Penyelesaian Berbagai Permasalahan Algoritma dengan Kombinasi Algoritma *Brute Force* dan *Greedy*

Anggriawan Sugianto [1], David Susanto [2], Zakka Fauzan Muhammad [3]

Laboratorium Ilmu dan Rekayasa Komputasi
Program Studi Teknik Informatika, Institut Teknologi Bandung (ITB)
Kampus ITB Jl Ganesha No. 10 Bandung
Email: if14018@students.if.itb.ac.id [1], if14019@students.if.itb.ac.id [2],
if14020@students.if.itb.ac.id [3]

Abstrak

Pada masa sekarang ini, terdapat banyak sekali cara-cara penyelesaian persoalan yang dapat dilakukan oleh komputer. Mulai dari cara penyelesaian persoalan dengan cara yang mengutamakan kemudahan pemikiran manusia, sederhana, pasti menemukan solusi tetapi memiliki kompleksitas algoritma yang sangat besar seperti penyelesaian dengan *brute force* dan *exhaustive search*, penyelesaian dengan mengutamakan kecepatan menemukan satu solusi, yang dianggap optimal, sangat tinggi, seperti algoritma *greedy*, ataupun algoritma-algoritma lain yang mengutamakan kemangkusan algoritma tetapi memiliki tingkat kerumitan cukup tinggi, seperti *BFS*, *DFS*, *Backtracking*, *Branch and Bound* dan *dynamic programming*. Secara ideal, jika ditemukan penyelesaian suatu permasalahan yang cepat, mudah, kemangkusannya tinggi, dan pasti menemukan solusi, pasti algoritma tersebut akan selalu dipakai, akan tetapi sayangnya sampai saat ini hal tersebut belum dapat terjadi. Pada beberapa kasus, kadang diinginkan algoritma yang mudah (seperti pada *brute force* ataupun *greedy*), cepat (seperti pada algoritma *greedy*), dan sedekat mungkin dengan solusi optimum global (seperti pada *brute force*). Pada makalah ini akan dibahas penyelesaian masalah dengan menggunakan kombinasi dari algoritma *brute force* dan *freedy*, memiliki ketepatan solusi yang cukup optimal dan kecepatan yang cukup tinggi.

***Kata kunci**: brute force, greedy, penyelesaian masalah, kombinasi algoritma*

1. Pendahuluan

Penggunaan algoritma yang tepat pada permasalahan yang tepat (atau pada sebuah program) akan sangat menguntungkan bagi pengguna program tersebut. Dengan penggunaan algoritma yang tepat, waktu yang dibutuhkan untuk penyelesaian suatu permasalahan akan semakin tinggi. Sayangnya, sampai saat ini suatu penyelesaian yang paling sederhana, yaitu algoritma *brute force*, memiliki kemangkusan yang sangat rendah. Sayangnya lagi, algoritma dengan kemangkusan paling tinggi, yaitu algoritma *greedy*, memiliki ketepatan penyelesaian yang tidak 100% benar. Penggabungan dari kedua algoritma tadi sangat mungkin dapat menghasilkan sebuah algoritma baru yang lebih mangkus dari *brute Force* dan lebih optimum dari *greedy*.

2. Perbandingan *Brute Force* dan *Greedy*

2.1. Karakteristik *Brute Force dan Greedy*

Secara umum, algoritma *Brute Force* dapat dilihat pada "penampilan luar"-nya (pada *source code* dan kecepatan program) yang pendek, mudah dimengerti dan membutuhkan cukup banyak waktu (terutama pada program yang besar). Atau dengan bahasa lain, *brute force* adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan [1].

Sedangkan algoritma *greedy* dapat dilihat terutama pada kecepatan program yang sangat tinggi, akan tetapi kadang-kadang hasilnya agak mengecewakan karena tidak sesuai dengan solusi optimum yang diharapkan. Atau dengan kata lain, algoritma *greedy* membentuk solusi langkah per langkah (*step by step*), dan prinsip *greedy* adalah "*take what you can get now!"* [1].

2.2. Penyelesaian Persoalan dengan *Brute Force* dan *Greedy*

2.2.1. Persoalan Penukaran Uang

Persoalan: Misalkan kita ingin menukarkan uang senilai *A* dengan sekumpulan uang koin dari berbagai satuan (dengan asumsi tersedia banyak koin

untuk setiap satuan, sehingga tidak mungkin terjadi kekurangan koin). Berapa jumlah minimum koin yang diperlukan tersebut?

Contoh: koin-koin yang tersedia bernilai 12, 7, 4, dan 2 (dalam jumlah yang banyak untuk tiap satuan). Bagaimana cara menukarkan uang sebanyak 18 dengan koin-koin tersebut?

Dengan sedikit pengetahuan, dapat diketahui bahwa jumlah koin yang diperlukan tidak akan lebih jumlah uang dibagi dengan pecahan terkecil koin dan tidak akan kurang dari jumlah uang dibagi dengan pecahan terbesar koin. Jadi bisa dipastikan bahwa jumlah koin yang diperlukan tidak akan lebih dari 9. Untuk memudahkan, dipergunakan sebuah koin maya bernilai 0, sehingga jika hanya dipilih 2 koin (anggap 12 dan 4) sama seperti dipilih S = {0, 0, 0, 0, 0, 0, 0, 4, 12} dengan S adalah himpunan koin-koin yang diambil.

Dengan *brute force*, dapat dicoba semua kemungkinan pilihan, mulai dari S = {0, 0, 0, …, 0}, S = {0, 0, …, 0, 2}, sampai S = {12, 12, 12, …, 12}, dengan rumus kombinasi berulang, berarti harus dicoba sebanyak 5^9 = 1.953.125 cara, untuk masing-masing cara, harus dihitung jumlah tiap-tiap elemen himpunan, jika sama dengan 18 diambil, jika tidak dibuang. Berarti jumlah perhitungan yang harus dilakukan adalah sebanyak 2 x 5^9 ≈ 4.000.000 cara. Pada kondisi akhir, akan diperoleh semua kemungkinan pilihan koin yang bisa dijadikan solusi, yaitu

S_1 = (0, 0, 0, 0, 0, 0, 2, 4, 12)
S_2 = (0, 0, 0, 0, 0, 0, 4, 7, 7)
S_3 = (0, 0, 0, 0, 0, 2, 2, 2, 12)
S_4 = (0, 0, 0, 0, 0, 2, 2, 7, 7)
…
S_n = (2, 2, 2, 2, 2, 2, 2, 2, 2)

Kemudian dipilihlah solusi yang paling sedikit menggunakan koin nyata, atau dengan kata lain dipilih solusi yang paling banyak menggunakan "koin 0". Sehingga diperoleh dua solusi yaitu: (2, 4, 12) dan (4, 7, 7).

Cara kedua, dengan algoritma *greedy*, akan dipilih satu koin terbesar dan jumlahnya dengan koin sebelumnya yang telah diambil tidak lebih dari 18 (pada kondisi awal, jumlah koin sebelumnya adalah 0).

Langkah-langkah algoritma *greedy*:
1. Ambil koin 12, jumlah = 12
2. Ambil koin 4, jumlah = 16
3. Ambil koin 2, jumlah = 18

Ditemukan solusi, stop.

Sehingga diperoleh satu solusi, yaitu (12, 4, 2).

Pada kasus umum, jika diketahui bahwa koin-koin yang tersedia adalah $k_1 < k_2 < ... < k_n$ dan uang yang akan ditukarkan adalah u, serta $m - 1 < u / k_1 \leq m$, m adalah bilangan bulat. Berarti banyak koin tidak akan lebih dari m. Sehingga komplesitas algoritma untuk penyelesaian secara *brute force* adalah $O((n+1)^m) \approx O(n^m)$.

Sedangkan untuk algoritma *greedy*, pemilihan bilangan terbesar yang memenuhi tidak akan memakan waktu lebih dari n, serta banyaknya koin yang dipilih tidak akan lebih dari m, berarti kompleksitasnya adalah $O(nm)$.

Dari 2 perhitungan diatas, dapat disimpulkan bahwa penyelesaian secara *greedy* jauh lebih mangkus daripada penyelesaian secara *brute force*, meski dalam beberapa kasus lainnya yang tidak dituliskan disini, ada kejadian dimana solusi dari algoritma *brute force* lebih memuaskan daripada solusi dari algoritma *greedy* (dan tidak mungkin terjadi sebaliknya), atau bahkan solusi ditemukan hanya dengan algoritma *brute force* saja, tidak dengan *greedy*, misalnya koin yang tersedia adalah 12 dan 7 saja sedangkan uang yang ingin ditukarkan adalah 14.

2.2.2. Persoalan Integer Knapsack

Persoalan: Misalkan kita diberikan beberapa buah barang dengan keuntungan serta beratnya masing-masing, akan tetapi container yang kita miliki hanya dapat memuat sejumlah berat tertentu. Pilih barang-barang yang akan dibawa sehingga keuntungan yang diperoleh maksimum.

Contoh: Terdapat 6 benda (labeli 1 sampai 6), dengan w_i adalah berat benda ke-i dan p_i adalah keuntungan benda ke-i.

$w_1 = 100$	$p_1 = 40$
$w_2 = 50$	$p_2 = 35$
$w_3 = 45$	$p_3 = 18$
$w_4 = 20$	$p_4 = 4$
$w_5 = 10$	$p_5 = 10$
$w_6 = 5$	$p_6 = 2$

Kapasitas knapsack $K = 100$

Tuliskan solusi sebagai $X = (x_1, x_2, ..., x_6)$ dengan x_i = 0 jika benda ke-i dibawa atau x_i = 1 jika benda ke-i tidak dibawa.

Penyelesaian dengan *brute force*:
Coba semua kemungkinan X, mulai X = (0, 0, 0, 0, 0, 0), sampai dengan X = (1, 1, 1, 1, 1, 1). Jumlah kemungkinan yang harus dicoba ada 2^6 = 64 cara. Tiap cara harus dihitung berat totalnya, dan dari semua cara yang berat totalnya tidak melebihi

kapasitas knapsack, pilih satu yang keuntungannya paling besar.

Pada akhirnya akan diperoleh solusi optimum X = (0, 1, 1, 0, 0, 1) dengan berat total 100 dan keuntungan total 55 [1].

Dengan *greedy*, bisa dipilih 3 cara sebagai berikut:

1. *Greedy by profit*, pilih benda-benda dengan keuntungan maksimum
2. *Greedy by weight*, pilih benda-benda dengan berat minimum
3. *Greedy by density*, pilih benda-benda dengan keuntungan per berat maksimum

Sehingga bisa dibuat tabelnya sebagai berikut: [1]

Properti objek				*Greedy by*		
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*
1	100	40	0,4	1	0	0
2	50	35	0,7	0	0	1
3	45	18	0,4	0	1	0
4	20	4	0,2	0	1	1
5	10	10	1,0	0	1	1
6	5	2	0,4	0	1	1
Total bobot				100	80	85
Total keuntungan				40	34	51

Sehingga diperolehlah solusi sebagai berikut:

- *greedy by profit*: X = (1, 0, 0, 0, 0, 0), keuntungan total = 40
- *greedy by weight*: X = (0, 0, 1, 1, 1, 1), keuntungan total = 34
- *greedy by density*: X = (0, 1, 0, 1, 1, 1), keuntungan total = 51

Dari sana diperoleh kenyataan, bahwa langkah *greedy* yang mengutamakan kecepatan dan hanya optimum di setiap langkah tidak ada yang menghasilkan keuntungan optimum di langkah akhirnya.

Secara umum, jika banyak barang adalah n, kompleksitas algoritma untuk *brute force* adalah $O(2^n)$.

Sedangkan untuk algoritma *greedy*, kompleksitas pada *greedy by profit* dan *greedy by weight* adalah sama, yaitu $O(n^2)$ (pemilihan k barang yang diambil, dengan maksimal n buah kalang, karena jumlah barang yang diambil tidak mungkin lebih dari n). Pada *greedy by density*, kompleksitasnya berbeda sedikit karena harus ada perhitungan p_i/w_i untuk setiap i barang-barang yang ada. Secara umum tidak terdapat perbedaan yang terlalu jauh karena $T(n) = n^2 + n$, sehingga kompleksitasnya juga $O(n^2)$.

3. Algoritma *Brudy* sebagai Penggabungan Algoritma *Brute Force* dan *Greedy*

3.1. Penggabungan *Brute Force* dan *Greedy*

Ide awal dari penggabungan kedua algoritma tersebut adalah membentuk sebuah algoritma baru yang serba sedang. Misalkan diperlukan algoritma dengan kecepatan yang cukup tinggi (lebih tinggi dari kecepatan *brute force*), kerumitan yang cukup sederhana, serta ketepatan penemuan solusi yang cukup baik (lebih baik dari *greedy*).

Algoritma ini (untuk selanjutnya, dinamakan sebagai algoritma *brudy* (*brute force-greedy*)) lebih berkaca pada algoritma *greedy*. Bedanya, algoritma *greedy* mencari **optimum lokal pada tiap langkahnya**, sedangkan algoritma *brudy* mencari **optimum lokal pada tiap b-langkah** (untuk selanjutnya b dinamakan sebagai nilai batas, dengan catatan $b > 1$ dan $b <$ *jumlah_tahap*), sehingga bisa juga dikatakan algoritma *brudy* ini sama dengan algoritma *b-greedy*.

Jadi, pada suatu keadaan, misalkan pada suatu permasalahan yang pengerjaannya bertahap-tahap (anggap saja 14 tahap dengan tahap ke-0 adalah kondisi awal), algoritma *brute force* akan mencari semua cara mencapai tahap ke-13 tersebut, algoritma *greedy* akan mencari optimum dari tahap ke-i menuju tahap ke-(i+1), sedangkan algoritma *brudy* akan mencari optimum dari tahap ke-i menuju tahap ke-(i+b) dan b bebas ditentukan oleh pengguna. Misalkan dipilih b adalah 3, berarti akan dicari optimum tahap ke-1 menuju tahap ke-4, setelah itu dicari optimum tahap ke-4 menuju tahap ke-7, dan seterusnya. Begitu diperoleh optimum dari tahap ke-1 menuju ke tahap ke-4, status di tahap ke-4 tersebut itulah yang akan diperluas untuk dicari optimumnya menuju tahap ke-7 dan seterusnya dan disinilah letak kemiripan algoritma *brudy* dengan algoritma *greedy*. Sedangkan untuk mencari optimum dari tahap ke-1 sampai tahap ke-4, akan dicoba semua cara yang mungkin dari tahap ke-1 sampai tahap ke-4, begitu seterusnya, dan disinilah letak kemiripan algoritma *brudy* dengan algoritma *brute force*.

Untuk meningkatkan pemahaman terhadap algoritma *brudy*, dapat dilihat pada bagian contoh pada subbab setelah subbab ini.

Sedikit perbedaan dengan algoritma *greedy*, yang sekaligus berguna menutupi kelemahan algoritma ini, adalah apabila tahap ke-(i+b) melebihi solusi akhir, *decrement* (kurangkan 1) b dan lakukan lagi untuk langkah selanjutnya. Untuk memudahkan, bisa melihat contoh 1 pada subbab **3.2.1**.

3.2. Penyelesaian Persoalan dengan Algoritma *Brudy*

3.2.1 Persoalan Penukaran Uang

Untuk lebih memahami mengenai persoalan penukaran mata uang, bisa dilihat lagi pada subbab **2.2.1** mengenai persoalan ini.

Selanjutnya untuk memudahkan pembandingan algoritma *brudy* dengan algoritma *brute force* dan algoritma *greedy*, lihat contoh yang sama seperti pada contoh pada subbab **2.2.1**.

Contoh: koin-koin yang tersedia bernilai 12, 7, 4, dan 2 (dalam jumlah yang banyak untuk tiap satuan). Bagaimana cara menukarkan uang sebanyak 18 dengan koin-koin tersebut?

Pertama, dipilih nilai batas (penjelasan mengenai nilai batas, baca subbab **3.1**) untuk algoritma *brudy* yang akan dipakai, misalkan dipilih $b = 2$.

Langkah-langkah penyelesaian algoritma:
1. Pilih 2 koin sehingga jumlah dari 2 koin tersebut optimum (prinsip *greedy*) dan tidak lebih dari 18. Cara pemilihan 2 koin dilakukan secara *brute force*.

Ada 4^2 cara pemilihan 2 koin, mulai dari (12,12), (12,7), (12,4), (12,2), (7,7), sampai dengan (2,2). Nilai terbesar yang kurang dari 18 adalah (12,4) dengan jumlah 16. Jadi sisa uang yang belum tertukar adalah 2.

2. Pilih 2 koin sehingga jumlah dari 2 koin tersebut optimum dan tidak lebih dari 2. Cara pemilihan sama dengan No.1, pada akhirnya tidak akan ditemukan solusi yang menyebabkan 2 koin jumlahnya tidak lebih dari 2.

3. Kurangkan b dengan 1, $b = 1$.

4. Pilih 1 koin sehingga jumlah dari 1 koin tersebut optimum dan tidak lebih dari 2. Coba semua kemungkinan, yaitu (12), (7), (4), (2) dan pada akhirnya yang memenuhi adalah (2). Jadi sisa uang yang belum tertukar adalah 0.
Solusi ditemukan. Stop.

Koin yang diambil (12,4,2).

Pada kasus umum, kita pilih nilai n dan m seperti pada subbab **2.2.1**, jika nilai batas kita anggap sebagai b, maka jumlah langkah yang diperlukan adalah:

1. *brute force* untuk mencapai tahap ke-b dari tahap ke-0 dan demikian selanjutnya.
2. *greedy* untuk mencari optimum tiap b langkah.

Kompleksitasnya adalah $O(mn^b/b)$.

Yang jika dipilih $b = m$, kompleksitasnya akan menjadi $O(n^m)$ atau sama dengan kompleksitas pada algoritma *brute force* (karena memang dengan memilih $b = m$ berarti memilih *greedy* dengan mencari optimum dari tahap awal ke tahap akhir, sama saja dengan *brute force*).

Pada kasus lain, jika dipilih $b = 1$, kompleksitasnya menjadi $O(mn)$ dan pemilihan pencarian optimum satu tahap memang sama dengan algoritma *greedy*.

3.2.2. Persoalan Integer Knapsack

Sekali lagi, untuk memudahkan membandingkan algoritma *brudy* dengan algoritma *brute force* ataupun *greedy*, akan digunakan contoh yang sama seperti pada subbab **2.2.2**.

Contoh: Terdapat 6 benda (labeli 1 sampai 6), dengan w_i adalah berat benda ke-i dan p_i adalah keuntungan benda ke-i.

$w_1 = 100$; $p_1 = 40$
$w_2 = 50$; $p_2 = 35$
$w_3 = 45$; $p_3 = 18$
$w_4 = 20$; $p_4 = 4$
$w_5 = 10$; $p_5 = 10$
$w_6 = 5$; $p_6 = 2$

Kapasitas knapsack $K = 100$.

Pertama, pilih nilai batas (penjelasan mengenai nilai batas, baca subbab **3.1**). Misalkan saja nilai batasnya dipilih $b = 2$.

Langkah-langkah penyelesaian algoritma:

1. Karena dipilih $b = 2$, berikutnya tentukan semua himpunan bagian dari 6 benda tersebut ({1, 2, 3, 4, 5, 6}) yang banyak anggotanya adalah 2.
2. Cari profit, berat, dan profit/berat untuk tiap 2 benda tersebut. Untuk memudahkan, buatlah dalam tabel.

Properti objek			
i	w_i	p_i	p_i/w_i
{1,2}	150	75	0,5
{1,3}	145	58	0,4
{1,4}	120	44	0,37
{1,5}	110	50	0,45
{1,6}	105	42	0,4
{2,3}	95	53	0,56
{2,4}	70	39	0,56
{2,5}	60	45	0,75
{2,6}	55	37	0,67
{3,4}	65	22	0,34
{3,5}	55	28	0,51
{3,6}	50	20	0,4
{4,5}	30	14	0,47
{4,6}	25	6	0,24
{5,6}	15	12	0,8

Brudy by weight:

1. Pilih *i* dengan berat terkecil dan tidak lebih dari 100, dipilih {5,6}, berat sisa = 85.
2. Pilih *i* berikutnya dengan berat terkecil, tidak lebih dari 85, dan $i \cap \{5,6\} = \{\}$, dipilih {3,4}, berat sisa = 20
3. Pilih *i* berikutnya dengan berat terkecil, tidak lebih dari 20, dan $i \cap \{3,4,5,6\} = \{\}$. Tidak ada yang dipilih.
4. *Decrement b*, $b = 1$, gunakan tabel pada subbab **2.2.2**, pilih *i* berikutnya dengan berat terkecil, tidak lebih dari 20, dan $i \cap \{3,4,5,6\} = \{\}$. Tidak ada yang dipilih.
5. *Decrement b*, $b = 0$.
   Stop

Solusi = (0, 0, 1, 1, 1, 1). Berat = 80, profit = 34.

Untuk selanjutnya, penyelesaian dengan *brudy by profit* dan *brudy by density* dapat dikerjakan dengan cara yang sama dengan cara yang digunakan pada *brudy by density*.

Secara kasarnya, kompleksitas untuk algoritma ini adalah $O(2^b n^2/b^2)$.

Jadi, seperti pada masalah penukaran mata uang (baca subbab **3.2.1**), jika $b = n$, maka kompleksitasnya menjadi $O(2^n)$, dan ini sama dengan algoritma *brute force*.

Sebaliknya, jika $b = 1$, maka kompleksitasnya pun akan sama pula dengan algoritma *greedy*, yaitu $O(2n^2) = O(n^2)$.

3.3. Kekuatan dan Kelemahan Algoritma *Brudy*

Algoritma *brudy* sebagai hasil penggabungan dari algoritma *brute force* dan algoritma *greedy* memiliki kekuatan di beberapa hal, misalnya di bagian kecepatan, kemangkusan, ketepatan mendapatkan solusi, dan lain-lain.

Dalam banyak hal, algoritma *brudy* bisa ditempatkan ditengah-tengah dari algoritma *brute force* dan algoritma *greedy*.

Misal dalam hal kecepatan, kecepatan algoritma *greedy* lebih cepat dari algoritma *brudy* dan algoritma *brudy* lebih cepat dari algoritma *brute force*. Beda lagi dengan ketepatan solusi, ketepatan solusi algoritma *brute force* lebih baik dari algoritma *brudy* dan ketepatan solusi algoritma *brudy* lebih baik dari algoritma *greedy*.

Kedua hal diatas (kecepatan dan ketepatan solusi) tidak bisa dianggap kelemahan ataupun kekuatan suatu algoritma, semua tergantung pada *user* yang memakai, apakah ingin menggunakan program yang memiliki kecepatan tinggi ataukah program yang memiliki ketepatan tinggi ataukah program yang memiliki ketepatan dan kecepatan lumayan?

Dan seperti algoritma hasil pewarisan dua induk, algoritma ini pun memiliki kelemahan induknya, terutama dari algoritma *greedy*. Masalah utama yang terasa adalah dengan menggunakan algoritma ini, hampir dapat dipastikan bahwa solusi yang dapat ditemukan tidak semuanya (bahkan, cukup besar bahwa solusi yang ditemukan hanya satu).

Terakhir, dan yang paling disayangkan adalah ternyata program ini membawa kelemahan yang bahkan tidak dimiliki induk-induknya, karena terkadang ia memiliki kecepatan yang lebih rendah dari algoritma *greedy* dan ternyata ia menghasilkan jawaban yang lebih buruk dari algoritma *greedy* pula.

Sebagai contoh, pada masalah penukaran uang, jika terdapat koin 5, 4, 2 dalam jumlah banyak, dan uang yang ditukar adalah 14. Misalkan nilai batas = 2. Maka koin yang diambil adalah (5, 5, 2, 2), dan salah satu metode untuk memperbaikinya adalah sekali lagi menggunakan koin maya bernilai 0 (baca subbab **2.2.1**).

Ataupun pada kasus yang lain, pada subbab **3.2.2** langkah ke-4 *brudy by weight* ternyata masih membutuhkan tabel *weight* dari setiap benda, meskipun sebenarnya itu tidak dibutuhkan karena pasti diberi di awal soal. Akan tetapi jika data yang dibutuhkan adalah data *profit per weight*, maka kemangkusan algoritma akan berkuran karena harus membuat data *profit per weight* terlebih dahulu.

4. Kesimpulan

Algoritma *brudy* masih perlu diujicoba kelayakannya, karena ternyata memiliki beberapa kelemahan. Akan tetapi, dari pengetesan selama ini terhadap beberapa persoalan yang diberikan, semuanya dapat terselesaikan secara benar oleh algoritma tersebut.

Untuk mengantisipasi kelemahan-kelemahan yang dimiliku oleh algoritma ini (terutama kelemahan-kelemahan yang tidak dimiliki baik oleh algoritma *brute force* maupun algoritma *greedy*), sudah sepantasnya kedepannya perlu dibuat penelitian untuk lebih mengembangkan algoritma *brudy* ini, sehingga nantinya algoritma ini benar-benar dapat dimanfaatkan pada kehidupan nyata, bukan hanya tertanam sebagai tulisan. Pada akhirnya, sangat besar kemungkinan algoritma ini menjadi *heuristic brudy* karena untuk menutupi kelemahan-kelemahan yang dimiliki oleh algoritma ini, perlu beberapa *heuristic* untuk memperbaikinya. Misalnya, seperti pada

penambahan koin maya bernilai 0 pada masalah penukaran uang, ataupun penambahan data agar pada persoalan *integer knapsack*, tidak dibutuhkan data tabel jika terjadi *decrement* nilai batas, dan pada persoalan-persoalan lainnya (sampai saat ini, penulis baru menguji kesahihan algoritma ini pada persoalan penukaran uang dan *integer knapsack*).

Penulis sendiri menganggap, semua persoalan yang dapat diselesaikan oleh algoritma *greedy* seharusnya dapat diselesaikan pula oleh algoritma *brudy*, karena seperti yang pernah dituliskan sebelumnya (baca subbab **3.1**), algoritma *brudy* bisa juga dianggap sebagai algoritma *b-greedy*.

Daftar Pustaka

[1] Rinaldi Munir, *Diktat Kuliah Strategi Algoritmik*. Departemen Teknik Informatika ITB : 2005.

[2] Rinaldi Munir, *Diktat Kuliah Matematika Diskrit*. Departemen Teknik Informatika ITB : 2005.