Penerapan Algoritma *Backtracking* pada Pencarian Solusi *Fill-in Crossnumber*

Yollanda Sekarrini - 13512051
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512051@std.stei.itb.ac.id

***Abstrak*—*Fill-in crossnumber* adalah kuis asah otak yang menggunakan metoda *fill-in* dan angka sebagai pengisinya. *Fill-in* berarti mengisi secara penuh *grid* yang kosong berdasarkan kata atau angka yang telah disediakan. Pengisian dengan angka yang dilakukan harus sesuai antara jumlah kotak satu baris/kolom putih (*white grid*) dengan jumlah digit angka yang dimasukkan. Kuis ini adalah salah satu variasi dari *fill-in crossword* yang menggunakan huruf sebagai pengisinya. Selain menggunakan angka, perbedaan yang mencolok antara *fill-in crossnumber* dengan *crossword* 'Teka-teki Silang' adalah petunjuk yang diberikan untuk mengisi *grid* yang kosong tersebut. Algoritma *Backtracking* adalah algoritma yang berbasis pada DFS (*Deep First Search*) yang pencarian hanya mengarah ke solusi sehingga menjadi lebih efisien. Algoritma *Backtracking* sering digunakan dengan menggunakan fungsi rekursif, yaitu fungsi yang memanggil dirinya sendiri. Algoritma *Backtracking* banyak digunakan untuk penyelesaian masalah pada bidang kecerdasan buatan (*Artificial Intelligence)* seperti permainan *tic-tac-toe,* labirin dan catur. Pencarian solusi *fill-in crossnumber* ini dapat diselesaikan dengan algoritma *Backtracking*.**

***Kata Kunci*—Algoritma, *Backtracking, Fill-in Crossnumber, Grid,* Solusi.**

I. PENDAHULUAN

Algoritma *Backtracking* yang disebut juga dengan algoritma Runut-balik, merupakan algoritma yang umum digunakan untuk menemukan semua solusi pada persoalan komputasi. Algoritma ini banyak diterapkan pada pencarian solusi *puzzle* seperti *eight queens puzzle, crossword, maze, sudoku* dan *verbal arithmetic.* Selain *puzzle,* algoritma *Backtracking* juga digunakan pada masalah optimasi kombinatorial seperti *parsing, sum of subsets* dan *integer knapsack.* Algoritma ini juga diterapkan pada *Logic Programming* (pemrograman berbasis logika) seperti Prolog, Icon, dan Planner yang digunakan untuk men-*generate* jawaban dari pernyataan yang diberikan.

Backtracking pertama kali dikenalkan oleh D.H.Lehmer, matematikawan Amerika, pada tahun 1950. Beliau adalah pionir *string-processing language* yaitu SNOBOL yang pertama kali menyediakan fasilitas *backtracking*. Kemudian teknik pencarian ini dikembangkan oleh R.J.Walker, Golomb, dan Baumert yang diterapkan diberbagai persoalan.

Algoritma *Backtracking* disebut sebagai perbaikan dari algoritma *Brute Force*. Algoritma *Brute Force* adalah algoritma yang *straightforward* (lempang atau lurus) yang menyelesaikan masalah berdasarkan pernyataan dan definisi masalah tersebut. Biasanya algoritma ini memecahkan masalah dengan sederhana dan dengan cara yang jelas seperti mengenumerasi semua kemungkinan solusi (termasuk solusi yang salah) kemudian memilih solusi yang optimal. Hampir semua persoalan dapat diselesaikan dengan menggunakan teknik pencarian ini seperti pencarian elemen terbesar/terkecil, *sorting, searching* dan *string matching.*

Berbeda dengan algoritma *Brute Force* yang harus mencek semua kemungkinan, *Backtracking* hanya mencek solusi yang mungkin benar sehingga jika ditemukan, saat pencarian, solusi yang salah maka pencarian langsung dihentikan (*prune*) dan dikembalikan ketahap pencarian sebelumnya, dilanjutkan dengan tidak mengambil tahap yang telah *prune*. Terbukti bahwa pencarian dengan *Bactracking* akan lebih efektif dibandingkan *Brute Force.*

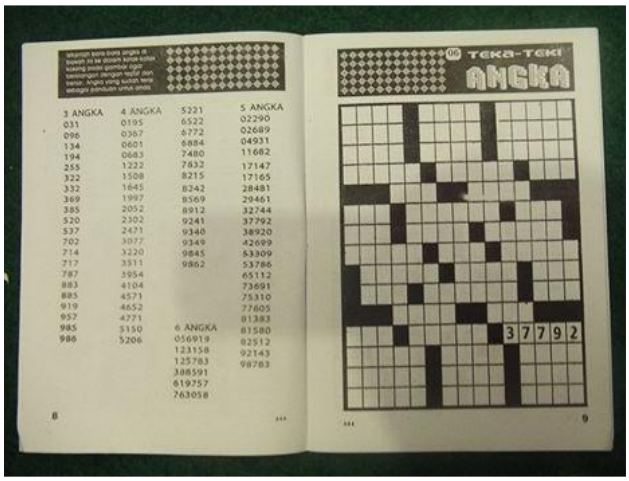
Algoritma *Backtracking* merupakan algoritma yang berbasis pada DFS. DFS (*Deep First Search*) adalah algoritma yang mencari solusi dengan mengutamakan kedalaman, berarti bahwa pencarian akan terus dilanjutkan sedalam mungkin sampai ke bagian akar baru lanjut ke cabang berikutnya.

Selain yang telah disebutkan di atas, algoritma *Backtracking* juga dapat digunakan untuk mencari solusi kuis Tebak Angka. Tebak Angka, lebih dikenal dengan sebutan *crossnumber*, merupakan permainan asah otak yang mengharuskan pemain mengisi *grid* yang kosong dengan angka yang telah ditentukan. *Grid* tersebut terdiri dari *white grid* (kotak putih) dan *black grid* (kotak hitam). *Grid* yang dapat diisi adalah *grid* yang putih. Sedangkan *grid* yang hitam berfungsi sebagai pembatas.

Crossword, induk dari Tebak Angka *'crossnumber',* merupakan permainan asah otak yang sangat terkenal di dunia. Permainan ini pertama kali dipublikasikan oleh

Arthur Wynne pada tahun 1913 yang terbit dikoran New York World. Permainan ini menjadi sangat popular dikalangan masyarakat zaman itu. Pertama kali *crossword* berbentuk *diamond.* Tetapi seiring berjalannya waktu, Arthur Wynne terus memodifikasi *crossword* hingga sampai kebentuk *rectangle* 'kotak' seperti sekarang. Perkembangan *crossword* tidak hanya terhenti pada bentuknya. Banyak variasi yang telah dikembangkan berbasis *crossword*, seperti *chipper crossword, diagramless crossword, fill-in crosswords, fill-in cross numbers.*

Contoh buku *fill-in crossnumbers* yang dijual dimasyarakat umum:



Gambar 1.Buku teka teki angka yang dijual dipasaran

Permainan *crossnumber* memang sangat menarik perhatian pecinta *puzzle solving*. Pemain mengambil langkah awal dikotak yang telah diisi sebelumnya sebagai petunjuk kemudian sedikit demi sedikit mengisi kotak yang lain berdasarkan angka-angka yang telah disediakan hingga semua kotak putih yang awalnya kosong menjadi terisi dengan angka. Tetapi sering saat pengerjaan *puzzle* tersebut para pemain *stuck* karena terdapat kemungkinan solusi yang banyak disatu kolom/baris sehingga membuat para pemain berhenti untuk melanjutkan permainan. Karena itulah, disusun ide dari suatu *solver fill-in crossnumber*, dengan menggunakan algoritma *Backtracking*, sebagai bantuan bagi pemain yang *stuck* dalam pengerjaan *puzzle* ini. Selain itu juga bisa sebagai pemicu timbulnya *puzzle* berbasis *crossword* yang baru sehingga perkembangan *puzzle* ini menjadi semakin berkembang.

II. TEORI

A. Algoritma *Backtracking*

Algoritma *Backtrcking* merupakan algoritma yang digunakan untuk mencari solusi berdasarkan ruang solusi yang ada tetapi tidak semua ruang solusi dieksekusi, hanya yang mengarah kesolusi yang akan diproses. Seperti yang telah dijelaskan sebelumnya bahwa algoritma *Backtracking* berbasis *Deep First Search* yang mengutamakan kedalaman pencarian hingga ke akar, kemudian baru dilanjutkan ke cabang berikutnya. Algoritma ini sering dinyatakan dengan fungsi rekursif (fungsi yang memanggil dirinya sendiri).

1. Properti Umum

Properti umum yang terdefinisi pada algoritma *Backtracking* adalah

a. Solusi persoalan

Menyatakan solusi-solusi dari persoalan yang terdefinisi. Solusi tersebut dinyatakan sebagai vektor dengan n-tuple:

$$X = (x1, x2, \ldots, xn),\ xi \in Si$$

dan mungkin saja,

$$S1 = S2 = \ldots = Sn$$

contoh:

Pada persoalan *Integer Knapsack* terdapat solusi persoalan untuk tiap barang yang dinyatakan dengan xi dan kemungkinan solusi dengan Si,

$$Si = \{0, 1\},$$
$$xi = 0 \text{ atau } 1$$

b. Fungsi Pembangkit nilai x_k

Dinyatakan sebagai:

$$T(k)$$

x_k merupakan komponen vektor solusi. T(k)-lah yang membangkitkan nilai untuk x_k tersebut.

c. Fungsi pembatas

Pada beberapa persoalan fungsi ini dinamakan fungsi criteria. Fungsi pembatas adalah fungssi yang akan menentukan langkah selanjutnya berupa penerusan pencarian solusi ataupun melakukan backtrack.

Dinyatakan sebagai

$$B(x1, x2, \ldots, xk)$$

B bernilai true jika (x1, x2, …, xk) mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika *false*, maka (x1, x2, …, xk) dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi. Inilah yang disebut dengan *pruning*

2. Pengorganisasian Solusi

Semua kemungkinan solusi yang didapat dari persoalan tersebut dinamakan ruang solusi (*solution space*). Contoh pada persoalan *Integer Knapsack* dengan n = 3. Solusi persoalan dinyatakan sebagai

(x1, x2, x3) dengan xi ϵ {0,1}.

sehingga didapat ruang solusinya adalah:

{(0, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1) }

Pada algoritma *Brute Force*, semua kemungkinan solusi akan diuji apakah ia memenuhi kriteria yang terdefinisi atau tidak. Akibatnya, untuk persoalan di atas (n = 3), dapat dihitung bahwa pencarian yang dilakukan seanyak $2^3 = 8$. Tetapi untuk algoritma *Backtracking*, solusi yang dicek hanya solusi yang mengarah ke solusi sebenarnya sehingga jika dalam tahap pencarian cabang yang bernilai *false*, kemungkinan solusi tersebut tidak akan dicek, langsung dilanjutkan kesolusi yang lain. Contoh untuk persoalan *Integer Knapsack* di atas, jika $x_1 = 1$ telah memberikan nilai *false*, maka kemungkinan solusi yang harus diuji algoritma ini menjadi:

{(0, 0, 0), (0, 1, 0), (0, 0, 1), (0, 1, 1)}

Dari hasil *pruning* tersebut, terbukti bahwa algoritma *Backtracking* lebih mangkus dibandingkan *Brute Force* karena jumlah pengujian algoritma *Backtracking* lebih sedikit ($< 2^3$) dibandingkan algoritma *Brute Force* (2^3).

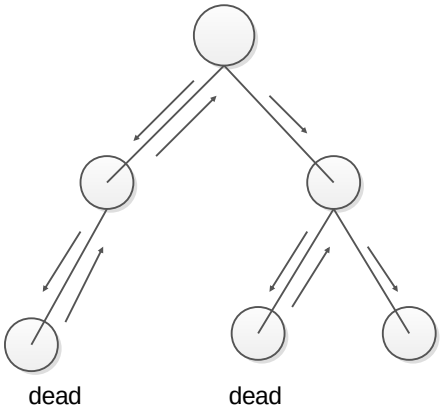
Pencarian solusi menggunakan algoritma *Backtracking* akan menghasilkan pohon ruang status (pohon yang menggambarkan tahapan pencarian semua kemungkinan solusi) yang dinamis.

3. Prinsip Pencarian

Langkah-langkah pencarian solusi sebagai berikut :

a. Solusi, dalam penggambarannya pada pohon ruang status, merupakan lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan *depht-first order* (DFS) yang mengutamakan kedalaman. Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*). Simpul hidup yang *sedang* diperluas dinamakan simpul-E (*Expand-node*).
b. Karena algoritma *Backtraking* berbasis pada DFS, tiap kali simpul-E diperluas, ia hanya akan membangkitkan satu simpul anak sehingga lintasan yang dibangun olehnya bertambah panjang.
c. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*) disebut juga dengan *pruning*. Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*). Pencarian ke lintasan tersebut akan dihentikan, kemudian *backtrack* ke simpul anak berikutnya. Selanjutnya simpul ini menjadi simpul-E yang baru.
d. Pencarian dihentikan bila kita telah sampai pada *goal node*.

Jalur pencarian algoritma *Backtracking* dapat dilihat pada gambar berikut:



Gambar 2. Jalur pencarian *Backtracing*

B. *Fill-in Crossnumbers*

Fill-in Crossnumbers merupakan teka teki asah otak yang banyak diminati oleh masyarakat umum. Tujuan dari teka teki ini adalah mengisi kotak-kotak (*grid*) yang awalnya kosong menjadi penuh dengan angka yang telah disediakan (dikelompokkan berdasarkan *digit*-nya) dan angka-angka tersebut harus terintegrasi satu sama lain.

Tahapan pengerjaan teka-teki ini adalah:

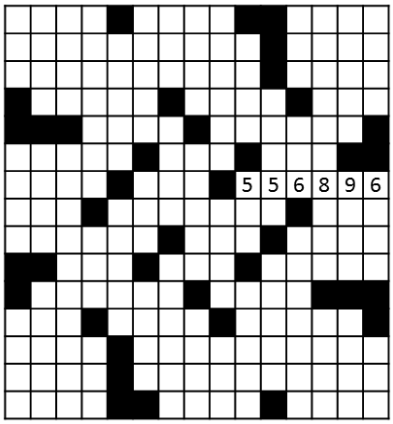
1. Mulai pengerjaan dari petunjuk yang telah diberikan. Tetapi tidak selalu pengerjaan yang dimulai dari titik petunjuk diberikan menjadi prioritas utama. Terkadang di dalam *puzzle*, terdapat hanya terdapat satu buah angka dalam kelompok *digit* tertentu sehingga bisa langsung diisi dengan angka tersebut di-*grid* yang sesuai.
2. Mencari angka yang sesuai dengan *grid* yang telah terisi. Pengisian harus melakukan pencocokan tidak pada satu baris, tapi juga pada satu kolom. Biasanya akan lebih mudah bila bisa mengisi angka dengan *digit* yang besar terlebih dahulu karena akan memperbesar persensi penyelesaian *puzzle*.
3. Mengulangi langkah 2 secara terus menerus sehingga *puzzle* menjadi susunan angka-angka yang terintegrasi satu sama lainnnya.

Kesalahan-kesalahan umum yang terjadi saat pengerjaan teka-teki ini adalah:

1. Tidak teliti. Sikap tidak teliti sering sekali terjadi dalam pengerjaan teka-teki ini. Kesalahan yang timbul dari sikap ini yaitu salah dalam menuliskan angka yang akan dimasukkan ke dalam *grid.* Akibatnya, tahapan pengerjaan yang diawal seharusnya dapat mengantarkan pemain menuju penyelesaian menjadi berakhir (*stuck*) karena kesalahan kecil tersebut.
2. Tergesa-gesa.
3. Memilih angka yang dimasukkan ke dalam *grid*

tanpa pikir panjang. Misalnya terdapat dua angka dengan *digit* lima dan berawalan "22". Pemain kerap langsung memilih angka pertama dari dua angka tersebut. Padahal jawaban yang benar adalah angka yang kedua.

Salah satu contoh *fill-in crossnumber* yaitu:



Gambar 3. Contoh *fill-in crossnumber*
Sumber: http://www.puzzles-to-print.com/number-puzzles/number-fill-in-puzzles.shtml

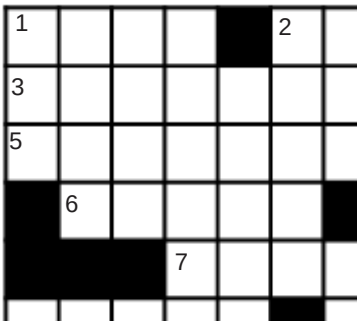
Istilah *grid* dalam makalah ini mengacu kepada pendefinisian "satu kotak" dalam TTS (Teka-teki Silang).

III. Penerapan Algoritma *Bactracking* pada *Puzzle Fill-in Crossnumber*

Seperti yang telah dijelaskan sebelumnya, algoritma *Backtracking*, sama halnya dengan *Sudoku* dan *crosswords*, bisa menyelesaikan *fill-in crossnumbers*. Penerapan algroitma ini berpusat pada pencarian angka yang tepat yang akan diisikan ke dalam *grid*.

Dengan menggunakan contoh *fill-in crossnumber* pada bab kedua, gambaran umum penyelesaian *puzzle* ini yaitu:

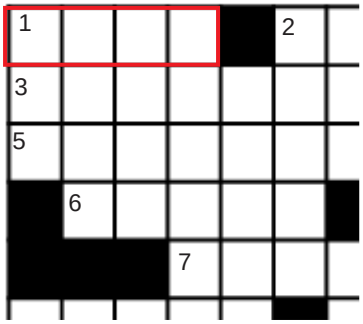
- Melakukan penomoran terlebih dahulu terhadap awal batas *grid* dan *grid* yang disebelah kirinya terdapat kotak hitam (*black grid*) berfungsi sebagai penanda awalan *grid* yang harus diisi.



Gambar 4. Penomoran *grid*

Penomoran *grid* dilakukan secara *logic* oleh *solver*. Selain melakukan penomoran, *solver* juga sekaligus menghitung jumlah *grid* yang harus diisi untuk tiap nomor *grid*. Penghitungan *grid* berhenti ketika bertemu dengan *black grid* atau telah mencapai batas *grid*.

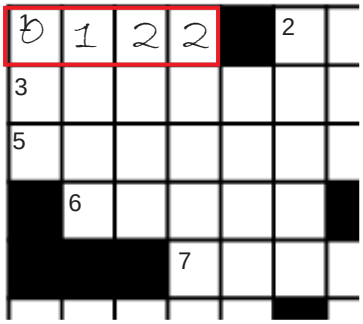
- Memulai pengisian dari nomor *grid* terkecil yang belum diisi oleh angka (nomor *grid* terkecil mengacu pada penomoran yang telah dilakukan sebelumnya).



Gambar 5. Men-*set current grid*

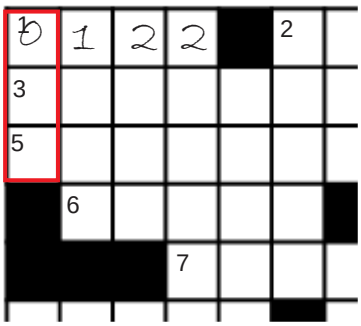
Karena pengisian *grid* diambil berdasarkan penomoran *grid*, Petunjuk yang diberikan diawal menjadi tidak terlalu berguna.

- Mengambil satu angka yang belum dimasukkan ke dalam *grid* yang sesuai antara jumlah *digit* dengan jumlah *grid*-nya. Misalnya, untuk *digit* yang berjumlah empat angka, terdefinisi angka "0122", "1567", dan "4422". Maka *solver* akan mengambil angka paling atas, yaitu "0122".



Gambar 6. Mengisi angka pada *grid*

- Melakukan pengecekkan pada kolom. Apakah angka yang telah terisi di-*grid* terpenuhi untuk perspektif kolom. Pengecekkan dilakukan sebanyak jumlah *grid* baris yang diisi. Jika hasil semua pengujian bernilai *true*, maka pengisian angka dilanjutkan ke nomor *grid* selanjutnya. Jika salah, maka kembali le langkah sebelumnya dan memilih kembali angka yang akan dimasukkan ke dalam *grid*.



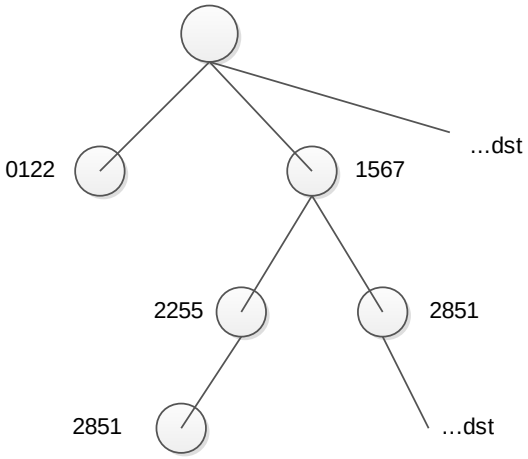
Gambar 7. Mencek kolom, apakah untuk angka ber-*digit* tiga terdapat yang berawalan "0"

Pada contoh di atas, jika pengujian terhadap angka ber-*digit* tiga yang berawalan "0" bernilai *true*, maka pengecekkan dilanjutkan ke-*grid* sebelahnya, yaitu mencek kolom, apakah untuk angka ber-*digit* empat terdapat yang berawalan "1". Jika hasil pengujian bernilai *false*, maka *solver* akan mengganti *grid* baris dari "0122" menjadi "1567". Kemudian melakukan pengecekkan kembali. Disinilah algoritma *Backtracking* diterapkan.

- Pengisian dan pengecekkan dilakukan terus menerus sampai semua *grid* terisi oleh angka.

Ide dari *solver fill-in crossnumber* pada makalah ini memang berpusat pada pengisian berdasarkan baris karena seiring dengan pengisian baris, sedikit demi sedikit kolom juga akan terisi. Ketika pengecekkan terhadap kolom bernilai *true* untuk setiap *grid* kolom, maka angka yang ditunjukkan oleh satu kolom *grid* tersebut ditandai menjadi "telah terisi".

Gambaran kecil pohon ruang status yang terbentuk dari pencarian solusi *puzzle fill-in crossnumber* adalah:



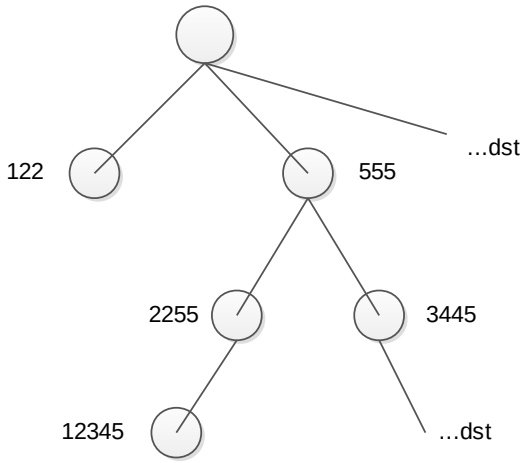
Gambar 8. Gambaran kecil pohon ruang status untuk algoritma *Backtracking* pada *puzzle* tersebut

Terdapat beberapa analisis terhadap pohon ruang status yang terbentuk dari teka-teki *fill-in crossnumber* tersebut. Pertama, untuk setiap *level* (aras) yang sama *digit* angka yang diambil adalah sama dengan *digit* angka yang lain. Kedua, nomor *level* sama dengan nomor *grid* yang diisi sehingga jika pada pohon di atas *level* ke-1 terdapat angka "1567", maka dapat didefinisikan bahwa pada *grid* dengan nomor satu, terisi angka yaitu "1567".

Karena pada contoh *puzzle* di atas *grid* ke satu, dua dan tiga menggunakan angka ber-*digit* empat, perbedaan antar *level* jadi tidak terlalu kelihatan. Tetapi jika digunakan contoh *puzzle* yang lain, yang pada *grid* kesatu terdefinisi untuk angka ber-*digit* tiga, *grid* kedua untuk angka ber-*digit* empat dan *grid* ketiga untuk angka ber-*digit* lima, serta terdefinisi *digit* angka sebagai berikut:

3 ANGKA	4 ANGKA	5 ANGKA
122	2255	12345
555	3445	45334
678	…	…

maka kemngkinan pohon ruang status baru yang terbentuk adalah:



Gambar 9. Kemungkinan pohon ruang status baru yang terbentuk dari *puzzle* lainnya

Dengan pendefinisian *puzzle* seperti di atas, terlihat dengan jelas bahwa tiap *level* mendefinisikan masing-masing nomor *grid*.

Dari penjelasan di atas, penggunaan algoritma *Bactracking* berpusat pada pengisian dan pengeekkan. Gambaran umum kelas-kelas yang digunakan dalam ide penyelesaian *fill-in crossnumber* adalah:

```
class Angka
attribute:
digit : integer
data : array of integer

method:
hitungDigit(data : integer)
{ menghitung digit dari data }

{ bersambung kehalaman berikutnya }
```

```
{sambungan dari halaman sebelumnya}

setDigit(digit : integer)
{ menset digit yang terdefinisi }

konversiIntKeArr(data : integer)
{ mengonversi data ke array of
integer (tiap digit mewakili satu
integer) }
```

Kelas Angka merupakan representasi dari angka yang terdefinisi dalam *puzzle*. Kelas lainnya:

```
class Crossnumber
attribute:
grid : array of Angka
dataAngka : list of array of Angka
isDataAngkaDiisi : list of array of
boolean

method:
procedure initCrossnumber()
{ menginisialisasi bentuk grid,
petunjuk yang diketahui, dan
memasukkan data angka yang
terdefinisi. Pembacaan dilakukan
melalui file eksternal }

function hitungGrid() → integer
{ menghitung jumlah grid yang
terdefinisi untuk masing-masing
nomor grid }

procedure nomorinGrid()
{ menomori grid yang berada di tepi
batas awal dan berada tepat sebelah
kanan grid hitam  }

procedure setGrid(angka : Angka)
{ menset angka pada grid yang
kosong }
```

Kelas Crossnumber merupakan representasi dari *field puzzle*. Terakhir, kelas Solver:

```
class Solver
attribute:
currentGrid : integer
currentAngka : array of integer
puzzle : Crossnumber

method
function ambilAngka(digit: integer)
→ Angka
{ mengambil angka yang terdapat
didalam dataAngka, digit berfungsi
sebagai index }

{ bersambung ke kolom selanjutnya }
```

```
{ sambungan kolom sebelumnya }

function cekKolom() → boolean
{ mencek apakah pada kolom tersebut
terdefinisi angka yang diketahui
pada dataAngka }

procedure isiGrid(nomor: integer)
{ mengisi grid dengan angka yang
diambil dari dataAngka }

procedure backtracking(nomor:
integer)
{ melakukan algoritma backtracking}
```

Kelas Solver adalah kelas yang akan menangani penyelesaian *puzzle* berikut dengan algoritma *Backtracking*-nya.

Skema umum penerapan algoritma *Backtracking* dari kelas Solver tersebut adalah:

```
procedure backtracking(nomor)
for each dataAngka yang bernilai
true do
      setGrid(Angka)
      if(cekKolom())
            isiGrid()
      else
            backtracking(nomor+1)
```

IV. KESIMPULAN

Algoritma *Backtracking* adalah algoritma yang berdasar kepada algoritma DFS (*Deep First Search*) yang mengutamakan kedalaman pencarian dibandingkan pelebaran pencarian. Algoritma ini mencari solusi dengan cara yang lebih mangkus (efesien) dibandingkan algoritma *Brute Force* karena algoritma *Backtracking* mempunyai properti seperti solusi persoalan, fungsi pembangkit, dan fungsi pembatas yang tidak dimiliki oleh algoritma *Brute Force*. Algoritma ini memiliki dua versi skema. Pertama versi skema dengan iteratif dan kedua versi skema dengan rekursif. Biasanya, algoritma *Backtracking* disajikan dalam versi iteratif. Penerapan algoritma *Backtracking* kerap diterapkan pada pencarian solusi *puzzle* seperti *eight queens puzzle*, *crossword*, *maze*, *Sudoku* dan *fill-in crossnumber*. Selain *puzzle*, algoritma *Backtracking* juga digunakan pada masalah optimasi kombinatorial. Algoritma ini juga diterapkan pada *Logic Programming*.

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF2211 Strategi Algoritma". Program Studi Teknik Informatika STEI ITB, 2009.

[2] Donald E. Knuth .1968. *The Art of Computer Programming*. Addison-Wesley.

[3] http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html#QQ1-51-128 Diakses pada tanggal 17 Mei 2014

[4] http://www.worksheetworks.com/puzzles/crossnumber.html
Diakses pada tanggal 16 Mei 2014
[5] http://www.puzzles-to-print.com/number-puzzles/number-fill-in-puzzles.shtml
Diakses pada tanggal 18 Mei 2014

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, 19 Mei 2014

ttd

Yollanda Sekarrini - 13512051