Penyelesaian Puzzle Kakurasu dengan Algoritma Brute Force

Yanfa Adi Putra(13512037)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512037@std.stei.itb.ac.id

***Abstract*— Makalah ini berisikan tentang bagaimana pengaplikasian dari mata kuliah Strategi Algoritma. Makalah ini bertemakan brute force, makalah ini membahas hal-hal untuk menyelesaikan permasalahan puzzle Kakurasu. Dalam makalah ini akan dijelaskan teori-teori mengenai *Brute force* dan tentang implementasinya pada permainan puzzle Kakurasu.**

I. PENDAHULUAN

A. . Latar Belakang

Makalah ini berawal dari ketertarikan penulis mengenai algoritma untuk memecahkan masalah secara umumnya dan brute force secara khususnya. Brute force merupakan sebuah cabang ilmu yang unik yang menerapkan sistem bagaimana kita bisa memecahkan masalah walau dengan algoritma yang memang mungkin tidak paling mangkus.

Selain karena ketertarikan penulis terhadap algoritma brute force, penulis juga sangatlah hobi bermain game puzzle, terutama game-game yang mengasah otak seperti puzzle Kakurasu.

Penulis juga ingin mencoba menerapkan bagaimana menyelesaikan puzzle Kakurasu dengan menggunakan algoritma brute force

B. Tujuan

Tujuan penulis membuat makalah ini adalah sebagai berikut :

- Menambah pemahan mengenai teori-teori pemeceahan permasalahan terutama pemecahan masalah menggunakan algoritma *brute force*.
- Menerapkan implementasi *brute force*.
- Memenuhi tugas mata kuliah Strategi Algoritma.

II. TEORI DASAR

A. Algoritma Brute force

Kombinatorial adalah cabang ilmu matematika untuk menghitung jumlah penyusun objek-objek tanpa harus mengenumerasi semua kemungkinan yang ada. Dimana

Brute force merupakan salah satu dari banyak algoritma pemecahan masalah yang ada. Algoritma *brute force* ini merupakan sebuah algoritma yang menerapkan metode yang sangat lempang.

Algoritma *brute force* biasanya didasarkan oleh *problem statement* dan definisi konsep yang dilibatkan. Algoritma *brute force* memecahkan persoalan dengan sangat sederhana, langsung dan sangat jelas.

Karakteristik pada algoritma *brute force* umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan jumlah komputasi yang besar dan waktu yang lama dalam penyelesaiannya dan juga umumnya mencoba semua kemungkinan yang memungkinakan untuk terjadi yang menyebabkan ada hal-hal yang seharusnya tidak perlu tapi tetap dilakukan. Dibalik ketidak-"cerdas"-an dari algoritma *brutefore*, algoritma *brute force* juga memiliki kelebihan, algoritma *brute force* sangat cocok untuk digunakan untuk program yang berukuran kecil, hal ini dikarenakan jalan penyelesaian dari algoritma *brute force* yang sangat lempang, sederhana, dan implemetasinya sederhana.

Algoritma *brute force* yang merupakan algoritma yang tidak terlalu mangkus ini umumnya bisa menyelesaikan segala permsalahan yang ada. Beberapa permasalahan memang akan tidak mangkus jika diselesaikan dengan menggunakan algoritma *brute force*, tetapi tidak semua algoritma untuk memecahkan permasalahan dapat diterapkan untuk sembarang permasalahan seperti algoritma *brute force* ini.

Selain algoritma *brute force* merupakan algoritma yang mudah untuk diterapkan untuk berbagai macam permasalahan yang ada, algoritma *brutefore* ini juga memiliki kelemahan dan kelebihan, berikut ini adalah kelebihan dan kelemahan dari algoritma *brute force* :

Kelebihan :

1. Algoritma *brute force* merupakan algoritma yang dapat diterapkan hampir untuk seluruh permasalahan yang ada
2. Algoritma *brute force* merupakan algoritma yang sederhana dan mudah dimengerti dalam penggunaannya
3. Algoritma *brute force* dapat menghasilkan algoritma-algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan, dan sebagainya.
4. Algoritma *brute force* dapat menghasilkan algoritma baku/standar untuk tugas-tugas

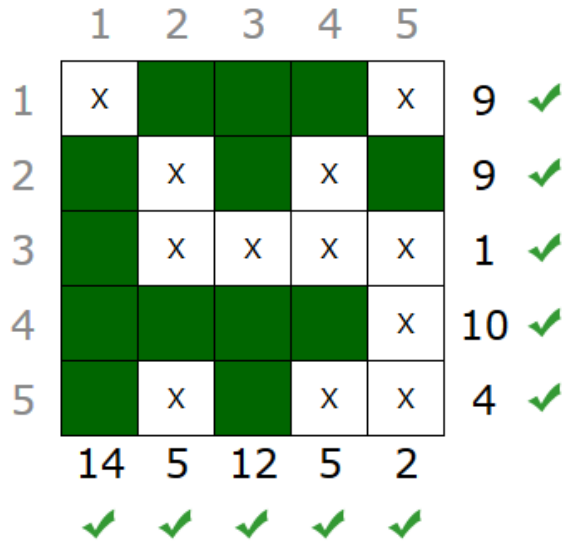
komputasi seperti penjumlahan/perkalian dari sebuah bilangan atau menentukan minimum/maksimum didalam sebuah data.

Kelemahan :

1. Algoritma *brute force* jarang menghasilkan algoritma yang mangkus dan sangkil untuk menyelesaikan masalah yang ada.
2. Kebanyakan dari algoritma *brute force* sangatlah lambat untuk komputasi dalam sekala besar sehingga banyak yang tidak dapat diterima.
3. Algoritma *brute force* tidak sekontruktif/sekreatif teknik pemecahan masalah yang lainnya.

B. Puzzle Kakurasu

Puzzle Kakurasu merupakan sebuah puzzle yang terdiri dari kotak-kotak kosong yang pada pinggir masing-masing kolom dan baris terdapat angka yang merepresentasikan jumlah yang harus didapatkan jika kotak tersebut di ambil.

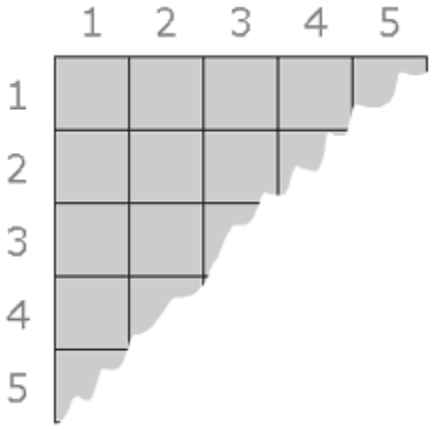


Tujuan pada puzzle Kakuraru adalah memilih kotak-kotak yang tepat dengan cara memberinya warna yang menandakan bahwa kotak tersebut telah dipilih. Petunjuk yang dimiliki hanya angka-angka yang berada pada pinggir masing-masing kotak pada masing-masing kolom dan baris. Angka-angka pada pinggir masing-masing kolom dan baris tersebut merupakan jumlah yang harus diperoleh agar permainan terselelsaikan.

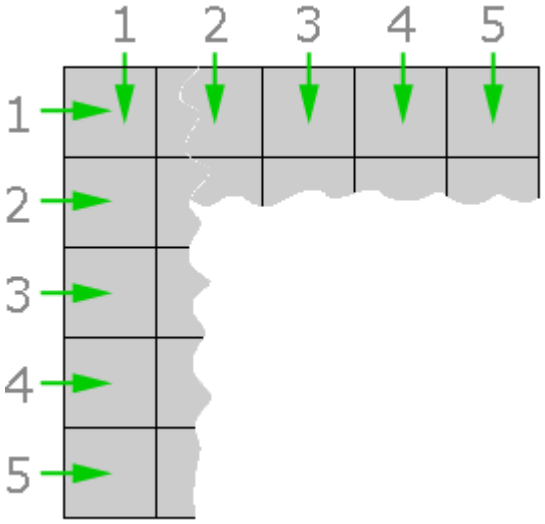
Puzzle kakurasu memiliki berbagai macam jenis, diantaranya adalah puzzle yang memiliki kotak sebanyak 16 buah yang terdiri dari 4 x 4 kotak kecil, jenis yang terdapat 25 kotak yang disusun menjadi 5 x 5 kotak kecil, jenis yang terdapat 36 kotak kecil yang disusun menjadi 6 x 6 kotak kecil, dan jenis-jenis lain yang lebih banyak kotak kecilnya untuk orang-orang yang menyukai tantangan puzzle kakurasu dan ingin menyelesaikannya.

Puzzle Kakurasu ini memiliki peraturan-peraturan dasar untuk memainkan permainannya. Berikut merupakan aturan-aturan dasar yang dimiliki oleh puzzle Kakurasu :

- Angka yang terdapat pada atas dan kiri dari kotak merepresentasikan berapanilai dari kotak tersebut jika anda memilih kotak tersebut.

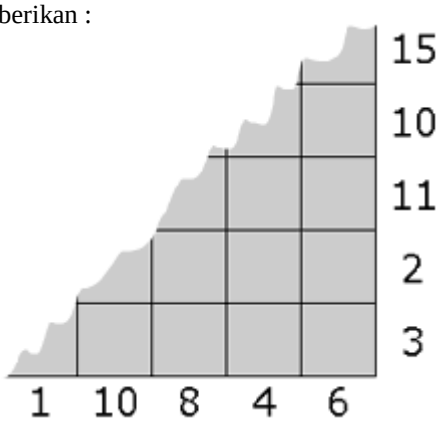


Gambar 2.1.1 : Contoh bagian dari puzzle Kakurasu (http://www.brainbashers.com/kakurasuhelp.asp)



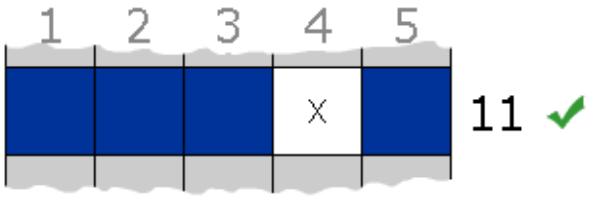
Gambar 2.1.2 : Ilustrasi nilai pada kotak (http://www.brainbashers.com/kakurasuhelp.asp)

- Angka yang terdapat pada kanan dan bawah kotak merupakan petunjuk untuk menyelesaikan puzzle ini, berikut adalah contoh gambar dari petunjuk yang biasa diberikan :

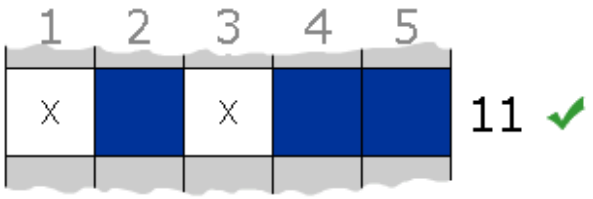


Gambar 2.2.4 : Ilustrasi nilai petunjuk kotak
(http://www.brainbashers.com/kakurasuhelp.asp)

Digambar tersebut terdapat angka yang merepresentasikan petunjuk dari puzzle, misalkan kita ambil contoh angka 11 pada baris pertama, maka pada baris pertama, pemain diharapkan untuk memilih kotak yang jika dijumlah kan dalam satu barisan pertama tersebut akan menghasilkan angka 11, berikut adalah contoh gambar dari pemilihan kotak berdasarkan petunjuk :



(a)



(b)

Gambar 2.2.5 Contoh dari pemilihan kotak Kakurasu
(http://www.brainbashers.com/kakurasuhelp.asp)

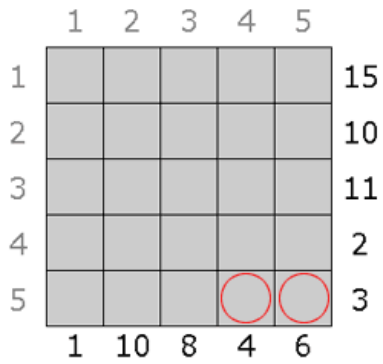
Pada gambar tersebut dapat dilihat bahwa kotak yang dipilih untuk mendapatkan jumlah yang sesuai memiliki variasi yang beragam dan pemilihan kotak biasa digambarkan dengan memberikan warna kepada kotak yang kita pilih dan tanda silang kepada kotak yang kita anggap merupakan kotak yang tidak kita pilih.

III. PENJELASAN ALGORITMA DAN IMPLEMENTASI

A. Teori Penyelesaian Puzzle Kakurasu

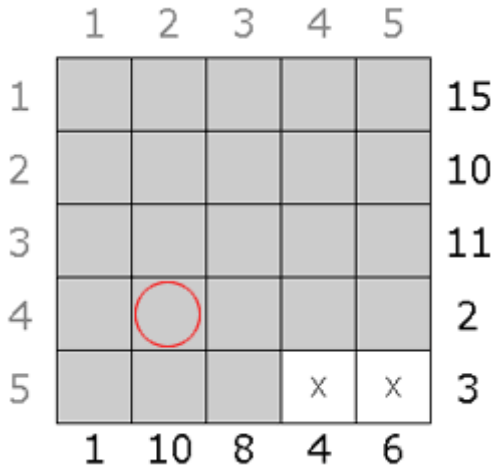
Pada permainan puzzle Kakurasu terdapat berbagai macam cara untuk menyelesaikan dan yang paling biasa dilakukan untuk menyelesaikan puzzle kakurasu adalah dengan mengurangi kotak-kotak yang tidak mungkin dipilih terlebih dahulu.

Jika kita ambil contoh sebuah kotak 5x5, dan terdapat sebuah petunjuk yang menyatakan bahwa baris kelima haruslah berjumlah 3, maka kita secara otomatis harus dapat menakumulasi bahwa untuk kotak yang bernilai 4 dan 5 tidak akan kita pilih, seperti pada contoh dibawah ini :



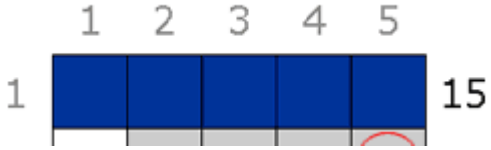
Gambar 3.1 contoh kotak yang tidak mungkin untuk dipilih
(http://www.brainbashers.com/kakurasuhelp.asp)

Selain cara tersebut, kita juga dapat memilih kotak yang memang sudah pasti akan kita pilih, mari kita ambil sebuah contoh pada puzzle kukarasu 5x5, jika pada baris keempat petunjuk yang diberikan sebagai jumlah yang harus didapatkan untuk sebuah baris adalah 2, maka kita bisa secara otomatis menilai bahwa pada baris tersebut hanya kotak yang bernilai 2 yang dapat dipilih karena jika kotak lain yang dipilih akan menyebabkan baris tersebut tidak memenuhi jumlah yang menjadi tujuan permainan, berikut adalah ilustrasi dari contoh permasalahan tersebut :



Gambar 3.2 contoh kotak yang pasti dipilih
(http://www.brainbashers.com/kakurasuhelp.asp)

Cara lain yang memudahkan untuk memilih kotak yang sudah pasti dipilih adalah jika sebuah petunjuk memiliki nilai yang merupakan jumlah dari seluruh kotak yang ada di baris atau kolom, misalkan kita ambil contoh jika sebuah baris memiliki petunjuk jumlah sebesar 15 dan puzzle kakurasu tersebut merupakan puzzle 5x5, maka kita bisa mengasumsikan bahwa seluruh kotak yang ada pada baris atau kolom yang harus berjumlah 15 tersebut harus dipilih.



Gambar 3.3 contoh kotak yang pasti dipilih
(http://www.brainbashers.com/kakurasuhelp.asp)

Setelah memilih semua kotak yang pasti bisa kita pilih

dan menandai kotak yang tidak mungkin kita pilih, maka kita akan mendapatkan sebuah persoalan yang tidak terlalu susah dibandingkan sebelumnya, hanya tinggal memilih kotak yang kira-kira akan memenuhi tujuan jumlah yang menjadi soal tanpa melanggar setiap kotak yang memang sudah kita tandai sebagai kotak yang tidak mungkin kita pilih dari puzzle dan mengulangi step-step sebelumnya, yaitu menandai setiap kotak yang memang sudah tidak mungkin lagi dan memilih semua kotak yang sudah pasti dipilih.

B. Implementasi *Brute force* pada Kakurasu

Seperti yang telah dijelaskan disubbab sebelumnya bahwa kakurasu memiliki beberapa cara mudah untuk menyelesaikannya. Namun hal itu tidak bisa kita lakukan jika kita tidak memiliki algoritma untuk mengira-ngira kotak yang tepat untuk dipilih.

Jika kita mengimplementasikan cara-cara tersebut menggunakan *brute force* maka kita akan mendapatkan sebuah penyelesaian dengan cepat dan dapat dipercaya, karena semua kokmputasi dilakukan oleh komputer, tanpa harus kita khawatirkan terdapat salah hitung atau semacamnya.

Pseudo-code algoritma *brute force* untuk penyelesaian Kakurasu adalah sebagai berikut :

Representasi kelas kakurasu :

```
class Kakurasu

nJumlah : integer
matriks : array[1..nJumlah][1..nJumlah] of integer
col : array[1..nJumlah] of integer
row : array[1..nJumlah] of integer

Kakurasu(int)
finishState() : boolean
countCol(int) : integer
countRow(int) : integer
fixCol(int) : void
fixRow(int) : void
solveCol(int) : void
solveRow(int) : void
solveProblem() : void
```

- Pada kelas Kakurasu tersebut, nJumlah merepresentasikan banyaknya elemen dalam satu baris, atau bisa dibilang sebagai patokan dimensi dari matriks yang akan dibuat
- matriks tersebut sendiri adalah sebuah representasi dari penyelesaian persoalan puzzle kakurasu, jawaban penyelesaian menggunakan algoritma *brute force* akan di but didalam matriks tersebut
- col pada kelas tersebut menandakan tujuan dari masing-masing kolom, dimana col[1] menandakan jumlah yang harus didapatkan oleh kolom 1.
- row pada kelas tersebut menandakan tujuan dari msaing-masing baris, dimana row[1] menandakan jumlah yang harus didapatkan oleh baris 1.

Konstruktor dari kelas Kakurasu :

```
Kakurasu(input n : integer)

nJumlah = n
For i = 1 to nJumlah
  For j = 1 to nJumlah
    matriks[i][j] = -1
```

Pada konstruktor tersebut, diinisialisasi dari masing-masing nilai matriks, dimana -1 merepresentasikan bahwa matriks tersebut masih bisa untuk diisi.

Fungsi untuk memeriksa apakah puzzle telah didapatkan penyelesaiannya :

```
function finishState() => boolean
finish : boolean
finish = true
for i = 1 to nJumlah
  if countCol(i) != getCol(i) or countRow(i) !=
     getRow(i)
     finish = false
=> finish
```

Fungsi finishState mengembalikan nilai true jika matriks yang merupakan representasi dari jawaban sudah sesuai dengan angka-angka yang menjadi tujuan dari masing-masing kolom dan baris.

Fungsi untuk menghitung jumlah pada kolom dan baris :

```
function countCol(input j : integer) => integer
n : integer = 0
for i = 1 to nJumlah
  if  matriks[i][j] == 1 then
    n += matriks[i][j] * (i)
=>  n
```

```
function countRow(input i : integer) => integer
n : integer = 0
for i = 1 to nJumlah
  if  matriks[i][j] == 1 then
    n += matriks[i][j] * (i)
=>  n
```

Fungsi countCol dan countRow tersebut berfungsi untuk mengecek berapa jumlah dari kotak-kotak yang berada di suatu kolom atau baris, seperti kita ambil contoh fungsi countCol(1) maka akan mengembalikan nilai integer yang merupakan jumlah dari kotak-kotak yang telah dipilih pada kolom 1, dan jika fungsi countRow(1) maka akan mengembalikan jumlah dari kotak-kotak yang telah dipilih pada baris kesatu.

Prosedur untuk menandai semua kotak yang memang sudah tidak mungkin untuk dipilih :

```
procedure fixRow(input i : integer)
for j = 1 to nJumlah
  if   j > row[i] then
    matriks[i][j] = 0
```

```
procedure fixCol(input j : integer)
for i = 1 to nJumlah
  if   i > col[j] then
    matriks[i][j] = 0
```

Prosedur fixCol dan fixRow ini adalah sebuah prosedur untuk memudahkan penyelesaian masalah dari puzzle kakurasu, dimana jika kita memanggil fixCol(3), maka komputer akan secara otomatis memeriksa apakah ada pada kotak tersebut yang memiliki nilai yang melebihi nilai tujuan dari kolom 3, misal pada kolom 3 tujuan jumlahnya bernilai 3, maka semua kotak dengan nilai lebih besar dari 3 akan ditandai 0, dimana 0 ini adalah tanda bahwa kotak tersebut sudah tidak boleh dipakai lagi.

Prosedur untuk menyelesaikan baris atau kolom yang dituju :

```
procedure solveCol(input j : integer)
while countCol(i) != getCol(i)
 for i = 1 to nJumlah
  if  matriks[i][j] != 0  then
    matriks[i][j] = 1
    if  countCol(i) > getCol(i)  then
      if j == 0 then
        matriks[nJumlah][j] = -1
      else
        matriks[i - 1][j] = -1
```

```
procedure solveRow(input i : integer)
while countRow(i) != getRow(i)
 for j = 1 to nJumlah
  if  matriks[i][j] != 0  then
    matriks[i][j] = 1
    if  countRow(i) > getRow(i)  then
      if i == 0 then
        matriks[i][nJumlah] = -1
      else
        matriks[i][j - 1] = -1
```

Prosedur solveCol dan prosedur solveRow adalah sebuah prosedur untuk memilih kotak-kotak yang memungkinkan untuk dipilih sesuai dengan soal, misal kita memanggil solveRow(4) maka prosedur akan memilih kotak-kotak pada baris keempat yang memungkinkan untuk menghasilkan jumlah sebesar row[4].

Algoritma brute force yang diimplementasikan pada penyelesaian permasalahan puzzle kakurasu terdapat pada prosedur ini, dimana algoritma yang saya sampaikan disini memiliki alur sebagai berikut :

1) Pilih kotak pertama, lalu geser satu kotak, sudah mencapai ujung dari baris atau kolom, maka kembali ke baris atau kolom pertama
2) Jumlahkan baris atau kolom yang sedang diperiksa, jika jumlah nilai kotak yang dipilih memiliki jumlah yang lebih besar dibandingkan dengan nilai tujuan yang ingin diperoleh dari baris atau kolom tersebut, maka baris atau kolom sebelumnya dibuat tidak dipilih, kembali ke nomor 1)
3) Ulangi sampai menemukan solusi yang tepat untuk baris atau kolom yang ingin diperiksa

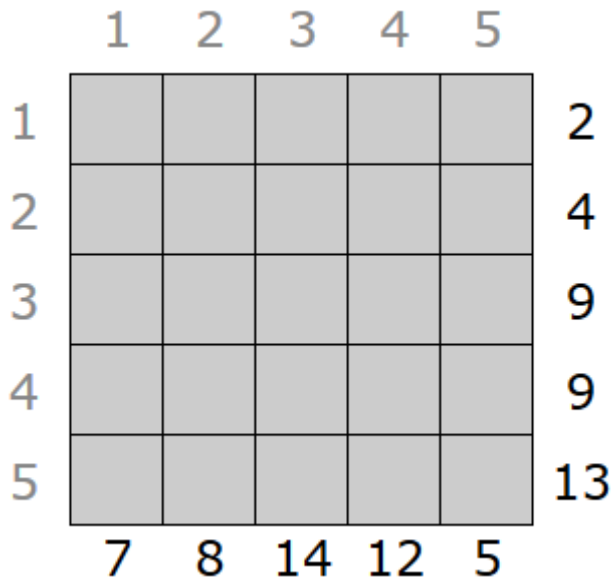
Prosedur untuk menyelesaikan puzzle Kakurasu secara keseluruhan:

```
procedure solveProblem()
while !finishState() do
 for i = 1 to nJumlah
   solveCol(i)
 for i = 1 to nJumlah
   solveRow(i)
```

Prrosedur solveProblem ini merupakan perluasan dari prosedur solveCol dan solveRow dimana jika di solveProblem ini adalah prosedur untuk menyelesaikan seluruh tujuan pada puzzle kakurasu, jika solveProblem terselesaikan maka matriks[][] akan bernilai 1 jika merupakan kotak yang dipilih dan -1 atau 0 yang merepresentasikan kalau kotak tidak dipilih.

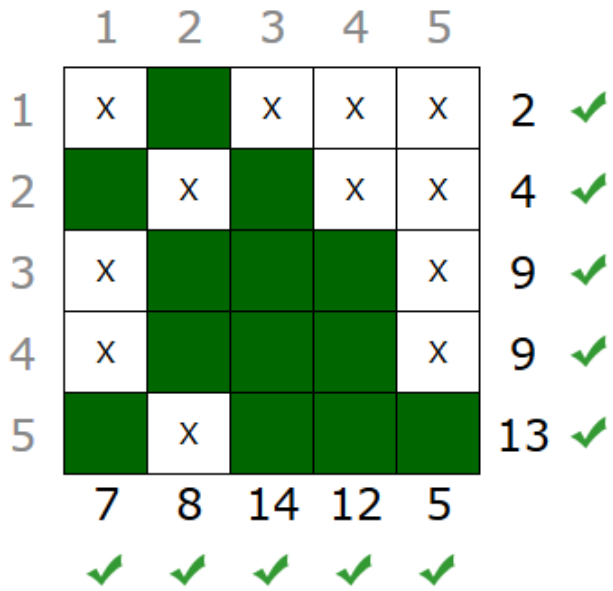
IV. HASIL ANALISIS

Dengan menggunakan algoritma ini puzzle kakurasu yang sederhana dapat diselesaikan dengan baik, terutama untuk puzzle kakurasu yang merupakan puzzle 5x5, contoh untuk kasus persoalan :



Gambar 4.1 Contoh soal Kakurasu
http://www.brainbashers.com/showkakurasu.asp

Pada saat menjalaankan prosedur solveProblem maka program akan memilih kotak sebagai berikut :



Gambar 4.2 Contoh jawaban Kakurasu
http://www.brainbashers.com/showkakurasu.asp

Algoritma *brute force* yang seperti disampaikan barusan dapat berjalan dengan baik jika puzzle merupakan puzzle yang masih sederhana cara penyelesaiannya, program akan memunculkan jawaban dari puzzle dengan waktu kurang dari 1 ms karena kebutuhan komputasi yang tidak terlalu banyak.

Algoritma *brute force* ini jika diterapkan pada puzzle merupakan puzzle yang sudah tidak sederhana, seperti diatas 6x6, maka algoritma tersebut sering tidak berjalan dengan baik karena terjadi saling ganti yang rekursif antara satuu prosedur solveCol dengan prosedur solveRow yang menyebabkan program tidak berhenti.

V. KESIMPULAN

Penerapan algoritma *brute force* pada penyelesaian puzzle Kakurasu ini sangat tepat, karena komputasi yang diperlukan tidaklah terlalu banyak, namun masih diperlukan algoritma *brute force* yang lebih bisa mencakup seluruh kasus dari puzzle Kakurasu agar tidak terjadi keselahan-keselahan seperti program yang tidak berhenti.

VI. UCAPAN TERIMA KASIH

Pertama dan yang paling utama penulis mengucapkan terima kasih kepada Allah SWT yang telah memberikan kesehatan dan banyak berkah kepada penulis, lalu penulis juga ingin mengucapkan terima kasih kepada orang tua penulis yang selalu mendukung penulis, lalu kepada Dr. Masayu Leylia Khodra, S.T, M.T dan Ir. Rinaldi Munir, M.T. selaku dosen Strategi Algoritma.

REFERENSI

[1] http://www.brainbashers.com/kakurasuhelp.asp
[2] Munir, Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2009

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2014

Yanfa Adi Putra(13512037)