Aplikasi dan Analisis Algoritma BFS dan DFS dalam Menemukan Solusi pada Kasus *Water Jug*

Rizkydaya Aditya Putra – NIM : 13506037

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha no. 10, Bandung
e-mail: if16037@students.if.itb.ac.id

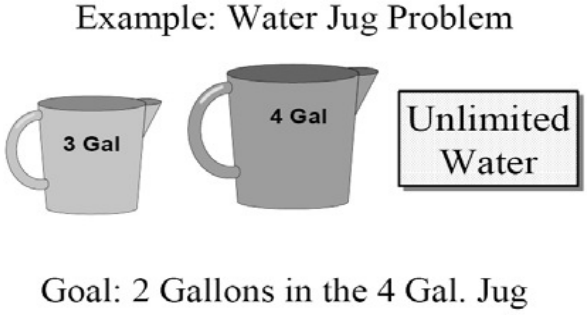
ABSTRAK

Dalam sebuah kasus *water jug* (kendi air) diberikan dua buah kendi air yang masing-masing memiliki kapasitas isi 3 liter dan 4 liter. Solusi yang diharapkan adalah bagaimana memperoleh 2 liter air di dalam ember yang berkapasitas 4 liter dengan penakaran hanya memakai dua buah ember tersebut. Kemungkinan solusi dari permasalahan ini dapat direpresentasikan dengan pohon pencarian atau *tree* search. Salah satu algoritma yang bisa diterapkan untuk menelusuri setiap simpul dalam tree search ini adalah *Breadth First Search* dan *Depth First Search* atau biasa disingkat BFS dan DFS. Inti dari algoritma ini yaitu mengeksplor simpul yang levelnya lebih bawah.
Bagaimanakah caranya? Dalam menemukan solusinya digunakan algoritma BFS dan DFS sekaligus analisa algoritmanya.

Kata kunci: *Water Jug*, Algoritma BFS dan DFS, Analisis Algoritma.

1. PENDAHULUAN

Kasus *water jug* yang terdapat pada IEEE Gamebook yaitu jika ada 2 kendi kosong dengan kapasitas 5 liter dan 3 liter, bagaimana caranya mendapatkan tepat 4 liter pada kendi yang berkapasitas 5 liter ? Masalah ini sangat terkenal dan pernah diangkat ke layar lebar dalam film *Die Hard* 3.



Gambar 1. Contoh *Water Jug Problem*

Melalui paper ini diasumsikan *state* awal kedua kendi dalam keadaan kosong. Misalkan kendi pertama berkapasitas p liter dan kendi kedua berkapasitas q liter dengan syarat $1 < p < q$. Dari p dan q dapat kita peroleh sejumlah bilangan bulat antara 1 sampai p+q. Kita mulai dengan menyederhanakan masalah tersebut dengan memberikan suatu contoh untuk menggambarkan algoritmanya. Kita akan mencari secara sequensial setiap aksi yang nantinya menjadi sebuah solusi.

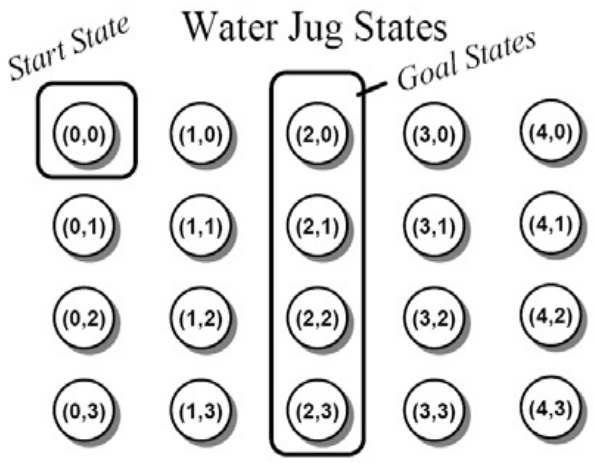
2. METODE

2.1 Rumus Pencarian

State : deskripsi kasus dalam dunia nyata
- beriisikan kombinasi dari volume kendi

State Space : himpunan semua *state* yang mungkin

P={0,1,2,3} dan q={0,1,2,3,4}
- 4 * 5 = 20 *state*



Gambar 2. Contoh State-state dalam Water Jug

Search Node : struktur data dimana informasi *state* disimpan untuk pencarian
Search Tree : Graph berarah G=(V,E)
- V adalah himpunan dari node (*state*)
- E adalah himpunan dari busur (perubahan aksi dari *state* ke *state*)

Initial State : Inisial *state* awal
- Kedua kendi kosong.

Successor States : Kumpulan dari *state* yang dibangkitkan oleh aksi di *state* tertentu.
Goal State : *State* yang memenuhi objek pencarian
- Kendi dengan isi 4 liter

Goal Test : Cara memutuskan *state* tujuan.
Open list : List dari *state-state* yang dibangkitkan oleh *state* sebelumnya pada saat pencarian solusi.
Closed list : List dari *state-state* yang telah dikunjungi pada saat pencarian solusi.
Solution : Jalur *(path)* dari inisial *state* ke goal *state*.

2.2 Metode Pencarian *Water Jug*

State : *Order* "AB"
- Dimisalkan *state* A adalah kendi dengan kapasitas 3 liter dan *state* B adalah kendi dengan kapasitas 4 liter.

State space : himpunan semua *state* yang mungkin.
- Terdapat 14 kemungkinan *state*.

Edges (sisi) : Semua kemungkinan aksi dari *state*.

1. Isi kendi 3 liter. [A,B] → [3,B]
2. Isi kendi 4 liter. [A,B] → [4,B]
3. Pindahkan seluruh isi kendi 3 liter ke 4 liter [A,B]→[0,A+B] Jika A+B≤4
4. Pindahkan seluruh isi kendi 4 liter ke 3 liter [A,B]→[A+B,0] Jika A+B≤3
5. Isikan kendi 3 liter dari kendi 4 liter [A,B]→[3,A+B-3] Jika A+B>3
6. Isikan kendi 4 liter dari kendi 3 liter [A,B]→[A+B-4,4] Jika A+B>4
7. Kosongkan kendi 3 liter. [A,B]→[0,B]
8. Kosongkan kendi 4 liter. [A,B]→[A,0]

2.3 Algoritma Pencarian *Water Jug*

```
Initial state : State "00".
OPEN={00}//List dari state-state
   yang dibangkitkan
CLOSED={}//List dari state-state
   yang telah dikunjungi
while OPEN is not empty {
X = state removed from OPEN
if X is (n2) or (2n) then
return solution
else {
add X to CLOSED
generate successor states via
   actions on X
ignore successors already in OPEN
   or CLOSED
add successors to OPEN
}
}
return FAILURE // Tidak ada solusi
```

Goal state : Dengan *state* "n2" atau "2n".
- Dimana n adalah bilangan bulat berapapun.

Open List:

Pada contoh ini, kita menggunakan metoda FIFO atau antrian untuk membangkitkan *state* pada *list open*. Cara ini disebut juga dengan *breadth-first search (BFS).*

Ada cara lain untuk menangani *state* pada *list open* yaitu dengan menggunakan metoda LIFO atau *stack*. Cara ini disebut juga dengan *depth-first search (DFS).*

2.4 Pohon Pencarian *Water Jug*

Dalam strategi pencarian dapat dilakukan dengan menggunakan :

2.4.1. *BREADTH-FIRST SEARCH* (BFS)

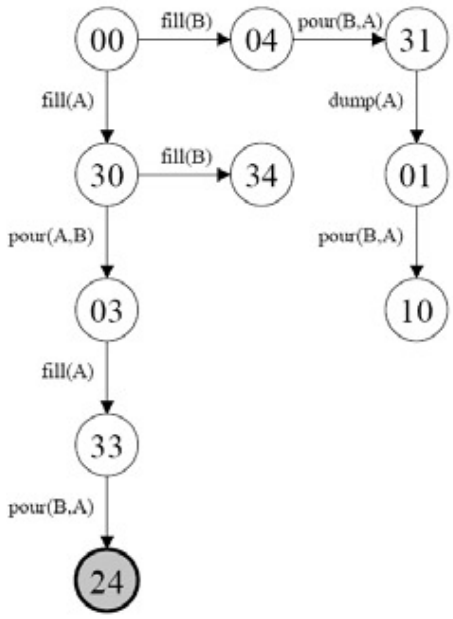
Breadth-First Search akan mengunjungisemua node pada kedalaman n sebelummengunjungi node lain pada kedalaman n+1 atau dengan kata lain akan dilakukan penelusuran per level.

Algoritma BFS:

create nodelist (a queue) and initialize to the start state.
repeat until a goal state is found or nodelist = {}
remove the first element from nodelist:
apply all possible rules and add resulting states to nodelist

Tabel 1. Tabel *Water Jug* dengan BFS

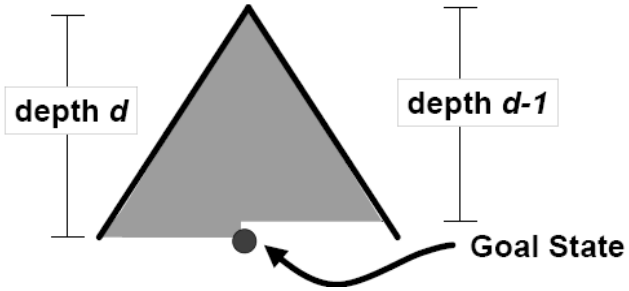
State	Open	Closed
--	00	--
00	30, 04	00
30	04, 03, 34	00, 30
04	03, 34, 31	00, 30, 04
03	34, 31, 33	00, 30, 04, 03
34	31, 33	00, 30, 04, 03, 34
31	33, 01	00, 30, 04, 03, 34, 31
33	01, 24	00, 30, 04, 03, 34, 31, 33
01	24, 10	00, 30, 04, 03, 34, 31, 33, 01
24	**Goal!!**	



Gambar 3. Diagram *State Water Jug* dengan BFS

Karakteristik BFS:

- Jika ada solusi, BFS akan menemukannya
- BFS akan menemukan solusi dengan jalur terpendek
- BFS tidak akan terjebak dalam *looping*
- BFS membutuhkan *space* untuk menyimpan *node list* antrian dan *space* yang dibutuhkan dan mungkin space yang dibutuhkan adalah cukup besar.
  Analisis waktu dan *space* BFS
- Asumsi :
  - Terdapat solusi dalam pohon.
  - Pencarian *tree* adalah secara terurut
    dan memiliki faktor pencabangan b.
  - *Goal state* terletak pada kedalaman d pada *tree*.
  - *Goal state* pada kasus umum biasanya
    - terdapat pada bagian tengah dari pohon.



Gambar 4. *Goal State* pada kasus rata - rata

BFS *Space Requirement*:

Antrian diinisialisasi dengan satu *state*. Kemudian antrian akan berisi b *state* dan kemudian semua b states akan diproses berdasarkan antriannya dalam satu level.

BFS *Time Analysis (Complexity)*

Merupakan pengukuran waktu dalam mengunjungi sejumlah *state*. Diasumsikan pada kasus *water jug* setiap proses node memiliki waktu yang sama.

Time = # level 1 nodes + #level 2 nodes + …
+ # level *d-1* nodes + (# level *d* nodes/2).
= 1 + b + b2 + b3 + … + bd-1 + bd/2
Є O(bd)

2.4.2 *Depth-First Search* (DFS)

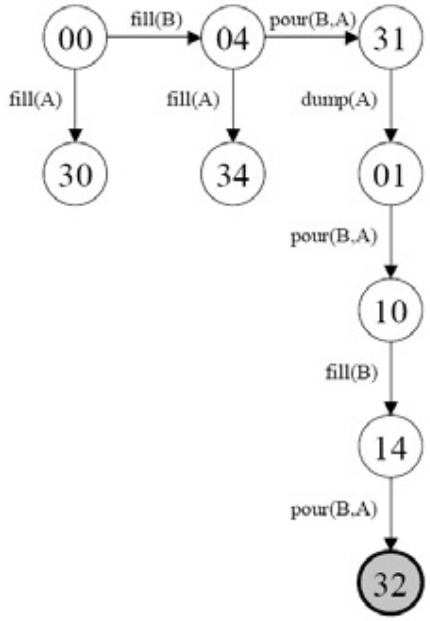
Depth-First Search akan memproses semua anak/pilihan dari sebuah node sebelum mempertimbangkan node saudaranya (node pada kedalaman yang sama). *Depth-First Search* menggunakan *stack*, lain halnya dengan BFS yang menggunakan *queue*. *Depth-First Search* dapat dengan mudah digambarkan secara rekursif.

Algoritma DFS

create *nodelist* (a stack) and initialize to the start state.
repeat until a goal state is found or *nodelist* = { }
pop the first element from nodelist,:apply all possible rules and add (push)
resulting states to nodelist.

Tabel 2. Tabel *Water Jug* dengan DFS

State	Open	Closed
--	00	--
00	04, 30	00
04	31, 34, 30	00, 04
31	01, 34, 30	00, 04, 31
01	10, 34, 30	00, 04, 31, 01
10	14, 34, 30	00, 04, 31, 01, 10
14	32, 34, 30	00, 04, 31, 01, 10, 14
32	Goal!!	



Gambar 5. Diagram *State Water Jug* dengan DFS

Karakteristik *Depth-First Search*

- Tidak perlu menyimpan jalur dari list yang terdiri dari banyak *state.*
- Dapat menemukan solusi dengan sangat cepat (bisa juga tidak pada kasus tertentu).
- *Pruning/pemutusan* dapat terjadi
- – contoh: branch-and-bound
- Dapat terjebak dalam *looping*

DFS *Space Analysis*

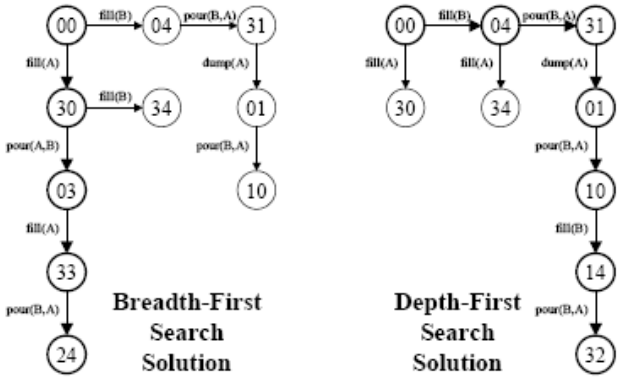
- Setelah langkah pertama *stack* akan berisikan sejumlah b *node*
- Setelah langkah kedua *stack* akan berisikan sejumlah *(b - 1) + b nodes*.
- Setelah langkah ke tiga *stack* akan berisikan *(b-1) + (b-1) + b nodes*.
- Setalah d(kedalaman) pertama *stack* akan bernilai *(b-1)*d + 1 nodes* (kedalaman maksimum)

DFS *Time Analysis (Complexity)*

- Dalam kasus terbaik, *goal state* berada pada pencarian pertama kali pada kedalaman d
- Dalam kasus terburuk, *goal* state akan terakhir kali
  diperiksa yaitu akan melihat seluruh node

$$1+b+b2+b3+\ldots+bd = (bd+1-1)/(b-1)$$
$$\in O(bd)$$

2.5 BFS vs DFS

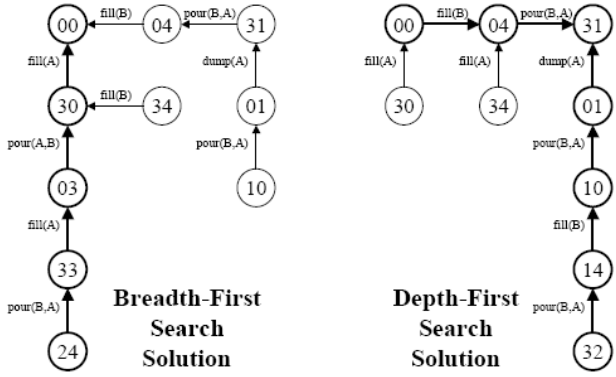


Gambar 6. Perbandingan BFS dan DFS

Hal-hal penting dalam menemukan solusi.

- Dalam kasus ini, solusi kita adalah sebuah
- keterurutan aksi-aksi atau jalur dari inisial *state* ke goal *state*.
- Dalam pencarian node adalah irresponsible, karena jalur node anak dari parent *state* tidak disimpan.
- Dalam prakteknya, kita membuat setiap node pengingat parentnya, kemudian dilakukan backtrack dari goal *state* ke initial *state*.

Responsible Children:



Gambar 7. *Responsible Children*

3. KESIMPULAN

Dari uraian penyelesaian masalah *water jug* dengan menggunakan algoritma BFS dan DFS diperoleh bahwa kompleksitas waktu algoritma BFS dan DFS yang dinotasikan dengan Big-oh yaitu sebanding dengan O(bd). Ini berarti kedua algoritma tersebut memerlukan waktu yang relatif sama dalam proses pencarian solusi. Namun dari karakteristiknya, *space* requirement, dan *cost* yang dibutuhkan untuk DFS adalah lebih baik meskipun *cost* yang dibutuhkan untuk melakukan setiap aksi dalam *water jug* adalah sama.

Dari uraian di atas dapat ditarik beberapa kesimpulan mengenai algoritma *Depth First Search* yang diterapkan pada permasalahan *water jug* di antaranya :

Kelebihan *Depth First Search* adalah:

1. Pemakain memori hanya sedikit, berbeda jauh dengan *Breadth First Search* yang harus menyimpan semua node yang pernah dibangkitkan.
2. Jika solusi yang dicari berada pada *level* yang dalam dan paling kiri, maka *Depth First Search* akan menemukannya secara cepat.

Kelemahan *Depth First Search* adalah:

1. Jika pohon yang dibangkitkan mempunyai level yang dalam (tak terhingga), maka tidak ada jaminan untuk menemukan solusi (*incomplete*).
2. Jika terdapat lebih dari satu solusi yang sama tetapi berada pada level yang berbeda, maka pada *Depth First Search* tidak ada jaminan untuk menemukan solusi yang paling baik (Tidak Optimal).

Selain itu, dari uraian di atas dapat diketahui bahwa metode *Depth First Search* dengan *backtrack* dapat lebih mengefisienkan penelusuran pada pohon pencarian untuk menemukan solusi.

REFERENSI

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF2251 Strategi Algoritmik. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[2] http://blue.utb.edu/ltang/cosc4350/ch3.ppt. Tanggal akses 18 Mei 2008 pukul 14.21 WIB.

[3] http://www.cs.rpi.edu/~hollingd/ai/lectures/blind_search.pdf. Tanggal akses 18 Mei 2008 pukul 18.00 WIB.

[4]http://en.wikipedia.org/wiki/Depth-first_search. Tanggal akses 19 Mei 2008 pukul 12.11 WIB.

[5] Kusdiarti, Ellyana. Penerapan Algoritma Depth First Search pada Permasalahan Water Jug. Volume I, 2005, halaman 2-3.