PENGUNAAN DUA VERSI ALGORITMA BACKTRACK DALAM MENCARI SOLUSI PERMAINAN SUDOKU

Aditia Dwiperdana – 13505014

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
deepest_peak@yahoo.com

ABSTRAK

Makalah ini membahas tentang algoritma *backtrack* dan penggunaannya dalam mencari solusi permainan sudoku. Algoritma *backtrack* adalah algoritma pencari solusi yang berdasarkan kepada algoritma DFS atau Depth First Search. Namun pada algoritma *backtrack* ini terdapat suatu nilai batas yang dapat menentukan apakah suatu simpul mengarah menuju solusi atau tidak.

Sudoku adalah permainan yang memiliki ruang solusi yang sangat banyak, yaitu sebanyak (n^{n*n}-1) / (n-1). Jumlah ruang solusi tersebut didapat dari rumus deret geometrik. Dengan ruang solusi sebanyak itu, maka mencari solusi dengan algoritma *brute force* akan sangat tidak mangkus. Begitu pula dengan metode exhaustive search.

Dalam makalah ini akan digunakan dua versi algoritma *backtrack*, dimana keduanya berbeda pada fungsi pembatasnya, fungsi maju ke elemen berikutnya, dan fungsi mundur(*backtrack*). Algoritma *backtrack* versi pertama menggunakan pendekatan pengisian berurutan dalam mencari solusi, yaitu dimulai dari kotak kiri atas sampai ke kotak kanan bawah. Sedangkan versi kedua menggunakan pendekatan kepastian, yaitu mengisi mulai dari kotak yang memiliki kemungkinan jawaban paling sedikit.

Kata Kunci : Sudoku, DFS, *Backtrack*

1. PENDAHULUAN

Banyak sekali jenis permainan yang membutuhkan pemainnya untuk berpikir keras dalam menemukan jawaban. Umumnya permainan-permainan tersebut memiliki aturan yang jelas, sehingga dapat dipecahkan dengan logika. Sudoku adalah salah satu permainan yang membutuhkan pemainnya memeriksa aturan setiap melakukan pengisian jawaban. Permainan sudoku dan aturan-aturannya akan dijelaskan di bagian selanjutnya.

Untuk mencari solusi permainan sudoku dengan komputasi, dapat digunakan berbagai algoritma pencarian solusi. Diantaranya adalah algoritma *backtrack*. Dengan menggunakan algoritma ini, pencarian solusi dapat dilakukan dengan mangkus. Algoritma *backtrack* adalah algoritma DFS atau Depth First search yang memiliki fungsi pembatas. Algoritma DFS dan algoritma *backtrack* akan dijelaskan pada bagian lain.

2. DASAR TEORI

2.1 Sudoku

Sudoku adalah suatu permainan teka-teki yang berasal dari Jepang. Permainan ini mengharuskan pemain mengisi sejumlah kotak-kotak kecil sebanyak n^2 x n^2 dengan angka dari 1 sampai n^2 dengan memenuhi syarat permainan. Syarat pengisian kotak-kotak tersebut adalah :

1. Tidak boleh ada angka yang berulang dalam suatu **baris**, dan dalam satu baris harus ada angka 1 sampai n^2.
2. Tidak boleh ada angka yang berulang dalam suatu **kolom**, dan dalam satu kolom harus ada angka 1 sampai n^2.
3. Tidak boleh ada angka yang berulang dalam **kotak besar**, dan dalam satu kotak besar harus ada angka dari 1 sampai n^2.
4. Sebuah kotak besar adalah kumpulan n^2 kotak kecil yang membentuk kotak yang lebih besar. Dalam permainan terdapat n^2 kotak besar.

			3
2			
1	2		

Gambar 1. Contoh papan Sudoku

			3
2			
1	2		3

Gambar 2. Contoh masukan jawaban yang salah

4	1	2	3
2	3	4	1
3	4	1	2
1	2	3	4

Gambar 3. Contoh papan sudoku yang sudah dipecahkan

Permainan sudoku ini memiliki ruang solusi yang sangat banyak, yaitu $(n^{n*n}-1) / (n-1)$. Jadi untuk papan sudoku dengan sisi 4 akan memiliki 1.431.655.765 simpul dalam ruang solusi. Sedangkan untuk papan sudoku dengan sisi 9 memiliki lebih dari 10^{77} simpul dalam ruang solusi.

2.2 Depth First Search

DFS adalah algoritma yang digunakan dalam melakukan traversal dalam sebuah graf, yang dalam masalah kita adalah graf ruang solusi. Misalkan penelusuran dimulai dari simpul A. Simpul berikutya yang dikunjungi adalah simpul B yang bertetangga dengan A. Lalu penelusuran dimulai lagi secara rekursif dari simpul B. Ketika mencapai simpul X sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul Y yang belum dikunjungi. Pencarian mendalam dimulai lagi dari simpul Y. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Algoritma DFS

```
procedure DFS(input v:integer)
Deklarasi:
  w:integer
Algoritma:
  Write(v)
  dikunjungi[v]<-true
  for tiap simpul w yang bertetangga dengan
      simpul v do
    if not dikunjungi then
      DFS(w)
    endif
  endfor
```

2.3 *Backtrack*

Backtrack adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Selain itu, *backtrack* merupakan perbaikan dari algoritma *brute force* , secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dalam metode ini, kita tidak perlu memeriksa semua kemungkinan solusi yang ada, namun hanya yang mengarah pada solusi saja yang dipertimbangkan. Dengan begitu, waktu pencarian dapat dihemat.

Properti metode runut-balik:

a. Solusi persoalan

Solusi dinyatakan sebagai vektor dengan n-tuple :

X={x1,x2,…,xn} x adalah elemen berhingga dari S

b. Fungsi Pembangkit nilai xk

Dinyatakan sebagai :

T(k)

T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi

c. Fungsi Pembatas

Dinyatakan sebagai :

B(x1,x2,…,xk)

Fungsi pembatas menentukan apakah (x1,x2,…,xk) mengarah ke solusi. Jika ya, maka pembangkitan nilai untik xk-1 dilanjutkan, tapi jika tidak, maka (x1,x2,…,xk) dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi. Fungsi pembatas tidak selalu dinyatakan sebagai fungsi matematis, dapat juga dinyatakan sebagai fungsi sebagai predikat *true* atau *false*, atau dalam bentuk lain yang ekivalen.

Langkah-langkah pencarian solusi :

a. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah metode pencarian mendalam atau DFS. Simpul-simpul yang sudah dibangkitkan disebut simpul hidup, Simpul hidup yang sedang diperluas disebut simpul-E. Simpul dinomori dari atas ke bawah sesuai urutan kelahirannya.
b. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpu-E tersebut “dibunuh” sehingga menjadi simpul mati. Fungsi yang digunakan untuk “membunuh” simpul-E adalah dengan menerapkan fungsi pembatas. Simpul yang sudah mati tidak akan pernah diperluas lagi.
c. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak ayng 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 kenmbali sampai lintasan tersebut membentuk solusi.
d. Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

3. ALGORITMA VERSI 1

Dalam algoritma versi 1 ini, ada beberapa hal penting yang akan dijelaskan, yaitu :

a. Pengisian jawaban dilakukan secara berurutan dari kotak kiri atas sampai kotak kanan bawah. Dalam aplikasinya, digunakan metode iterasi yang lebih mudah digunakan.
b. Jawaban yang dimasukkan selalu merupakan jawaban yang valid, yaitu valid terhadap baris, kolom, dan kotak besar dari kotak yang bersangkutan. Jawaban yang dimasukkan mulai dari jawaban valid yang paling kecil.
c. Jika pengisian jawaban gagal dilakukan dan harus terjadi runut-balik, maka mundur sampai kotak sebelumnya yang memiliki nilai valid untuk dimasukkan.
d. Pencarian solusi dinyatakan berhasil jika mencapai kotak kanan bawah.

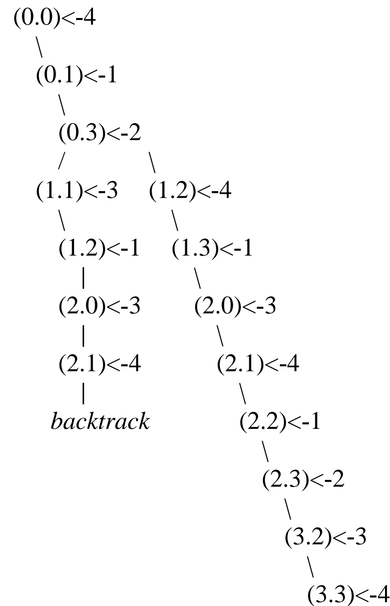
e. Pencarian solusi dinyatakan gagal jika terjadi runut balik sampai melewati kotak kiri atas.

Algoritma versi 1 dalam bahasa Java

```
public static void backtrack(Point p){
  int brs=p.x;
  int klm=p.y;
  int val;
  boolean ansFound,nextFound;
  counter=0;
  while(!stop){
    counter++;
    val=matrix.getGridValue(brs,klm);
    if(val==0){
      matrix.generateValid(brs,klm);
      if(!matrix.isEmptyValidGrid(brs,klm)){
        matrix.setGridValue(matrix.
getGridValid (brs, klm), brs, klm);
        ansFound=true;
      }else{
        ansFound=false;
      }
    }else{
      if(!matrix.isEmptyValidGrid(brs,klm)){
        matrix.setGridValue(matrix.
getGridValid(brs, klm), brs, klm);
       ansFound=true;
      }else{
        ansFound=false;
      }
    }
    nextFound=false;
    if(ansFound){while(brs<matrix.getSize()
*matrix.getSize()&&!nextFound){
      klm++;
      if(klm==matrix.getSize()
*matrix.getSize()){
         klm=0;
         brs++;
         if(brs==matrix.getSize()
*matrix.getSize()){
           nextFound=true;
           solusi=true;
           stop=true;
         }
       }
       if(!nextFound&&matrix.
getGridLock(brs, klm)==0){
         nextFound=true;
       }
     }
   }else{
      matrix.setGridValue(0, brs, klm);
      while(brs>=0&&!nextFound){
         klm--;
         if(klm<0){
             klm=matrix.getSize()
*matrix.getSize()-1;
             brs--;
             if(brs<0){
                 nextFound=true;
                 solusi=false;
                 stop=true;
             }
          }
          if(!nextFound&&matrix.
getGridLock(brs, klm)==0){
               nextFound=true;
          }
      }
```

```
      }
    }
  }
```

Pohon solusi yang dihasilkan dengan alrogitma versi 1 ini untuk contoh papan sudoku pada Gambar 1 adalah sebagai berikut



Gambar 4. Pohon solusi untuk algoritma versi 1

Dari berbagai percobaan dengan berbagai ukuran matriks dan berbagai kerumitan soal, algoritma ini mampu menyelesaikan semua persoalan. Namun waktu yang diperlukan untuk memecahkan masalah masih cukup lama. Sehingga perlu dilakukan optimasi pada algoritmanya. Optimasi tersebut direalisasikan pada algoritma versi 2.

4. ALGORITMA VERSI 2

Algoritma ini merupakan optimasi dari algoritma versi 1, yaitu dengan mengubah beberapa pendekatan dalam mencari solusi. Hal yang perlu diperhatikan dari algoritma versi 2 ini adalah :

a. Pengisian jawaban dilakukan untuk kotak dengan kemungkinan nilai valid terkecil. Kemungkinan nilai valid ini dibangkitkan setiap iterasi *backtrack*.

b. Jawaban yang dimasukkan selalu merupakan jawaban yang valid, yaitu valid terhadap baris, kolom, dan kotak besar dari kotak yang bersangkutan. Jawaban yang dimasukkan mulai dari jawaban valid yang paling kecil.

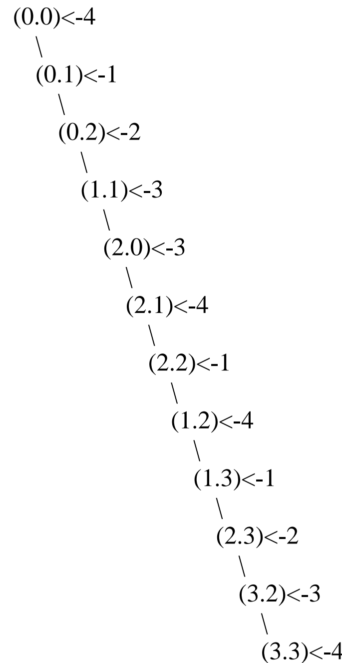
c. Setiap pengisian kotak, kotak yang bersangkutan dimasukkan ke dalam stack simpulHidup.

d. Jika pengisian berhasil dilakukan, pengisian selanjutnya dilakukan untuk kotak dengan kemungkinan nilai valid terkecil.
e. Jika pengisian jawaban menyebabkan ada kotak kosong yang memiliki kemungkinan nilai validnya nol, maka harus dilakukan *backtrack*.
f. Pencarian dihentikan jika tidak ada lagi kotak kosong yang tersisa atau stack simpul hidup menjadi kosong.

Algoritma Versi 2 dalam bahasa Java

```
public int backtrack(Point p){
  int brs=p.x;
  int klm=p.y;
  int val;
  int counter=0;
  Point temP;
  boolean ansFound;
  boolean stop=false;
  Stack<Point> simpulHidup=new Stack
    <Point>();
  while(!stop){
    counter++;
    val=getValueGrid(brs,klm);
    if(val==0){
      generateValid(brs,klm);
      setValueGrid(brs, klm, getGrid(brs,
klm).getNextVal());
      if(getMinNum()==0){
         ansFound=false;
      }else{
         ansFound=true;
         simpulHidup.push(new
           Point(brs,klm));
      }
    }else{
      if(!isEmptyValidGrid(brs, klm)){
         setValueGrid(brs, klm, getGrid(brs,
           klm).getNextVal());
         if(getMinNum()==0){
           ansFound=false;
         }else{
            ansFound=true;
            simpulHidup.push(new Point(brs,
klm));
         }
       }else{
          ansFound=false;
       }
     }
     if(ansFound){
        if(!isFull()){
           brs=getMinPoint().x;
           klm=getMinPoint().y;
        }else{
           stop=true;
        }
     }else{
        setValueGrid(brs,klm,0);
        if(simpulHidup.empty()){
          stop=true;
        }else{
           temP=simpulHidup.pop();
           brs=temP.x;
           klm=temP.y;
        }
     }
  }
  return counter;
}
```

Pohon solusi yang dihasilkan dengan alrogitma versi 2 ini untuk contoh papan sudoku pada Gambar 1 adalah sebagai berikut



Gambar 5. Pohon solusi untuk algoritma versi 2

Terlihat dari pohon solusi yang dihasilkan, algoritma ini lebih mangkus ketimbang algoritma sebelumnya. Perlu diperhatikan bahwa tidak terjadi runut-balik pada pohon solusi diatas, karena algoritma versi 2 ini sudah memperkecil kemungkinan terjadi runut-balik.

5. KESIMPULAN

Dari hasil percobaan dapat disimpulkan bahwa oprtimasi yang dilakukan terhadap algoritma versi 1 yang diterapkan pada algoritma versi 2 berhasil dan dapat memecahkan persoalan sudoku dengan lebih mengkus.

REFERENSI

[1] Munir,Rinaldi. “Strategi Algoritmik”. 2007.

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.