PENGGUNAAN ALGORITMA *BACKTRACKING* DALAM PENYELESAIAN PERMAINAN *SUDOKU*

Wahyu Adhi Arifiyanto (13505024)

Program Studi Teknik Informatika ITB
Alamat : Jl. Ganesha 10, Bandung
e-mail: if15024@students.if.itb.ac.id

ABSTRAK

Komputer diciptakan untuk mengerjakan beragam pekerjaan komputasi. Seiring dengan perkembangan teknologi, fungsi komputer juga mengalami perkembangan untuk dapat menjalankan aplikasi *game*. *Game* menjadi sangat beragam jenisnya dan populer karena minat dari *user* sendiri. *Game* tadinya hanya dimainkan untuk sekedar melepaskan penat dari pekerjaan. Tetapi sekarang *game* menjadi sangat digemari karena beragamnya jenis aplikasi yang ditawarkan. *User* bisa memilih aplikasi *game* yang disukai berdasarkan kriteria yang diinginkan, misalnya petualangan, strategi, dan lain-lain.

Pada makalah ini, penulis mencoba membahas tentang *game "Sudoku"* yang memungkinkan *user* untuk memasukkan angka yang tepat pada kotak – kotak yang telah tersedia dan menganalisa penerapan algoritma *backtracking* sebagai salah satu cara penyelesaian persoalan pada *game* ini.

Melalui pembahasan dalam makalah ini, algoritma *backtracking* (runut balik) 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 *user*). Jika dalam proses pengisian ternyata terjadi ketidaksinambungan maka akan dilakukan proses *backtracking* (runut balik).

Kata kunci: *game*,*sudoku*,*backtracking*.

1. PENDAHULUAN

Permainan atau *game* sudah merupakan kegiatan yang hampir disukai oleh masyarakat di seluruh penjuru dunia. Dengan berkembangnya berbagai bentuk *game*, baik berupa *game* yang tradisional maupun yang sudah menggunakan teknologi yang canggih, *game* menjadi sangat populer di berbagai tempat.

Salah satu bentuk *game* yang sudah tidak asing lagi adalah *game* komputer. *Game* disini merupakan aplikasi yang cukup diminati sebagai ajang *refreshing* bahkan olah otak. Bahkan tidak jarang, aplikasi *game* pada komputer menimbulkan kecanduan bagi *user* sehingga mengembangkannya sebagai hobi yang menantang.

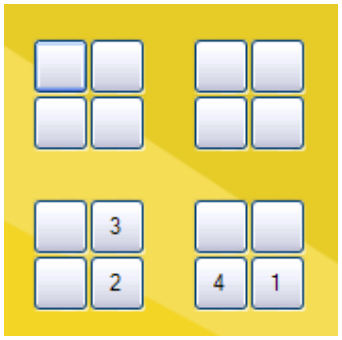
Salah satu aplikasi *game* yang sering dijumpai adalah *Game* "*Sudoku*". *Game* ini pada dasarnya merupakan suatu aplikasi *game* yang memungkinkan *user* mengisi angka pada kotak – kotak yang telah tersedia berdasarkan fungsi batasan yang dimiliki *game* ini.

Pada makalah ini, analisa penulis berkenaan dengan pengisian angka - angka ke dalam kotak yang telah disediakan oleh program sebagai *player*. Dalam proses ini, algoritma yang dipergunakan adalah algoritma *backtracking* (runut balik) yang menangani ketidaksinambungan yang dapat terjadi jika memasukkan angka pada kotak yang salah.

2. METODE

2.1 Deskripsi Umum *Game Sudoku*

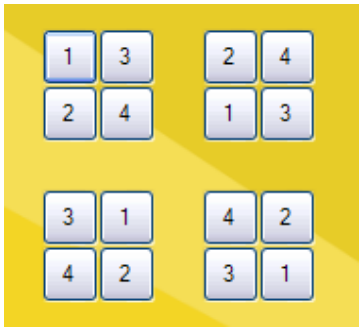
Sudoku merupakan salah satu permainan yang sangat populer. Pada permainan ini akan terdapat kotak besar yang disusun oleh kotak-kotak kecil. Kota kecil akan tersusun oleh n^2 *space* sebagai tempat untuk memposisikan angka. Kemudian kotak – kotak kecil terebut akan menyusun kotak besar sehingga ukuran kotak besar adalah n^2 x n^2.



Gambar 1.Ilustrasi pada tengah permainan

Objektif dari permainan ini adalah bagaimana kita bisa mengatur susunan angka – angka yang telah ditentukan kedalam suatu *space* pada kotak, dengan syarat bahwa angka pada kolom dan baris kotak besar tidak boleh sama demikian juga angka pada satu kotak kecil tidak boleh sama.

Cara penyelesaian permainan ini dirancang dengan menggunakan algoritma *backtracking* dalam menemukan susunan angka yang sesuai untuk mengisi *space* – *space* yang ada sesuai dengan batasan yang telah dijelaskan sebelumnya, yaitu angka pada satu baris, satu kolom dan satu kotak kecil tidak boleh sama.



Gambar 2.Ilustrasi pada akhir permainan

2.2 Deskripsi Umum Algoritma *Backtracking*

Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus. Runut-balik merupakan perbaikan dari algoritma *exhaustive-search* yang secara sistematis akan melakukan pencarian solusi permasalahan di antara 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.

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.

2.3 Algoritma *Backtracking* Sebagai Pemecahan Solusi

2.3.1 Konsep Dasar

Pada permainan ini algoritma *backtracking* menjadi komponen dalam penerapan intelegensia buatan kepada program, sehingga program dapat menyelesaikan permainan dengan mendapatkan suatu solusi. Program akan mengisi tiap kotak dengan nilai tertentu yang dirasa tepat, kemudian akan melanjutkan mengisi kotak berikutnya atau akan kembali ke kotak sebelumnya jika tidak mengarah ke solusi yang memungkinkan.

Dalam implementasinya , proses pencarian pada algoritma *backtracking* sebenarnya identik dengan penggunaan algoritma *exhaustive-search* namun dengan proses yang lebih sistematis. Secara garis besar langkah – langkah yang dilakukan oleh program dalam pencarian solusi adalah sebagai berikut :

1. Menguji sembarang nilai (0 < nilai < n+1) dengan fungsi pembatas yang ada.
2. a. Jika nilai tersebut memenuhi fungsi pembatas maka kotak akan terisi dengan nilai uji tersebut kemudian proses akan berpindah ke kotak berikutnya.
   b. Jika nilai tersebut tidak memenuhi fungsi pembatas, maka program akan mengambil nilai lain yang belum dipakai.
3. Jika semua opsi nilai pada domain nilai telah dicoba dan tidak ada satupun yang memenuhi fungsi pembatas, maka terjadi *backtracking* ke kotak sebelumnya.

4. Pada kotak hasil *backtracking* ini, program akan menguji fungsi pembatas dengan nilai baru.
5. a. Jika nilai baru tersebut dapat memenuhi fungsi pembatas, maka nilai tersebut akan diisikan ke dalam kotak. Kemudian proses pengisian akan berpindah ke kotak berikutnya.
   b. Jika semua opsi nilai baru yang ada pada domain tidak dapat memenuhi fungsi pembatas maka akan terjadi lagi *backtracking* ke kotak sebelumnya.
6. Program akan melakukan proses di atas secara rekursif sampai ditemukan penyelesaian atau tidak terdapat penyelesaian.
   a. Terdapat penyelesaian : program berhasil mengisi semua kotak kosong yang tersedia dengan nilai yang memenuhi fungsi pembatas.
   b. Tidak terdapat penyelesaian : setelah beberapa saat mencari solusi, ternyata program melakukan *backtracking* hingga kembali ke kotak pertama dan saat nilai indeks baris = -1 menandakan bahwa program tidak berhasil menemukan solusi.

2.3.2 Implementsi domain solusi permasalahan

Himpunan ruang solusi terdiri :

{nilai| 0 < nilai < n+1; nilai ϵ integer}

dengan **n** adalah ukuran kotak kecil.

2.3.3 Implementsi fungsi –fungsi pembatas

Sebagaimana telah kami sampaikan di atas *game sudoku* mempunyai fungsi pembatas yang cukup menarik dan unik, yaitu :

a. angka pada satu baris tidak boleh sama,
b. angka pada satu kolom tidak boleh sama, dan
c. angka pada satu kotak kecil tidak boleh sama.

Berikut saya tampilkan source code dari ketiga fungsi pembatas tersebut, sengaja dituliskan dalam bahasa **java** untuk memudahkan pembaca bila ingin mencoba menjalankan ketiga fungsi tersebut tanpa perlu menerjemahkannya terlebih dahulu ke suatu bahasa pemrograman. Semoga dapat membantu dalam memahami fungsi batasan yang dimiliki permainan ini.

a. Cek Kolom.
   Spesifikasi : Pada suatu kolom tidak boleh ditemukan 2 (dua) nilai yang sama.

```
Boolean cekKolom (int x,int y,int nilai){
    // n adalah ukuran papan
    Boolean found1= true;
    int j = 0;
    while (found1 && (j < (n * n)))
    {
        if (j == y)
        {
                // do nothing
        }
        else{
              if (getSel(x, j) == nilai){
                found1 = false;
              }
        }
        j++;
    }
    Return found1;
}
```

b. Cek Baris.
   Spesifikasi : Pada suatu baris tidak boleh ditemukan 2 (dua) nilai yang sama.

```
Boolean cekBaris (int x,int y,int nilai){
    // n adalah ukuran papan
    Boolean found2= true;
    int i = 0;
    while (found2 && (i < (n * n)))
    {
        if (i == x)
        {
                // do nothing
        }
        else{
              if (getSel(i, y) == nilai){
                found1 = false;
              }
        }
        i++;
    }
    Return found2;
}
```

c. Cek Kotak Kecil.
Spesifikasi : Pada suatu kotak kecil tidak boleh ditemukan 2 (dua) nilai yang sama.

```
Boolean cekKotakKecil (int x,int y,int nilai){
    // n adalah ukuran papan
    Boolean found3= true;

    // [xbwh,ybwh]=koordinat dari bagian
    //             kiri atas kotak kecil
    // [xats,yats]=koordinat dari bagian
    //              kanan bawah kotak kecil

   int xtemp = (x % n);
   int xbwh  = (x - xtemp);
   int xats  = (xbwh + n - 1);

   int ytemp = (y % n);
   int ybwh  = (y - ytemp);
   int yats  = (ybwh + n - 1);

   i = xbwh; j = ybwh;

   while (found3 && i <= xats){
        while (j <= yats){
               if (i == x && j == y){
                 // do nothing
               }else{
                 if (getSel(i, j) == nilai){
                       found3 = false;
                 }
               }
               j++;
        }
        j = ybwh;
        i++;
   }

   Return found3;
}
```

2.3.4 Implementasi *backtracking*

Sebagaimana pada sub bagian implementasi fungsi batasan, berikut saya sertakan source code dari algoritma backtracking yang telah dituliskan dalam bahasa **java ,** semoga memudahkan pembaca dalam memahami algoritma tersebut.

```
void BackTrack(){

   //ArrBool [i,j]==true,artinya kotak tersebut
  //         telah diisi dengan masukan user

   nilai = 1;
   int i = 0; int j = 0;
   berhasil = true;

   while (i < (n * n) && i >= 0 && berhasil){
     while (j < (n * n) && berhasil){
       if (ArrBool[i, j] == false){
          while ((!(cekMemenuhi(i, j, nilai))) &&
                 nilai <= (n * n))){
               nilai++;
               setSel(i, j, nilai);
          }
          if (nilai > (n * n)){
               setSel(i, j, 0);
               do{
                  j--;
                  if (j < 0){
                       j += (n * n);
                       i--;
                       if (i < 0){
                          berhasil = false;
                          i++;
                       }
                  }
               }while (ArrBool[i, j] == true);
               nilai = getSel(i, j);
               setSel(i, j, 0);
               nilai++;
          }else{
               setSel(i, j, nilai);
               j++;
               nilai = 1;
          }
       }
       else{
            j++;
       }
     }
     i++;
     j = 0;
   }

   if (berhasil == false){
      output("Tidak ditemukan solusi!");
   } else {
      output("Solusi berhasil ditemukan!");
   }
}
```

3. KESIMPULAN

Algoritma *backtracking* merupakan algoritma yang cukup mangkus untuk menyelesaikan berbagai persoalan. Hal ini disebabkan karena pada prinsipnya, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Pencarian hanya mengarah pada solusi yang dipertimbangkan saja. Oleh karena algoritma ini cukup efektif, maka *backtracking* banyak diterapkan dalam berbagai program *game* dan persoalan yang berkaitan dengan bidang kecerdasan buatan (*artificial inteligence*).

Hasil analisis kemampuan algoritma *backtracking* dalam menyelesaikan persoalan pengisian *sudoku* menunjukkan bahwa algoritma ini cukup efektif untuk mendapatkan solusi persoalan tersebut. Sistem kerja algoritma *backtracking* yang sistematis dan ciri khasnya yang hanya memeriksa kemung-kinan solusi yang memang dapat dipertimbangkan untuk menjadi solusi akhir, diperkirakan dapat menjadi solusi yang efektif dan efisien untuk persoalan ini.

REFERENSI

[1] Ir. Rinaldi Munir,M.T , "Diktat kuliah IF2251 - Strategi Algoritmik", Informatika ITB, 2007.

[2] Ajeng Wirasati,Ronny Adry, "Teka – teki silang backtracking",2007.

[3] http://www.en.wikipedia.org/wiki/Backtracking