Penerapan Algoritma Program Dinamis dalam Optimasi Urutan Penggunaan *Skill* pada Game RPG

Muntaha Ilmi (13512048)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
muntahailmi@students.itb.ac.id

***Abstrak*—Game RPG adalah salah satu permainan dimana pemainnya akan menggunakan satu karakter, atau lebih, yang akan dipakai untuk mengalahkan musuhnya dalam pertarungan. Pada RPG yang tidak menggunaka *turn-based system*, *timing* penggunaan *skill* dan urutannya akan sangat menentukan hasil dari pertarungan tersebut. Optimasi urutan penggunaan skill ini sangat sulit untuk diperkirakan tanpa penghitungan yang lama ataupun pengalaman menggunakan karakter tersebut yang memadai. Makalah ini akan memberikan salah satu cara penyelesaian permasalahan optimasi urutan penggunaan *skill* ini dengan menggunakan algoritma program dinamis.**

***Kata Kunci*—program dinamis, optimasi *skill*, RPG, *video game*.**

I. PENDAHULUAN

RPG adalah salah satu permainan dimana pemainnya mengendalikan satu karakter atau lebih dan menjalankan suatu cerita yang biasanya sudah ditentukan. Pada proses perjalanan cerita ini, karakter tersebut mungkin akan bertarung melawan berbagai macam musuh. Dalam melawan musuh ini, karakter yang dikendalikan pemain akan dibantu dengan *skill*, yang didapat karakter tersebut seiring dengan perkembangan cerita dan kemampuan karakter tersebut. *Skill* inilah yang sangat menentukan hasil pertarungan tersebut, apakah akan menang atau kalah.

Gambar 1.1 Contoh *skill-skill* yang dapat dimiliki oleh suatu karakter
Sumber : http://rfonline.gamescampus.com/

Skill adalah kemampuan karakter yang dapat digunakan pada saat melawan musuh. *Skill* bentuknya bervariasi, mulai dari sekedar menendang musuh, hingga menggunakan misil untuk meledakkan lawan. *Skill* secara umum dibagi menjadi tiga kategori berdasarkan efek yang dihasilkan, yaitu *Recovery Skill*, *Ailment Skill*, dan *Damaging Skill*. *Recovery Skill* adalah *skill* yang akan menyembuhkan dan mengembalikan kemampuan target dari *skill* ini untuk melanjutkan pertarungan. Contoh sederhananya adalah *skill* yang akan menyembuhkan luka targetnya. *Ailment Skill* adalah *skill* yang akan mengubah kekuatan target dan kemampuan target tersebut dalam melanjutkan pertarungan. Contohnya adalah *skill* yang memberikan racun ke target atau membutakan mata targetnya. *Damaging skill* adalah *skill* yang memberikan kerusakan, atau melukai target, dan secara langsung mengurangi kemampuan target tersebut dalam melanjutkan pertarungan.

Gambar 1.2 Karakter menggunakan *Damaging Skill*
Sumber : http://www.ragnarokbattle.com/

Penggunaan *skill* ini juga memiliki batasan tertentu. Setiap penggunaan *skill* biasanya memakai energi dari karakter tersebut yang jumlahnya terbatas. Setiap karakter memiliki laju pemulihan energi yang bervariasi, namun biasanya tidak akan cukup untuk menggunakan banyak skill berturut-turut.

Pada game RPG yang memakai sistem pertarungan *real-time action*, dimana waktu penggunaan *skill* tidak dibatasi oleh giliran seperti pada sistem *turn-based*, *skill* juga diberikan batasan *cooldown*, yaitu waktu jeda hingga skill tersebut dapat digunakan kembali. Karena kedua batasan itulah, *timing* penggunaan *skill* akan sangat menentukan hasil pertarungan tersebut. Jika *timing*-nya meleset, energi yang terpakai dapat bertambah dan kerusakan yang dapat diberikan dapat berkurang sangat

jauh. Belum lagi masalah penyesuaian *timing* dengan *timing* bertahan dan *skill* lawan yang dapat mengganggu penggunaan *skill* karakter.

Gambar 1.3 *Skill* yang sedang *Cooldown*
Sumber : http://www.aq.com/play-now/

Makalah ini hanya akan membahas penyelesaian masalah optimasi *skill* berkategori *Damaging Skill*, karena penggunaan *skill* berkategori lain akan sangat bergantung pada situasi pertarungan dan lawan yang dihadapi saat itu. Dengan alasan yang sama juga, *timing* bertahan dan penggunaan *skill* lawan akan diabaikan.

Optimasi yang dimaksud disini adalah pemaksimalan kerusakan yang dihasilkan ke musuh, yang dibatasi dengan energi yang dimiliki karakter dan dalam rentang waktu yang ditentukan. Optimasi ini dimaksudkan untuk memaksimalkan penggunaan energi yang dipulihkan dalam rentang waktu tertentu. Urutan penggunaan *skill* yang optimal ini kemudian dapat digunakan berulang-ulang untuk rentang waktu selanjutnya.

.

II. DASAR TEORI

2.1 Program Dinamis

Program dinamis adalah sebuah metode pemecahan masalah dengan membagi masalah-masalah tersebut menjadi tahap-tahap yang setiap tahapnya akan menghasilkan rangkaian pemilihan keputusan. Rangkaian keputusan ini harus dibangun dari pemecahan masalah pada tahap sebelumnya. Setiap tahap ini ditentukan dan dibedakan dengan sejumlah status yang berhubungan dengan tahap tersebut[1].

Syarat pertama yang harus dipenuhi agar suatu permasalah dapat diselesaikan menggunakan algoritma progam dinamis adalah permasalahan tersebut harus bisa dipandang sebagai kumpulan dari keputusan-keputusan. Sejumlah keputusan tersebut juga harus berhingga agar program dinamisnya dapat selesai, tidak terus menerus mencari kemungkinan selanjutnya.

Syarat berikutnya adalah permasalahan tersebut dapat dibagi berdasar prinsip optimasi, dimana masalah tersebut dapat dipecah menjadi sub-masalah yang lebih sederhana. Solusi optimal dari sub-masalah itulah yang kemudian membangun solusi optimal dari masalah yang sebenarnya.

Permasalahan yang ada pada tiap tahap mungkin ada lebih dari satu, karena itu solusi yang dipertimbangkan untuk membangun solusi optimal juga ada lebih dari satu, sehingga program dinamis akan mempertimbangkan beberapa rangkaian keputusan sekaligus pada tiap langkahnya, bahkan sesungguhnya program dinamis akan mempertimbangkan seluruh kemungkinan yang ada pada akhirnya, yang membuatnya berbeda dengan algoritma greedy yang hanya mempertimbangkan satu rangkaian keputusan. Hasilnya, tidak seperti greedy yang hanya menghasilkan solusi yang pseudo-optimal, program dinamis dipastikan akan menghasilkan solusi yang optimal.

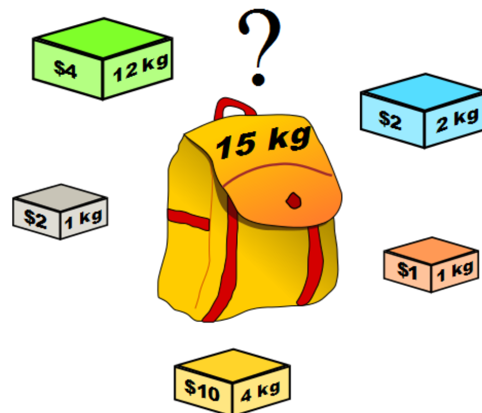
Keuntungan lain dari algoritma program dinamis adalah adanya *overlapping* pada sub-masalah[2] sehingga dengan karakteristik program dinamis yang menyimpan solusi dari sub-masalah yang sudah pernah dikalkulasi, waktu kalkulasi yang diperlukan dapat berkurang cukup jauh. Berbeda dengan algoritma *divide and conquer* yang tidak memiliki *overlapping* sub-masalah, sehingga tidak mendapat penambahan performa walaupun melakukan penyimpanan solusi submasalahnya.

Terdapat beberapa karakteristik utama dari algoritma program dinamis, yaitu sebagai berikut.

1. Program dinamis membagi permasalahan menjadi beberapa tahap.
2. Tiap tahap program dinamis terdiri dari sejumlah status yang juga membedakannya. Status ini dapat dianggap sebagai kemungkinan masukan pada tahap tersebut.
3. Hasil dari keputusan tiap tahap didapat dari hasil transformasi suatu status yang bersangkutan pada tahap itu ke status berikutnya di tahap selanjutnya.
4. Keputusan pada tiap tahap bersifat independen dari tahap-tahap sebelumnya.
5. Hanya solusi optimal pada tahap sebelumnyalah yang akan membentuk solusi optimal pada tahap selanjutnya, tidak ada solusi optimal pada suatu tahap yang dibentuk dari solusi non-optimal (prinsip optimalitas).

2.2 Knapsack Problem

Knapsack problem adalah salah satu masalah optimasi kombinatorial, dimana diberikan sebuah set barang, masing-masing memiliki massa dan nilai, tentukan jumlah barang yang harus diambil untuk setiap barang sehingga total nilai yang diambil sebesar mungkin, namun total berat yang diambil tidak melebihi batasan yang ditentukan. Nama *Knapsack* diambil dari permasalahan memasukkan barang-barang ke suatu karung dengan kapasitas yang ditentukan sehingga nilai barang yang dimasukkan setinggi mungkin.



Gambar 2.2.1 Contoh Ilustrasi *Knapsack problem*
Sumber : http://en.wikipedia.org/wiki/File:Knapsack.svg

Knapsack problem bersifat *NP-Complete*, dan *NP-Hard*. Dalam kata lain, pada sisi pemilihan keputusan maupun optimasi, belum ditemukan algoritma polinomial yang dapat menyelesaikannya. Pada saat ini, Algoritma program dinamis hanya dapat menyelesaikannya dalam waktu pseudo-polinomial.

2.2.1 Unbounded Knapsack

Unbounded knapsack problem adalah permasalahan *knapsack* dimana setiap barangnya bisa diambil sebanyak mungkin, jumlahnya tak terbatas, dan beratnya tidak ada yang negatif. Algoritma program dinamis menyelesaikan masalah ini dalam $O(nW)$, dengan n adalah banyak barang yang ada dan W adalah berat batas total barang yang boleh diambil, sehingga menjadikan algoritma ini algoritma pseudo-polinomial.

Penyelesaiannya adalah sebagai berikut.

Definisikan $f(w)$ adalah fungsi yang menghasilkan nilai tertinggi yang bisa didapat dengan total berat kurang dari atau sama dengan w, dengan $w \leq W$.

$f(w)$ terdefinisi sebagai berikut.

$f(0) = 0$ (tidak mengambil barang apapun)

$f(w) = \max_{w_i \leq w}\{v_i + f(w - w_i)\}$

Hasil yang dicari akan ada pada $f(W)$.

2.2.2 0/1 Knapsack

0/1 Knapsack adalah bentuk lain knapsack dimana setiap barang hanya tersedia satu buah, sehingga hanya hanya dapat diambil satu atau tidak diambil (0/1). Algoritma program dinamis untuk masalah ini juga memakan waktu pseudo-polinomial, tepatnya $O(nW)$.

Penyelesaiannya adalah sebagai berikut.

Definisikan $f(i, w)$ adalah fungsi yang menghasilkan nilai tertinggi yang bisa dihasikan dengan total berat barang kurang dari atau sama dengan w dan hanya menggunakan sampai barang ke-i. v_i adalah nilai barang ke-i dan w_i adalah berat barang ke-i.

$f(i, w)$ terdefinisi sebagai berikut.

$f(0, w) = 0$

$f(i, w) = f(i-1, w), \quad w_i > w$

$f(i, w) = \max(f(i-1, w), v_i + f(i-1, w - w_i)), \quad w_i \leq w$

Hasil yang dicari akan ada pada $f(n, W)$.

Memori yang dipakai sekilas terlihat sebesar $O(nW)$, namun jika menggunakan optimasi flying table, dan pengisian dimulai dari $f(W)$ ke $f(0)$, maka memori yang digunakan dapat dikurangi menjadi hanya $O(W)$.

III. Deskripsi Masalah

Diberikan jumlah *skill* yang ada, energi karakter yang boleh dipakai, dan waktu batas untuk penggunaan *skill*. Untuk masing-masing *skill*, diberikan jumlah kerusakan yang akan diberikan, waktu *cooldown* skill, dan energi yang diperlukan untuk memakai skill tersebut.

Dari data tersebut, hasilkan total kerusakan tertinggi yang dapat dicapai serta urutan penggunaan skill untuk mencapai angka tersebut.

Untuk menyederhanakan soal, anggap setiap *skill* akan langsung memberikan efek segera setelah digunakan dan karakter dapat menggunakan lebih dari satu *skill* dalam satu waktu, jika *skill-skill* tersebut sedang tidak *cooldown*. Tentu saja, tidak ada skill yang memiliki *cooldown* 0. Tidak ada juga skill yang memerlukan energi 0.

IV. Implementasi Program Dinamis

Berikut adalah daftar simbol yang akan digunakan dan maknanya.

$N =$ jumlah skill yang ada

$E =$ energi yang boleh digunakan

$T =$ waktu batas yang ditentukan

$d_i =$ kerusakan yang diberikan oleh *skill* ke-i

$t_i =$ waktu *cooldown skill* ke-i

$e_i =$ energi untuk menggunakan *skill* ke-i

Pertama-tama, jika diperhatikan, sebenarnya persoalan ini mirip dengan persoalan *knapsack*, namun batasan yang diberikan ada dua, energi karakter dan waktu. Analisis berikutnya, karena *skill* dapat digunakan bersama-sama dalam satu waktu, maka penggunaan *skill* tidak akan mempengaruhi bisa/tidaknya menggunakan *skill* lain, sehingga penghitungan waktu *cooldown skill*, dan, sebagai penurunannya, penghitungan jumlah penggunaan maksimal untuk tiap *skill* dapat dihitung paralel, tidak memengaruhi dan dipengaruhi satu sama lain.

Berdasar kedua analisis tersebut, masalah tersebut dapat disederhanakan menjadi seperti berikut.

Diberikan set barang (*skill*), dimana setiap barang memiliki massa (energi yang diperlukan), nilai (kerusakan yang diberikan), dan jumlah yang dapat diambil (T/t_i), tentukan jumlah barang yang harus diambil untuk masing-masing barang sehingga total nilai yang didapat (total kerusakan yang diberikan) setinggi mungkin, namun total berat yang diambil tidak lebih dari berat yang ditentukan (energi yang boleh dipakai).

Masalah ini menjadi sangat mirip dengan *Unbounded knapsack problem*, hanya saja setiap barang diberikan jumlah maksimal yang dapat diambil. *0/1 Knapsack problem* adalah bentuk lain masalah ini dengan jumlah maksimal barang yang dapat diambil adalah 1 untuk semua barang.

Karena masalah ini dapat di-reduce menjadi *Knapsack problem*, yang merupakan *NP-Complete* dan *NP-Hard*, dapat dipastikan bahwa algoritma solusi dari masalah ini tidak akan mencapai waktu polinomial.

Penyelesaian masalah ini merupakan sedikit modifikasi dari masalah *0/1 Knapsack problem*.

Pada *0/1 Knapsack problem*, algoritma penyelesaian dengan menggunakan program dinamis adalah sebagai berikut.

Bentuk rekurensnya.

$f_0(w) = 0$

$f_i(w) = f_{i-1}(w), \quad w_i > w$

$f_i(w) = \max(f_{i-1}(w), v_i + f_{i-1}(w - w_i)),$
$w_i \leq w$

Hasil yang dicari akan ada pada $f_n(W)$.

Algoritma itu akan dimodifikasi, dan diubah simbolnya, sehingga dari yang tadinya hanya membandingkan apakah suatu barang tidak diambil atau diambil 1, menjadi dibandingkan antara tidak diambil dengan maksimal dari menggunakan x *skill* dengan $e_i * x \leq e \,\&\, t_i * x \leq T$.

Bentuk rekurensnya sekarang.

$f_0(e) = 0$

$f_i(e) = f_{i-1}(e),$
$e_i > e$

$f_i(e) = \max(f_{i-1}(e),$
$\max_{(e_i*x\leq e \,\&\, t_i*x\leq T)}\{d_i * x + f_{i-1}(e - e_i * x)\}),$
$e_i \leq e$

$i = 1,2,..,N$

$x = 1,2,..$

Hasil yang dicari akan ada pada $f_N(E)$.

Definisi dari $f_i(e)$ adalah fungsi yang menghasilkan total kerusakan tertinggi yang bisa dihasikan dengan total energi yang dipakai kurang dari atau sama dengan e, dalam waktu kurang dari T dan hanya menggunakan sampai barang ke-i. d_i adalah nilai kerusakan yang dihasilkan *skill* ke-i dan w_i adalah energi yang diperlukan oleh *skill* ke-i.

Setelah didapat tabel hasil dari program dinamis tersebut, maka untuk mendapatkan jumlah penggunaan tiap skill cukup dilakukan backtracking pemilihan keputusan program dinamisnya seperti berikut.

1. Inisialisasi tabel sum berukuran N dengan 0.
2. Inisialisasi nilai e dengan E.
3. Anggap tabel memorisasi program dinamis bernama $memo$.
4. Mulai dari baris ke-N.
5. Anggap baris sekarang baris ke-i.
6. Pilih nilai x dimana nilai dari $d_i * x + memo[i-1][e - e_i * x]$ paling besar, dengan $e_i * x \leq e \,\&\, t_i * x \leq T$.
7. x adalah banyak penggunaan skill ke-i yang optimal. Isi tabel $sum[i]$ dengan x.
8. Ubah nilai e menjadi $e - e_i * x$
9. Pindah ke baris $i - 1$.
10. Ulangin langkah 5-9 hingga mencapai baris ke-0.
11. Isi tabel sum sekarang adalah jumlah penggunaan untuk tiap skill.

Sekarang masalah mengubah dari tabel jumlah penggunaan masing-masing skill menjadi urutan penggunaan skill. Hal itu dapat dilakukan dengan mudah menggunakan pseudo-code berikut.

```
procedure ubah(sum : array[1..n] of integer)
var
n, jum, numarr, kecil: integer;
i, j: integer;
{ pada deklarasi array }
{ n_skill maksudnya adalah jumlah skill yang
ada }
{ byk maksudnya adalah jumlah penggunaan skill }
{ byk paling buruk bernilai jumlah dari tabel
sum }
delay : array[1..n_skill] of integer;
waktu : array[0..byk] of integer;
arr : array[0..byk][0..n_skill] of integer;

begin
n := length of sum;
{ hitung jumlah skill yang belum digunakan }
jum := 0;
for i := 1 to n do begin
        jum := jum + sum[i];
        { inisialisasi waktu untuk penggunaan
skill }
        delay[i] := 0;
end
{ inisialisasi array penyimpan urutan skill yang
digunakan }
for i := 0 to jum do begin
        waktu[i] := 0;
        arr[i][0] := 0;
end
numarr := 0;
while (jum > 0) do begin
        kecil := -1;
        { cari skill yang waktu untuk penggunaan
berikutnya tercepat }
        for i:=1 to n do begin
                { cek skill tersebut masih bisa
digunakan/tidak }
                if (sum[i] <> 0 ) then begin
                        if (kecil = -1) then begin
                                kecil = delay[i];
                        end     else if (waktu[i]
< kecil) then begin
                                kecil = delay[i];
                        end
                end
        end
        if (kecil = -1) then begin
                { tidak ada skill yang bisa
dipakai lagi }
                jum := 0;
        end     else begin
                { set waktu penggunaan skill }
                waktu[numarr] := kecil;
                { mempersiapkan array untuk
dimasukkan elemen }
                arr[numarr][0] := 0;
                nummarr := numarr + 1;
                { mencari skill yang bisa
digunakan pada waktu yang sama }
                for i:=1 to n do begin
                        if (waktu[i] = kecil) then
begin
                                { menambah jumlah
elemen di array }
                                arr[numarr][0] :=
arr[numarr][0] + 1;
                                { menambah skill
ke array }

        arr[numarr][arr[numarr][0]] := i;
                                { menambah waktu
penggunaan skill selanjutnya }
                                delay[i] :=
delay[i] + t[i];
                                { mengurangi
jumlah penggunaan skill }
                                sum[i] := sum[i] -
1;
                                { mengurangi
jumlah skill yang belum dimasukkan array }
                                jum := jum - 1;
                        end
```

```
                end
            end
end
{ outputkan urutan penggunaan skill }
for i:=0 to (numarr-1) do begin
            output('pada detik ke-'+waktu[i]);
            output('skill yang harus digunakan:');
            for j:=1 to arr[i][0] do begin
                output('skill ke-'+arr[i][j]);
            end
end
end.
```

Inti dari algoritma di atas adalah sebagai berikut.

1. Inisialisasi tabel $delay$ untuk menampung waktu suatu skill dapat digunakan lagi.
2. Inisialisasi tabel $waktu$ untuk menampung waktu penggunaan skill
3. Inisialisasi tabel arr yang berisi set angka yang dikosongkan dahulu. arr digunakan untuk menampung skill apa saja yang akan digunakan pada waktu tertentu.
4. Pada tabel $delay$, cari nilai tabel $delay$ terkecil yang nilai pada tabel sum yang bersangkutan masih lebih besar dari 0. Jika tidak ada, loncat ke langkah 11.
5. Anggap nilai tersebut nilai x.
6. Tambah nilai x ke tabel $waktu$.
7. Buat set angka yang berisi semua skill yang memiliki delay sama dengan x. Anggap set tersebut set S.
8. Tambahkan isi tabel $delay$ yang skill bersangkutannya ada di set S dengan t_i.
9. Tambahkan set S ke tabel arr.
10. Kembali ulangi ke langkah 4.
11. Tabel $waktu$ dan arr berisi urutan penggunaan skill. $waktu[i]$ berisi detik menggunakan skill dan $arr[i]$ berisi set skill yang akan digunakan pada detik itu.

Memori yang digunakan untuk algoritma ini adalah $O(NE)$.

Kompleksitas pengubah dari jumlah penggunaan skill ke urutan penggunaan skill adalah $O(N * \min(T,E))$.

Kompleksitas algoritma *backtracking* untuk mencari jumlah penggunaan skill adalah $O(N * \min(T,E))$.

Kompleksitas algoritma program dinamisnya sendiri adalah $O(NE * \min(T,E))$.

Maka, kompleksitas total dari algoritma ini adalah

$$O(NE * \min(T,E) + 2N * \min(T,E)) \\ = O(NE * \min(T,E))$$

Algoritma ini masih memenuhi kriteria *NP-Hard*, karena kompleksitasnya yang pseudo-polinomial.

V. EVALUASI IMPLEMENTASI ALGORITMA

Pada sebagian besar game RPG, jumlah skill yang ada tidak terlalu banyak, karena semakin banyak skill yang ada, maka pemainnya juga akan bingung untuk menggunakannya. Batas atasnya dapat dianggap adalah 30 skill.

Untuk waktu batasan, semakin lama batasan yang diberikan, maka pemain akan lebih sulit menghafal urutan penggunaan skill tersebut. Selain itu, tidak banyak juga musuh yang akan masih hidup setelah sekian lama pertarungan. Karena itu, batas atas waktu ini dapat dianggap sekitar 5 menit, atau 300 detik.

Untuk batasan energi, laju pemulihan energi ini biasanya tidak akan terlalu tinggi, karena dapat merusak keseimbangan game. Dalam waktu 5 menit, batas atas waktu, kira-kira dapat dianggap sekitar 10,000 energi.

Untuk itu, perkiraan waktu terlama algoritma ini berjalan adalah 30*300*10,000 = 90,000,000 operasi. Sebuah komputer biasa dapat menjalankan sekitar 100,000,000 operasi dalam satu detik, sehingga algoritma ini hanya perlu kurang dari satu detik untuk menghitung urutan penggunaan skill unuk 5 menit ke depan. Algoritma ini sudah dapat dikatakan cukup baik.

Untuk masalah penggunaan memori, program ini akan memakan kira-kira 30*10,000 = 300,000 ukuran integer, yang berarti 1,200,000 byte, atau kira-kira 1,2 MB, yang cukup kecil.

VI. CONTOH LANGKAH ALGORITMA

Berikut adalah contoh pengisian tabel program dinamis. Pada contoh, terdapat tiga skill, dengan batasan waktu 12 detik dan batasan energi 100.

Keterangan masing-masing skill ada pada tabel berikut.

Tabel 6.1 Data Input Contoh

N=3; E=100; T=12

Skill ke-*i*	d_i	e_i	t_i
1	100	10	5
2	500	75	2
3	75	30	2

Berikut tabel program dinamis pada semua tahap.

Tabel 6.2 Tabel Program Dinamis Tahap 1

			Solusi optimum
y	$f_0(y)$	$f_1(y)$	(x_1*,x_2*,x_3*)
0 ↓ 9	0 ↓ 0	0 ↓ 0	(0,0,0) ↓ (0,0,0)
10 ↓ 19	0 ↓ 0	100 ↓ 100	(1,0,0) ↓ (1,0,0)
20 ↓ 100	0 ↓ 0	200 ↓ 200	(2,0,0) ↓ (2,0,0)

Tabel 6.3 Tabel Program Dinamis Tahap 2

			Solusi optimum
y	$f_1(y)$	$f_2(y)$	(x_1*,x_2*,x_3*)
0	0	0	(0,0,0)

↓	↓	↓	↓
9	0	0	(0,0,0)
10	100	100	(1,0,0)
↓	↓	↓	↓
19	100	100	(1,0,0)
20	200	200	(2,0,0)
↓	↓	↓	↓
74	200	200	(2,0,0)
75	200	500	(0,0,1)
↓	↓	↓	↓
84	200	500	(0,0,1)
85	200	600	(1,0,1)
↓	↓	↓	↓
94	200	600	(1,0,1)
95	200	700	(2,0,1)
↓	↓	↓	↓
100	200	700	(2,0,1)

Tabel 6.4 Tabel Program Dinamis Tahap 3

		Solusi optimum	
y	$f_2(y)$	$f_3(y)$	(x_1^*,x_2^*,x_3^*)
0	0	0	(0,0,0)
↓	↓	↓	↓
9	0	0	(0,0,0)
10	100	100	(1,0,0)
↓	↓	↓	↓
19	100	100	(1,0,0)
20	200	200	(2,0,0)
↓	↓	↓	↓
49	200	200	(2,0,0)
50	200	240	(2,0,1)
↓	↓	↓	↓
74	200	240	(2,0,1)
75	500	500	(0,0,1)
↓	↓	↓	↓
84	500	500	(0,0,1)
85	600	600	(1,0,1)
↓	↓	↓	↓
94	600	600	(1,0,1)
95	700	700	(2,0,1)
↓	↓	↓	↓
100	700	700	(2,0,1)

.

VII. KESIMPULAN

Algoritma kalkulasi urutan penggunaan skill yang optimal setelah disederhanakan ternyata merupakan salah satu turunan Knapsack problem, yang merupakan masaalh yang masih NP-Complete. Algoritma ini masih pseudo-polinomial, namun berdasarkan dengan kemungkinan input yang akan diterima, algoritma ini sudah cukup efisien untuk menjalankan tugasnya.

REFERENCES

[1] Munir, Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika, STEI, ITB, 2009.

[2] Dasgupta, S., C.H. Papadimitriou, dan U.V. Vazirani, "Algorithms", hlm. 173. Tersedia juga di link di bawah. http://www.cs.berkeley.edu/~vazirani/algorithms.html

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

Muntaha Ilmi (13512048)