EKSPLORASI ALGORITMA *BRUTE FORCE, GREEDY* DAN PEMROGRAMAN DINAMIS PADA PENYELESAIAN MASALAH 0/1 KNAPSACK

Prasetyo Andy Wicaksono - 13505030

Program Studi T. Informatika, STEI, Institut Teknologi Bandung
Jl. Ganesha 10 Bandung 40132
e-mail: if15030@students.if.itb.ac.id

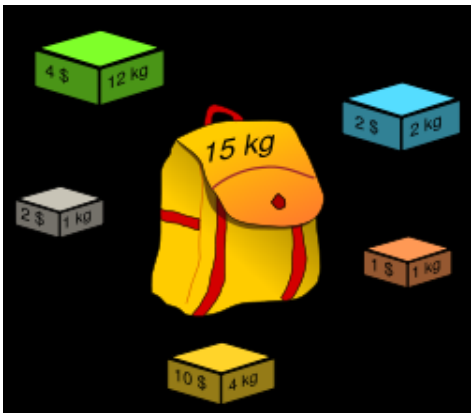
ABSTRAK

Setiap manusia menginginkan keuntungan yang sebanyak-banyaknya dengan hanya menggunakan sumber daya yang seefisien mungkin dengan berbagai batasan yang ada. Salah satu contohnya dalam kehidupan sehari-hari manusia adalah dalam permasalahan saat seseorang ingin memilih benda apa saja yang sesuai untuk dimasukkan ke dalam suatu wadah yang mempunyai keterbatasan ruang dan daya tampung, sehingga dengan konfigurasi beberapa benda yang dimasukkan, solusi yang didapatkan menghasilkan keuntungan yang maksimal. Pada beberapa kasus, benda-benda yang dimasukkan adalah benda yang berbentuk satuan dan tidak bisa dipecah menjadi beberapa bagian. Sehingga jika ingin memasukkan benda tersebut, maka satu kesatuan benda harus masuk ke dalam wadah. Permasalahan seperti ini disebut *Integer Knapsack*. Pada makalah ini akan dibahas eksplorasi dan perbandingan metode pencarian solusi permasalahan *Integer Knapsack* dengan menggunakan algoritma *Brute Force, Greedy* dan *Dynamic Programming* sehingga didapatkan algoritma yang paling mangkus untuk dibandingkan satu dengan yang lainnya. Perbandingan algoritma-algoritma ini meliputi tingkat kompleksitas dan kesulitan implementasi dari setiap algoritma.

Kata kunci: *Integer Knapsack*, *Brute Force, Greedy, Dynamic Programming*

1. PENDAHULUAN

Setiap manusia menginginkan keuntungan yang sebanyak-banyaknya dengan hanya menggunakan sumber daya yang seefisien mungkin dengena berbagai batasan yang ada. Salah satu contohnya dalam kehidupan sehari-hari menusia adalah dalam permasalahan saat seseorang ingin memilih benda apa saja yang sesuai untuk dimasukkan ke dalam suatu wadah yang mempunyai keterbatasan ruang dan daya tampung, sehingga dengan konfigurasi beberapa benda yang dimasukkan, solusi yang didapatkan menghasilkan keuntungan yang maksimal. Pada beberapa kasus, benda-benda yang dimasukkan adalah benda yang berbentuk satuan dan tidak bisa dipecah menjadi beberapa bagian. Sehingga jika ingin memasukkan benda tersebut, maka satu kesatuan benda harus masuk ke dalam wadah. Permasalahan ini disebut *Integer Knapsack*.Persoalan *Integer Knapsack* ini pada dasarnya dapat dicari solusinya dengan beberapa macam algoritma yang ada. Namun terdapat tiga algoritma yang bersesuaian dengan permasalahan ini untuk dianalisis, yaitu dengan menggunakan algoritma *Brute Force, Greedy* dan *Dynamic Programming.*



Gambar 1. Contoh Persoalan *Integer Knapsack* pada kehidupan sehari-hari, yaitu penentuan benda pada tas.

2. METODE

Pada bab ini akan dibahas lebih mendalam satu persatu mengenai apa itu *Integer Knapsack*, algoritma *Brute Force*, *Greedy,* dan *Dynamic Programming* pada pencarian solusi permasalahan *Integer Knapsack.*

2.1 Integer Knapsack

Integer knapsack dalam konteks ini dimisalkan diberikan *n* buah benda, x_1 sampai x_n, dan sebuah *knapsack* (karung, tas, dsb.) dengan kapasitas bobot K. Setiap benda memiliki bobot w_i dan keuntungan p_i. Objektif persoalan adalah bagaimana memilih benda-benda yang dimasukkan ke dalam *knapsack* sehingga tidak melebihi kapasitas *knapsack*, namun tetap memaksimumkan total keuntungan.[1]

Permasalahan *integer knapsack* mempunyai solusi persoalan dinyatakan sebagai himpunan:

$$X = \{x_1, x_2, x_3, \ldots, x_n\}$$

Dimana setiap elemen x ke-n (x_n) diisi dengan angka 0 atau 1. Misalkan $x_1 = 1$, berarti benda x_1 dimasukkan ke dalam *knapsack*, dan jika $x_1 = 0$, maka benda x_1 tidak dimasukkan ke dalam *knapsack*. Jika solusi yang ditemukan adalah X = {1,0,1}, maka benda ke-1 dan ke-3 dimasukkan ke dalam *knapsack*, dan benda ke-2 tidak dimasukkan.

Secara matematis persoalan *integer knapsack* dirumuskan sebagai berikut:

maksimasi $$\sum_{j=1}^{n} p_j x_j. \qquad (1)$$

dengan batasan

$$\sum_{j=1}^{n} w_j x_j \le c, \qquad x_j = 0 \text{ or } 1, \quad j = 1, \ldots, n. \qquad (2)$$

2.2 Algoritma *Brute Force*

Algoritma *Brute Force* adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya langsung pada pernyataan masalah (*problem statement*), dan definisi konsep yang dilibatkan.[1]

Prinsip pencarian solusi permasalahan *Integer Knapsack* menggunakan algoritma *brute force* adalah:

1. Mengenumerasikan list semua hiimpunan bagian dari himpunan dengan *n* objek
2. Menghitung total keuntungan dari setiap himpunan bagian dari langkah 1
3. Memilih himpunan bagian yang memerbikan total keuntungan terbesar

Misalkan pada kasus diberikan *n* = 3. Misalkan objek-objek yang ada kita berikan nomor 1, 2, 3. Sehingga objek ke-1 adalah x_1, dan begitu seterusnya. Properti dari setiap objek ke-*i* dan kapasitas dari *knapsack* adalah sebagai berikut:

$w_1 = 3$; $p_1 = 30$
$w_2 = 2$; $p_2 = 25$
$w_3 = 5$; $p_3 = 20$

Kapasitas *knapsack* adalah K = 8

Langkah-langkah pencarian solusi secara *brute force* digambarkan dalam tabel berikut:

Tabel 1. Tabel Contoh Pencarian Solusi *Integer Knapsack* menggunakan *Brute Force*

Himpunan Bagian	Total Bobot	Total Keuntungan
{}	0	0
{1}	3	30
{2}	2	25
{3}	5	20
{1,2}	**5**	**55**
{2,3}	7	45
{1,3}	8	50
{1,2,3}	10	Tidak layak

Catatan:

Himpunan bagian {1,2,3} tidak menghasilkan solusi karena menghasilkan total bobot yang lebih besar dari kapasitas *knapsack* K.

Dari tabel di atas didapatkan himpunan bagian yang memberikan keuntungan maksimum jika konfigurasi objek yang dimasukkan adalah {1,2}. Dengan total keuntungan 55. Solusi dari permasalahan ini adalah X = {1,1,0}. Artinya objek ke-1 dan ke-2 dimasukkan ke dalam *knapsack*, sedangkan objek ke-3 tidak dimasukkan.

2.3 Algoritma *Greedy*

Algoritma *greedy* membentuk solusi langkah per langkah. Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya. [1]

Untuk pencarian solusi permasalahan *integer knapsack* menggunakan algoritma *greedy*, masalah dipecahkan dengan memasukkan objek satu per satu ke dalam *knapsack*. Sekali objek tersebut dimasukkan ke dalam *knapsack,* maka objek tersebut tidak dapat dikeluarkan lagi. Ada beberapa strategi algoritma *greedy* yang dapat digunakan untuk pemecahan masalah ini, bergantung pada properti objek yang akan dijadikan parameter *greedy*:

1. *Greedy by weight*
   Pada setiap langkah pada strategi ini, *knapsack* diisi dengan objek yang memiliki bobot lebih ringan terlebih dahulu. Tujuan dari strategi ini adalah

untuk memaksimumkan jumlah kuantitas objek yang dapat masuk ke dalam *knapsack*.

2. *Greedy by profit*
   Pada setiap langkah pada strategi ini, *knapsack* diisi dengan objek yang memiliki keuntungan lebih besar terlebih dahulu. Tujuan dari strategi ini adalah untuk memaksumumkan jumlah keuntungan dari objek yang dimasukkan ke dalam *knapsack*.
3. *Greedy by density*
   Pada setiap langkah pada strategi ini, *knapsack* diisi dengan objek yang memiliki rasio keuntungan dibagi dengan bobot (p_i/w_i) yang paling besar. Tujuan dari strategi ini adalah untuk memaksimumkan keuntungan dari objek yang dimasukkan, namun *knapsack* tetap diisikan dengan jumlah objek sebanyak mungkin.

Dimisalkan dari contoh persoalan yang dipakai di subbab *brute force.*Diberikan kasus $n = 3$. Misalkan objek-objek yang ada kita berikan nomor 1, 2, 3. Sehingga objek ke-1 adalah x_1, dan begitu seterusnya. Properti dari setiap objek ke-*i* dan kapasitas dari *knapsack* adalah sebagai berikut:

$w_1 = 3$; $p_1 = 30$
$w_2 = 2$; $p_2 = 25$
$w_3 = 5$; $p_3 = 20$

Kapasitas *knapsack* adalah K = 8

Maka tabel solusi dengan menggunakan algoritma *greedy* adalah sebagai berikut:

Tabel 2. Tabel Contoh Pencarian Solusi *Integer Knapsack* menggunakan *greedy*dengan n = 4

Properti Objek				*Greedy by*			Solusi Optimal
i	w_i	p_i	p_i/w_i	*Weight*	*Profit*	*Density*	
1	3	30	0,1	1	1	1	1
2	2	25	0,08	1	1	0	1
3	5	20	0.25	0	0	1	0
Total Bobot				5	5	8	5
Total Keuntungan				55	55	50	55

Dari tabel di atas algoritma *greedy* dengan strategi pemilihan *greedy by weight dan profit* menghasilkan solusi optimal (solusi optimal diperoleh dari algoritma *brute force* yang dijabarkan pada subbab sebelumnya), sedangkan strategi *greedy by density* tidak menghasilkan solusi optimal.

Misalkan pada contoh lainnya dengan menggunakan enam objek:

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

Kapasitas *knapsack* $W = 100$

Tabel 3. Contoh langkah pencarian solusi *Integer Knapsack* secara *greedy* dengan n=6

Properti Objek				*Greedy by*			Solusi Optimal
i	w_i	p_i	p_i/w_i	*Weight*	*Profit*	*Density*	
1	100	40	0,4	1	0	0	0
2	50	35	0,7	0	0	1	1
3	45	18	0,4	0	1	0	1
4	20	4	0,2	0	1	1	0
5	10	10	1,0	0	1	1	0
6	5	2	0,4	0	1	1	0
Total bobot				100	80	85	100
Total keuntungan				40	34	51	55

Dari tabel di atas, algoritma *greedy* dengan ketiga strategi yang ada tidak ada satu pun yang menghasilkan solusi optimal. Solusi optimal yang dicari dengan algoritma *brute force* adalah X = {0,1,1,0,0,0}, dengan total bobot = 100, dan total keuntungan = 55.

2.3 Algoritma *Dynamic Programming*

Dynamic programming, atau dalam bahasa Indonesia berarti program dinamis, adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. [1]

Pada penyelesaian persoalan dengan metode ini:

1. Terdapat sejumlah bilangan berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya
3. Kita menggunakan persyaratan optimasi dan kendalauntuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Metode ini mirip dengan algoritma *greedy* karena program dinamis dan *greedy* sama-sama membentuk solusi secara bertahap.

Dua pendekatan yang digunakan adalah dengan *dynamic programming* adalah maju (*forward atau top-down*) dan mundur (*backward* atau *bottom-up*). Misalkan x_1, x_2, … ,x_n menyatakan peubah keputusan yang harus dibuat masing-masing untuk tahap 1 ,2 , …, *n*. Maka,

a. Program dinamis maju. Program dinamis bergerak mulai tahap 1 terus maju ke tahap 2, dan seterusnya sampai tahap n. Runtuntan peubah keputusan adalah x_1, x_2, … ,x_n.

b. Program dinamis mundur. Program dinamus bergerak mulai tahap n, lalu mundur ke tahap $n - 1$, tahap $n - 2$, dan seterusnya sampai tahap 1. Runtutan peubah keputusan adalah x_n, x_{n-1}, $x_{n-2},\ldots,x_1$

Secara umum, ada empat langkah yang dilakukan untuk mengembangkan algoritma program dinamis, yaitu:

1. Mengkarateristik struktur solusi optimal
2. Mendefinisikan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Konstruksi solusi optimal

Penerapan metode program dinamis maju pada persoalan *integer knapsack* adalah:

1. Tahap (k) adalah proses memasukkan benda ke dalam tas
2. Status (y) menyatakan kapasitas muat tas yang tersisa setelah memasukkan benda pada tahap sebelumnya
   - Dari tahap ke-1, kita masukkan objek ke-1 ke dalam karung untuk setiap satuan kapasitas tas sampai batas maksimumnya.Karena kapasitas karung adalah bilangan bulat, maka pendekatan ini praktis
   - Misalkan ketika memasukkan objek pada tahap k, kapasitas muat karung sekarang adalah $y - w_k$.
   - Untuk mengisi kapasitas sisanya, kita menerapkan prinsip optimalitas dengan mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas sisa $y - w_k$ (yaitu $f_{k-1}(y - w_k)$)
   - Selanjutnya kita bandingkan nilai keuntungan dari objek pada tahap k (yaitu p_k) plus nilai $f_{k-1}(y - w_k)$ dengan keuntungan pengisian hanya $k - 1$ macam objek, $f_{k-1}(y)$
   - Jika $p_k + f_{k-1}(y - w_k)$ lebih kecil dari $f_{k-1}(y)$, maka objek yang ke-k tidak dimasukkan ke dalam karung, tetapi jika lebih besar, maka objek yang ke-k dimasukkan

Relasi rekurens untuk persoalan ini adalah

$$f_0(y) = 0,\ y = 0, 1, 2, \ldots, M \qquad \text{(basis)}$$

$$f_k(y) = -\infty,\ y < 0 \qquad \text{(basis)}$$

$$f_k(y) = \max\{f_{k-1}(y), p_k + f_{k-1}(y - w_k)\},\ k = 1, 2, \ldots, n \qquad \text{(rekurens)}$$

yang dalam hal ini,

$f_k(y)$ adalah keuntungan optimum dari persoalan *integer knapsack* pada tahap k untuk kapasitas karung sebesar y.

$f_0(y) = 0$ adalah nilai dari persoalan *knapsack* kosong dengan kapasitas y

$f_k(y) = -\infty$ adlah nilai dari persoalan *knapsack* untuk kapasitas negatif. Solusi optimum dari persoalan *integer knapsack* adalah $f_n(M)$.

Misalkan digambarkan dalam contoh persoalan memuatkan tiga macam benda ke dalam tas. Tiap macam benda memiliki berat w_i dan akan memberikan keuntungan p_i, dimana i = (1,2,3). Kapasitas muat benda adalah M (dalam satuan berat). Tentukan benda-benda apa saja yang dimasukkan ke dalam tas sehingga memberikan keuntungan penjualan yang maksimum, namun total berat barang tidak boleh melebihi M.

$M = 5$

Tabel 4. Bobot dan Keuntungan Barang ($n = 3$)

Benda ke – i	w_i	p_i
1	2	65
2	3	80
3	1	30

Dari data pada tabel di atas, maka dapat dicari solusi dengan bertahap.

Tahap 1:

$$f_1(y) = \max\{f_0(y), p_1 + f_0(y - w_1)\}$$
$$= \max\{f_0(y), 65 + f_0(y - 2)\}$$

Tabel 5. Tahap 1 Pencarian Solusi *integer knapsack* dengan Pemrograman Dinamis

			Solusi Optimum	
y	$f_0(y)$	$65 + f_0(y-2)$	$f_1(y)$	(x1*,x2*,x3*)
0	0	$-\infty$	0	(0,0,0)
1	0	$-\infty$	0	(0,0,0)
2	0	65	65	(1,0,0)
3	0	65	65	(1,0,0)
4	0	65	65	(1,0,0)
5	0	65	65	(1,0,0)

Tahap 2:

$$f_2(y) = \max\{f_1(y), p_2 + f_1(y - w_2)\}$$
$$= \max\{f_1(y), 80 + f_1(y - 3)\}$$

Tabel 6. Tahap 2 Pencarian Solusi *integer knapsack* dengan Pemrograman Dinamis

			Solusi Optimum	
y	$f_1(y)$	$80 + f_1(y-3)$	$f_2(y)$	(x1*,x2*,x3*)
0	0	$80 + (-\infty) = -\infty$	0	(0,0,0)
1	0	$80 + (-\infty) = -\infty$	0	(0,0,0)
2	65	$80 + (-\infty) = -\infty$	65	(1,0,0)
3	65	80 + 0 = 80	80	(0,1,0)

4	65	80 + 0 = 80	80	(0,1,0)
5	65	80 + 65 = 145	145	(0,1,0)

Tahap 3:

$$f_3(y) = \max\{f_2(y), p_3 + f_2(y - w_3)\}$$
$$= \max\{f_2(y), 30 + f_2(y - 1)\}$$

Tabel 7. Tahap 3 Pencarian Solusi *integer knapsack* dengan Pemrograman Dinamis

			Solusi Optimum	
y	f_2y)	$30 + f_2(y\text{-}1)$	$f_3(y)$	(x1*,x2*,x3*)
0	0	$30 + (-\infty) = -\infty$	0	(0,0,0)
1	0	$30 + (-\infty) = -\infty$	0	(0,0,0)
2	65	30 +0 = 30	65	(1,0,0)
3	80	30 + 65 = 95	95	(1,0,1)
4	80	30 + 80 = 110	110	(0,1,1)
5	**145**	**30 + 80 = 110**	**145**	**(0,1,0)**

Solusi optimum dari persoalan *integer knapsack* di atas adalah (1,1,0) dengan nilai keuntungan maksimum adalah $f = 145$.

2.4 Perbandingan Ketiga Algoritma

Setelah menganalisis ketiga algoritma untuk persoalan *integer knapsack*, terlihat bahwa ketiga algoritma ini mempunyai karakteristik kerja yang berbeda dan mempunyai kompleksitas waktu yang berbeda pula.

Penyelesaian permasalahan *integer knapsack* dengan algoritma *brute force* memiliki kompleksitas waktu yang paling besar. Hal ini terjadi karena algoritma ini mengecek semua kemungkinan himpunan bagian yang terjadi pada himpunan benda-benda tersebut, yang merupakan karakteristik algoritma ini. Dari banyak pengecekan himpunan bagian sebanyak 2^n. Setelah semua kemungkinan didapat lalu dilakukan lagi pencarian hasil optimum yang merupakan operasi perbandingan sebanyak *n* kali, maka kompleksitasnya menjadi O ($n.2^n$).

Pencarian dengan algoritma *greedy*, jumlah langkah yang diperlukan untuk mencapai solusi permasalahan *integer knapsack* ini bergantung pada pada jumlah objek yang dimiliki. Untuk menentukan objek dengan keuntungan maksimal diperlukan proses perbandingan sebanyak *n* kali, lalu setiap obejk akan diuji apakah dipilih untuk masuk ke dalam *knapsack* atau tidak dengan proses sebanyak *n* kali. Sehingga kompleksitas waktunya menjadi O(n^2).

Pada algoritma pemrograman dinamis. Pemrograman dinamis tidak mempunyai algoritma yang pasti untuk permasalahan *integer knapsack*. Karena status yang dibangkitkan tidak selalu sama jumlahnya. Namun dari langka-langkah penyelesaiannya dapat dilihat bahwa penggunaan algoritma pemrograman dinamis mangkus untuk persoalan *integer knapsack.*

IV. KESIMPULAN

Setelah menganalisis ketiga algoritma di atas, dihasilkan bahwa penyelesaian permasalahan *integer knapsack* denga pemrograman dinamis adalah metode penyelesaian yang lebih mangkus dibandingkan dengan dua algoritma yang lainnya. Namun akan terasa rumit saat fase pengimplementasian ke bahasa pemrograman.

Algoritma *greedy* merupakan algoritma yang mempunyai tingkat kemangkusan kedua untuk permasalahan ini, dibanding dengan dua algoritma yang lainnya, saat pengimplementasiannya ke dalam bahasa pemrograman tidak terasa kesulitan yang berarti, namun solusi yang dihasilkan algoritma ini tidak selalu merupakan solusi optimal.

Algorita *brute force* merupakan algoritma yang paling mudah untuk diimplementasi, namun butuh usaha yang sangat besar untuk kasus dengan masukan yang besar, karena kompleksitas waktu yang dimiliki eksponensial.

REFERENSI

[1] Munir, Rinaldi, *Diktat Kuliah IF2251 : Strategi Algoritmik,* 2006

[2] Martello, Silvano, *New Trends in exact algorithms for the 0-1 knapsack problem,* 1997

[3] http://wikipedia.org/ , diakses pada 17 Mei 2007