong then
      fill(x,y)
      {fungsi fill mengisi Matriks
      pada kolom x dan baris y dengan
      penempatan bagian kapal yang mungkin}

   if y adalah baris terakhir then
      if checkjumlahkolom(x) then
      {mengecek apakah jumlah bagian kapal
      sudah memenuhi angka pinggir,
      fungsi checkjumlahkolom mengembalikan
      true jika sudah memenuhi angka pinggir}
         findsolution(x+1,y)
      else
         clear(x,y)
         backtracking(x,y)
         {prosedur backtracking akan melakukan
         proses runut-balik dan apabila masih
         ada simpul orangtua yang masih hidup
         maka akan dilanjutkan dengan prosedur
         findsolution. apabila tidak ada, maka
         berarti tidak ada solusi untuk kasus
         teka-teki Battleship ini}
   else if x adalah kolom terakhir then
      if checkjumlahbaris(y) then
      {mengecek apakah jumlah bagian kapal
      sudah memenuhi angka pinggir,
      fungsi checkjumlahbaris mengembalikan
      true jika sudah memenuhi angka pinggir}
         findsolution(0,y+1)
      else
         clear(x,y)
         backtracking(x,y)
   else if y adalah baris terakhir dan x adalah
kolom terakhir then
      if checkjumlahbaris(y) and
checkjumlahkolom(x) then
         {solusi sudah ditemukan}
      else
         clear(x,y)
         backtracking(x,y)
   else
   {x dan y bukan kolom ataupun
   baris terakhir}
      findsolution(x+1,y)
```

Matriks di algoritma tersebut merupakan matriks yang merepresentasikan kisi dari teka-teki *Battleship* tersebut. Matriks tersebut berisi bagian kapal, bagian air dan bagian yang belum diisi (kosong).

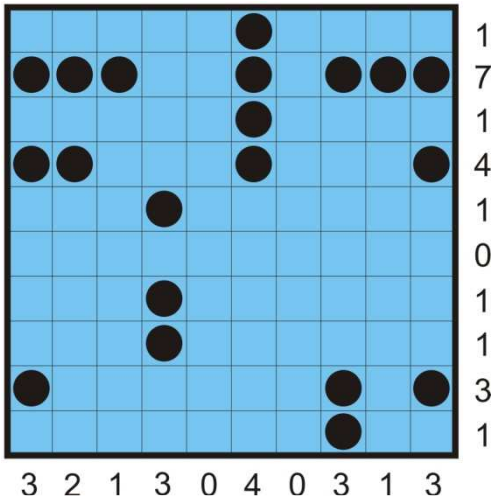
Pada algoritma tersebut, terdapat fungsi fill yang berfungsi mengisi matriks tersebut dengan komponen yang mungkin dengan mengamati kotak-kotak di sekitarnya.

Pada algoritma di atas, juga terdapat fungsi checkjumlahbaris dan checkjumlahkolom. Kedua fungsi tersebut berfungsi untuk mengecek apakah jumlah bagian kapal pada baris atau kolom yang bersangkutan sudah memenuhi angka pinggir yang terkait. Apabila sudah memenuhi, maka kedua fungsi akan mengembalikan nilai *true*. Sebaliknya, apabila jumlah bagian kapal melebihi ataupun kurang dari angka pinggir yang terkait, maka kedua fungsi akan mengembalikan nilai *false*.

Fungsi clear merupakan fungsi yang mengosongkan kotak pada kolom x dan baris y. Selanjutnya, terdapat prosedur backtracking. Prosedur ini merupakan prosedur yang melakukan proses runut-balik. Prosedur ini akan terus melakukan runut-balik hingga menemukan simpul

orangtua yang masih hidup atau dengan kata lain, komponen bagian kapal yang berbeda ataupun bagian air yang masih mungkin ditempatkan pada posisi sebelumnya. Apabila tidak ditemukan, maka kesimpulan yang dapat ditarik adalah kasus teka-teki *Battleship* kali ini tidak memiliki solusi. Hal ini mungkin terjadi, apabila terdapat kesalahan kombinasi angka pinggir.

Algoritma ini akan membuat matriks terisi penuh dengan bagian kapal beserta bagian air apabila solusi dari teka-teki *Battleship* telah ditemukan. Namun, algoritma ini akan mengembalikan matriks yang kosong apabila solusi dari teka-teki *Battleship* tidak dapat ditemukan karena kesalahan kombinasi angka pinggir. Contohnya untuk kasus pada gambar 5, akan dihasilkan solusi seperti pada gambar 7.



Gambar 6. Solusi Teka-Teki *Battleship*

4. KESIMPULAN

Dibandingkan dengan menggunakan penerapan algoritma *BruteForce* yang mencari kemungkinan dari semua kombinasi penempatan kapal yang ada, pencarian solusi teka-teki *Battleship* ini lebih optimal dengan menggunakan penerapan algoritma Runut-Balik. Karena dengan menerapkan algoritma Runut-Balik, kita dapat menghilangkan semua kemungkinan penempatan bagian kapal yang tidak mengarah ke solusi dari teka-teki. Sehingga dengan penerapan algoritma ini, akan lebih menghemat waktu pencarian solusi serta ruang memori yang digunakan tidak banyak.

REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF2251 Strategi Algoritmik*. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. 2009.

[2] *Battleship Puzzle*: http://www.conceptispuzzles.com/index.aspx?uri=puzzle/battleships/classic diakses pada tanggal 29 Desember 2009 pukul 21:11 WIB.

[3] *English Wikipedia* : http://en.wikipedia.org/wiki/Battleship_(puzzle) http://en.wikipedia.org/wiki/Depth-first_search diakses pada tanggal 29 Desember 2009 pukul 21:13 WIB.