Penggunaan Algoritma DFS dan BFS pada Permainan Three Piles of Stones

Muharram Huda Widaseta
NIM 13508033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
hyouda@students.itb.ac.id

***Abstract*—*Three Piles of Stone* adalah salah satu mini game yang menarik pada *game tales of destiny 2*, *game* ini merupakan *game* yang sederhana, tidak membutuhkan alat bantuan macam-macam, dapat dimainkan oleh 2 atau lebih pemain, dan *game* ini menarik untuk dimainkan. game ini mempunyai inisialisasi 3 bilangan bulat positif sembarang yang dilambangkan dengan jumlah batu, dan kondisi pemenang adalah yang mengambil batu terakhir.**

Algoritma DFS dan BFS dipakai untuk melakukan penelusuran solusi yang akan digunakan pada saat pengambilan batu, setiap turn setelah musuh mengambil batu, algoritma DFS atau BFS akan mencari solusi, dan mengambil batu sesuai solusi yang ditunjukkan.

***Index Terms*—permainan, Batu, tumpukan, algoritma**

1. PENDAHULUAN

Three Piles of Stone adalah sebuah permainan yang memakai 3 tumpukan batu, dan setiap tumpukan batu mempunyai sembarang jumlah batu. Permainan ini ditunjukkan pada permainan berjudul *Tales of destiny 2* sebagai sebuah mini games. *Tales of destiny 2* adalah permainan dengan genre rpg yang dirilis untuk konsol *Sony Playstation*.

Cara Bermain *Three Piles of Stone* adalah dengan kita menyiapkan 3 buah tumpukan batu (batu dalam hal ini bisa diganti oleh koin, kue, atau bahkan hanya angka dalam kertas). Terdapat 2 pemain untuk memainkan permainan ini. Setiap pemain mempunyai giliran masing-masing untuk memainkan permainan ini. Setiap giliran, pemain bisa mengambil 1 sampai 3 buah batu dalam 1 tumpukan. Pemain yang berhasil mengambil batu terakhir pada tumpukan-tumpukan batu tersebut itulah pemain yang menang.

Gambar 1: Permainan *Three piles of stone*

Peraturan dalam *game* ini adalah, dalam setiap giliran pemain, harus mengambil batu dari tumpukan batu tersebut. Tidak boleh mengembalikan batu jika sudah terambil, tidak boleh mengambil batu lebih dari yang sudah ditentukan, tidak boleh mengambil batu dari lebih dari 1 tumpukan.

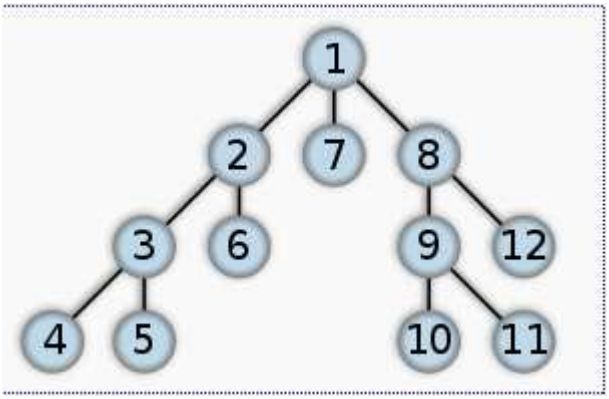
Dalam memenangkan permainan ini, dapat diterapkan banyak strategi. Kebanyakan strategi yang tersebar luas di internet adalah dengan menggunakan *heuristic*. *Heuristic* yang digunakan adalah dengan membuat batu pada masing-masing tumpukan menjadi kelipatan 4. Pada saat mengaplikasikan, cara ini kurang berhasil, karena mempunyai banyak lubang kemungkinan. Saya akan mencoba membuat solusi cara memenangkan permainan ini dengan menggunakan DFS.

2. ALGORITMA DFS DAN BFS

2.1. Depth First Search

DFS adalah algoritma untuk melakukan traversal atau pencarian pada sebuah graf atau pohon. Dimulai dari simpul akar, pencarian akan dilakukan dan mengeksplorasi sejauh mungkin pada tiap cabang sebelum akhirnya melakukan *backtracking*.

DFS bekerja dengan mengembangkan simpul anak pertama pada pohon kemudian mencari lagi lebih dalam pada anak simpul tersebut sampai simpul yang dicari ditemukan atau sampai tidak ada lagi simpul anak yang bisa dikunjungi. Setelah itu, *backtrack* dilakukan, kembali ke simpul terakhir yang masih memiliki simpul anak yang belum dikunjungi, dan seterusnya. Pada implementasi non-rekursif, simpul-simpul yang baru dikembangkan dimasukkan pada sebuah *stack* untuk eksplorasinya.

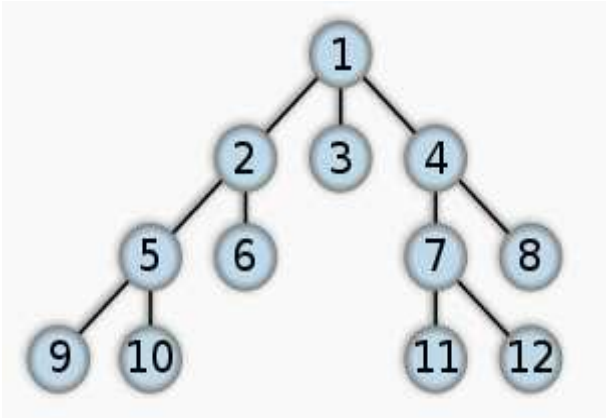


Gambar 2: Contoh urutan kunjungan algoritma DFS

Secara umum, algoritma DFS adalah sebagai berikut:
1. Masukkan simpul akar ke dalam *stack*.
2. Ambil sebuah simpul pada antrian kemudian diperiksa.
a. Jika elemen yang dicari ditemukan pada simpul ini, hentikan pencarian.
b. Jika tidak, masukkan simpul-simpul anak simpul ini ke dalam antrian.
3. Jika antrian kosong, semua simpul graf telah diperiksa dan elemen tidak ditemukan. Hentikan pencarian.
4. Ulangi mulai langkah 2.

2.2. Breadth First Search

Dalam teori graf, BFS adalah algoritma pencarian graf yang mulai mencari dari simpul akar dan kemudian mencari ke semua simpul yang bertetangga. Untuk setiap simpul-simpul tetangga yang telah dikunjungi, simpul-simpul lain yang belum dikunjungi yang bertetangga dengan simpul tersebut akan ditelusuri. Begitu seterusnya sampai pencarian berakhir. Contoh kasus persoalan graf yang diselesaikan menggunakan algoritma BFS adalah pada gambar 1.



Gambar 3: Contoh urutan kunjungan algoritma BFS

Dari sudut pandang algoritma ini, semua simpul tetangga atau simpul anak yang didapat dengan mengembangkan sebuah simpul akan dimasukkan ke dalam FIFO *queue*. Pada implementasinya, informasi simpul-simpul akan disimpan pada sebuah kontainer yang berupa *queue* atau *linked list*.
Secara umum, algoritma BFS adalah sebagai berikut:
1. Masukkan simpul akar ke dalam antrian.
2. Ambil sebuah simpul pada antrian kemudian diperiksa.
a. Jika elemen yang dicari ditemukan pada simpul ini, hentikan pencarian.
b. Jika tidak, masukkan simpul-simpul anak simpul ini ke dalam antrian.
3. Jika antrian kosong, semua simpul graf telah diperiksa dan elemen tidak ditemukan. Hentikan pencarian.
4. Ulangi mulai langkah 2.

3. METODE

3.1 Pemecahan dengan Heuristik

Pada *heuristic*, pemecahan masalah ini adalah dengan membuat musuh melakukan giliran saat kelipatan dari semua tumpukan adalah 4. Karena untuk mengambil batu harus 1 sampai dengan 3 buah batu, misalnya musuh mengambil N-batu, dan setelah musuh mengambil batu tersebut, kita ambil pada tumpukan yang sama dengan musuh batu sebanyak 4-N batu. Dengan cara apapun, jika kita sudah berada pada keadaan tersebut. Dengan melakukan cara tersebut, kita sudah pasti menang.

Ambil	Giliran	Tumpukan 1	Tumpukan 2	Tumpukan 3
inisial	Kita	8	12	4
2	musuh	6	12	4
2	Kita	4	12	4
1	musuh	4	11	4
3	Kita	4	8	4
3	musuh	4	8	1
1	Kita	4	8	0
2	musuh	2	8	0
2	Kita	0	8	0
3	musuh	0	5	0

1	kita	0	4	0	e.
2	musuh	0	2	0	
2	kita	0	0	0	f.

Tabel 1: Tabel urutan giliran bermain pada *Three Piles of Stone*

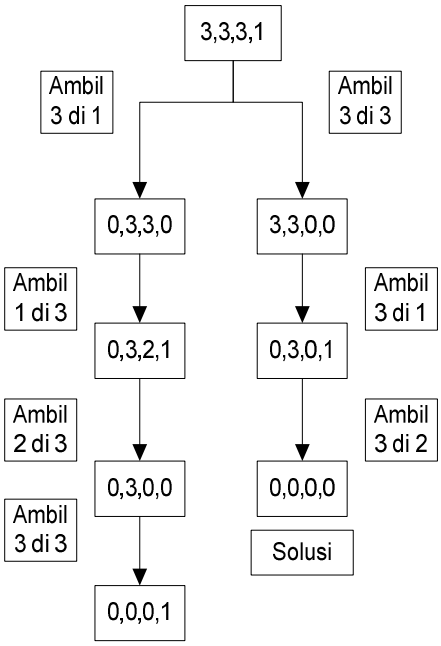
Sekilas algoritma *heuristic* ini sangat mangkus, dan terlihat bahwa game ini mudah sekali. Tapi pada aplikasi permainan tersebut, musuh dan kita tidak akan pernah memberikan keadaan tersebut kepada lawan bermain. Kadang juga terjadi perhitungan yang lain, karena pada keadaan sebenarnya, kita dan musuh tidak membiarkan hal ini terjadi, sehingga pengambilan tidak beraturan dan kemungkinan menang kembali menjadi *random*.

3.2 Pemecahan dengan Algoritma DFS

Secara sederhana, penerapan algoritma DFS adalah dengan melakukan perubahan *state* pada salah satu tumpukan diambil satu-persatu. Sampai tumpukan tersebut habis, lalu pada daun sebelahnya pengambilan dilakukan 2 atau 3 jika memungkinkan, lalu mengambil batu pada tumpukan berikutnya.
Pada algoritma DFS tidak diperlukan penanda, karena algoritma dilakukan pada program dengan menggunakan rekursif. Secara garis besar, algoritmanya sebagai berikut:

a. Membuat global yang menandakan rekursif belum berakhir <bool end> dan *vector* yang menandakan stack langkah pada giliran <vector <int[]>langkah>, pastikan *vector* kosong, dan global end bernilai false sebelum memanggil fungsi.
b. Fungsi memuat beberapa parameter yaitu: int tumpukan1 yang menunjukkan banyaknya batu pada tumpukan 1 ,int tumpukan2 menunjukkan jumlah batu pada tumpukan 2, int tumpukan3 menunjukkan jumlah batu pada tumpukan 3, bool player menunjukkan player mana yang sedang berjalan true untuk kita, false untuk musuh.
c. Basisnya adalah saat tumpukan1, tumpukan2, dan tumpukan3 jumlah totalnya adalah 0, maka global end akan diisi dengan !player, dan tumpukan-tumpukan batu akan disimpan state-nya, ditambahkan pada *vector*.
d. Rekursi-nya adalah jika jumlah tumpukan-tumpukan lebih dari 0. Didalamnya jika tumpukan1 lebih dari 0, maka dilakukan perulangan oleh i dari 1-3. Didalamnya lagi, jika sisa tumpukan – i >=0 maka dilakukan pengecekan pada global end, jika end masih false maka akan dilakukan penambahan state tumpukan kedalam *vector*, rekursi fungsi dfs dengan parameter: tumpukan1 – I,tumpukan2,tumpukan3, dan !player, dan melakukan delete *vector* terakhir, jika dicek global end adalah false, karena jika keluar dari rekursi dan global end adalah false maka belum menemukan solusi.
e. Didalam rekursi juga ditambahakan prosedur seperti diatas untuk tumpukan2 dan tumpukan3.
f. Untuk menunjukkan *next step*, tinggal menunjukkan state pertama yang disimpan.

Penerapan algoritma DFS di atas untuk menyelesaikan permainan dengan kondisi seperti pada gambar dibawah akan menghasilkan pohon yang menyimpan *state* seperti dibawah.



Gambar 3: Contoh urutan kunjungan algoritma BFS

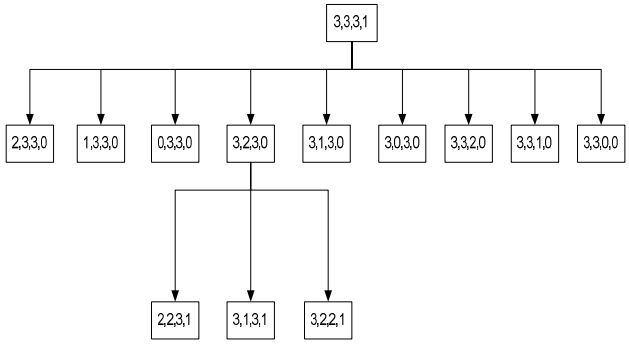
3.3 Pemecahan dengan Algoritma BFS

Pada dasarnya, penerapan algoritma BFS adalah dengan memeriksa semua kemungkinan dari awal adalah dari tumpukan 1 ke tumpukan 3, setiap tumpukan dicari kemungkinan dari pengambilan 1-3. Dan setiap state mempunyai player yang berbeda dengan daunnya. Jadi, jika langkah pencarian solusi dibuat menjadi sebuah pohon, akan terbentuk pohon merentang yang urutan pembentukan daunnya adalah secara horizontal terlebih dahulu.
Algoritma BFS yang dapat diterapkan untuk menyelesaikan permainan ini memerlukan beberapa parameter yang sama seperti parameter pada fungsi DFS, yaitu, tumpukan1, tumpukan2, tumpukan3, dan player.
Algoritmanya adalah sebagai berikut:

a. Membuat global *vector* yang menandakan *queue* langkah pada giliran <vector <int[]>langkah>, pastikan vector kosong sebelum memanggil fungsi.
b. Fungsi memuat beberapa parameter yaitu: int tumpukan1 yang menunjukkan banyaknya batu pada tumpukan 1 ,int tumpukan2 menunjukkan jumlah batu pada tumpukan 2, int tumpukan3 menunjukkan jumlah batu pada tumpukan 3, bool

player menunjukkan *player* mana yang sedang berjalan true untuk kita, false untuk musuh.

c. Pada awal kita melakukan inisiasi pembuatan *vector* yang akan digunakan untuk meyimpan *state*.
d. Pertama kita masukkan state awal keadaan tumpukan-tumpukan batu kedalam *vector*
e. Lalu melakukan perulangan dari awal *vector* ke akhir *vector*
f. Hal yang diulang adalah yang pertama mengecek apakah elemen pada *vector* itu solusi atau bukan dengan melihat jumlah tumpukan1, tumpukan2, dan tumpukan3 apakah sama dengan nol, lalu melihat player, jika player adalah false, maka itu adalah solusi ,lalu fungsi akan menampilkan semua *vector* sampai ke *vector state* tersebut. jika tidak, maka akan menghapus *vector* tersebut. lalu melakukan perulangan untuk i=1sampai i=3 pada tumpukan 1, jika tumpukan1 – i>=0 maka akan menambahkan *vector*.
g. Ditambahkan juga perosedur diatas untuk tumpukan2 dan tumpukan3.
h. Saat diakhir, dilakukan pengecekan, apakah *vector* itu berada di belakang daun atau tidak, jika berada dibelakang daun dan bukan solusi, maka akan menghapus *vector parent*.
i. Saat diakhir program, jika tidak menemukan solusi diberi penanganan untuk mengeluarkan “tidak bisa menang”, sebaliknya, jika menemukan solusi maka akan ditampilkan semua *vector* sampai *vector* yang dilacak tersebut, sehingga tahu urutan *state* untuk mencapai solusi.



Gambar 4: Contoh urutan kunjungan algoritma BFS

Penerapan algoritma BFS di atas adalah gambaran singkat langkah untuk menyelesaikan permainan, untuk urutan pemeriksaannya adalah dari tumpukan pertama, mengambil dari 1-3 yang mungkin, lalu cari pada tumpukan2 dan tumpukan3. Solusi pada bagian ini akan langsung memperlihatkan semua langkah yang bisa diikuti untuk menunjuk kepada solusi.

4. Analisis

Jika melihat contoh kasus diatas, terlihat bahwa penggunaan algoritma DFS lebih efisien dalam menyelesaikan permainan ini dibandingkan dengan algoritma BFS. Ini adalah contoh kasus yang membuat algoritma DFS lebih bagus daripada algoritma BFS. Sebenarnya tergantung kasus yang diberikan juga, BFS akan lebih bagus jika diberikan kasus yang kemungkinan selusinya berada pada tingkat yang sedikit dan jauh dari pencarian awal, sedangkan DFS akan lebih bagus jika kasusnya mempunyai solusi yang dalam dan berada di dekat inisial pencarian.

Contoh kasus dengan pemakaian BFS yang lebih bagus

Gambar 5: Contoh kasus dengan algoritma BFS

Ambil	Giliran	Tumpukan 1	Tumpukan 2	Tumpukan 3
inisial		7	11	6
2	Kita	7	11	4
3	musuh	7	8	4
3	kita	4	8	4
2	musuh	4	6	4
2	kita	4	4	4
1	musuh	4	3	4
3	kita	4	0	4
1	musuh	3	0	4
3	kita	0	0	4
2	musuh	0	0	2
2	kita	0	0	0

Tabel 1: Tabel urutan giliran bermain pada kasus pada gambar 5

Pada table diatas dapat dilihat bahwa penangan tumpukan 3 terlebih dahulu, sehingga dapat diasumsikan bahwa pencarian pada tumpukan 3 diaplikasikan, kalau dengan DFS, dengan tumpukan 3 terlebih dahulu, waktu pencarian menjadi lebih lama.

Contoh kasus dengan pemakaian DFS yang lebih bagus

Gambar 6: Contoh kasus dengan algoritma DFS

Ambil	Giliran	Tumpukan 1	Tumpukan 2	Tumpukan 3
Inisial		5	7	9
3	kita	5	4	9
2	musuh	5	4	7
2	kita	5	4	5
3	musuh	2	4	5
3	kita	2	4	2
1	musuh	2	3	2
3	kita	2	0	2
1	musuh	1	0	2
1	kita	1	0	1
1	musuh	0	0	1
1	kita	0	0	0

Tabel 3: Tabel urutan giliran bermain pada kasus pada gambar 6

Dari table diatas dapat diketahui bahwa pohon mempunyai tingkat yang tinggi, dan banyak pengurangan 1 pada tumpukan bawah, sehingga dapat diasumsikan bahwa kasus diatas lebih baik dengan menggunakan DFS.

5. Kesimpulan

Permainan seperti *Three Piles of stone* ini dapat dipecahkan dengan algoritma DFS dan BFS yang sesuai. Namun untuk masalah jumlah urutan langkah akan berbeda-beda tergantung beberapa hal, misalnya ketinggian pohon, lebarnya pencarian. Oleh karena itu, algoritma mana yang secara umum lebih baik untuk menyelesaikan permainan seperti ini masih sulit ditentukan. Tapi dari penanganan kasus diatas, dengan menggunakan algoritma DFS dan BFS, kita bisa lebih banyak menyelesaikan kasus daripada metode *heuristic*.

REFERENSI

[1] Munir, Rinaldi, “Diktat Kuliah IF3051 Strategi Algoritma”, Program Studi Teknik Informatika, 2009.

[2] http://www.gamefaqs.com/psp/920817-tales-of-eternia/faqs/41557
Waktu akses : 12/8/2010 12:31 AM

[3] http://en.wikipedia.org/wiki/Breadth-first_search
Waktu akses : 12/8/2010 2:10 AM

[4] http://en.wikipedia.org/wiki/Depth-first_search
Waktu akses : 12/8/2010 2:10AM

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, 9 Desember 2010

ttd

Muharram Huda W. / 13508033