PENCARIAN SOLUSI DENGAN ALGORITMA *BACKTRACKING* UNTUK MENYELESAIKAN *PUZZLE* KAKURO

Oleh: Teguh Budi Wicaksono

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
if15028@students.if.itb.ac.id

ABSTRAK

Kakuro merupakan sejenis *puzzle* logika. Permasalahan dalam Kakuro merupakan permasalahan *integer*. Kakuro juga biasa disebut sebagai transliterasi matematis dari *puzzle* silang. *Puzzle* ini merupakan *puzzle* logika yang terpopuler di Jepang diantara berbagai *puzzle* yang diterbitkan oleh Nikoli setelah Sudoku. Tujuan dari *puzzle* ini adalah untuk mengisi setiap kolom yang tersedia dengan angka bulat dari 1 hingga 9 dengan beberapa syarat dalam pengisiannya. Oleh karena itu permasalahan ini dapat diselesaikan secara komputasi yaitu dengan menggunakan matriks. Walaupun biasanya *puzzle* ini dikerjakan secara manual dengan tangan. Pemecahan masalah ini dapat diselesaikan dengan algoritma DFS (*Depth Fisrt Search*), tapi dengan menggunakan metoda DFS masih menghasilkan komputasi yang lama, untuk itu perlu dikembangkan lagi dengan menggunakan algoritma *backtracking*, yaitu dengan memberikan sebuat acuan yang dapat mengeliminasi pohon yang tidak dapat menghasilkan solusi. Dengan begitu kalkulasi yang tidak perlu dapat dihindari sehingga mengurangi lamanya komputasi. Sehingga didapat algoritma yang lebih efisien.

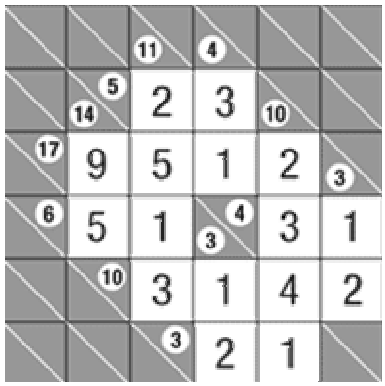
Kata kunci: Kakuro, sudoku, *Depth First Search*, *backtracking*

1 PENDAHULUAN

Kakuro adalah sebuah perminan *puzzle* yang bermula pada mulanya bernama *Cross Sums*. *Puzzle* pertama dikeluarkan pada tahun 1966 oleh Dell Magazine di AS. Sepuluh tahun kemudian, Dell Magazine memperkenalkan *puzzle* Sudoku pada dunia.

Maki Kaji, presiden dari Nicoli *Puzzle*, pada tahun 1980 membawa masuk *Cross Game* ke Jepang. Maki Kaji memberi nama *puzzle* ini Kasan Kurosu (penjumlahan silang), yang merupakan gabungan kata "penjumlahan" dalam bahasa jepang dan kata "cross" (penyilangan) dalam ejaan bahasa jepang. Enam tahun kemudian, yaitu pada tahun 1986, Nikoli memberi nama dagang game ini dengan nama Kakuro, sebuah kependekan dari *Kasan Kurosu*. Nikoli juga menerbitkan booklet pertama Kakuro. Setelah itu, Nikoli menerbitkan lebih dari 22 booklet-booklet dan berhasil menjual lebih dari 1 juta buku-buku.

September 2005 adalah titik penentunya. Game baru ini diperkenalkan ke barat oleh The Guardian dan The Daily Mail yang menerbitkan *puzzle* Kakuro setiap harinya di Inggris. Setelah itu Kakuro menyebar keseluruh dunia.



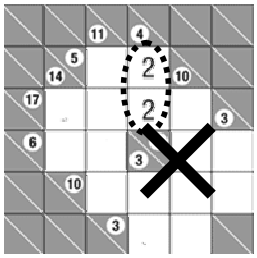
Gambar 1. Contoh sebuah *puzzle* kakuro

2 DESKRIPSI MASALAH

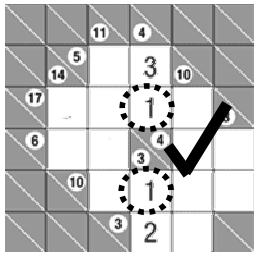
Kakuro adalah sebuah *puzzle* yang selesai jika seluruh kotak jawaban telah terisi. Adapun syarat-syarat dalam pengisian kotak jawaban, yaitu sebagai berikut:

1. Setiap kotak hanya boleh diisikan dengan bilangan bulat dari 1 sampai 9.

2. Setiap kotak yang berurutan dalam satu lajur (lajur yang dimaksudkan adalah deretan kotak yang berurutan dalam satu baris atau kolom) tidak boleh memiliki angka yang sama.
3. Isi dari setiap lajur harus memiliki jumlah yang sama pada ujung kiri (untuk baris) atau ujung atas (untuk kolom) lajur tersebut.

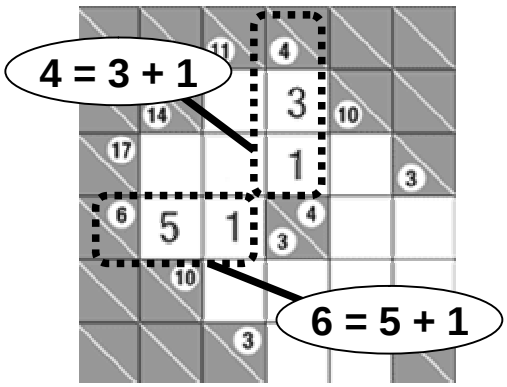


Gambar 2a.
nilai sama pada 1 lajur



Gambar 2a.
nilai sama pada lajur berbeda

Pengisian jawaban seperti pada Gambar 2a. merupakan contoh yang salah karena terdapat dua angka atau lebih pada satu lajur. Sedangkan pada gambar 2b. diperbolehkan karena terdapat pada lajur yang berbeda, walaupun dalam kolom yang sama.



Gambar 3. Cara pengisian jawaban.
Jumlah nilai jawaban sesuai soal

3 METODE

Metode yang digunakan untuk menyelesaikan masalah ini adalah dengan menggunakan algoritma *backtracking*. Algoritma *backtracking* adalah sebuah pengembangan dari algoritma DFS biasa dan merupakan perbaikan dari algoritma bruteforce, yaitu dengan menambahkan fungsi pembatas yang mempertimbangkan apakah jawaban mengarah ke ruang solusi. Sehingga jika jawaban tidak mengarah ke ruang solusi maka jawaban akan dibuang dan tidak akan dipertimbangkan kembali.

Penerapannya dalam masalah ini yaitu dengan cara mencoba berbagai kemungkinan untuk setiap kotak secara rekursif dan akan melakukan runut balik bila isi dari jawaban suatu kotak tidak memenuhi aturan yang telah ditetapkan. Aturan-aturan di sini adalah syarat-syarat dalam pengisian kotak, seperti yang disebutkan di atas.

Pencarian dimulai dari kotak pada baris paling kecil, pada sisi paling kiri. Kemudian dimulai percobaan jawaban. Apabila pencarian telah mencapai ujung dan isi valid maka solusi ditemukan, apabila terjadi *backtrack* hingga melebihi awal pencarian, maka solusi tidak ditemukan.

Pada setiap kotak dilakukan percobaan jawaban dari 1 hingga 9. Tetapi sebelum mengisi matriks akan diperiksa nilai tersebut, apakah menuju ke solusi atau tidak.

Pertama kali diperiksa apakah nilai yang dimasukkan pada suatu lajur melebihi jumlah pada lajur tersebut. Bila melebihi maka dilakukan backtrack. Jika tidak, diperiksa kembali apakah ada lajur yang telah terisi dengan lengkap, jika ya dan jumlahnya lebih kecil dari soal, maka coba nilai selanjutnya. Jika tidak, diperiksa lagi apakah ada nilai yang sama pada satu lajur. Jika ya, maka maju ke nilai selanjutnya. Jika semua kotak lulus semua pemeriksaan maka lanjut ke kotak setelahnya.

Proses rekursif dalam pencarian jawaban, memiliki basis jika rekursif telah mencapai batas ujung kanan bawah atau backtrack hingga ujung kiri atas.

Source Code (tanpa optimasi) dalam bahasa java

```
private boolean solveRekursif
   (int _brs,int _klm,int jawab) {
//mengembalikan true jika ada solusi
//false jika tidak ada solusi
  if (_brs==(totBrs)) {
    //melewati ujung dengan nilai yang valid
    //solusi ditemukan
    return true;
  }else if(matriks.getGrid(_brs,_klm).getLock()){
    //jika grid kotak bukan grid jawaban
    if(_klm==(totKlm-1)) {
      return solveRekursif(_brs+1,0,1);
    } else {
      return solveRekursif(_brs,_klm+1,1);
    }
  } else {
    //jika grid kotak grid jawaban, coba nilai
    boolean done = false;
    while (!done && jawab<=9) {
    matriks.getGrid(_brs,_klm).setValue(jawab);
      //matriks.printMat();
      int valid = valid();
      //mengembalikan 0=valid;
      //1=tidak valid, jawab++;
      //2=tidak valid, backtrack
      if (valid == 1) {
        jawab++;
      } else if (valid == 2) {
        matriks.getGrid(_brs,_klm).setValue(-1);
```

```
        return false;
      } else {
        if(_klm==(totKlm-1)) {
          done = solveRekursif(_brs+1,0,1);
        } else {
          done = solveRekursif(_brs,_klm+1,1);
        }
        jawab++;
      }
    }
    if (!done)
      matriks.getGrid(_brs,_klm).setValue(-1);
    return done;
  }
}
```

4 HASIL ANALISIS

Dengan menggunakan algoritma di atas, dilakukan percobaan untuk menyelesaikan beberapa contoh *puzzle* Kakuro. Yaitu sebagai berikut:

	3	6		24	16
4	1	3	15	8	7
3	2	1	3 16	7	9
	12	2	1	9	
			2		

Gambar 4. *Puzzle* 1 (6x5)

		11	4		
	14 5	2	3	10	
17	9	5	1	2	3
6	5	1	3 4	3	1
	10	3	1	4	2
		3	2	1	

Gambar 5. *Puzzle* 2 (6x6)

	4	22		16	3
3	1	2	16 6	4	2
18	3	5	7	2	1
	17 23	8	9	6	14
9	8	1	6	1	5
15	9	6	12	3	9

Gambar 6. *Puzzle* 3 (6x6)

	23	30			27	12	16
16	9	7		17 24	8	7	9
17	8	9	15 29	8	9	5	7
35	6	8	5	9	7	12	
	7	6	1	7 8	2	6	7
	11	10 16	4	6	1	3	2
21	8	9	3	1	5	1	4
6	3	1	2		3	2	1

Gambar 7. *Puzzle* 4 (8x8)

		12	12		3	16	12		15	4
	4	1	3	19	2	9	8	3	2	1
	8 5	4	1	13 11	1	7	3	30 7	4	3
29	7	5	8	9	14	22 11	1	7	3	10
3	1	2	28 7	1	2	4	22	9	5	8
	11	15 11	5	3	1	2	6 9	6	1	2
9	2	1	6	20	3	7	2	8	23	17
22	9	5	8	17 20	8	9	3	9 16	7	5
	10 18	3	9	6	5	16 14	1	2	3	8
3	1	2	11	3	1	7	15	6	9	
13	9	4	21	8	4	9	5	1	4	

Gambar 8. *Puzzle* 5 (11x11)

			7	16		26	16	17	
	34	25 4	1	3	23	8	9	6	7
12	6	3	2	1	8 13	3	9	1	2
39	9	7	4	8	6	5	4	3	1
15	7	8	23 7	4	2	1	14 11	7	4
11	4	1	6		11	9	2	16	32
23	8	6	9	16		18	8	3	7
	19	29 12	8	4	4	10 7	4	1	2
8	3	5	7	2	1	4	7 10	2	9
17	9	8	3 23	1	3	2	4	6	5
23	7	9	1	6	16	1	2	4	9
	12	7	2	3	4	3	1		

Gambar 9. *Puzzle* 6 (6x5)

Pada *puzzle* 7 yang berukuran 14x14 terjadi rekursif sebanyak lebih dari 8 juta kali dan dengan waktu lebih dari 20 menit.

Dengan melakukan optimasi terhadap algoritma diharapkan dapat menghasilkan kalkulasi yang lebih sedikit. Berikut merupakan tabel hasil dari kalkulasi dengan atau tanpa optimasi, di mana optimasi mendahulukan pengisian kotak bergantung pada jenis optimasinya.

Tabel 1 Hasil perhitungan berbagai optimasi algoritma

No *Puzzle*	**Jumlah rekursif dengan dan tanpa optimasi**					
	N	**S1**	**S2**	**J1**	**J2**	**S1+J1**
1	51	22	96	18	469	24
2	276	65	658	66	2384	42
3	193	295	2723	128	525212	317
4	8403	30493	30424	277117	> 6juta	1097091
5	17289	> 2juta	> 2juta	> 2juta	-	> 6juta
6	412045	-	-	-	-	-
7	> 8juta	-	-	-	-	-

Keterangan: **N = tanpa optimasi**
S1 = optimasi soal (kecil -> besar)
S1 = optimasi soal (besar -> kecil)
J1 = optimasi jlh kotak dalam 1 lajur (kecil -> besar)
S1 = optimasi jlh kotak dalam 1 lajur (besar -> kecil)

*Beberapa perhitungan tidak dilanjutkan karena memakan waktu yang lama.

*Waktu kalkulasi untuk 1-20000 jumlah rekursif adalah kurang dari 1 detik. Untuk jumlah kalkulasi +/- 400000 adalah 11 detik

Dari hasil di atas diketahui dengan algoritma ini solusi dari Kakuro ini dapat ditemukan. Dari nilai jumlah rekursif dan waktu kalkulasi diketahui bahwa algoritma ini untuk menyelesaikan papan berukuran hingga 12x10 masih terbilang cepat (tanpa optimasi). Tetapi setelah melewati ukuran papan lebih dari itu akan terlihat membutuhkan kalkulasi yang banyak dan memakan waktu yang lama.

Dari tabel tersebut dapat dilihat untuk ukuran yang kecil (*puzzle* 1 - 3) jumlah rekursif dengan menggunakan optimasi J1 merupakan hasil yang terkecil. Tetapi pada *puzzle* 4 keatas jumlah kalkulasi langsung naik secara drastis. Begitu juga dengan pengoptimasian algoritma yang lainnya. Hal ini disebabkan karena dengan melakukan pengoptimasian seperti ini, maka pengerjaan akan terpencar-pencar karena pengurutan pengerjaan berdasarkan optimasi, sedangkan tanpa optimasi pengerjaan tidak terpencar-pencar karena berurutan dari atas ke bawah.

5 KESIMPULAN

Algoritma *backtracking* tanpa optimasi ini cukup bagus untuk mencari solusi dari *puzzle* kakuro ini dengan ukuran kecil hingga sedang (kurang dari 14x14). Karena dapat menyelesaikannya kurang dari 1 detik (kurang dari 11x11). Tetapi masih terbilang lambat untuk permasalahan yang lebih besar. Semakin kecil nilai dari soal akan mempercepat kalkulasi karena akan meghasilkan kemungkinan yang sedikit. Begitu juga dengan jumlah kotak per lajur, semakin sedikit kotak semakin memperkecil kemungkinan jawaban. Sehingga semakin besar *puzzle* semakin banyak membutuhkan kalkulasi

Untuk mengatasi permasalahan yang lebih besar dapat dilakukan dengan algoritma yang lain atau dengan pengoptimasia algoritma *backtracking* yang lain. Karena dengan optimasi yang dilakukan pada percobaan pada bagian analisis, didapatkan hasil yang lebih buruk untuk kasus ukuran *puzzle* yang besar.

REFERENSI

.

[1] Munir, Rinaldi. 2005. Strategi Algoritmik. Teknik Informatika ITB : Bandung

[2] http://www.wikipedia.org

[3] http://www.nikoli.co.jp/en/puzzles/kakuro/

[4] http://www.passionforpuzzles.com/strategy/kakuro.php