PENERAPAN ALGORITMA RUNUT-BALIK (BACKTRACKING) DALAM PENYELESAIAN PERMAINAN SUDOKU

Sibghatullah Mujaddid

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha No. 10, Bandung
e-mail: if17124@students.if.itb.ac.id

ABSTRAK

Algoritma runut-balik (backtracking) adalah algoritma yang berbasis pada DFS untuk mencari persoalan secara lebih mangkus. Runut-balik (*backtracing*), yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Hanya saja algoritma ini merupakan pencarian yang mengarah ke solusi yang dipertimbangkan saja.

Algoritma runut-balik banyak digunakan pada program permainan (*game*) dan masalah pada bidang kecerdasan buatan. Salah satu jenis permainan yang dapat diselesaikan dengan algoritma runut-balik (*backtracking*) adalah permainan Sudoku.

Pada makalah ini, penulis akan membahas tentang penerapan algoritma runut-balik (*backtracking*) dalam penyelesaian permainan sudoku. Dengan menggunakan algoritma runut-balik, solusi dapat ditemukan lebih cepat tanpa harus mencoba semua kemungkinan solusi.

Melalui pembahasan dalam makalah ini, algoritma runut-balik (*backtracking*) digunakan pada saat program diminta untuk menyelesaikan permainan dengan mengisikan angka–angka tertentu yang memenuhi fungsi batasan pada kotak–kotak yang masih kosong (belum diisi atau bernilai nol). Jika dalam proses pengisian ternyata terjadi ketidaksinambungan maka akan dilakukan proses runut-balik (*backtracking*).

Kata kunci: runut-balik, *backtracking*, sudoku .

1. PENDAHULUAN

Melakukan permainan atau bermain game merupakan hal yang menarik bagi hampir seluruh masyarakat di dunia ini. Sebagian masyarakat tersebut bermain game hanya untuk melepas lelah, dan ada juga yang ingin bersenang-senang, dan ada juga yang hanya mengisi waktu kosong, serta ada yang menganggapnya sebagai lahan untuk mengolah otak. Apalagi dengan berkembangnya teknologi di masyarakat ini, semakin berkembang juga jenis permainan yang ada pada saat ini. Dan permainan yang sering dilakukan masyarakat pada jaman ini adalah permainan yang dilakukan di komputer.

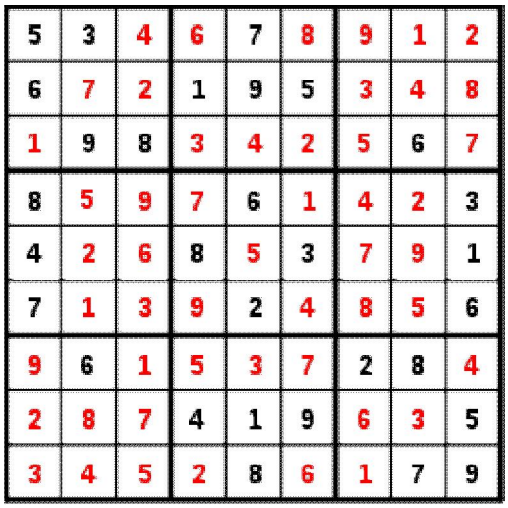
Salah satu permainan tersebut adalah Sudoku. Sudoku adalah permainan teka-teki angka berbasis logika yang pertama kali didesain oleh seorang arsitek berkebangsaan Amerika Serikat, Howar Garns, pada tahun 1979. Barulah kemudian permainan ini menjadi populer di Jepang. Kata "Sudoku" sendiri merupakan singkatan dari sebuah frase kalimat dalam bahasa Jepang, "*Suuji wa dokushin ni kagiru*", yang berarti "angka-angkanya harus tetap tunggal".

1.1 Deskripsi Umum Permainan Sudoku

Sudoku paling umum berbentuk matriks 9x9 (n^2 x n^2 dengan n=3). Aturan permainannya sederhana, isi semua matriks sampai penuh, dengan catatan, untuk setiap kolom, baris, maupun submatriks berukuran 3x3 (n x n) hanya boleh terisi dengan angka 1 sampai 9 yang berjumlah masing-masing satu. Sesuai namanya, maka tidak boleh ada angka/digit yang sama.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Gambar 1. Contoh permainan sudoku, berupa matriks yang masih belum terisi penuh. (Sumber: http://en.wikipedia.org/wiki/Sudoku)



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Gambar 2. Contoh permainan sudoku, yang merupakan penyelesaian dari sudoku pada gambar 1. (Sumber: http://en.wikipedia.org/wiki/Sudoku)

Kesulitan tingkat permainan Sudoku, selain diukur dari besar ukuran matriksnya, biasanya diukur dari masalah perumusan angka-angka awal yang telah diatur posisi dan nilainya. Semakin sedikit angka-angka awal yang diberikan, tentunya akan semakin sulit.

Untuk memecahkan teka-teki Sudoku, dapat digunakan algoritma runut-balik (*backtracking*). Algoritma ini berbasis pada DFS (*Depth First Search*), dimana solusi dapat ditemukan dengan penelusuran yang lebih sedikit dan dapat mencari solusi permasalahan secara lebih mangkus. Algoritma runut-balik sendiri merupakan perbaikan dari algoritma *brute-force* dan DFS.

1.2 Strategi Umum Penyelesesain Sudoku

Secara umum, Sudoku dapat diselesaikan dengan kombinasi teknik pemindaian (*scanning*), penandaan (*marking*), dan analisa (*analyzing*). Beberapa teki-teki Sudoku yang tergolong mudah dapat diselesaikan hanya dengan salah satu proses, namun pada umumnya kita harus mengkombinasikan ketiga teknik tersebut.

a. Pemindaian

Berupa proses memindai baris atau kolom untuk mengindentifikasi baris mana dalam suatu blok yang terdapat angka-angka tertentu. Proses ini kemudian diulang pada setiap kolom (atau baris) secara sistematis. Kemudian menentukan nilai dari suatu sel dengan membuang nilai-nilai yang tidak mungkin.

b. Penandaan

Berupa analisa logika, dengan menandai kandidat angka yang dapat dimasukkan dalam sebuah sel.

c. Analisa

Berupa eliminasi kandidat, dimana kemajuan dicapai dengan mengeliminasi kandidat angka secara berturut-turut hingga sebuah sel hanya punya 1 kandidat.

2. METODE

2.1 Deskripsi Umum Algoritma Runut-Balik

Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Runut-balik (*backtracking*) merupakan perbaikan dari algoritma *brute-force* yang secara sistematis akan melakukan pencarian solusi permasalahan di antara semua kemungkinan solusi yang ada. Dengan metode ini, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang akan dipertimbangkan. Akibatnya, waktu pencarian dapat dihemat. Runut-balik lebih alami dinyatakan dalam algoritma rekursif. Kadang-kadang disebutkan pula bahwa runut balik merupakan bentuk tipikal dari algoritma rekursif.

Untuk memfasilitasi pencarian ini, maka ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai – nilai x. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Seluruh lintasan dari dari akar ke daun membentuk ruang solusi.

Algoritma Backtracking membentuk sebuah pohon ruang status selama prosesnya Struktur pohon inilah, yang juga merupakan sebuah graf tak berarah, yang ditraversal dengan prinsip DFS (*Depth First Search*). Simpul-simpul pada pohon ruang status yang tidak mengarah ke solusi maka akan "dimatikan". Sedangkan simpul-simpul pohon ruang status yang masih mengarah ke solusi maka akan terus berkembang. Pematian simpul pohon ruang status yang tidak mengarah kepada solusi ini sering disebut dengan istilah *prunning*. Dengan demikian, seluruh lintasan dari akar ke daun yang melalui simpul-simpul yang tidak "dimatikan" akan membentuk sebuah ruang solusi.

2.2 Prinsip Pencarian Solusi dengan Metode Runut-Balik

Di sini kita hanya akan meninjau pencarian solusi pada pohon ruang status yang dibangun secara dinamis. Langkah-langkah pencarian solusi adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah

mengikuti metode pencarian mendalam (*DFS*). Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live mode*). Simpul hidup yang sedang diperluas dinamakan **simpuel-E** (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.

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 "dimatikan" 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 hidup terdekat (simpul orangtua). Selanjutnya simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

2.3 Penerapan Algoritma Runut-Balik dalam Penyelesaian Sudoku

Penerapan algoritma runut-balik dalam penyelesaian Sudoku adalah sebagai berikut:

1. Algoritma dimulai pada elemen kosong pertama pada matriks.
2. Periksa seluruh kemungkinan angka yaitu n (dengan $n \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$) yang dapat diisi oleh elemen kosong tersebut dengan memeriksa batasan (dengan fungsi pembatas).
3. Jika nilai n tersebut memenuhi fungsi pembatas (*valid*), maka elemen kosong tersebut diisi dengan nilai n, dan lanjutkan pemeriksaan ke elemen kosong berikutnya.
4. Jika nilai n tersebut tidak memenuhi fungsi pembatas, maka uji dengan nilai n lain.
5. Jika seluruh nilai n telah diuji dan tidak ada nilai n yang memenuhi fungsi pembatas, maka lakukan backtracking ke elemen sebelumnya.
6. Elemen ini akan diuji lagi dengan nilai n baru berdasarkan fungsi pembatas.
7. Lakukan cara yang sama dengan poin nomor 3.
8. Proses diatas akan dilakukan terus-menerus secara rekursif hingga ditemukan suatu solusi atau tidak ditemukan suatu solusi.
9. Suatu solusi dinyatakan benar jika seluruh elemen kosong telah diisi dengan nilai *valid*.

Pseudo-code algoritma runut-balik (*backtracking*) untuk penyelesaian Sudoku adalah sebagai berikut:

1. Deklarasi konstanta umum

```
{ Kamus global }
Deklarasi
   const NOL : integer = 0
   const NBARIS : integer = 9
   const NKOLOM : integer = 9
   const NKOTAK : integer = 3
   type matriks : array[1..NBARIS,
        1..NKOLOM] of integer
   CELL : matriks
```

Bagian ini merupakan deklarasi konstanta umum yang akan digunakan.

- Konstanta NOL menyatakan nilai 0 atau suatu elemen belum memiliki nilai *valid* yaitu $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Konstanta NBARIS menyatakan jumlah baris atau ukuran baris pada matriks.
- Konstanta NKOLOM menyatakan jumlah kolom atau ukuran kolom pada matriks.
- Konstanta NKOTAK menyatakan ukuran baris dan kolom untuk sub matriks (kotak kecil).

Pada kasus ini, matriks Sudoku berukuran 9x9 sehingga ukuran sub matriks (kotak kecil) berukuran 1/3 yaitu 3x3.

2. Implementasi fungsi untuk mencari tempat/elemen kosong

```
function CariTempatKosong (output baris,
   kolom : integer) → boolean

Algoritma:
   for baris ← 1 to NBARIS do
      for kolom ← 1 to NKOLOM do
         if CELL[baris][kolom] = NOL then
            return true
         endif
      endfor
   endfor
   return false
```

Fungsi ini digunakan untuk mencari elemen kosong berikutnya pada matriks `CELL`. Elemen kosong dinyatakan sebagai elemen yang bernilai 0.

3. Implementasi fungsi pembatas

```
function IsTempatValid(input baris, kolom,
   num : integer) → boolean
```

```
Algoritma:
   for kol ← 1 to NKOLOM do
      if CELL[baris][kol] = num then
         return false
      endif
   endfor

   for bar ← 1 to NBARIS do
      if CELL[kolom][bar] = num then
         return false
      endif
   endfor

   for bar ← 1 to NKOTAK do
      for kol ← 1 to NKOTAK do
         if CELL[bar + baris - baris mod
            NKOTAK][kol + kolom - kolom mod
            NKOTAK] = num then
               return false
         endif
      endfor
   endfor

   return true
```

Fungsi ini merupakan fungsi pembatas bagi penyelesaian solusi. Fungsi pembatas ini berdasarkan batasan-batasan dalam Sudoku yaitu:

a. Tidak ada angka yang sama dalam satu baris.
b. Tidak ada angka yang sama dalam satu kolom.
c. Tidak ada angka yang sama dalam satu kotak kecil (sub matriks).

Fungsi diatas akan memeriksa apakah suatu angka/nilai `num` memenuhi batasan-batasan dalam Sudoku.

Pertama akan diperiksa apakah angka `num` berada pada baris `baris`.

```
   for kol ← 1 to NKOLOM do
      if CELL[baris][kol] = num then
         return false
      endif
   endfor
```

Jika angka `num` ditemukan maka fungsi akan mengembalikan nilai `false`..

Kemudian akan diperiksa apakah angka `num` berada pada kolom `kolom`.

```
   for bar ← 1 to NBARIS do
      if CELL[kolom][bar] = num then
         return false
      endif
   endfor
```

Jika angka `num` ditemukan maka fungsi akan mengembalikan nilai `false`.

Terakhir akan diperiksa apakah angka `num` berada dalam satu kotak kecil yang sama dengan baris `baris` dan kolom `kolom`.

```
   for bar ← 1 to NKOTAK do
      for kol ← 1 to NKOTAK do
         if CELL[bar + baris - baris mod
            NKOTAK][kol + kolom - kolom mod
            NKOTAK] = num then
               return false
         endif
      endfor
   endfor
```

Jika angkaa `num` ditemukan maka fungsi akan mengembalikan nilai `false`.

Jika setelah semua batasan telah diperiksa dan semuanya memenuhi, maka fungsi akan mengembalikan nilai `true`.

4. Implementasi fungsi utama backtracking

```
function SolveSudoku() → boolean

{ Kamus lokal }
Deklarasi
   baris : integer
   kolom : integer
   num : integer

Algoritma:
   if CariTempatKosong(baris, kolom) =
      false then
         return true
   endif

   for num ← 1 to 9 do
      if IsTempatValid(baris, kolom, num) =
         true then
            CELL[baris][kolom] = num;
            if SolveSudoku() = true then
               return true
            endif
            CELL[baris][kolom] = NOL
      endif
   endfor
   return false
```

Fungsi ini merupakan fungsi utama yaitu sebagai backtracking dalam penyelesaian Sudoku. Fungsi ini menggunakan dua fungsi sebelumnya yaitu fungsi `CariTempatKosng` dan fungsi `IsTempatValid`.

Pertama fungsi akan mencari elemen kosong pada matriks.

```
   if CariTempatKosong(baris, kolom) =
      false then
         return true
```

```
    endif
```

Jika tidak ditemukan elemen kosong, maka fungsi akan mengembalikan nilai `true`, fungsi tak perlu dilanjutkan.

Jika ditemukan elemen kosong, lakukan pengujian untuk angka-angka $num \in \{1,2,3,4,5,6,7,8,9\}$ dengan memeriksa batasan untuk num dengan fungsi pembatas `IsTempatValid`.

```
    if IsTempatValid(baris, kolom, num) =
       true then
          ...
    endif
```

Jika `num` valid maka isi elemen kosong tersebut dengan nilai `num` dan teruskan pengujian untuk elemen kosong berikutnya.

```
          CELL[baris][kolom] = num;
          if SolveSudoku() = true then
                ...
          Endif
```

Jika sukses, maka fungsi akan mengembalikan nilai `true`.

```
                return true
```

Jika gagal, maka kosongkan kembalik elemen tadi dan uji dengan angka lain.

```
          CELL[baris][kolom] = NOL
```

Jika setelah semua angka telah diuji dan tidak ada yang memenuhi, lakukan proses *backtracking* dengan mengembalikan nilai `false` pada fungsi sekarang. Nilai `false` ini akan memicu (*trigger*) backtracking pada elemen sebelumnya.

```
    for num ← 1 to 9 do
       ...
    endfor
    return false
```

Proses ini akan terus berulang hingga seluruh elemen kosong telah diperiksa dan diuji.

3. KESIMPULAN

Algoritma runut-balik (*backtracking*) merupakan algoritma yang cukup mangkus untuk menyelesaikan berbagai persoalan. Hal ini disebabkan pada algoritma runut-balik, tidak perlu mencari semua kemungkinan solusi untuk diuji. Hanya kemungkinan solusi yang mengarah pada solusi yang dipertimbangkan saja yang perlu diuji, sehingga algoritma ini cukup mangkus untuk digunakan. Berbeda dengan algoritma *brute-force* yang mencari semua kemungkinan solusi untuk diuji, sehingga tidak mangkus. Oleh karena itu, algoritma runut-balik banyak diterapkan dalam berbagai persoalan, terutama program game dan persoalan *artificial intelligence*.

Hasil analisis penggunaan algoritma runut-balik dalam menyelesaikan persoalan pengisian angka-angka Sudoku menunjukkan bahwa algoritma ini cukup mangkus untuk mendapatkan solusi persoalan tersebut. Sistem kerja algoritma runut-balik yang sistematis dan ciri khasnya yang hanya memeriksa kemungkinan solusi yang memang dapat dipertimbangkan untuk menjadi solusi akhir, diperkirakan dapat menjadi solusi yang efektif dan efisien untuk persoalan ini.

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Teknik Informatika, 2006.

[2] http://en.wikipedia.org/wiki/Sudoku diakses pada Jum'at, 01 Januari 2010.

[3] http://id.wikipedia.org/wiki/Sudoku diakses pada Jum'at, 01 Januari 2010.