PENERAPAN
ALGORITMA *GREEDY* DAN *DYNAMIC PROGRAMMING* PADA PERMASALAHAN *INTEGER KNAPSACK*

SKRIPSI

Oleh:
Winda Mega Arista
NIM. 071810101019

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2013

PENERAPAN
ALGORITMA *GREEDY* DAN *DYNAMIC PROGRAMMING* PADA PERMASALAHAN *INTEGER KNAPSACK*

SKRIPSI

Diajukan guna melengkapi tugas akhir dan memenuhi salah satu syarat
untuk menyelesaikan Program Study Matematika (S1)
dan mencapai gelar Sarjana Sains

Oleh:
Winda Mega Arista
NIM. 071810101019

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER

2013

PERSEMBAHAN

Skripsi ini saya persembahkan untuk :

1. kedua orang tua tercinta Ayahanda Sugito dan Bunda Puji Hernanik;
2. guru-guruku sejak taman kanak-kanak sampai dengan perguruan tinggi;
3. almamater Fakultas Matematika dan Ilmu Pengetahuan Alam.

MOTTO

Never put off till tomorrow what may be done today. [*)]

(Proverbs)

Friendship makes prosperity more shining and lessens adversity by deviding and sharing it. [**)]

(Cicero (106 BC-43 BC) on Friendship, 44 B.C.)

Sesungguhnya dalam kesulitan ada kemudahan. Bila engkau telah selesai dari suatu pekerjaan maka kerjakan urusan yang lainnya dengan tekun. Namun kepada Tuhanmulah engkau berharap. [***)]

(Terjemahan Q.S. surat Al- Insyirah: 6-8)

---

*) B.A, Evelyn Len and P.S. Yue.1988. *Secondary 1 vocabulary Guide& Practice*. Singapore: Preston Corporation (PTE) LTD

**) Departemen Agama Republik Indonesia. 1998. *Al Qur'an dan terjemahannya.* Semarang: PT. Kumudasmoro Grafindo.

PERNYATAAN

Saya yang bertanda tangan dibawah ini :

Nama : Winda Mega Arista

NIM : 071810101019

Menyatakan dengan sesungguhnya bahwa karya ilmiah yang berjudul "Penerapan Algoritma *Greedy* dan *Dynamic Programming* pada Permasalahan *Integer Knapsack*" adalah benar-benar hasil karya sendiri, kecuali kutipan yang sudah saya sebutkan sumbernya, belum pernah diajukan pada institusi mana pun, dan bukan karya jiplakan. Saya bertanggung jawab atas keabsahan dan kebenaran isinya sesuai dengan sikap ilmiah yang harus dijunjung tinggi.

Demikian pernyataan ini saya buat dengan sebenarnya, tanpa ada tekanan dan paksaan dari pihak manapun serta bersedia mendapat sanksi akademik jika ternyata di kemudian hari pernyataan ini tidak benar.

Jember,
Yang Menyatakan

Winda Mega Arista
NIM. 071810101019

SKRIPSI

PENERAPAN ALGORITMA *GREEDY* DAN *DYNAMIC PROGRAMMING* PADA PERMASALAHAN *INTEGER KNAPSACK*

Oleh

Winda Mega Arista
NIM. 071810101019

Pembimbing

Dosen Pembimbing Utama : Kiswara Agung Santoso, S.Si, M.Si.

Dosen Pembimbing Anggota : Kusbudiono, S.Si, M.Si.

PENGESAHAN

Skripsi berjudul "Penerapan Algoritma *Greedy* dan *Dynamic Programming* pada permasalahan *Integer Knapsack"* telah diuji dan disahkan pada:

Hari, tanggal :

Tempat : Ruang Sidang FMIPA Universitas Jember

Tim Penguji,

Ketua,	Sekretaris,
Kiswara Agung Santoso, S.Si, M.Si. NIP. 197209071998031003	Kusbudiono, S.Si, M.Si. NIP. 197704302005011001
Anggota 1,	Anggota 2,
Prof. Drs. Kusno, DEA., PhD. NIP. 196101081986021001	Yuliani Setia Dewi, S.Si, M.Si. NIP. 197407162000032001

Mengesahkan
Dekan,

Prof. Drs. Kusno, DEA., PhD.
NIP. 196101081986021001

RINGKASAN

Penerapan Algoritma *Greedy* dan *Dynamic Programming* pada Permasalahan *Integer Knapsack;* Winda Mega Arista, 071810101019; 2013; 52 Halaman; Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember.

Knapsack merupakan suatu permasalahan bagaimana memilih objek dari sekian banyak objek dan berapa besar objek tersebut akan disimpan sehingga diperoleh suatu penyimpanan yang optimal. *Knapsack* dapat diilustrasikan sebagai suatu kantong atau media penyimpanan. Kantong atau media penyimpanan tersebut hanya dapat menyimpan beberapa objek dengan batasan objek tersebut sama atau lebih kecil dari kapasitas media penyimpannya. Terkadang keterbatasan manusia dalam menyelesaikan masalah *knapsack* tanpa menggunakan alat bantu merupakan salah satu kendala dalam pencarian solusi optimum. Dengan adanya algoritma penyelesaian pada masalah *integer knapsack* diharapkan dapat membantu dalam proses pemilihan barang. Dengan adanya proses pemilihan barang yang tepat maka dapat membantu mendapatkan keuntungan maksimum.

Penelitian ini dilakukan di industri perdagangan UD. BINTANG TANI di Jl. Yos. Sudarso Kecamatan Semboro Kabupaten Jember. Pengambilan data dilakukan dengan metode wawancara dan data yang diambil berupa data harga beli, harga jual, dan banyaknya barang. Untuk menerapkan data tersebut dilakukan pengidentifikasian untuk mencari keuntungan (p_i) dan (w_i). Algoritma yang digunakan pada permasalahan *integer knapsack* ini adalah algortima *Greedy* dan *Dynamic Programming.* Data penelitian yang digunakan yaitu data sekunder. Tujuan dari peneliti adalah untuk mencari keuntungan maksimum di UD. BINTANG TANI pada permasalahan *integer knapsack* dengan menggunakan algoritma *Greedy* dan *Dynamic Programming,* serta membandingkan algoritma *Greedy* dan *Dynamic Programming* pada permasalahan *integer knapsack* dari segi hasil dan kompleksitas

waktu. Hasil penelitian menunjukkan: (1) Keuntungan maksimum penggunaan algoritma *Greedy* adalah sebesar Rp 687.500,- dengan bobot 479 kg. (2) Keuntungan maksimum penggunaan algoritma *Dynamic Programming* adalah sebesar Rp 691.500,- dengan bobot 499 kg. (3) Algoritma *Greedy* dan *Dynamic Programming* pada kasus permasalahan *integer knapsack* berdasarkan banyak langkah yang dibutuhkan diperoleh hasil pencarian bahwa pada algoritma *Greedy* diperlukan proses perbandingan sebanyak n^2 kali, maka kompleksitas waktunya adalah $O(n^2)$. Pada algoritma *Dynamic Programming* jumlah langkah yang diperlukan untuk mencapai solusi optimal adalah sebanyak m^3n^2, maka kompleksitas waktunya adalah $O(m^3n^2)$. Sehingga algoritma *Dynamic Programming* mempuyai jumlah kompleksitas yang lebih besar dibandingkan dengan algoritma *Greedy*.

Dari hasil di atas dapat disimpulkan bahwa dari segi hasil algortima *Dynamic Programming* lebih mencapai hasil yang maksimum daripada algoritma *Greedy* tetapi dalam segi kompleksitas waktu algoritma *Dynamic Programming* mempunyai kompleksitas waktu yang lebih besar daripada algoritma *Greedy*.

PRAKATA

Puji syukur kehadirat Allah SWT, atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul "Penerapan Algoritma *Greedy* dan *Dynamic Programming* pada Permasalahan *Integer Knapsack*". Skripsi ini disusun untuk memenuhi salah satu syarat menyelesaikan pendidikan Strata satu (S1), pada Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember.

Penyusunan skripsi ini tidak lepas dari bantuan berbagai pihak, oleh karena itu penulis ingin menyampaikan ucapan terima kasih kepada:

1. Prof. Drs. Kusno, DEA., PhD., selaku Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam yang telah memberikan perijinan dalam menyelesaikan karya ilmiah tertulis ini;
2. Kiswara Agung Santoso, S.Si, M.Si., selaku Dosen Pembimbing Utama, Kusbudiono, S.Si, M.Si., selaku Dosen Pembimbing Anggota, Prof. Drs. Kusno, DEA., PhD., selaku Dosen Penguji I, Yuliani Setia Dewi, S.Si, M.Si., selaku Dosen Penguji II yang telah meluangkan waktu, pikiran, dan perhatian dalam penulisan skripsi ini;
3. Prof. Drs. Kusno, DEA., PhD., selaku Dosen Pembimbing Akademik yang telah membimbing selama penulis menjadi mahasiswa;
4. Drs. Rusli Hidayat, M.Sc, selaku Ketua Jurusan Matematika yang telah memberikan bantuan sarana dan prasarana dalam menyelesaikan karya ilmiah tertulis ini;
5. seluruh Dosen Pengajar dan karyawan Fakultas Matematika dan Ilmu Pengetahuan Alam;
6. Orang tuaku tercinta Ayahanda Sugito dan Bunda Puji Hernanik yang selama ini telah memberikan dukungan dan nasihat baik materiil maupun moril serta doa dan kasih sayang;

7. semua keluarga serta adik-adikku yang selalu memberikan masukan;
8. temanku Ana Imadil Bilad, A.Md, yang selalu memberikan dorongan dan motivasi dalam penyelesaian skripsi ini;
9. semua teman Strata 1 Matematika yang telah memberikan saran dan kritiknya serta dorongan semangat dalam penyelesaian skripsi ini;
10. semua pihak yang telah membantu terselesaikannya penulisan skripsi ini.

Semoga Allah SWT memberikan balasan kepada mereka yang telah memberikan bantuan kepada penulis dalam menyelesaikan skripsi ini. Akhir kata, semoga skripsi ini dapat bermanfaat bagi pembaca.

Jember, Januari 2013 Penulis

DAFTAR ISI

Halaman

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

Halaman

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Setiap manusia pasti menginginkan keuntungan yang sebanyak-banyaknya dengan hanya menggunakan sumber daya yang seefisien mungkin dari berbagai batasan yang ada. Salah satu contohnya dalam kehidupan sehari-hari adalah dalam permasalahan saat seseorang memilih objek dari sekumpulan objek yang masing-masing mempunyai bobot/berat (*weight*) dan nilai/profit (*value*) untuk dimuat dalam sebuah media penyimpanan dengan keterbatasan ruang tertentu sehingga di dapat keuntungan maksimum dari objek-objek tersebut. Permasalahan seperti ini disebut problem *knapsack*.

Terkadang keterbatasan manusia dalam menyelesaikan masalah *knapsack* tanpa menggunakan alat bantu merupakan salah satu kendala dalam pencarian solusi optimum. Apalagi jika dihadapkan dengan jumlah objek terlampau banyak maka perhitungannya semakin sulit. Efisiensi waktu juga sering menjadi pertimbangan. Oleh karena itu, dibutuhkan suatu metode sekaligus program aplikasi penerapan metode tersebut yang dapat membantu menyelesaikan masalah *knapsack*.

Permasalahan *knapsack* terbagi menjadi tiga, yaitu *integer knapsack*, *bounded knapsack,* dan *unbounded knapsack*. Persoalan *integer knapsack* adalah menentukan objek mana saja yang harus dimuat atau tidak di muat sama sekali dalam media penyimpanan. Persoalan *bounded knapsack* adalah menentukan berapa bagian dari masing-masing objek yang akan dimuat dalam media penyimpanan, sedangkan pada *unbounded knapsack* persoalannya adalah penentuan untuk barang yang tidak terbatas (Martello, 2006).

Knapsack problem dapat diselesaikan dengan berbagai cara. Terdapat beberapa strategi algoritma yang dapat menghasilkan solusi optimal diantaranya algoritma *greedy*, *dynamic programming*, *branch and bound*, *brute force,* dan *genetik*. Tetapi

algoritma-algoritma tersebut mempunyai karakteristik berbeda dan mempunyai kompleksitas waktu yang berbeda pula. Pernah dilakukan penelitian dengan judul masalah *integer knapsack* diselesaikan dengan menggunakan algoritma *greedy* (Paryati, 2009). Pada penelitian tersebut algoritma *greedy* untuk persoalan *integer knapsack* diterapkan pada data primer yaitu data sejumlah barang yang akan dibeli dengan kapasitas dana yang tersedia. Pada penelitian tersebut disimpulkan bahwa algoritma *greedy* adalah algoritma yang baik untuk diaplikasikan pada persoalan *integer knapsack*.

Algoritma *greedy* merupakan salah satu metode dari sekian banyak metode yang dapat digunakan untuk menyelesaikan permasalahan *integer knapsack*. Contoh metode lain yang dapat digunakan untuk menyelesaikan permasalahan *integer knapsack* yaitu dengan algoritma *dynamic programming*. Dari latar belakang tersebut penulis tertarik untuk membahas penyelesaian persoalan tersebut dengan algoritma *greedy* dan *dynamic programming*. Tegasnya, untuk mengetahui algoritma yang terbaik maka dilakukan analisis antara algoritma *greedy* dan *dynamic programming* untuk menyelesaikan permasalahan *knapsack*. Hasil akhir dari skripsi ini diharapkan dapat mengetahui hasil perbandingan kedua algoritma.

Setiap perusahaan perlu menentukan pilihan masing-masing barang yang akan dijual dalam menjamin kelangsungan hidup usahanya. Tanpa adanya penentuan pemilihan barang, perusahaan akan dihadapkan pada resiko bahwa suatu waktu perusahaan tidak dapat memeuhi keinginan pelanggan yang memerlukan barang tersebut. Penentuan pemilihan barang sangat mempengaruhi omset penjualan dan perolehan keuntungan. Perolehan keuntungan maksimal merupakan tujuan yang diharapkan suatu perusahaan dalam menggunakan modalnya secara efektif dan efisien.

Salah satu perusahaan UD. BINTANG TANI, selalu berusaha untuk menentukan pilihan barang yang tepat guna menjamin kebutuhan bagi kelancaran usahanya. Terdapat berbagai macam pilihan barang yang harus di beli oleh

UD. BINTANG TANI. Dalam proses pemilihan barang tersebut tedapat batasan yang harus diperhatikan yaitu besar kapasitas maksimum muat barang yang diangkut.

Berdasarkan masalah diatas, maka permasalahan yang digunakan dalam skripsi ini adalah permasalahan *integer knapsack* untuk mencari keuntungan maksimum di UD. BINTANG TANI dengan menggunakan algoritma *greedy* dan *dynamic programming* dan bagaimana menentukan pembelian barang yang tepat karena sangat penting dalam pemenuhan kebutuhan bagi kelancaran usahanya yakni untuk memenuhi kebutuhan para konsumen sehingga dapat mencapai keuntungan yang optimal.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah di uraikan di atas, maka dapat dirumuskan permasalahan sebagai berikut.

a. Bagaimana menentukan keuntungan maksimum pada permasalahan *knapsack* di UD. BINTANG TANI dengan menggunakan algoritma *greedy*.
b. Bagaimana menentukan keuntungan maksimum pada permasalahan *knapsack* di UD. BINTANG TANI dengan menggunakan algoritma *dynamic programming*.
c. Bagaimana cara mempercepat perhitungan dengan algoritma *greedy* dan *dynamic programming* untuk mencari keuntungan maksimum pada permasalahan *knapsack* di UD. BINTANG TANI.
d. Diantara algoritma *greedy* dan *dynamic programming*, algoritma mana yang memberikan solusi paling optimal untuk permasalahan *knapsack* (dari segi hasil maupun waktu yang dibutuhkan).

1.3 Batasan Masalah

Adapun batasan permasalahan yang harus dibatasi pada penyelesaian permasalahan *integer knapsack* di UD.BINTANG TANI yakni tingkat penjualan/permintaan konsumen untuk masing-masing barang diasumsikan sama.

1.4 Tujuan

Berdasarkan rumusan masalah di atas, maka tujuan yang akan dibahas adalah.

a. Mencari keuntungan maksimum pada permasalahan *knapsack* di UD. BINTANG TANI dengan menggunakan algoritma *greedy*.
b. Mencari keuntungan maksimum pada permasalahan *knapsack* di UD. BINTANG TANI dengan menggunakan algoritma *dynamic programming.*
c. Membuat program penyelesaian permasalahan *knapsack* menggunakan software MATLAB R2009a.
d. Membandingkan algoritma *greedy* dan *dynamic programming* (dari segi hasil dan waktu).

1.5 Manfaat

Manfaat yang diperoleh dari penulisan skripsi ini adalah.

a. Memberikan solusi dalam menentukan pemilihan barang yang harus dibeli oleh UD. BINTANG TANI untuk pemenuhan kebutuhan konsumen agar keuntungan maksimal.
b. Dengan adanya program maka dapat mempercepat perhitungan pencarian keuntungan maksimal di UD. BINTANG TANI.
c. Mengetahui perbandingan antara algoritma *greedy* dan *dynamic programming* dalam penyelesaian masalah *integer knapsack*.

BAB 2. TINJAUAN PUSTAKA

2.1 Optimasi

Optimasi ialah suatu proses untuk mencapai hasil yang ideal atau optimal (nilai efektif yang dapat dicapai) pada masalah yang berhubungan dengan keputusan yang terbaik, maksimum, minimum, dan memberikan cara penentuan solusi yang memuaskan. Untuk dapat mencapai nilai optimal baik minimal atau maksimal tersebut, secara sistematis dilakukan pemilihan nilai variabel integer atau nyata yang akan memberikan solusi optimal.

Persoalan yang berkaitan dengan optimasi sangat kompleks dalam kehidupan sehari-hari. Nilai optimal yang didapat dalam optimasi dapat berupa besaran panjang, waktu, jarak dan lain-lain. Berikut ini adalah beberapa persoalan yang memerlukan optimasi:

a. penentuan pemilihan barang pada masalah *knapsack*;
b. menentukan lintasan terpendek dari suatu tempat ke tempat yang lain;
c. menentukan jumlah pekerja seminimal mungkin untuk melakukan suatu proses produksi agar pengeluaran biaya pekerja dapat diminimalkan dan hasil produksi tetap maksimal;
d. mengatur jalur kendaraan umum agar semua lokasi dapat dijangkau.

2.2 Definisi *Knapsack*

Knapsack merupakan suatu permasalahan bagaimana memilih objek dari sekian banyak objek dan berapa besar objek tersebut akan disimpan sehingga diperoleh suatu penyimpanan yang optimal dengan memperhatikan objek yang terdiri dari n objek (1, 2, 3, …, n) dimana setiap objek memiliki bobot (w_i) dan nilai profit (p_i) dengan memperhatikan juga kapasitas dari media penyimpanan sebesar (M).

Masalah *knapsack* merupakan sebuah persoalan yang menarik. Dalam dunia nyata permasalahan *knapsack* ini sering sekali digunakan pada bidang (jasa) pengangkutan barang seperti pengangkutan peti kemas dalam sebuah media pengangkut. Dalam usaha tersebut, diinginkan keuntungan yang maksimal untuk mengangkut barang yang ada dengan tidak melebihi kapasitas yang ada. Berdasarkan persoalan tersebut, diharapkan ada suatu solusi yang secara otomatis dapat mengatasi persoalan itu. *Knapsack* adalah permasalahan mengenai optimalisasi kombinatorial dimana kita harus mencari solusi terbaik dari banyak kemungkinan yang dihasilkan. (Dimyati, 2004).

Knapsack sendiri terdiri dari beberapa persoalan, yaitu.

a. *Knapsack* 0-1 (*integer knapsack*)
   Objek yang dimasukkan ke dalam media penyimpanan dimensinya harus dimasukkan semua atau tidak sama sekali.
b. *Knapsack* terbatas (*bounded knapsack*)
   Objek yang dimasukkan ke dalam media penyimpanan dimensinya bisa dimasukkan sebagian atau seluruhnya.
c. *Knapsack* tak terbatas (*unbounded knapsack*).
   Jumlah objek yang dimasukkan ke dalam media penyimpanan macamnya tidak terbatas.

Knapsack yang akan dibahas pada skripsi ini adalah jenis *knapsack* 0-1 (*integer knapsack*). Variabel keputusan yang diperoleh yaitu x_i bernilai 1 jika objek dipilih dan x_i bernilai 0 jika objek tidak dipilih.

2.2.1 *Integer Knapsack* (0-1 *knapsack*)

Dalam persoalan ini, kita diberikan *n* buah objek dan sebuah media penyimpanan yang memiliki daya tampung maksimal senilai *M*. setiap benda memiliki bobot (w_i) dengan nilai keuntungan profit (p_i). Objektif dari permasalahan ini adalah bagaimana memilih objek-objek yang dimasukkan ke dalam media penyimpanan sehingga tidak melebihi kapasitas dari media penyimpanan namun

memaksimalkan total keuntungan yang diperoleh. Pada persoalan integer *knapsack* barang yang diangkut dimensinya harus diangkut seluruhnya atau tidak sama sekali.

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

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

Yang dalam hal ini, x_i = 1 jika benda ke-*i* dimasukkan ke dalam media penyimpanan, atau x_i = 0 jika benda ke-*i* tidak dimasukkan ke dalam media penyimpanan. Karena itulah persoalan ini dinamakan 0-1 *knapsack*. Sebagai contoh, X = {1, 0, 0, 1} adalah sebuah solusi yang ditemukan, maka benda ke-1 dan ke-4 dimasukkan ke dalam media penyimpanan, dan benda ke-2 dan ke-3 tidak dimasukkan ke dalam media penyimpanan.

Secara matematis, persoalan *integer knpasack* dapat dirumuskan seagai berikut:
Fungsi tujuan Maks/Min

$$Z = \sum_{i=1}^{n} p_{i\,x_i i}$$

Kendala

$$z = \sum_{i=1}^{n} w_{i x^i} \leq M$$

dengan x_i = 0 atau 1, i = 1, 2, …, *n*,
keterangan:

Z = nilai optimum dari fungsi tujuan,
z = kendala fungsi tujuan,
p_i = keuntungan barang-i, dengan i = 1, 2, …, *n*,
w_i = berat (weight) barang, dengan i = 1, 2, …, *n*,
M = kapasitas media penyimpanan (*knapsack*),
x_i = banyaknya barang jenis ke-i.

2.3 Algoritma

2.3.1 Definisi dan Sifat Algoritma

Algoritma merupakan serangkaian kata atau instruksi untuk mendapatkan hasil khusus dalam beberapa langkah berhingga (Chartrand & Oellerman, 1993). Algoritma juga bisa diartikan sebagai langkah-langkah penyelesaian masalah secara sistematis. Sebuah algoritma tidak hanya harus benar tetapi juga harus efisien (Munir, 2005).

Sifat-sifat algoritma meliputi hal-hal sebagai berikut:

a. jelas, yaitu setiap langkah pada setiap algoritma harus dinyatakan dengan jelas, tidak bermakna ganda dan ditetapkan dengan cermat;
b. logis (urut), yaitu algoritma dibuat berdasarkan aturan yang tetap, berdasarkan pada alur berfikirnya;
c. terhingga, yaitu sebuah algoritma harus berhenti setelah melakukan satu langkah atau lebih dalam suatu interval waktu tertentu. Tanpa sifat ini, sebuah algoritma tidak bisa diimplementasikan oleh mannusia maupun mesin;
d. menyelesaikan masalah, yaitu dengan input tertentu suatu algoritma akan menyelesaikan masalah dalam kelasnya;
e. efektif, yaitu sebuah algoritma harus menegaskan tindakan sederhana yang dapat dilakukan secara efektif. Sifat ini menjamin bahwa setiap langkah pada suatu algoritma secara nyata dapat dijalankan.

2.3.2 Efisiensi Algoritma

Dalam menganalisis suatu algoritma yang menjadi perhatian utama adalah berapa waktu tempuh dan berapa ruang dalam memori yang dibutuhkan untuk menjalankan algoritma tersebut. Meskipun suatu algoritma memberikan hasil yang mendekati optimal tetapi waktu yang dibutuhkan sangat lama maka algoritma tersebut biasanya jarang dipakai.

Beberapa keadaan dalam kompleksitas waktu adalah:

a. *best case* (suatu keadaan terbaik), yaitu waktu minimum yang dibutuhkan untuk menjalankan algoritma;
b. *worst case* (suatu keadaan terburuk), yaitu waktu maksimum yang dibutuhkan untuk menjalankan algoritma;
c. *average case* yaitu suatu keadaan rata-rata dari waktu untuk beberapa nilai masukan.

Kompleksitas waktu (T(n)) :

a. diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalanan algoritma sebagai fungsi dari ukuran masukan n;
b. dihitung dari jumlah operasi dasar yang dilakukan di dalam algoritma sebaga fungsi ukuran masukan (n).

Ada aspek lain yang berhubungan dengan efisiensi algoritma yaitu memori yang digunakan. Memori yang dibutuhkan dalam pemrograman berhubungan dengan perangkat keras komputer. Dengan semakin pesatnya perkembangan teknologi saat ini maka efisiensi memori bukanlah masalah yang serius. Oleh karena itu waktu proses merupakan fektor yang lebih penting dibanding memori (Siang, 2004)

2.4 Algoritma *Greedy*

Algoritma *greedy* digunakan untuk memperoleh penyelesaian dari suatu permasalahan optimasi. Suatu permasalahan dengan *n* masukkan data dilakukan secara bertahap. Pertama dilakukan pemilihan solusi yang mungkin kemudian dari himpunan solusi yang mungkin tersebut akan diperoleh solusi optimal.

Metode ini bekerja secara bertahap dengan memperhatikan setiap input data pada setiap keadaan. Pada setiap tahap, dibuat keputusan dengan memperhatikan ada atau tidak sebuah input data yang memberikan solusi optimal, dan memperhatikan pula urutan data dalam proses pengambilannya. Pendekatan yang digunakan pada algoritma *Greedy* adalah membuat pilihan yang dapat memberikan perolehan yang

terbaik yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan tujuan bahwa sisanya mengarah ke solusi optimum global (Springer V, 2005).

2.4.1 Prosedur Perhitungan Algoritma *Greedy* Pada Persoalan *Integer Knapsack*

Algoritma *greedy* adalah algoritma untuk menyelesaikan permasalahan secara bertahap (Brassard G, 1996).

Terdapat beberapa strategi *Greedy* yang dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *M* antara lain.

a. Greedy by profit

Algoritma *greedy by profit*:

1) tetapkan nilai kapasitas maksimum *knapsack*;
2) urutkan objek-objek berdasarkan keuntungan (*profit*) dari yang terbesar;
3) isi *knapsack* dengan objek yang memiliki keuntungan terbesar terlebih dahulu;
4) ambil satu-persatu objek yang dapat ditampung sampai kapasitas *knapsack* penuh;
5) hitung jumlah bobot dan keuntungan.

b. Greedy by weight

Algoritma *greedy by weight*:

1) tetapkan nilai kapasitas maksimum *knapsack*;
2) urutkan objek-objek berdasarkan berat dari yang teringan;
3) isi *knapsack* dengan objek yang memiliki berat teringan terlebih dahulu;
4) ambil satu-persatu objek yang dapat ditampung sampai kapasitas *knapsack* penuh;
5) hitung jumlah bobot dan keuntungan.

c. Greedy by density

Algoritma *greedy by density*:

1) tetapkan nilai kapasitas maksimum *knapsack*;
2) hitung rasio (p_i/w_i) dari tiap-tiap barang;
3) urutkan objek-objek berdasarkan rasio (*density*) terbesar terlebih dahulu;
4) ambil satu-persatu objek yang dapat ditampung sampai kapasitas *knapsack* penuh;
5) hitung jumlah bobot dan keuntungan.

2.5 Algoritma *Dynamic programming*

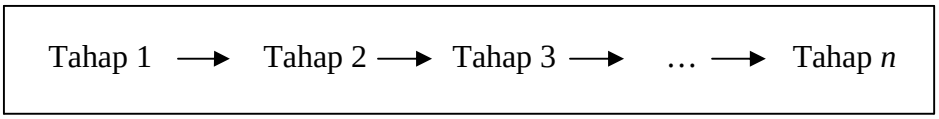
Dynamic programming adalah prosedur matematis yang terutama dirancang untuk memperbaiki efisien perhitungan masalah pemrograman matematis tertentu dengan menguraikannya menjadi bagian-bagian masalah yang lebih kecil, sehingga lebih sederhana dalam perhitungan. Suatu masalah yang diselesaikan dengan pemrograman dinamik dapat dibagi dalam tahap-tahap. Setiap tahap mewakili kemungkinan kapasitas dari masalah. Banyaknya tahap ditentukan pada awal pemrograman dinamik. Selanjutnya kemungkinan kapasitas dimulai dengan mengambil kemungkinan kapasitas minimal (0) sampai kemungkinan kapasitas maksimal (kemungkinan kapasitas yang tersedia). Untuk mencari total keuntungan optimal, dicari terlebih dahulu keuntungan yang diperoleh dari tiap kelompok barang dengan kemungkinan kapasitasnya. Perhitungan disetiap kelompok barang melalui persamaan rekursif dan selanjutnya menghasilkan penyelesaian optimal yang mungkin bagi seluruh barang (Taha, 1996).

Prosedur penyelesaian suatu masalah dengan *dynamic programming* yaitu dengan prosedur rekursif yang berarti bahwa setiap kali mengambil keputusan harus memperhatikan keadaan yang dihasilkan oleh keputusan optimal sebelumnya dan merupakan landasan bagi keputusan optimal berikutnya. Prosedur penyelesaian rekursif ada 2 macam yaitu prosedur maju (*forward procedure*) dan prosedur mundur

(*backward procedure*). Misalnya, jika diberikan n jumlah benda dinotasikan $i, i = 1,2, \dots, n, i \in N$, dengan syarat kapasitas yang tersedia dinotasikan dengan *M*, dimana $M \in N$, tiap-tiap benda mempunyai variabel keputusan dinotasikan $x_i, i = 1,2, \dots, n$.

Dikatakan prosedur maju karena dalam perhitungan dimulai dari tahap pertama ke tahap akhir. Perhitungan dimulai dengan mencari nilai keuntungan di tahap 1 lalu dilanjutkan mencari keuntungan di tahap 2 sampai tahap *n*. Setelah dilakukan perhitungan diperoleh keputusan optimal.

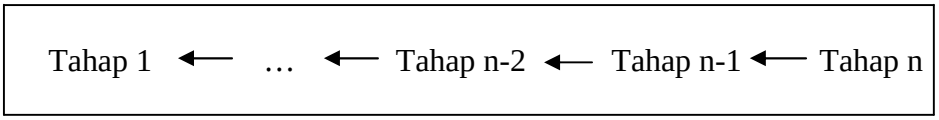
Model prosedur maju.



Gambar 2.1 Rekursif maju algoritma *Dynamic Programming*

Perhitungan yang dimulai pada tahap terakhir dan berlanjut ke belakang ke tahap 1 dinamakan prosedur mundur. Perhitungan dimulai dengan mencari nilai keuntungan di tahap *n* lalu dilanjutkan mencari nilai keuntungan *n-1* sampai tahap 1. Setelah dilakukan perhitungan diperoleh keputusan optimal.

Model prosedur mundur.



Gambar 2.2 Rekursif mundur algoritma *Dynamic Programming*

Dari ilustrasi di atas dimisalkan bahwa suatu masalah dapat dipecahkan dengan *n* tahapan. Pada ilustrasi Gambar 2.1 di atas $x_1, x_2, \dots, x_n$ menggambarkan rekursif maju (*forward recursion*), sedangkan Gambar 2.2 di atas $y_1, \dots, y_{n-2}, y_{n-1}, y_n$

menggambarkan rekursif mundur (*backward recursion*). Perbedaan pokok antara rekursif maju dan rekursif mundur terletak pada cara mendefinisikan tahap yang dipakai, apakah digunakan tahap pertama atau tahap terakhir untuk memulai perhitungan.

2.5.1 Prosedur Perhitungan Algoritma *dynamic programming* pada Persoalan *Integer knapsack*

Program dinamik (*dynamic programming*) merupakan 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.

Pada penyelesaian persoalan dengan metode *dynamic programming* ini terdapat beberapa persoalan, antara lain:

a. terdapat sejumlah berhingga pilihan yang mungkin;
b. solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya;
c. kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Penerapan algoritma *dynamic programming* rekursif maju (*forward recursion*) pada persoalan *integer knapsack* adalah sebagai berikut (Hiller, 1994):

a. tahap (k) adalah proses memasukkan objek ke dalam *M*,
b. status (y) menyatakan kapasitas muat *M* yang tersisa setelah memasukkan objek pada tahap sebelumnya.
    1) Dari tahap ke-1, kita masukkan objek ke-1 ke dalam *M* untuk setiap satuan kapasitas *M* sampai batas kapasitas maksimumnya. Karena kapasitas *M* adalah bilangan bulat, maka pendekatan ini praktis.
    2) Misalkan ketika memasukkan objek pada tahap *k*, kapasitas muat *M* sekarang adalah $y - w_k$,

3) Untuk mengisi kapasitas sisanya, kita menerapkan prinsip optimalitas dengan mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas sisa $y - w_k$ yaitu $f_{k-1}(y - w_k)$,
4) 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, $f_{k-1}(y)$
5) 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 *M*, tetapi jika lebih besar, makaobjek yang ke-*k* dimasukkan.

Relasi rekurens untuk persoalan ini adalah

$$f_0(y) = 0, y = 0, 1, 2, \dots, M$$

$$f_k(y) = -\infty, y < 0$$

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

yang dalam hal ini,

$f_k(y)$ = keuntungan optimum dari persoalan *integer knapsack* pada tahap k untuk kapasitas *M* sebesar y,

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

$f_k(y)$ = -∞ adalah nilai dari persoalan *knapsack* untuk kapasitas negative. Solusi optimum dari persoalan *integer knapsack* adalah $f_n(M)$.

2.6 Penerapan Algoritma pada Persoalan *Integer knapsack*

Untuk penyelesaian optimasi pada persoalan *Integer knapsack* menggunakan algoritma *Greedy* dan algoritma *Dinamic Programming* diperlukan beberapa tahapan. Terlebih dahulu dijelaskan penyelesaian secara manual untuk mengetahui cara kerja masing-masing algoritma dengan menggunakan sampel data yang kecil dengan 4 (empat) jenis barang yang diambil dari sampel data acak. Pengambilan sampel data yang kecil tersebut bertujuan untuk mempermudah dan mempercepat dalam melakukan perhitungan.

Diberikan data dengan persoalan memuatkan empat macam objek ke dalam *M*. Tiap macam barang memiliki berat w_i dan akan memberikan keuntungan p_i, dimana i= (1, 2, 3,4). Kapasitas muat *M* adalah 5 (dalam satuan berat). Tentukan benda-benda apa saja yang dimasukkan ke dalam *M* sehingga memberikan keuntungan penjualan yang maksimum, namun dengan total berat barang tidak boleh melebihi *M*.

Tabel 2.1 Data bobot dan keuntungan barang (n =4)

Barang ke-*i*	w_i	p_i
1	2	80
2	1	60
3	1	75
4	2	40

Kapasitas media (*M* = 5)

2.6.1 Penyelesaian masalah *integer knapsack* dengan menggunakan algoritma *Greedy*

Dari data pada tabel 2.1, maka penyelesaiannya adalah sebagai berikut:

a. Greedy by profit

Langkah pertama kali yang dilakukan adalah mengurutkan secara menurun objek-objek berdasarkan keuntungan yang terbesar. Kemudian baru diambil satu-persatu objek yang dapat ditampung oleh media sampai media penuh atau sudah tidak ada objek lagi yang bisa dimasukkan.

Tabel 2.2 Perhitungan *greedy by profit*

Barang ke-*i*	w_i	p_i	Status
1	2	80	Diambil (1)
3	1	75	Diambil (1)
2	1	60	Diambil (1)
4	2	40	Tidak (0)

b. Greedy by weight

Langkah pertama yang harus dilakukan adalah mengurutkan objek-objek berdasarkan bobot (*weight*) teringan. Kemudian baru diambil satu-persatu objek yang dapat ditampung oleh media sampai media penuh atau sudah tidak ada lagi yang bisa dimasukkan.

Tabel 2.3 Perhitungan *greedy by weight*

Barang ke-*i*	w_i	p_i	Status
3	1	75	Diambil (1)
2	1	60	Diambil (1)
1	2	80	Diambil (1)
4	2	40	Tidak (0)

c. Greedy by density

Langkah pertama yang dilakukan adalah mencari nilai per unit (*density*) dari tiap-tiap objek. Kemudian objek-objek tersebut diurutkan berdasarkan density (p_i/w_i) yang terbesar. Kemudian baru diambil satu-persatu objek yang dapat ditampung oleh media sampai media penuh atau sudah tidak ada objek lagi yang bisa dimasukkan.

Tabel 2.4 Perhitungan *greedy by density*

Barang ke-*i*	w_i	p_i	p_i/w_i	Status
3	1	75	75	Diambil (1)
2	1	60	60	Diambil (1)
1	2	80	40	Diambil (1)
4	2	40	20	Tidak (0)

Tabel 2.5 Hasil ringkasan perhitungan algoritma *greedy*

Properti objek				*Greedy by*			Solusi optimal
i	w_i	p_i	p_i/w_i	*Weight*	*Profit*	*Density*	
1	2	80	40	1	1	1	1
2	1	60	60	1	1	1	1
3	1	75	75	1	1	1	1
4	2	40	20	0	0	0	0
Total bobot				4	4	4	4
Total keuntungan				215	215	215	215

Berdasarkan Tabel 2.5 dapat dijelaskan bahwa algoritma *greedy* dengan ketiga strategi pemilihan objek memberikan solusi optimal. Solusi optimal permasalahan ini adalah X = (1, 1, 1, 0) yang artinya barang ke-1, 2, dan 3 di ambil dengan total keuntungan adalah 215.

2.6.2 Penyelesaian masalah *integer knapsack* dengan menggunakan algoritma *Dynamic Programming*

Program dinamik (*dynamic programming*) merupakan metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Dari data pada tabel 2.1, dapat dicari solusi dengan bertahap.

- Tahap 1

$$f_1(y) = max\{f_0(y), p_1 + f_0(y - w_1)\}$$
$$= max\{f_0(y), 80 + f_0(y - 2)\}$$

Tabel 2.6 Tahap 1 pencarian solusi *integer knapsack* dengan *Dynamic programming*

y	[illegible]	[illegible] $80 + f_0(y-2)$	[illegible]	Solusi Optimum (x_1*, x_2*, x_3*, x_4*)
0	0	-∞	0	(0, 0, 0, 0)
1	0	-∞	0	(0, 0, 0, 0)
2	0	80	80	(1, 0, 0, 0)
3	0	80	80	(1, 0, 0, 0)
4	0	80	80	(1, 0, 0, 0)
5	0	80	80	(1, 0, 0, 0)

- Tahap 2

$$f_2(y) = max\{f_1(y), p_2 + f_1(y - w_2)\}$$
$$= max\{f_1(y), 60 + f_1(y - 1)\}$$

Tabel 2.7 Tahap 2 pencarian solusi *integer knapsack* dengan *Dynamic programming*

y	[illegible]	[illegible] $60 + f_1(y-1)$	[illegible]	Solusi Optimum (x_1*, x_2*, x_3*, x_4*)
0	0	60 + (-∞) = -∞	0	(0, 0, 0, 0)
1	0	60 + 0 = 60	60	(0, 1, 0, 0)
2	80	60 + 0 = 60	80	(1, 0, 0, 0)
3	80	60 + 80 = 140	140	(1, 1, 0, 0)
4	80	60 + 80 = 140	140	(1, 1, 0, 0)
5	80	60 + 80 = 140	140	(1, 1, 0, 0)

- Tahap 3

$$f_3(y) = max\{f_2(y), p_3 + f_2(y - w_3)\}$$
$$= max\{f_2(y), 75 + f_2(y - 1)\}$$

Tabel 2.8 Tahap 3 pencarian solusi *integer knapsack* dengan *Dynamic programming*

y				Solusi Optimum
		75 +		(x_1*, x_2*, x_3*, x_4*)
0	0	75 + (-∞) = -∞	0	(0, 0, 0, 0)
1	60	75 + 0 = 75	75	(0, 0, 1, 0)
2	80	75 + 60 = 135	135	(0, 1, 1, 0)
3	140	75 + 80 = 155	155	(1, 0, 1, 0)
4	140	75 + 140 = 215	215	(1, 1, 1, 0)
5	140	75 + 140 = 215	215	(1, 1, 1, 0)

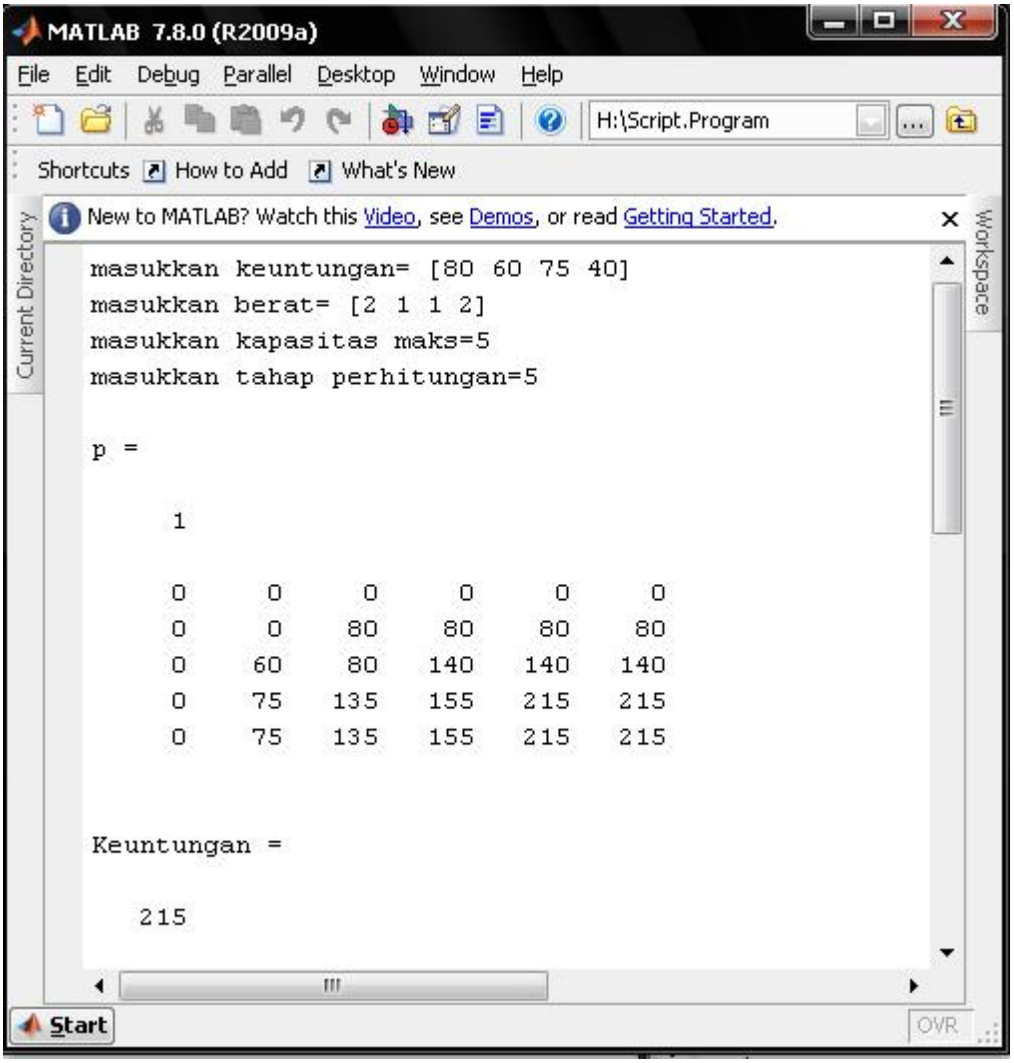
- Tahap 4

$$f_4(y) = max\{f_3(y), p_4 + f_3(y - w_4)\}$$
$$= max\{f_3(y), 40 + f_3(y - 2)\}$$

Tabel 2.9 Tahap 4 pencarian solusi *integer knapsack* dengan *Dynamic programming*

y				Solusi Optimum
		40 +		(x_1*, x_2*, x_3*, x_4*)
0	0	40 + (-∞) = -∞	0	(0, 0, 0, 0)
1	75	40 + (-∞) = -∞	75	(0, 0, 1, 0)
2	135	40 + 0 = 40	135	(0, 1, 1, 0)
3	155	40 + 75 = 115	155	(1, 0, 1, 0)
4	215	40 + 135 = 175	215	(1, 1, 1, 0)
5	215	40 + 155 = 195	215	(1, 1, 1, 0)

Hasil perhitungan dari tabel tahap 1 sampai tahap 4 diatas sesuai dengan penyelesaian algoritma *dynamic programming* menggunakan komputer seperti tampak pada Gambar 2.3 berikut



Gambar 2.3 Output Perhitungan Masalah *Knapsack* dengan algoritma *Dynamic Programming*

Keterangan hasil output diatas dapat diringkas pada Tabel 2.10.

Tabel 2.10 Ringkasan Hasil Perhitungan Algoritma Premrograman Dinamik

Janis Barang	Kapasitas Angkut					
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	80	80	80	80
2	0	60	80	140	140	140
3	0	75	135	155	215	215
4	0	75	135	155	215	215

Tabel di atas berisi kemungkinan-kemungkinan dari keuntungan yang diperoleh untuk masing-masing kapasitas angkut. Dari tabel di atas ditentukan keuntungan optimalnya adalah z(4, 5) = 215. penentuan ini berdasarkan pada barang terakhir dengan kapasitas angkut yang maksimal.

Dari langkah-langkah menentukan barang yang masuk solusi optimal di atas diperoleh nilai total keuntungan maksimal sebesar 215 pada z(4, 5) lalu dibandingkan dengan nilai 215 pada z(3, 5), karena nilai keduanya sama maka barang ke-4 tidak diangkut.untuk perhitungan barang ke-3 diperhatikan nilai 215 pada z(3,5) lalu bandingkan dengan nilai yan berada di atasnya yaitu nilai 140 pada z(2, 5).karena nilai keduanya tidak sama, maka menurut algoritma (langkah 4) barang ke-3 menjadi baran yang diangkut. Setelah itu kurangkan nilai kapasitas ankut 5 dengan berat angkut barang ke-3. untuk perhitungan barang ke-2 diperhatikan nilai 140 pada z(2, 4), nilai 140 pada z(2, 4) ini lalu dibandingkan dengan nilai yang berada di atasnya yaitu nilai 80 pada z(1, 4), karena nilai keduanya tidak sama, maka barang ke-2 menjadi barang yang diangkut. Setelah itu kurangkan nilai kapasitas angkut 4 dengan berat angkut barang ke-2. untuk perhitungan barang ke-1 diperhatikan nilai 80 pada z(1, 3), nilai 80 pada z(1, 3) ini lalu dibandingkan dengan nilai yang berada di atasnya yaitu nilai 0 pada z(0, 3), karena nilai keduanya tidak sama, maka barang ke-1 menjadi barang yang diangkut.

Jadi solusi optimal permasalahan dengan menggunakan algoritma *Dynamic programming* adalah X = (1, 1, 1, 0) yang artinya barang ke-1, 2, dan 3 di ambil dengan total keuntungan adalah 215.

BAB 3. METODE PENELITIAN

3.1 Data Penelitian

Data yang digunakan dalam tugas akhir ini adalah data sejumlah barang yang akan dibeli oleh UD. BINTANG TANI. Unit dagang tersebut bergerak di bidang perdagangan jenis obat-obatan tanaman yang terletak di Jl. Yos. Sudarso kecamatan Semboro kabupaten Jember. Unit dagang ini menjual berbagai jenis barang di antaranya macam-macam pupuk dan obat-obatan untuk tanaman. Pengambilan data dilakukan dengan metode wawancara langsung kepada pihak pemilik UD. BINTANG TANI. Data yang di ambil adalah data mentah yang berupa harga beli barang, harga jual barang, banyaknya barang yang dibeli, serta kapasitas angkut maksimumnya. Adapun kapasias angkut maksimumnya sebesar (M) = 500 kg. Untuk menerapkan data ini pada permasalahan *knapsack*, peneliti perlu melakukan pengidentifikasian dari data mentah yang didapat guna mencari keuntungan (p_i) dan berat (w_i) dari masing-masing barang.

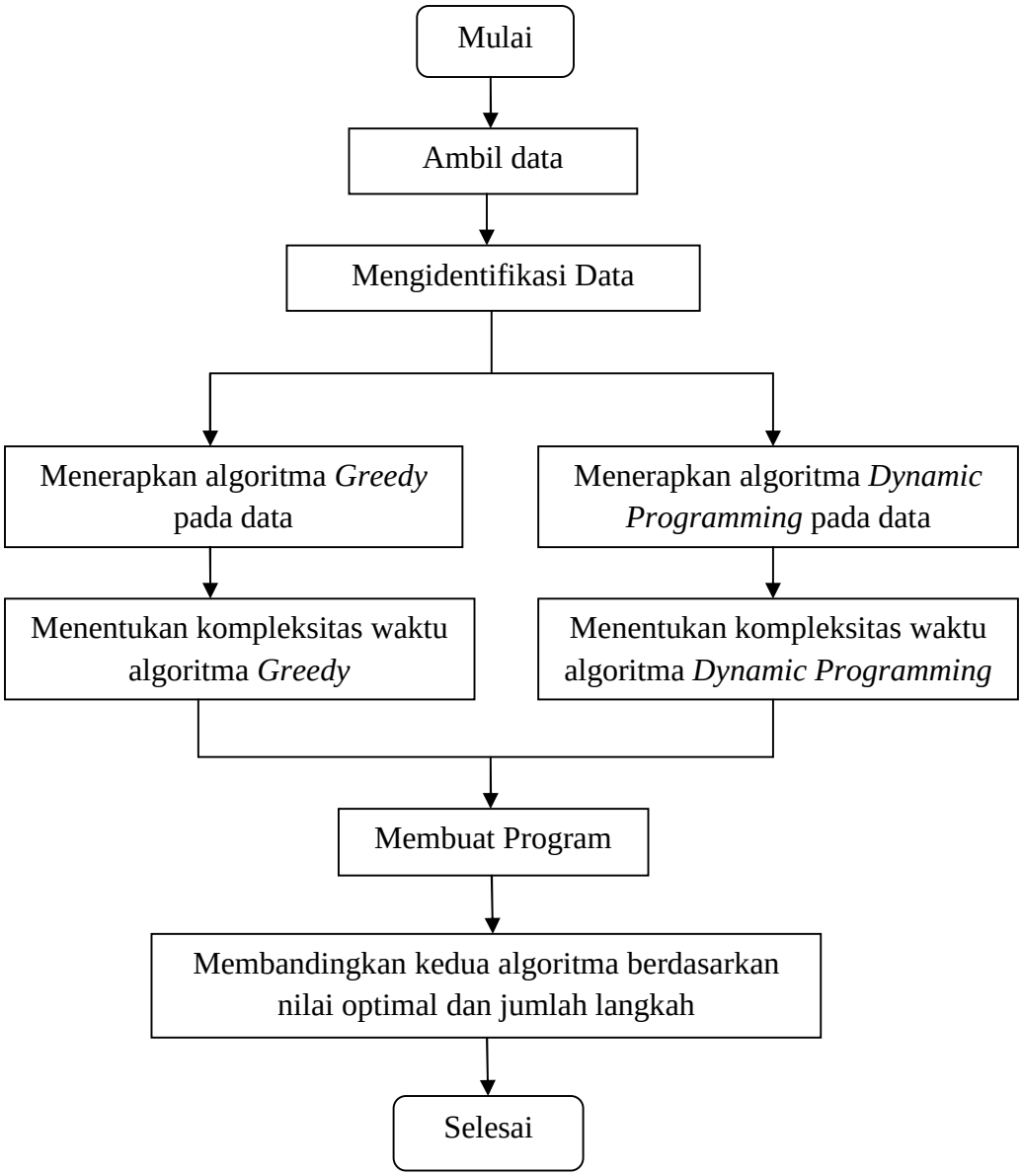
Data mentah yang di ambil dari UD. BINTANG TANI berupa data harga beli barang, harga jual barang, dan banyaknya barang yang dibeli serta kapasitas angkut maksimum. Data tersebut dapat disajikan dengan Tabel 3.1 berikut.

Tabel 3.1. Data barang di UD. BINTANG TANI

No	NAMA BARANG (*i*)	BANYAK BARANG	HARGA BELI	HARGA JUAL
1	PRIMAFUR 2kg	10	16500	24000
2	CALCIUM KUPU2 1kg	10	12500	37500
3	CALCIUM KUMBANG 1kg	10	8500	43500
4	CRASH 1ltr	10	39500	54500
5	MKP KRISTA 1kg	20	22500	72500
6	NITRAFOS 1kg	10	22500	97500
7	NITRABOR 1kg	3	14000	17000
8	CALCINIT 1kg	10	12500	37500
9	KALI CHILI YARA 2kg	9	19500	42000
10	PLASTIK MULSA 9kg	5	24000	34000
11	PLASTIK MULSA 18kg	5	24000	34000
12	PLASTIK 2KELINCI 5kg	5	95000	105000
13	PLASTIK 2KELINCI 10kg	4	178000	194000
14	AMEGRASS 5kg	4	317500	407500
15	AMEGRASS 20kg	2	63000	67000
16	SIDAMIN 1kg	10	46000	86000
17	SIDAFOS 1kg	10	31500	66500
18	P21 1kg	15	56000	71000
19	BISI 2 1kg	3	42000	43500
20	CIHERANG SS 10kg	3	73000	79000
21	INTANI 10kg	2	73000	87000
22	NPK MUTIARA 5kg	4	339000	383000
23	KCL SHS 5kg	4	255000	275000
24	ZA 5kg	4	70000	90000
25	UREA D/B 5kg	4	89000	93000
26	PHONSKA 5kg	4	115000	119000
27	NPK TAWON 5kg	4	325000	345000
28	KNO.3 MERAH 2kg	5	18900	24400
29	KNO.3 PUTIH 2kg	5	27800	28800
30	REGENT.03G 1kg	10	16250	23750
31	ALPHADIN 1kg	10	12500	17500
32	FURADAN 3G 2kg	5	19500	22000
33	ZK 1kg	5	16000	26000
34	MKP KNO.3 SAPRTAN 3kg	5	22800	33800
35	FERRTERA 2kg	5	77500	90000
36	PRIMA STICK 1kg	5	9000	24000
37	PRIMA STICK 5kg	5	35000	60000

3.2 Langkah-langkah Penyelesaian

Secara sistematik, skema langkah-langkah yang dilakukan dalam penelitian tentang permasalahan *integer knapsack* di UD. BINTANG TANI tampak pada Gambar 3.1.



Gambar 3.1 Skema langkah-langkah penyelesaian

Dari skema pada gambar 3.1, langkah-langkah penelitian dapat dijelaskan sebagai berikut.

a. Mengambil data sekunder di UD. BINTANG TANI.
b. Mengidentifikasi data untuk mencari keuntungan (p_i) dan berat (w_i) pada masing-masing barang.
    1) Untuk perhitungan mencari nilai keuntungan (p_i) dari masing-masing barang diperoleh dari selisih harga beli dan harga jual yang telah ditetapkan.
    2) Untuk perhitungan mencari berat (w_i) dari masing-masing barang diperoleh dengan cara mengalikan bobot dari tiap-tiap barang dengan banyaknya barang.
c. Menerapkan algoritma *Greedy* dan *Dynamic Programming* pada data yang sudah diidentifikasi.
d. Menentukan kompleksitas waktu dari algoritma *Greedy* dan algoritma *Dynamic Programming*.
e. Membuat program sesuai algoritma yang digunakan pada langkah c menggunakan bahasa pemrograman MATLAB R2009a.
f. Membandingkan kedua algoritma yakni.
    1) Membandingkan hasil optimum dari perhitungan algoritma *Greedy* dan *Dynamic Programming.*
    2) Membandingkan efektifitas algoritma dengan menghitung kompleksitas waktu (T(n)) dari kedua algoritma.

BAB 4. PEMBAHASAN

Pada bab ini akan dibahas masalah *integer knapsack* pada perusahaan UD. BINTANG TANI dengan menggunakan algoritma *Greedy* dan algoritma *Dynamic programming*. Alasan mengapa mengambil studi kasus industri di UD. BINTANG TANI karena dalam kasus pengangkutan dengan batas kapasitas maksimum terkadang kurang efektif misalnya ada beberapa barang yang seharusnya diangkut atau tidak diangkut tapi justru sebaliknya, dengan adanya algoritma penyelesaian pada masalah *integer knapsack* pada UD. BINTANG TANI ini diharapkan dapat membantu dalam proses pemilihan barang yang tepat harus diangkut atau tidak. Dengan adanya proses pemilihan barang yang tepat maka dapat membantu mendapatkan keuntungan maksimum.

Data pada Tabel 3.1 merupakan data mentah yang diambil dari UD. BINTANG TANI. Data tersebut berupa data harga beli barang, harga jual barang, dan bobot barang yang dibeli oleh UD. BINTANG TANI. Kapasitas angkut maksimum sebesar (M) = 500 kg. Untuk menerapkan data ini, maka penulis perlu melakukan identifikasi dari data Tabel 3.1 untuk mencari bobot (w_i) dan keuntungan (p_i) dari masing-masing barang. Untuk perhitungan mencari nilai keuntungan (p_i), peneliti melakukan pengidentifikasian dengan cara menghitung keuntungan masing-masing barang yang diperoleh dari selisih harga beli dan harga jual yang ditetapkan. Sedangkan untuk mencari bobot total (w_i), maka peneliti melakukan pengidentifikasian dengan cara mengalikan bobot dari tiap barang dengan banyaknya barang. Berikut ini merupakan data hasil identifikasi dari Tabel 3.1.

Tabel 4.1 Data identifikasi dari Tabel 3.1

No	NAMA BARANG (*i*)	BERAT (w_i)	KEUNTUNGAN (p_i)
1	PRIMAFUR 2kg	20	7500
2	CALCIUM KUPU2 1kg	10	25000
3	CALCIUM KUMBANG 1kg	10	35000
4	CRASH 1ltr	10	15000
5	MKP KRISTA 1kg	20	50000
6	NITRAFOS 1kg	10	75000
7	NITRABOR 1kg	3	3000
8	CALCINIT 1kg	10	25000
9	KALI CHILI YARA 2kg	18	22500
10	PLASTIK MULSA 9kg	45	10000
11	PLASTIK MULSA 18kg	90	10000
12	PLASTIK 2KELINCI 5kg	25	10000
13	PLASTIK 2KELINCI 10kg	40	16000
14	AMEGRASS 5kg	20	90000
15	AMEGRASS 20kg	40	4000
16	SIDAMIN 1kg	10	40000
17	SIDAFOS 1kg	10	35000
18	P21 1kg	15	15000
19	BISI 2 1kg	3	1500
20	CIHERANG SS 10kg	30	6000
21	INTANI 10kg	20	14000
22	NPK MUTIARA 5kg	20	44000
23	KCL SHS 5kg	20	20000
24	ZA 5kg	20	20000
25	UREA D/B 5kg	20	4000
26	PHONSKA 5kg	20	4000
27	NPK TAWON 5kg	20	20000
28	KNO.3 MERAH 2kg	10	5500
29	KNO.3 PUTIH 2kg	10	1000
30	REGENT.03G 1kg	10	7500
31	ALPHADIN 1kg	10	5000
32	FURADAN 3G 2kg	10	2500
33	ZK 1kg	5	10000
34	MKP KNO.3 SAPRTAN 3kg	15	11000
35	FERRTERA 2kg	10	12500
36	PRIMA STICK 1kg	5	15000
37	PRIMA STICK 5kg	25	25000

4.1 Penyelesaian Permasalahan *Knapsack* dengan Algoritma *Greedy*

Data pengamatan pada tabel 4.1 diolah menggunakan algoritma *Greedy*. Terdapat 3 strategi *greedy* pada perhitungan algoritma *Greedy*, diantaranya adalah strategi *greedy by weight, greedy by profit,* dan *greedy by density*. Untuk langkah penyelesaiannya adalah sebagai berikut.

4.1.1 Perhitungan strategi *Greedy by Weight*

Langkah pertama yang harus dilakukan adalah mengurutkan objek-objek berdasarkan bobot (*weight*) teringan. Kemudian baru diambil satu-persatu objek yang dapat ditampung oleh media sampai media penuh atau sudah tidak ada lagi yang bisa dimasukkan. Berikut disajikan perhitungan *greedy by weight*.

Tabel 4.2 Perhitungan *greedy by weight*

Barang ke-*i*	w_i.s	p_i.s	Status (s)
7	3	3000	1
19	3	1500	1
33	5	10000	1
36	5	15000	1
2	10	25000	1
3	10	35000	1
4	10	15000	1
6	10	75000	1
8	10	25000	1
16	10	40000	1
17	10	35000	1
28	10	5500	1
29	10	1000	1
30	10	7500	1
31	10	5000	1
32	10	2500	1
35	10	12500	1
18	15	15000	1
34	15	11000	1
9	18	22500	1
1	20	7500	1
5	20	50000	1
14	20	90000	1
21	20	14000	1
22	20	44000	1

Tabel 4.2 *Lanjutan*

Barang ke-*i*	w_i.s	p_i.s	Status (s)
23	20	20000	1
24	20	20000	1
25	20	4000	1
26	20	4000	1
27	20	20000	1
12	12	10000	1
37	25	25000	1
20	30	6000	1
13	0	0	0
15	0	0	0
10	0	0	0
11	0	0	0
Total	474	676500	

Keterangan :

1 = dipilih

0 = tidak dipilih

Menurut perhitungan dari Tabel 4.2 diatas dijelaskan bahwa pada strategi *Greedy* by *weight* diperoleh hasil keuntungan maksimum sebesar Rp 676.500,- dengan bobot 474 kg, dan pilihan barang yang diangkut adalah barang ke 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37.

4.1.2 Perhitungan strategi *Greedy by Profit*

Langkah pertama kali yang dilakukan adalah mengurutkan secara menurun objek-objek berdasarkan keuntungan yang terbesar. Kemudian baru diambil satu-persatu objek yang dapat ditampung oleh media sampai media penuh atau sudah tidak ada objek lagi yang bisa dimasukkan. Berikut disajikan perhitungan *greedy by Profit*.

Tabel 4.3 Perhitungan *greedy by Profit*

Barang ke-*i*	w_i.s	p_i.s	Status (s)
14	20	90000	1
6	10	75000	1
5	20	50000	1
22	20	44000	1
16	10	40000	1
3	10	35000	1
17	10	35000	1
2	10	25000	1
8	10	25000	1
37	25	25000	1
9	18	22500	1
23	20	20000	1
24	20	20000	1
27	20	20000	1
13	40	16000	1
4	10	15000	1
18	15	15000	1
36	5	15000	1
21	20	14000	1
35	10	12500	1
34	15	11000	1
10	45	10000	1
11	90	10000	1
12	25	10000	1
33	0	0	0
1	0	0	0
30	0	0	0
20	0	0	0
28	0	0	0
31	0	0	0
15	0	0	0
25	0	0	0
26	0	0	0
7	0	0	0
32	0	0	0
19	0	0	0
29	0	0	0
Total	498	655500	

Keterangan :

1 = dipilih

0 = tidak dipilih

Dari perhitungan Tabel 4.3 diatas dijelaskan bahwa pada strategi *Greedy by profit* diperoleh hasil keuntungan maksimum sebesar Rp 655.500,- dengan bobot 498 kg, dan pilihan barang yang diangkut adalah barang ke 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 21, 22, 23, 24, 27, 34, 35, 36, 37.

4.1.3 Perhitungan strategi *Greedy by Density*

Langkah pertama yang dilakukan adalah mencari nilai per unit (*density*) dari tiap-tiap objek. Kemudian objek-objek tersebut diurutkan berdasarkan density (p_i/w_i) yang terbesar. Kemudian diambil satu-persatu objek yang dapat ditampung oleh media sampai media penuh atau sudah tidak ada objek lagi yang bisa dimasukkan. Berikut disajikan perhitungan *greedy by density*.

Tabel 4.4 Perhitungan *greedy by Density*

Barang ke-*i*	p_i/w_i	w_i.s	p_i.s	Status (s)
6	375	10	75000	1
14	2500	20	90000	1
16	3500	10	40000	1
3	1500	10	25000	1
17	2500	10	35000	1
36	7500	5	15000	1
2	1000	10	25000	1
5	2500	20	50000	1
8	1250	10	25000	1
22	222.22	20	44000	1
33	111.11	5	10000	1
4	400	10	15000	1
9	400	18	22500	1
35	4500	10	12500	1
7	100	3	3000	1
18	4000	15	15000	1
23	3500	20	20000	1
24	1000	20	20000	1
27	500	20	20000	1
37	200	25	25000	1
30	700	10	7500	1
34	2200	15	11000	1
21	1000	20	14000	1
28	1000	10	5500	1
19	200	3	1500	1

Tabel 4.4 *Lanjutan*

Barang ke-*i*	p_i/w_i	w_i.s	p_i.s	Status (s)
31	200	10	5000	1
12	1000	25	10000	1
13	550	40	16000	1
1	100	20	7500	1
32	750	10	2500	1
10	500	0	0	0
20	250	0	0	0
25	2000	0	0	0
26	733.33	0	0	0
11	1250	0	0	0
15	3000	0	0	0
29	1000	0	0	0
Total		479	687500	

Keterangan :

1 = dipilih

0 = tidak dipilih

Dari perhitungan Tabel 4.4 diatas dijelaskan bahwa pada strategi *Greedy* by density diperoleh hasil maksimum sebesar Rp 687.500,- dengan bobot 479 kg, dan pilihan barang yang diangkut adalah barang ke 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37.

Setelah menghitung dengan ketiga starategi algoritma *Greedy* diperoleh hasil ringakasan dari ketiga strategi tersebut. Untuk hasil ringkasan perhitungan dengan ketiga strategi algoritma *Greedy* dapat disajikan pada Tabel 4.5 berikut.

Tabel 4.5 Ringkasan perhitungan dengan ketiga strategi algoritma *Greedy*

Barang		*Status Greedy by*			Hasil optimal *Greedy by*		
i	p_i	*Weight*	*Profi*	*Density*	*Weight.* p_i	*Profit.* p_i	*Density.* p_i
1	7500	1	0	1	7500	0	7500
2	25000	1	1	1	25000	25000	25000
3	35000	1	1	1	35000	35000	35000
4	15000	1	1	1	15000	15000	15000
5	50000	1	1	1	50000	50000	50000
6	75000	1	1	1	75000	75000	75000
7	3000	1	0	1	3000	0	3000
8	25000	1	1	1	25000	25000	25000
9	22500	1	1	1	22500	22500	22500
10	10000	0	1	1	0	10000	10000
11	10000	0	1	0	0	10000	0
12	10000	1	1	1	10000	10000	10000
13	16000	0	1	1	0	16000	16000
14	90000	1	1	1	90000	90000	90000
15	4000	0	0	0	0	0	0
16	40000	1	1	1	40000	40000	40000
17	35000	1	1	1	35000	35000	35000
18	15000	1	1	1	15000	15000	15000
19	1500	1	0	1	1500	0	1500
20	6000	1	0	0	6000	0	0
21	14000	1	1	1	14000	14000	14000
22	44000	1	1	1	44000	44000	44000
23	20000	1	1	1	20000	20000	20000
24	20000	1	1	1	20000	20000	20000
25	4000	1	0	0	4000	0	0
26	4000	1	0	0	4000	0	0
27	20000	1	1	1	20000	20000	20000
28	5500	1	0	1	5500	0	5500
29	1000	1	0	0	1000	0	0
30	7500	1	0	1	7500	0	7500
31	5000	1	0	1	5000	0	5000
32	2500	1	0	1	2500	0	2500
33	10000	1	0	1	10000	0	10000
34	11000	1	1	1	11000	11000	11000
35	12500	1	1	1	12500	12500	12500
36	15000	1	1	1	15000	15000	15000
37	25000	1	1	1	25000	25000	25000
		Total			676500	655500	687500

Keterangan :

1 = dipilih

0 = tidak dipilih

Dari hasil akhir perhitungan pada Tabel 4.5 dapat dijelaskan bahwa ada beberapa strategi *Greedy* yang dapat digunakan dalam permasalahan *knapsack* diantaranya adalah strategi *Greedy* by *weight*, *Greedy* by *profit*, dan *Greedy* by *density*. Pada strategi *Greedy* by *weight* diperoleh hasil maksimum sebesar Rp 676.500,- dengan bobot 474 kg, dan pilihan barang yang diangkut adalah barang ke 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37. Pada strategi *Greedy by profit* diperoleh hasil maksimum sebesar Rp 655.500,- dengan bobot 498 kg, dan pilihan barang yang diangkut adalah barang ke 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 21, 22, 23, 24, 27, 34, 35, 36, 37. Pada strategi *Greedy by density* diperoleh hasil maksimum sebesar Rp 687.500,- dengan bobot 479 kg, dan pilihan barang yang diangkut adalah barang ke 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37. Dari ketiga strategi tersebut terdapat hasil yang paling maksimal yaitu pada strategi *greedy by density*.

Untuk hasil optimal pilihan barang dengan menggunakan algoritma *Greedy* dapat disajikan pada Tabel 4.6 berikut.

Tabel 4.6 Hasil pilihan barang algoritma *Greedy*

No	NAMA BARANG (*i*)	BERAT (w_i)	KEUNTUNGAN (p_i)
1	PRIMAFUR 2kg	20	7500
2	CALCIUM KUPU2 1kg	10	25000
3	CALCIUM KUMBANG 1kg	10	35000
4	CRASH 1ltr	10	15000
5	MKP KRISTA 1kg	20	50000
6	NITRAFOS 1kg	10	75000
7	NITRABOR 1kg	3	3000
8	CALCINIT 1kg	10	25000
9	KALI CHILI YARA 2kg	18	22500
10	PLASTIK MULSA 9kg	45	10000
12	PLASTIK 2KELINCI 5kg	25	10000
13	PLASTIK 2KELINCI 10kg	40	16000
14	AMEGRASS 5kg	20	90000

Tabel 4.6 *Lanjutan*

No	NAMA BARANG (*i*)	BERAT (w_i)	KEUNTUNGAN (p_i)
16	SIDAMIN 1kg	10	40000
17	SIDAFOS 1kg	10	35000
18	P21 1kg	15	15000
19	BISI 2 1kg	3	1500
21	INTANI 10kg	20	14000
22	NPK MUTIARA 5kg	20	44000
23	KCL SHS 5kg	20	20000
24	ZA 5kg	20	20000
27	NPK TAWON 5kg	20	20000
28	KNO.3 MERAH 2kg	10	5500
30	REGENT.03G 1kg	10	7500
31	ALPHADIN 1kg	10	5000
32	FURADAN 3G 2kg	10	2500
33	ZK 1kg	5	10000
34	MKP KNO.3 SAPRTAN 3kg	15	11000
35	FERRTERA 2kg	10	12500
36	PRIMA STICK 1kg	5	15000
37	PRIMA STICK 5kg	25	25000
	JUMLAH	479	687500

Menurut Tabel 4.6 diatas dapat diketahui bahwa hasil maksimum pada algoritma *Greedy* terdapat pada strategi *greedy by density* dengan perolehan maksimum sebesar Rp 687.500,- dengan bobot 479 kg.

4.2 Penyelesaian Permasalahan *Knapsack* dengan Algoritma *Dynamic Programming*

Data pengamatan pada Tabel 4.1 diolah dengan menggunakan algoritma *dynamic programming*. Langkah-langkah penyelesaiannya adalah sebagai berikut.

a. Menentukan struktur dari masalah

Dari masalah diketahui n = 37 jenis barang dinotasikan i, i = 1, …, 37, $i \in N$ dengan berat barang i dinotasikan dengan w_i sehingga w_i= [20, 10, 10, 10, 20, 10, 3, 10, 18, 45, 90, 25, 40, 20, 40, 10, 10, 15, 3, 30, 20, 20, 20, 20, 20, 20, 20, 10, 10, 10, 10, 10, 5, 15, 10, 5, 25]. Kapasitas angkut dinotasikan dengan M = 500 kg, dimana $M \in N, w_i \leq M, i \in N$, dengan batasan $\sum_{i=1}^{n} w_i x_i \leq M$. Tujuan dari pemrograman dinamik yaitu mencari total keuntungan optimal (dinotasikan dengan Z) dari pengangkutan barang dimana total keuntungan optimal dipengaruhi oleh keuntungan dari barang yang dinotasikan dengan p_i= [7500, 25000, 35000, 15000, 50000, 75000, 3000, 25000, 22500, 10000, 10000, 10000, 16000, 90000, 4000, 40000, 35000, 15000, 1500, 6000, 14000, 44000, 20000, 20000, 4000, 4000, 20000, 5500, 1000, 7500, 5000, 2500, 10000, 11000, 12500, 15000, 25000], sehingga dapat diltulis $Z = \sum_{i=1}^{n} p_i x_i$.

Untuk memperoleh total keuntungan optimal dilakukan perhitungan dengan prosedur rekursif maju. Perhitungan dimulai dengan mencari nilai keuntungan barang ke-1 lalu dilanjutkan mencari nilai keuntungan barang ke-dua dan seterusnya sampai barang ke-n dimasukkan dalam matriks z dengan ukuran (n, m).

b. Menentukan persamaan rekursif

Dari persamaan Z diatas dapat ditulis suatu persamaan rekursif untuk mencari semua kemungkinan pengangkutan jenis barang ke-i sehingga diperoleh nilai total keuntungan optimal untk masalah. Nilai keuntungan yang maksimal inilah yang dimasukkan ke dalam z(i, j), sehingga dapat di tulis persamaan rekursifnya

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

c. Menghitung nilai dari solusi optimal

Untuk menghitung nilai dari solusi optimal pada masalah angkutan ini prosedur perhitungan yang digunakan adalah prosedur maju. Perhitungan dimulai dari mencari nilai keuntungan untuk barang ke-1 lalu dilanjutkan mencari nillai keuntungan barang ke-2 sampai tahap ke-37.

d. Menentukan keputusan optimal

Dari langkah c diperoleh matriks yang berisi keputusan-keputusan optimal yaitu matris z. Keuntungan optimal dari masalah di atas adalah nilai dari z(n, m) yang diilustrasikan seperti pada pada Tabel 4.7.

Langkah a dan b pada penyelesaian di atas dilakukan untuk memperoleh bentuk dari permasalahan *Dynamic Programming* dan menentukan persamaan rekursif yang digunakan. Pada langkah c yaitu menghitung nilai solusi optimal tiap tahap. Langkah d adalah mencari nilai solusi optimal atau jenis barang yang diangkut. Untuk hasil perhitungan algoritma *Dynamic Programming* pada data 4.1 dipergunakan program karena keterbatasan melakukan perhitungan manual untuk $n = 37$ dan kapasitas maksimum media pengangkut (M) = 500 kg.

Adapun langkah untuk menjalankan program algoritma *dynamic programming* adalah sebagai berikut:

1) Memasukkan jumlah data $n = 37$;
2) Memasukkan profit barang p_i = [7500, 25000, 35000, 15000, 50000, 75000, 3000, 25000, 22500, 10000, 10000, 10000, 16000, 90000, 4000, 40000, 35000, 15000, 1500, 6000, 14000, 44000, 20000, 20000, 4000, 4000, 20000, 5500, 1000, 7500, 5000, 2500, 10000, 11000, 12500, 15000, 25000];
3) input matrik berat barang w_i = [20, 10, 10, 10, 20, 10, 3, 10, 18, 45, 90, 25, 40, 20, 40, 10, 10, 15, 3, 30, 20, 20, 20, 20, 20, 20, 20, 10, 10, 10, 10, 10, 5, 15, 10, 5, 25];
4) input kapasitas maksimum M = 500.

Hasil ringkasan perhitungan penyelesaian algoritma *Dynamic Programming* menggunakan komputer dapat disajikan pada Tabel 4.7.

Untuk menghitung nilai solusi optimal permasalahan *knapsack* dengan menggunakan algoritma *Dynamic Programming* diperoleh dari output program. Output tersebut berisi kemungkinan-kemungkinan dari keuntungan yang diperoleh untuk masing-masing kapasitas angkut. Untuk mempermudah membahas output dari perhitungan algoritma *Dynamic Programming*, maka dibawah ini disajikan tabel ringkasan output perhitungannya.

Tabel 4.7 Ringkasan hasil output program algoritma *Dynamic Programming*

Janis Barang	Kapasitas Angkut									
	0	410	420	430	440	445	460	470	475	500
0										
⋮		⋮	⋮	⋮	⋮	⋮	⋮			
28	…	603000						⋮	⋮	
29	…	603000	605000							⋮
30	…	…	610500	612500						
31	…	…	…	615500	617500					
32	…	…	…	…	618000	618000				
33	…	…	…	…	…	628000	63000			
34	…	…	…	…	…	…	639000	641000		
35	…	…	…	…	…	…	…	651500	651500	
36	…	…	…	…	…	…	…	…	666500	670500
37	…	…	…	…	…	…	…	…	…	691500

Dari hasil Tabel 4.7 tersebut ditentukan keuntungan optimal adalah z(37, 500) = 691500. Penentuan ini berdasarkan pada barang terakhir dengan kapasitas angkut yang maksimal.

Untuk langkah-langkah menentukan barang yang dipilih solusi optimal pada data diperoleh nilai total keuntungan maksimum pada persoalan data 4.1 sebesar 691500 pada z(37,500). Lalu dibandingkan dengan nilai 670500 pada z(36,500). Karena nilainya tidak sama, maka menurut algoritma langkah ke-4 barang ke-37 menjadi barang yang diangkut. Setelah itu kurangkan nilai kapasitas angkut 500 dengan bobot barang ke-37, sehingga untuk perhitungan barang ke-36 diperhatikan nilai 666500 pada z(36,475). Nilai 666500 pada z(36,475) ini lalu dibandingkan

dengan nilai yang berada diatasnya yaitu nilai 651500 pada z(35,475). Karena nilai keduanya tidak sama maka barang ke-36 diangkut. Lalu kapasitas angkut sebesar 475 dikurangi dengan bobot barang ke-36. Untuk perhitungan barang ke-35 diperhatikan nilai 651500 pada z(35,470), nilai ini dibandingkan dengan nilai yang berada diatasnya yaitu nilai 641000 pada z(34,470). Karena nilainya tidak sama maka barang ke-35 menjadi barang yang diangkut. Lalu kapasitas angkut sebesar 470 dikurangi dengan bobot barang ke-35. Untuk perhitungan barang ke-34 diperhatikan nilai 639000 pada z(34,460), nilai ini dibandingkan dengan nilai yang berada diatasnya yaitu nilai 630000 pada z(33,460). Karena nilainya tidak sama maka barang ke-34 menjadi barang yang diangkut. Lalu kapasitas angkut sebesar 460 dikurangi dengan bobot barang ke-34. Untuk perhitungan barang ke-33 diperhatikan nilai 628000 pada z(33,445), nilai ini dibandingkan dengan nilai yanng berada diatasnya yaitu nilai 618000 pada z(32,445). Karena nilainya tidak sama, maka barang ke-33 menjadi barang yang diangkut. Lalu kapasitas angkut sebesar 445 dikurangi dengan bobot barang ke-33. Untuk perhitungan barang ke-32 diperhatikan nilai 618000 pada z(32,440), nilai ini dibandingkan dengan nilai yanng berada diatasnya yaitu nilai 617500 pada z(31,440). Karena nilainya tidak sama, maka barang ke-32 menjadi barang yang diangkut. Lalu kapasitas angkut sebesar 440 dikurangi dengan bobot barang ke-32. Untuk perhitungan barang ke-31 diperhatikan nilai 615500 pada z(31,430), nilai ini dibandingkan dengan nilai yanng berada diatasnya yaitu nilai 612500 pada z(30,430). Karena nilainya tidak sama, maka barang ke-31 menjadi barang yang diangkut. Untuk perhitungan status pemilihan barang ke-28 sampai barang ke-1 perhitungannya sama seperti sebelumnya.

Selanjutnya untuk mempercepat mencari perhitungan barang yang diangkut, peneliti membuat program sesuai dengan input data yang diinginkan seperti yang tertera pada sub bab 4.4.

Perolehan hasil optimal algoritma *Dynamic Programming* dapat dilihat pada Tabel 4.8.

Tabel 4.8 Hasil Pilihan barang dengan algoritma *Dynamic Programming*

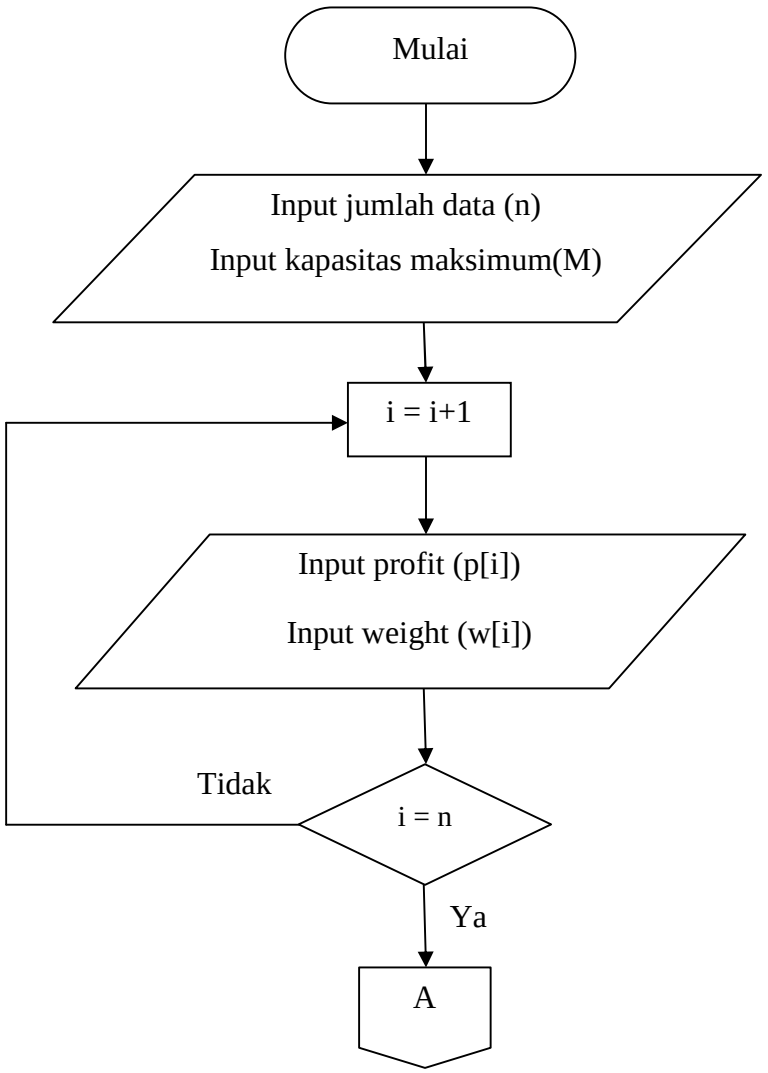
No	NAMA BARANG (*i*)	BERAT (w_i)	KEUNTUNGAN (p_i)
1	PRIMAFUR 2kg	20	7500
2	CALCIUM KUPU2 1kg	10	25000
3	CALCIUM KUMBANG 1kg	10	35000
4	CRASH 1ltr	10	15000
5	MKP KRISTA 1kg	20	50000
6	NITRAFOS 1kg	10	75000
7	NITRABOR 1kg	3	3000
8	CALCINIT 1kg	10	25000
9	KALI CHILI YARA 2kg	18	22500
10	PLASTIK MULSA 9kg	45	10000
12	PLASTIK 2KELINCI 5kg	25	10000
13	PLASTIK 2KELINCI 10kg	40	16000
14	AMEGRASS 5kg	20	90000
16	SIDAMIN 1kg	10	40000
17	SIDAFOS 1kg	10	35000
18	P21 1kg	15	15000
19	BISI 2 1kg	3	1500
21	INTANI 10kg	20	14000
22	NPK MUTIARA 5kg	20	44000
23	KCL SHS 5kg	20	20000
24	ZA 5kg	20	20000
25	UREA D/B 5kg	20	4000
27	NPK TAWON 5kg	20	20000
28	KNO.3 MERAH 2kg	10	5500
30	REGENT.03G 1kg	10	7500
31	ALPHADIN 1kg	10	5000
32	FURADAN 3G 2kg	10	2500
33	ZK 1kg	5	10000
34	MKP KNO.3 SAPRTAN 3kg	15	11000
35	FERRTERA 2kg	10	12500
36	PRIMA STICK 1kg	5	15000
37	PRIMA STICK 5kg	25	25000
	JUMLAH	499	691500

Dari keterangan diatas didapat hasil maksimal dengan perolehan maksimum sebesar Rp 691.500,- dengan bobot 499 kg.

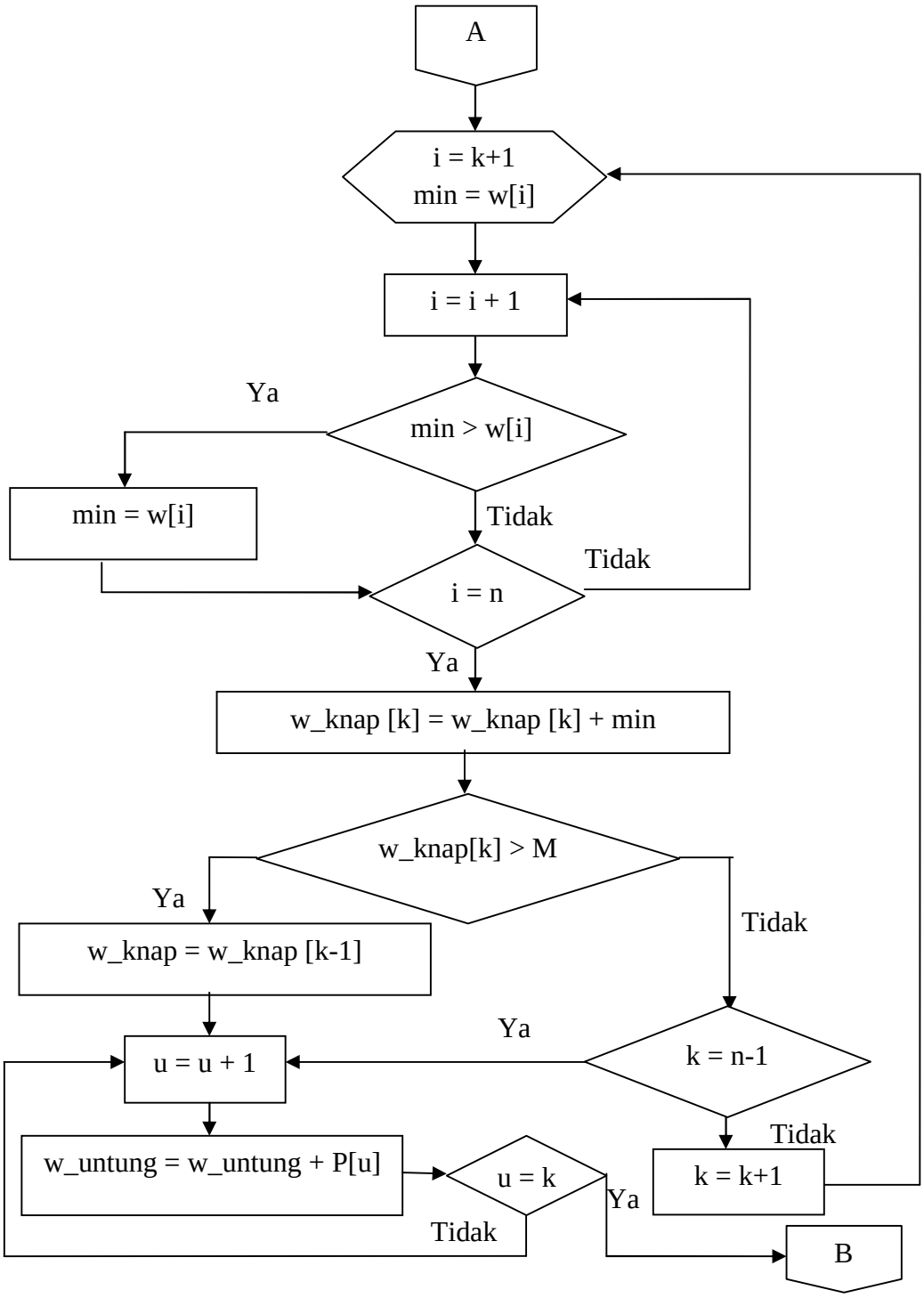
4.3 Perhitungan Kompleksitas Waktu

Untuk menentukan kemangkusan algoritma, penulis menyertakan perhitungan kompleksitas waktu algoritma dengan memperhatikan tahapan komputasi masing-masing algoritma pada *flowchart.*

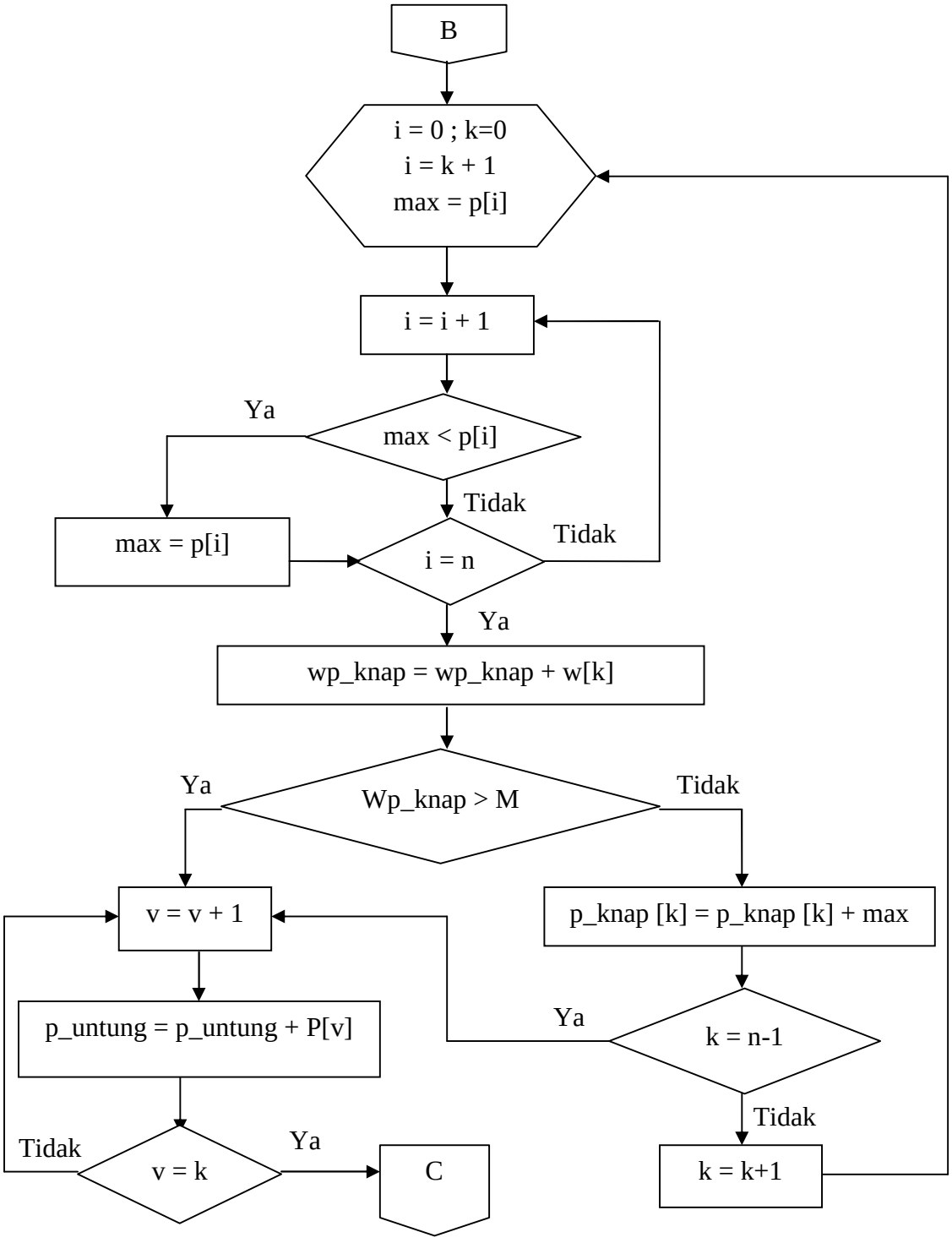
4.3.1 Flowchart Program Algoritma *Greedy*



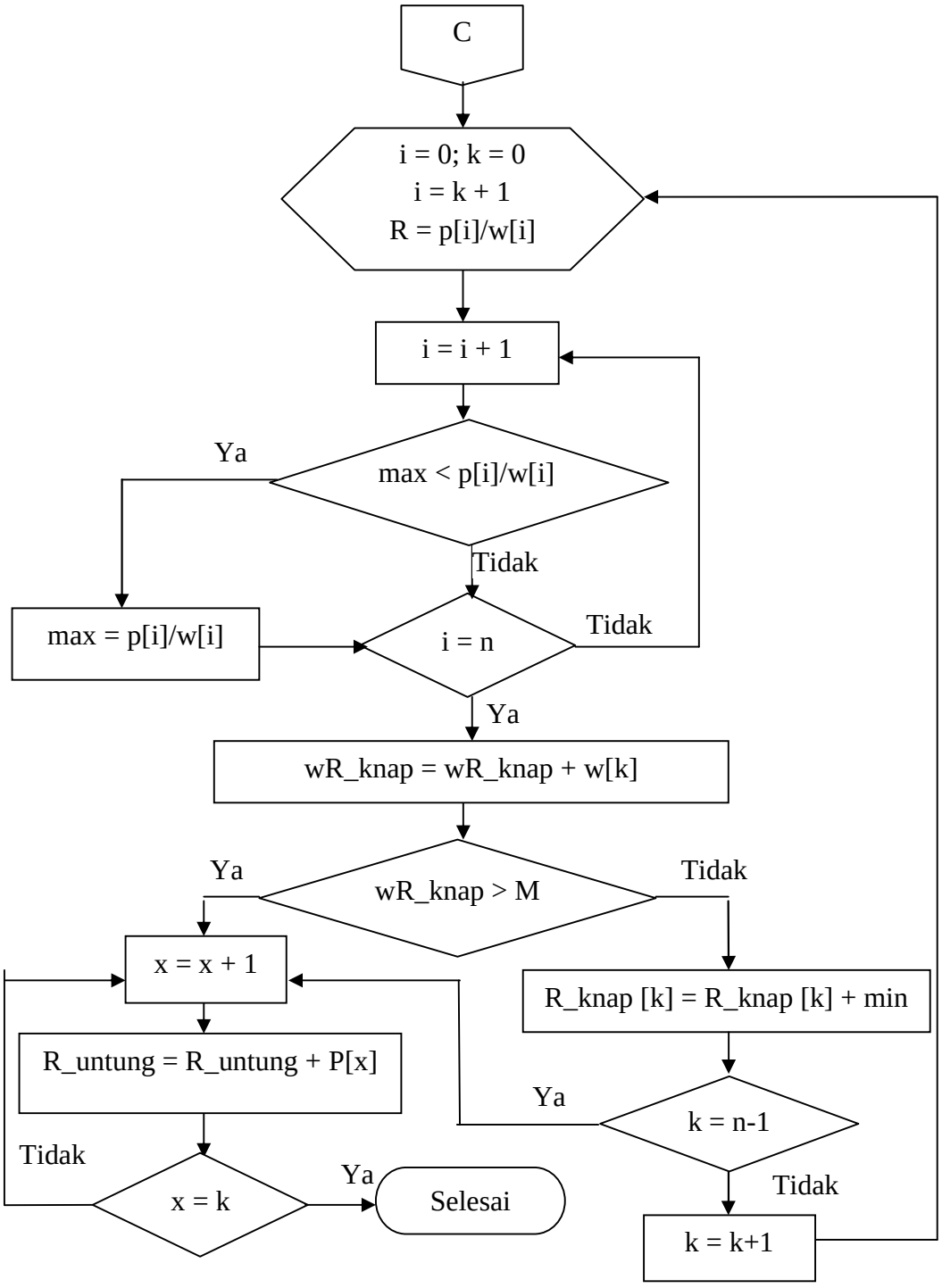
Gambar 4.1 Flowchart solusi awal algoritma *Greedy*



Gambar 4.2 Flowchart algoritma *Greedy by weight*

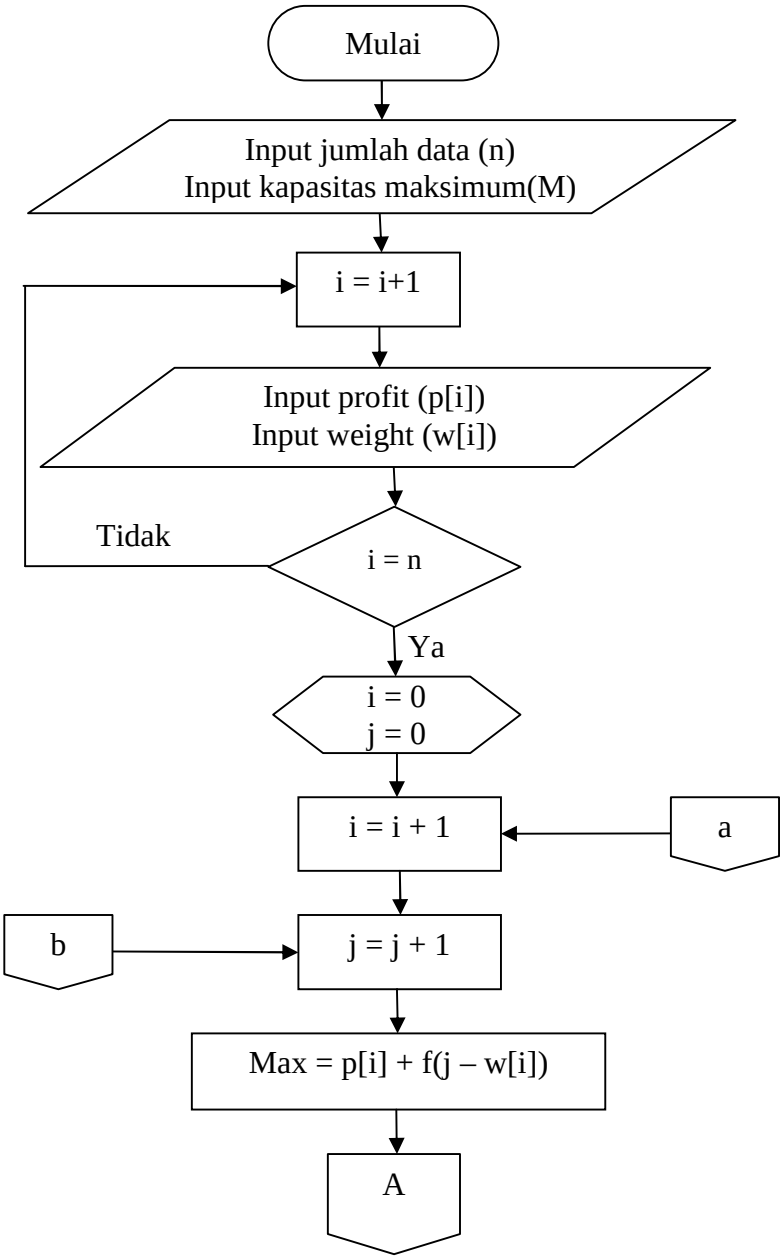


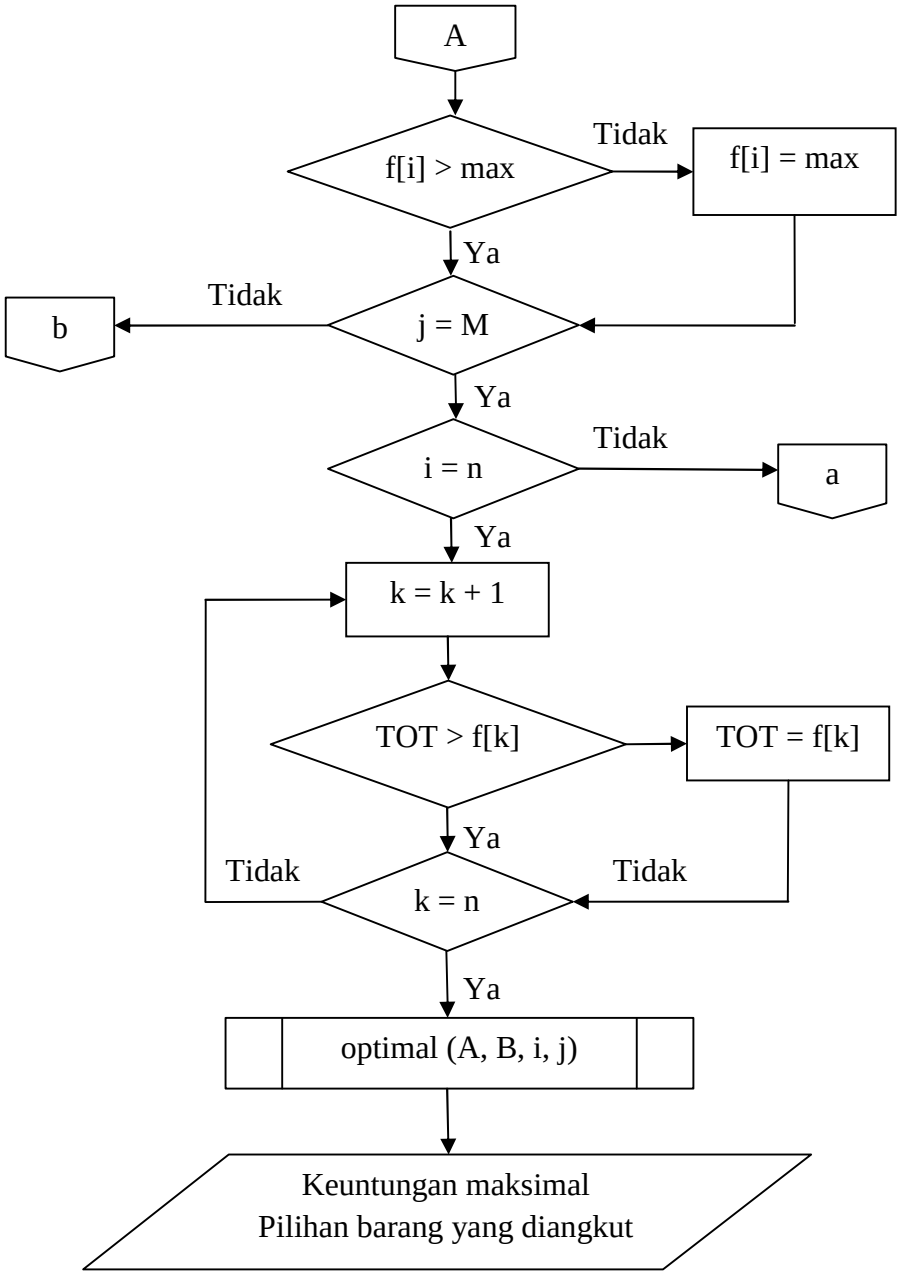
Gambar 4.3 Flowchart algoritma *Greedy by profit*

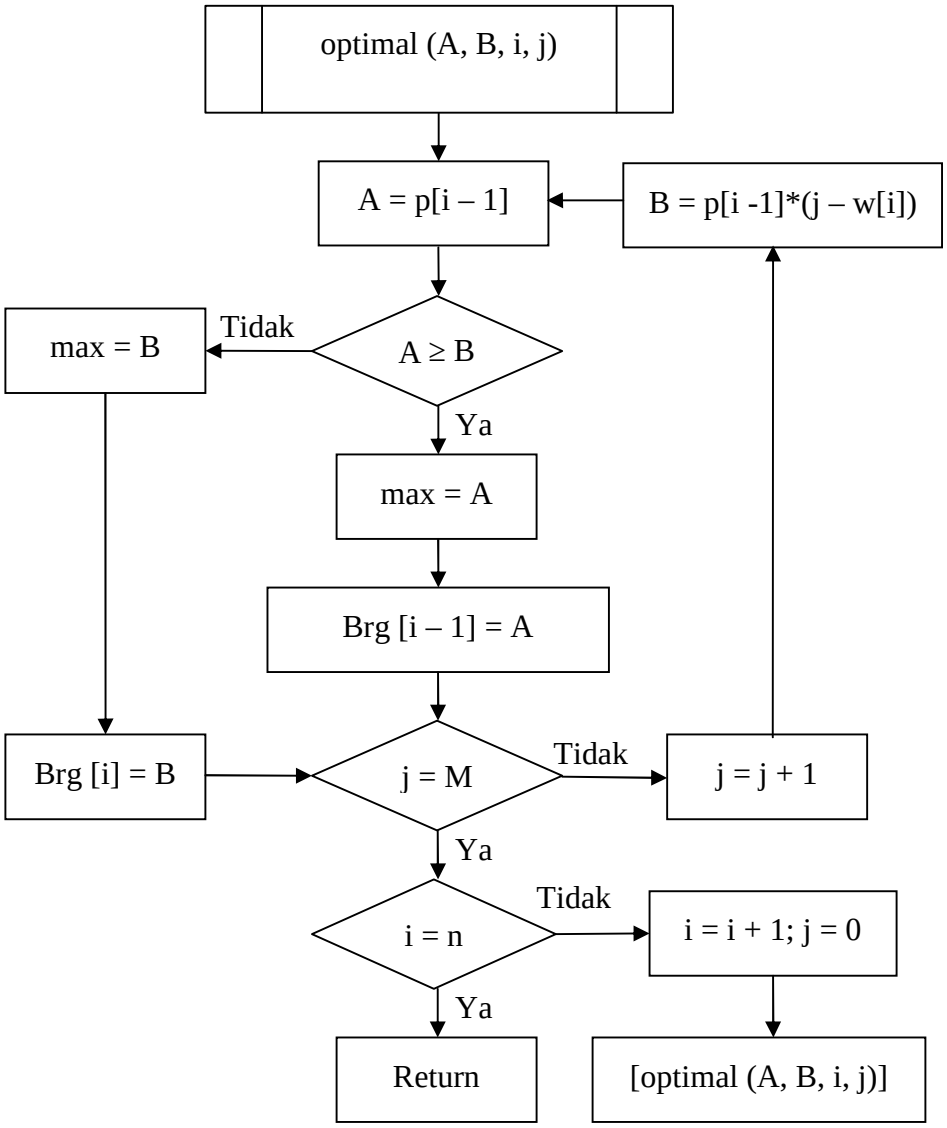


Gambar 4.4 Flowchart algoritma *Greedy by density*

4.3.2 Flowchart program Algoritma *Dynamic Programming*




A
f[i] > max
Tidak
f[i] = max
Ya
j = M
Tidak
b
Ya
i = n
Tidak
a
Ya
k = k + 1
TOT > f[k]
TOT = f[k]
Ya
k = n
Tidak
Tidak
Ya
optimal (A, B, i, j)
Keuntungan maksimal
Pilihan barang yang diangkut



Gambar 4.5 Flowchart algoritma *Dynamic Programming*

Dalam menganalisis suatu algoritma yang menjadi perhatian utama adalah berapa waktu tempuh dan berapa ruang dalam memori yang dibutuhkan untuk menjalankan algoritma tersebut. Untuk itu diperlukan proses pencarian kompleksitas waktu dari suatu algoritma. Untuk menghitung kompleksitas waktu dari algoritma dapat ditentukan dari *flowchart*. Pada sub bab ini akan dibahas tentang cara perhitungan kompleksitas dari algoritma yang saya pergunakan yaitu algoritma *Greedy* dan *Dynamic Programming*.

a. Perhitungan algoritma *Greedy* berdasarkan *flowchart* Gambar 4.1

Perhitungan loop input ($p[i]$) dan ($w[i]$)

Untuk i = 1, jumlah perhitungan = n

…

Untuk i = n, jumlah perhitungan = n

Jadi jumlah perhitungan kompleksitas waktunya adalah T(n) = n

1) *Greedy by weight* berdasarkan *flowchart* Gambar 4.2

- Perhitungan loop mencari $min > w[i]$
  - Proses pilihan ya

    Untuk mencari min = w[i], jumlah perhitungannya = 2 kali dan proses looping sampai $n - 1$ kali, jadi jumlah perhitungannya = $2(n - 1)$
  - Proses pilihan tidak

    Untuk i = 2, jumlah perhitungan = $n - 1$ kali

    …

    Untuk i = n, jumlah perhitungan = $n - 1$ kali

  Jadi jumlah perhitungan kompleksitas waktunya adalah

  $T(n) = 2(n - 1) + n - 1 = 2n - 2 + n - 1 = 3n - 3$
- Perhitungan $w\_knap[k] = w\_knap[k] + min$ = 1 kali

- Perhitungan syarat $w\_knap[k] > M$
  - Proses pilihan tidak

    Untuk mencari $w\_knap[k] > M$ dengan syarat $k = n - 1$ adalah $(3n - 3 + 1 + 1)n - 1 = (3n - 1)n - 1$

    Jadi $T(n) = 3n^2 - 4n + 1$
  - Proses pilihan ya

    Perhitungan $w\_knap = w\_knap[k - 1]$ = 1 kali,

    Perhitungan looping $w\_untung = w\_untung + p[u]$

    Untuk u = 1, jumlah perhitungan = k

    …

    Untuk u = k, jumlah perhitungan = k

    Jadi $T(n) = 3n - 3 + 1 + 1 + k = 3n - 1 + k$

  Jadi jumlah perhitungan kompleksitas waktunya adalah

  T(n) = $3n^2 - 4n + 1 + 3n - 1 + k = 3n^2 - n + k$

Jadi jumlah perhitungan kompleksitas waktu *Greedy by Weight* adalah

$T(n) = 3n - 3 + 1 + 3n^2 - n + k = 3n^2 + 2n + k - 2, O(n^2)$

2) *Greedy by profit* berdasarkan *flowchart* Gambar 4.3

- Perhitungan loop mencari $max < p[i]$
  - Proses pilihan ya

    Untuk mencari max = p[i], jumlah perhitungannya = 2 kali dan proses looping sampai $n - 1$ kali, jadi jumlah perhitungannya = $2(n - 1)$
  - Proses pilihan tidak

    Untuk i = 2, jumlah perhitungan = $n - 1$ kali

    …

    Untuk i = n, jumlah perhitungan = $n - 1$ kali

  Jadi jumlah perhitungan kompleksitas waktunya adalah

  T(n) =$2(n - 1) + n - 1 = 2n - 2 + n - 1 = 3n - 3$

- Perhitungan $wp\_knap[k] = wp\_knap[k] + w[k]$ = 1 kali
- Perhitungan syarat $wp\_knap[k] > M$
    - Proses pilihan tidak

      Perhitungan $p\_knap[k] = p\_knap[k] + max$ = 1 kali. Untuk mencari $wp\_knap[k] > M$ dengan syarat $k = n - 1$ adalah $(3n - 3 + 1 + 1 + 1 + 1)n - 1 = (3n + 1)n - 1 = 3n^2 - 2n - 1$

      Jadi $T(n) = 3n^2 - 2n - 1$
    - Proses pilihan ya

      Perhitungan looping $p\_untung = p\_untung + p[v]$

      Untuk v = 1, jumlah perhitungan = k

      …

      Untuk v = k, jumlah perhitungan = k

      Jadi $T(n) = 3n - 3 + 1 + k = 3n - 2 + k$

  Jadi jumlah perhitungan kompleksitas waktunya adalah

  T(n) = $(3n^2 - 2n - 1) + (3n - 2 + k) = 3n^2 + n + k - 1$

Jadi jumlah perhitungan kompleksitas waktu *Greedd by Profit* adalah

$T(n) = 3n - 3 + 1 + 3n^2 + n + k - 1 = 3n^2 + 4n - 3 + k, O(n^2)$

3) *Greedy by density* berdasarkan *flowchart* Gambar 4.4

- Perhitungan loop mencari $max < p[i]/w[i]$
    - Proses pilihan ya

      Untuk mencari $min = p[i]/w[i]$, jumlah perhitungannya = 2 kali dan proses looping sampai $n - 1$ kali, jadi jumlah perhitungannya = $2(n - 1)$
    - Proses pilihan tidak

      Untuk i = 2, jumlah perhitungan = $n - 1$ kali

      …

      Untuk i = n, jumlah perhitungan = $n - 1$ kali

Jadi jumlah perhitungan kompleksitas waktunya adalah

$T(n) = 2(n-1) + n - 1 = 2n - 2 + n - 1 = 3n - 3$

- Perhitungan $wR\_knap[k] = wR\_knap[k] + w[k]$ = 1 kali
- Perhitungan syarat $wR\_knap[k] > M$
  - Proses pilihan tidak

    Untuk mencari $wR\_knap[k] > M$ dengan syarat $k = n - 1$ adalah $(3n - 3 + 1 + 1 + 1)n - 1 = (3n)n - 1 = 3n^2 - 3$

    Jadi $T(n) = 3n^2 - 3$
  - Proses pilihan ya

    Perhitungan looping $R\_untung = R\_untung + p[x]$

    Untuk x = 1, jumlah perhitungan = k

    …

    Untuk x = k, jumlah perhitungan = k

    Jadi $T(n) = 3n - 3 + 1 + k = 3n - 2 + k$

Jadi jumlah perhitungan kompleksitas waktunya adalah

$T(n) = (3n^2 - 3) + (3n - 3 + k + 1) = 3n^2 + 3n + 1 + k$

Jadi jumlah perhitungan kompleksitas waktu *Greedy by Density* adalah

$T(n) = 3n - 3 + 1 + 3n^2 + 3n + 1 + k = 3n^2 + 6n + k - 1, O(n^2)$

Jadi total perhitungan kompleksitas algoritma *Greedy* adalah

$T(n) = n + (3n^2 + 2n + k - 2) + (3n^2 + 4n - 3 + k) + (3n^2 + 6n + k - 1)$

$= 9n^2 + 13n + 3k - 6\,, O(n^2)$

b. Perhitungan kompleksitas algoritma *Dynamic Programming* berdasarkan *flowchart* gambar 4.5

Perhitungan loop input ($p[i]$) dan ($w[i]$)

Untuk i = 1, jumlah perhitungan = n

Untuk i = n, jumlah perhitungan = n

Jadi jumlah perhitungan kompleksitas waktunya adalah T(n) = n

- Proses perhitungan syarat $j = M$
  - Pilihan tidak untuk syarat $f[i] > max$ adalah 1 + 1 + 1 + 1 = 4 dan looping sampai M kali, sehingga kompleksitas waktunya adalah $4m$ kali.
  - Pilihan ya untuk syarat $f[i] > max$ adalah 1 + 1 + 1 + 1 = 4 dan looping sampai M kali sehinnga $4m$, looping $j = M$ syarat tidak adalah $2m$.

  Jadi, perhitungan kompleksitas syarat $j = M$ adalah

  $n + (4m.\,2m) = n + 8m^2$
- Proses perhitungan syarat $i = n$
  - Proses pilihan tidak perhitungan kompleksitasnya adalah

    $(n + (8m^2)n) = n + 8m^2n$
  - proses syarat $TOT > f[k]$
    - Proses pilihan tidak dengan syarat $k = n$ perhitungan kompleksitasnya adalah

      $n + (8m^2n + 1 + 1) = n + 8m^2n + 2 = 8m^2n^2 + n + 2$
    - Proses pilihan ya dengan syarat $k = n$ perhitungan kompleksitasnya adalah

      $n + (8m^2n + 1)n = n + 8m^2n^2 + n = 8m^2n^2 + 2n$

  Jadi, perhitungan kompleksitas syarat $k = n$ adalah $8m^2n^2 + 2n$
- Untuk syarat perhitungan $j = M$
  - Proses tidak dengan syarat $A \geq B$ dengan pilihan:
    - tidak perhitungan kompleksitasnya adalah

      $(8m^2n^2 + 2n + 2 + 5)m = 8m^3n^2 + 2nm + 7m$
    - ya perhitungan kompleksitasnya adalah $8m^2n^2 + 2n + 4$

  jadi, perhitungan kompleksitas waktunya adalah

  $(8m^3n^2 + 2nm + 7m) + (8m^2n^2 + 2n + 4)$

  $= 8m^3n^2 + 8m^2n^2 + 2nm + 7m + 2n + 4$

- Untuk syarat perhitungan $i = n$
  - Untuk pilihan tidak, maka perhitungan kompleksitas waktunya adalah

$$(8m^3n^2 + 8m^2n^2 + 2nm + 7m + 2n + 4) + 2$$
$$= 8m^3n^2 + 8m^2n^2 + 2nm + 7m + 2n + 6$$

  - Untuk pilihan ya, maka perhitungan kompleksitas waktunya adalah

$$(8m^3n^2 + 8m^2n^2 + 2nm + 7m + 2n + 4)$$

Jadi total perhitungan kompleksitas algoritma *Dynamic Programming* adalah $T(n) = 8m^3n^2 + 8m^2n^2 + 2nm + 7m + 2n + 6, O(m^3n^2)$.

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

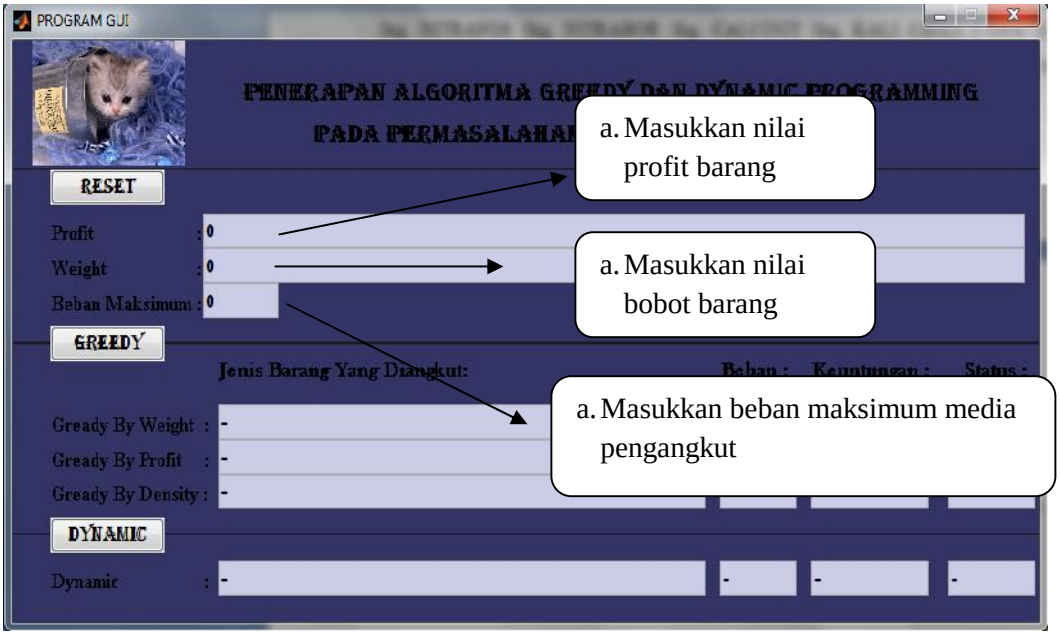
Penyelesaian persoalan *integer knapsack* dengan algoritma *Greedy*, jumlah langkah yang diperlukan untuk mencapai solusi permasalahan *integer knapsack* ini bergantung pada jumlah objek yang dimiliki. Untuk menentukan objek dengan keuntungan maksimal diperlukan proses perbandingan sebanyak n kali. Sehingga kompleksitas waktunya adalah $O(n^2)$. Sedangkan pada algoritma *Dynamic Programming* terdapat 2 variabel, yaitu m yang menyatakan jumlah kelipatan kapasitas angkut tiap tahap dan n yang menyatakan jumlah data dan memiliki kompleksitas waktu sebesar $O(m^3n^2)$. Dari keterangan tersebut dapat disimpulkan bahwa algoritma *Dynamic Programming* mempunyai kompleksitas yang lebih besar dibandingkan dengan algoritma *Greedy*.

4.4 Permasalahan *Knapsack* dengan Program MATLAB R2009a

Dalam skripsi ini akan disertai aplikasi penerapan algoritma *Greedy* dan *Dynamic Programming* pada permasalahan *integer knapsack* dengan menggunakan program MATLAB R2009a. Aplikasi yang telah dibuat pada skripsi ini bertujuan untuk mempermudah dan mempercepat perhitungan kedua algoritma yaitu algoritma *Greedy* dan *Dinamic Programming*. Gambar 4.3 merupakan tampilan awal dari program Aplikasi Permasalahan *Integer Knapsack.*

Untuk langkah-langkah menjalankan programnya adalah sebagai berikut:

a. Masukkan profit barang pada kolom profit.
b. Masukkan bobot (weight) barang pada kolom profit.
c. Masukkan beban maksimum media pengangkut (M).
d. Klik tombol *GREEDY* untuk melakukan perhitungan dengan algoritma *Greedy*.
e. Klik tombol DYNAMIC untuk melakukan perhitungan dengan algoritma *Dynamic Programming*.



Gambar 4.6 Aplikasi yang dibuat dari bahasa pemrograman Matlab R2009a

Terdapat beberapa langkah yang harus dilakukan untuk menjalankan program aplikasi permasalahan *integer knapsack*, berikut penjelasan dari tiap langkah dalam menjalankan program Aplikasi Permasalahan *Integer Knapsack*.

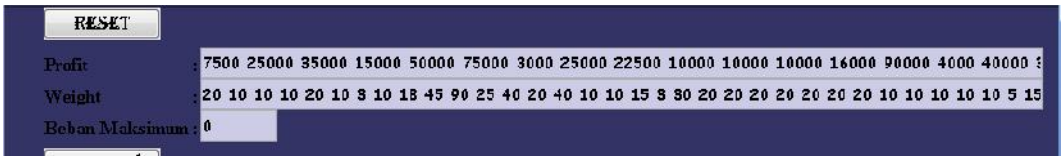
a. Input profit (p_i)

Isi tabel dengan data profit barang p_i = [7500, 25000, 35000, 15000, 50000, 75000, 3000, 25000, 22500, 10000, 10000, 10000, 16000, 90000, 4000, 40000, 35000, 15000, 1500, 6000, 14000, 44000, 20000, 20000, 4000, 4000, 20000, 5500, 1000, 7500, 5000, 2500, 10000, 11000, 12500, 15000, 25000].

Gambar 4.7 Langkah mengisi data profit (n = 37)

b. Input weight (w_i)

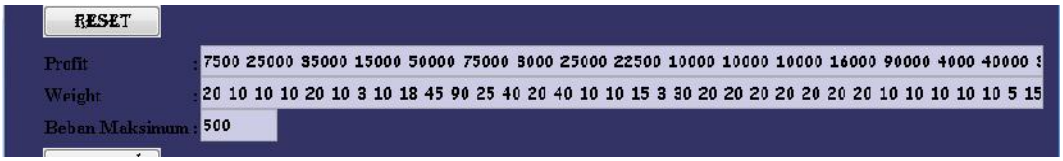
Isi tabel dengan data bobot barang w_i = [20, 10, 10, 10, 20, 10, 3, 10, 18, 45, 90, 25, 40, 20, 40, 10, 10, 15, 3, 30, 20, 20, 20, 20, 20, 20, 20, 10, 10, 10, 10, 10, 5, 15, 10, 5, 25]



Gambar 4.8 Langkah mengisi data berat (n = 37)

c. Input beban maksimum media pengangkut (M)

Isikan bobot maksimum dari media pengangkut (M)



Gambar 4.9 Langkah mengisi data kapasitas maksimum (n = 37)

d. Berikut gambar lengkap dari aplikasi yang telah dibuat mulai dari langkah 1 sampai langkah 3.

Gambar 4.10 Tampilan program untuk data pada Tabel 4.1

Hasil akhir yang ditampilkan dalam perhitungan menggunakan program ini adalah sebuah hasil keuntungan maksimum dari masing-masing algoritma. Dari Gambar 4.10 di atas dijelaskan bahwa perhitungan untuk algoritma *greedy* dari data yang telah di *input* dapat diperoleh hasil untuk strategi *Greedy* by *weight* diperoleh hasil maksimum sebesar Rp 676.500,- dengan bobot 474 kg. Pada strategi *Greedy by profit* diperoleh hasil maksimum sebesar Rp 655.500,- dengan bobot 498 kg. Pada strategi *Greedy by density* diperoleh hasil maksimum sebesar Rp 687.500,- dengan bobot 479 kg. Dari ketiga strategi tersebut terdapat hasil yanng paling maksimal yaitu pada strategi *greedy by density* dengan perolehan maksimum sebesar Rp 687.500,- dengan bobot 479 kg. Sedangkan untuk algoritma *Dynamic Programming* diperoleh hasil maksimum sebesar Rp 691.500,- dengan bobot 499 kg.

4.5 Perbandingan Algoritma *Greedy* dan *Dynamic Programming* pada Permasalahan *Knapsack*

Pada subbab 4.1 telah dijelaskan hasil perhitungan dari masing-masing algoritma. Pada algoritma Greedy terdapat 3 strategi greedy yang dihasilkan. Ketiga strategi *Greedy* itu diantaranya adalah strategi *Greedy* by *weight*, *Greedy* by *profit*, dan *Greedy* by *density*. Pada strategi *Greedy* by *weight* diperoleh hasil keuntungan maksimum sebesar Rp 676.500,- dengan bobot 474 kg, pada strategi *Greedy* by profit diperoleh hasil keuntungan maksimum sebesar Rp 655.500,- dengan bobot 498 kg, pada strategi *Greedy* by density diperoleh hasil keuntungan maksimum sebesar Rp 687.500,- dengan bobot 479 kg. Dari ketiga strategi tersebut terdapat hasil keuntungan yang paling maksimal yaitu pada strategi *greedy by density* dengan perolehan maksimum sebesar Rp 687.500,- dengan bobot 479 kg. Sedangkan Pada algoritma *Dynamic Programming* diperoleh hasil keuntungan maksimum sebesar Rp 691.500,- dengan bobot 499 kg. Dari hasil perhitungan diperoleh keuntungan yang maksimum pada permasalahan *knapsack* ialah sebagai berikut:

Tabel 4.9 Solusi Optimal dari Perhitungan Kedua Algoritma

No	Algoritma	Pilihan barang	Keuntungan
1	*Greedy*	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37	Rp 687.500,-
2	*Dynamic Programming*	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24, 25, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37	Rp 691.500,-

Menurut tabel 4.9 dapat diketahui bahwa hasil perhitungan dengan menggunakan algoritma *Greedy* menghasilkan keuntungan yang lebih kecil dibandingkan dengan nilai keuntungan yang dihasilkan oleh algoritma *Dynamic Programming.* Artinya, penggunaan algoritma *Dynamic Programming* lebih optimal jika diaplikasikan pada permasalahan *knapsack* di UD. BINTANG TANI.

Berbeda halnya jika ditinjau dari segi kompleksitas waktu yang dihasilkan. Algoritma *Dynamic Programming* memiliki kompleksitas waktu yakni $O(m^3n^2)$, sedangkan algoritma *Greedy* dengan $O(n^2)$. Artinya, perhitungan dengan algoritma *Dynamic Programming* membutuhkan waktu yang lebih lama jika dibandingkan dengan perhitungan menggunakan algoritma *Greedy* pada permasalahan *knapsack* di UD.BINTANG TANI. Dengan kata lain menurut perhitungan kompleksitas waktu yang diperoleh dapat dikatakan algoritma *Greedy* lebih efektif jika dibandingkan dengan algoritma *Dynamic Programming* untuk diterapkan pada permasalahan *knapsack* di UD. BINTANG TANI.

Jika ditinjau dari ruang pencarian solusi, perbedaan yang paling mendasar antara algoritma *Greedy* dan *Dynamic Programming* adalah pada algoritma *Greedy* solusi pencarian bersifat optimum lokal, sedangkan pada algoritma *Dynamic Programming* solusi pencarian bersifat optimum global.

BAB 5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil dan pembahasan dengan data saya dapat disimpulkan bahwa:

a. Untuk mencari keuntungan maksimum di UD. BINTANG TANI dengan menggunakan algoritma *Greedy* diperoleh hasil optimal pada strategi *greedy by density* yaitu sebesar Rp 687.500,- dengan jumlah bobot 479 kg.

b. Untuk mencari keuntungan maksimum di UD. BINTANG TANI dengan menggunakan algoritma *Dynamic programming* diperoleh hasil optimal yaitu sebesar Rp 691.500,- dengan jumlah bobot 499 kg.

c. Berdasarkan hasil perhitungan diatas maka diperoleh hasil bahwa perhitungan pada algoritma *Dynamic Programming* lebih menghasilkan keuntungan yang maksimal dibandingkan dengan algoritma *Greedy*. dengan demikian dapat disimpulkan bahwa algoritma *Dynamic Programming* lebih baik daripada algoritma *greedy* untuk penyelesaian pada permasalahan *integer knapsack.*

d. Algoritma *Greedy* dan *Dynamic Programming* pada kasus permasalahan *integer knapsack* berdasarkan banyak langkah yang dibutuhkan diperoleh hasil pencarian bahwa pada algoritma *Greedy* diperlukan proses perbandingan sebanyak n^2, maka kompleksitas waktunya adalah $O(n^2)$. Pada algoritma *Dynamic Programming* jumlah langkah yang diperlukan untuk mencapai solusi optimal adalah sebanyak m^3n^2, maka kompleksitas waktunya adalah $O(m^3n^2)$. Sehingga algoritma *Dynamic Programming* mempuyai jumlah kompleksitas yang lebih besar dibandingkan dengan algoritma *Greedy*.

5.2 Saran

Dalam penelitian ini masalah variabel integer kanpsack yang digunakan adalah profit dan bobot suatu barang dengan kapasitas bobot maksimal sehingga masih terbuka bagi peneliti lain untuk menggunakan variabel harga dasar barang dan profit barang dengan kapasitas dana yang tersedia sebagai kapasitas maksimum. Program komputer yang digunakan juga dapat lebih dikembangkan dengan menggunakan *software* yang lain seperti Java.

DAFTAR PUSTAKA

A. Taha, Hamdy. 1996. *Riset Operasi*. Jakarta: Binarupa Aksara.

Brassard, G. 1996. *Fundamentals of Algorithms.* New Jersey: Prentice Hall.

Diah, K., Mardhiah, F., dan CharloSutanto. 2010. *"Penyelesaian Knapsack Problem Menggunakan Algoritma Genetik"*. Riau: Teknik Komputer Politeknik Caltex Riu Pekanbaru.

Dimyati, Tjutju T., Ahmad Dimyati. 2004. *Operations Research*: *Model-model Pengambilan Keputusan*. Bandung: Sinar Baru Agesindo.

Martello, Silvano. 2006. *New Trends in Exact Algorithms for the 0-1 Knapsack Problem.* http://www.cise.ufl.edu/~sahni/papers/knap.pdf. [21Januari 2012]

Munir, Rinaldi. 2005. *Strategi Algoritmik*. Yogyakarta: Graha Ilmu.

Paryati. 2009. "*StrategiAlgoritma Greedy untuk Menyelesaikan Permasalahan Knapsack 0-1*". Yogyakarta: Teknik Informatika UPN "Veteran" Yogyakarta.

Shofwatul'uyun. 2009. *Kompleksitas Algoritma*. Teknik Informatika UIN Kalijaga. http://www.srcribd.com/doc/38673595/null. [25 Desember 2012]

Suyanto. 2010. *Algoritma Optimasi*. Yogyakarta: Graha Ilmu.

S. Hiller, Frederick,. Gerald J. Lieberman. 1994. *Pengantar Riset Operasi*. Jakarta: PT. Gelora Aksara Pratama.

V. Springer. 2005. *Knapsack 0-1 Problem*. Yogyakarta: Graha Ilmu.

Wardy, I.S. 2007.*Penggunaan Graph Dalam Algoritma Semutuntuk Melakukan Optimasi*. Bandung: Program Study Teknuik Informatika ITB Bandung.

Wiboeo, Ferry Wahyu. 2011. *Matematika Diskrit: Kompleksitas Algoritma*. STMIK Amikom Yogyakarta.

LAMPIRAN

LAMPIRAN A. SCRIPT PROGRAM ALGORITMA *GREEDY*

```
% clc
clear all
% input data
profit=[7500 25000 35000 15000 50000 75000 3000 25000 22500
10000 10000 10000 16000 90000 4000 40000 35000 15000 1500 6000
14000 44000 20000 20000 4000 4000 20000 5500 1000 7500 5000
2500 10000 11000 12500 15000 25000];%input('masukkan
keuntungan= ');
weight=[20 10 10 10 20 10 3 10 18 45 90 25 40 20 40 10 10 15 3
30 20 20 20 20 20 20 20 10 10 10 10 10 5 15 10 5
25];%input('masukkan berat= ');
M=500;%input('masukkan kapasitas maks=');
n=length(profit); m=length(weight);
weigt0=weight; M0=M; n0=n; m0=m;
profit0=profit;
maksm= sum(weight);
if M>=maksm
     disp('Semua barang terangkut');
     untung=sum(profit);
     disp(['jumlah keuntungan= ', num2str(untung)]);
    break
end
if n~=m
    disp('Error, matrik profit dan weight tidak sama')
    break
end
% disp('pilih 1: Greedy by Weight');
% disp('pilih 2: Greedy by Profit');
```

```
% disp('pilih 3: Greedy by Density');
% pilih=input('masukkan pilihan 1-3= ');
% switch pilih
%     case 1
    % Greedy by Weight
        W= sort(weight); % weight yang sudah urut dari kecil
ke besar
        % menyesuaikan urutan profit dan urutan barang
        for i=1:n

            for j=1:m
               a=W(i); b= weight(j);
               if a==b
                  P(i)=profit(j);
                  urutan(i)=j;
                  weight(j)=-b;
                  break;
               end

            end
        end
        berat=0;i=0;
        while berat<=M
           i=i+1;
            berat=berat+W(i);
        end

        if berat > M
            berat=berat-W(i);
            i=i-1;
        end
        untung= sum (P(1:i));
```

```
        disp('Greedy By Weight');
        disp('Hasil')
                urutan=sort(urutan(1:i));
        disp(['jenis barang yang di angkut:
',num2str(urutan(1:i))])
        disp(['jumlah beban yang di angkut: ',
num2str(berat)])
        disp(['jumlah keuntungan: ',num2str(untung)]);
%     case 2
% Greedy by profit
weight=weigt0; M=M0; n=n0; m=m0;
profit=profit0;

P= sort(profit); % profit yang sudah urut dari kecil ke besar
        P=P(n:-1:1);
        % menyesuaikan urutan weihgt dan urutan barang
        for i=1:n

            for j=1:m
               a=P(i); b= profit(j);
               if a==b
                  W(i)=weight(j);
                  urutan(i)=j;
                  profit(j)=-b;
                  break;
               end

            end
        end
        berat=0;i=0;
        while berat<=M
           i=i+1;
```

```
            berat=berat+W(i);
        end

        if berat > M
            berat=berat-W(i);
            i=i-1;
        end
        untung= sum (P(1:i));
        disp('Greedy By Profit');
        disp('Hasil')
        urutan=sort(urutan(1:i));
        disp(['jenis barang yang di angkut:
',num2str(urutan(1:i))])
        disp(['jumlah beban yang di angkut: ',
num2str(berat)])
        disp(['jumlah keuntungan: ',num2str(untung)]);

%     case 3
weight=weigt0; M=M0; n=n0; m=m0;
profit=profit0;

rasio=profit./weight;
            % Greedy by Weight
        R= sort(rasio); % weight yang sudah urut dari kecil ke
besar
        R=R(n:-1:1);
        % menyesuaikan urutan profit dan urutan barang
        for i=1:n

            for j=1:m
               a=R(i); b= rasio(j);
               if a==b
```

```
                P(i)=profit(j);
                W(i)=weight(j);
                urutan(i)=j;
                rasio(j)=-b;
                break;
            end

        end
    end
    berat=0;i=0;
    while berat<=M
       i=i+1;
        berat=berat+W(i);
    end

    if berat > M
        berat=berat-W(i);
        i=i-1;
    end
    untung= sum (P(1:i));
    disp('Greedy By Density');
    disp('Hasil')
            urutan=sort(urutan(1:i));
    disp(['jenis barang yang di angkut:
',num2str(urutan(1:i))])
    disp(['jumlah beban yang di angkut: ',
num2str(berat)])
    disp(['jumlah keuntungan: ',num2str(untung)]);

% end
```

LAMPIRAN B. SCRIPT PROGRAM ALGORITMA *DYNAMIC PROGRAMMING*

```
clc
clear all
% input data
profit=input('masukkan keuntungan= ');
weight=input('masukkan berat= ');
M=input('masukkan kapasitas maks=');
r=input('masukkan tahap perhitungan=');
p=r/M
n=length(profit); m=length(weight);
if n~=m
    disp('Error, matrik profit dan weight tidak sama')
end
n=37;
M=500;
m=500;
v=[7500 25000 35000 15000 50000 75000 3000 25000 22500 10000
10000 10000 16000 90000 4000 40000 35000 15000 1500 6000 14000
44000 20000 20000 4000 4000 20000 5500 1000 7500 5000 2500
10000 11000 12500 15000 25000 ];
w=[20 10 10 10 20 10 3 10 18 45 90 25 40 20 40 10 10 15 3 30
20 20 20 20 20 20 20 10 10 10 10 10 5 15 10 5 25 ];
p=M/m;
z=zeros(n+1,m);
%Tahap 1 sampai n
for i=2:n+1
   y(1)=0;
   z(i,1)=0;
   for j=2:m
      y(j)=y(j-1)+p;
```

```
      if w(i-1)<=y(j)
         z(i,j)=max([z(i-1,j) v(i-1)+z(i-1,j-w(i-1))]);
      else
         z(i,j)=z(i-1,j);
      end;
   end;
end;
disp(z);
% Mencari solusi optimum
Keuntungan=z(i,m)
beban=0
%Mencari Barang yang dibeli
for i=n+1:-1:2
   if z(i,m)~=z(i-1,m)
      a(i)=1;
      c=m.*p;
      q=c-w(i-1);
      m=round(q/p);
   else
      a(i)=0;
   end;
end;
pp=length(a);
disp('jenis barang yang di angkut')
a=a(2:pp)
for i=1:pp-1
    ai=a(i);
    if ai==1
   beban=beban+w(i);
    end
end
disp('Jumlah beban yang di angkut')
```

LAMPIRAN C. OUTPUT SEBAGIAN HASIL ALGORITMA *DYNAMIC PROGRAMMING*.

Columns 421 through 430

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
491500	491500	491500	491500	491500	491500	491500	491500	491500	491500
505500	505500	505500	505500	505500	505500	505500	505500	505500	505500
543500	543500	543500	543500	543500	543500	543500	543500	543500	543500
563500	563500	563500	563500	563500	563500	563500	563500	563500	563500
579500	579500	579500	579500	579500	579500	582000	582000	582000	583500
583500	583500	583500	583500	583500	583500	583500	583500	583500	583500
583500	583500	583500	583500	583500	583500	586000	586000	586000	587500
601500	601500	601500	601500	601500	601500	602000	602000	602000	603500
605000	605000	605000	605000	605000	605000	605500	605500	605500	607000
605000	605000	605000	605000	605000	605000	605500	605500	605500	607000
610500	610500	610500	610500	610500	610500	611000	611000	611000	612500
612500	612500	612500	612500	612500	612500	614000	614000	614000	615500
614000	614000	614000	614000	614000	614000	614000	614000	614000	615500
622000	622500	622500	622500	624000	624000	624000	624000	624000	624000
629000	629500	629500	629500	631000	631000	631000	631000	631000	632500
639000	640000	640000	640000	641500	641500	641500	641500	641500	641500
654000	654000	654000	654000	654000	654000	655000	655000	655000	656500
670500	673500	673500	673500	675000	675000	675000	675000	675000	675000

Columns 431 through 440

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
491500	491500	491500	491500	491500	491500	494000	494000	494000	495500
505500	505500	505500	505500	505500	505500	505500	505500	505500	505500
543500	543500	543500	545000	545000	545000	548000	548000	548000	549500
563500	563500	563500	563500	563500	563500	563500	563500	563500	563500
583500	583500	583500	583500	583500	583500	583500	583500	583500	583500
583500	583500	583500	583500	583500	583500	583500	583500	583500	583500
587500	587500	587500	587500	587500	587500	587500	587500	587500	587500
603500	603500	603500	603500	603500	603500	603500	603500	603500	603500
607000	607000	607000	607000	607000	607000	607500	607500	607500	609000
607000	607000	607000	607000	607000	607000	607500	607500	607500	609000
612500	612500	612500	612500	612500	612500	613000	613000	613000	614500
615500	615500	615500	615500	615500	615500	616000	616000	616000	617500
615500	615500	615500	615500	615500	615500	616500	616500	616500	618000
624000	624000	624000	624000	625500	625500	625500	625500	625500	625500
632500	632500	632500	632500	633000	633000	633500	633500	633500	635000
641500	642000	642000	642000	643500	643500	643500	643500	643500	645000
656500	656500	656500	656500	656500	656500	657000	657000	657000	658500
675000	676000	676000	676000	677500	677500	677500	677500	677500	677500

Columns 441 through 450

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
495500	495500	495500	495500	495500	495500	495500	495500	495500	495500

505500	505500	505500	505500	505500	505500	505500	505500	505500	505500
549500	549500	549500	549500	549500	549500	549500	549500	549500	549500
563500	563500	563500	563500	563500	563500	563500	563500	563500	563500
583500	583500	583500	583500	583500	583500	583500	583500	583500	583500
583500	583500	583500	583500	583500	583500	586000	586000	586000	587500
587500	587500	587500	587500	587500	587500	587500	587500	587500	587500
603500	603500	603500	603500	603500	603500	606000	606000	606000	607500
609000	609000	609000	609000	609000	609000	609000	609000	609000	609000
609000	609000	609000	609000	609000	609000	609000	609000	609000	610000
614500	614500	614500	614500	614500	614500	615000	615000	615000	616500
617500	617500	617500	617500	617500	617500	618000	618000	618000	619500
618000	618000	618000	618000	618000	618000	618500	618500	618500	620000
625500	626500	626500	626500	628000	628000	628000	628000	628000	628000
635000	635000	635000	635000	635000	635000	635000	635000	635000	636500
645000	645000	645000	645000	645500	645500	646000	646000	646000	647500
658500	658500	658500	658500	660000	660000	660000	660000	660000	660500
677500	677500	677500	677500	679000	679000	679000	679000	679000	679000

Columns 451 through 460

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
495500	495500	495500	495500	495500	495500	495500	495500	495500	495500
505500	505500	505500	505500	505500	505500	508000	508000	508000	509500
549500	549500	549500	549500	549500	549500	549500	549500	549500	549500
563500	563500	563500	565000	565000	565000	568000	568000	568000	569500
583500	583500	583500	583500	583500	583500	583500	583500	583500	583500
587500	587500	587500	587500	587500	587500	587500	587500	587500	587500
587500	587500	587500	587500	587500	587500	587500	587500	587500	587500
607500	607500	607500	607500	607500	607500	607500	607500	607500	607500
609000	609000	609000	609000	609000	609000	611500	611500	611500	613000
610000	610000	610000	610000	610000	610000	611500	611500	611500	613000
616500	616500	616500	616500	616500	616500	616500	616500	616500	617500
619500	619500	619500	619500	619500	619500	620000	620000	620000	621500
620000	620000	620000	620000	620000	620000	620500	620500	620500	622000
628000	628500	628500	628500	630000	630000	630000	630000	630000	630000
636500	636500	636500	636500	636500	636500	637500	637500	637500	639000
647500	647500	647500	647500	647500	647500	647500	647500	647500	649000
660500	661000	661000	661000	662500	662500	662500	662500	662500	662500
679000	680000	680000	680000	681500	681500	681500	681500	681500	681500

Columns 461 through 470

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
495500	495500	495500	495500	495500	495500	495500	495500	495500	495500
509500	509500	509500	509500	509500	509500	509500	509500	509500	509500
549500	549500	549500	549500	549500	549500	549500	549500	549500	549500
569500	569500	569500	569500	569500	569500	569500	569500	569500	569500
583500	583500	583500	583500	583500	583500	583500	583500	583500	583500
587500	587500	587500	587500	587500	587500	587500	587500	587500	587500
587500	587500	587500	587500	587500	587500	590000	590000	590000	591500
607500	607500	607500	607500	607500	607500	607500	607500	607500	607500
613000	613000	613000	613000	613000	613000	613000	613000	613000	613000
613000	613000	613000	613000	613000	613000	613000	613000	613000	614000
617500	617500	617500	617500	617500	617500	619000	619000	619000	620500
621500	621500	621500	621500	621500	621500	621500	621500	621500	622500
622000	622000	622000	622000	622000	622000	622500	622500	622500	624000
630000	630500	630500	630500	632000	632000	632000	632000	632000	632000
639000	639000	639000	639000	639000	639000	639500	639500	639500	641000
649000	649000	649000	649000	649000	649000	650000	650000	650000	651500
662500	662500	662500	662500	664000	664000	664000	664000	664000	664000
681500	682000	682000	682000	683500	683500	683500	683500	683500	685000

Columns 471 through 480

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500

235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
495500	495500	495500	495500	495500	495500	495500	495500	495500	495500
509500	509500	509500	509500	509500	509500	509500	509500	509500	509500
549500	549500	549500	549500	549500	549500	552000	552000	552000	553500
569500	569500	569500	569500	569500	569500	569500	569500	569500	569500
583500	583500	583500	585000	585000	585000	588000	588000	588000	589500
587500	587500	587500	587500	587500	587500	588000	588000	588000	589500
591500	591500	591500	591500	591500	591500	591500	591500	591500	591500
607500	607500	607500	607500	607500	607500	607500	607500	607500	607500
613000	613000	613000	613000	613000	613000	613000	613000	613000	613000
614000	614000	614000	614000	614000	614000	614000	614000	614000	614000
620500	620500	620500	620500	620500	620500	620500	620500	620500	621500
622500	622500	622500	622500	622500	622500	624000	624000	624000	625500
624000	624000	624000	624000	624000	624000	624000	624000	624000	625500
632000	632500	632500	632500	634000	634000	634000	634000	634000	634000
641000	641000	641000	641000	641000	641000	641500	641500	641500	643000
651500	651500	651500	651500	651500	651500	652000	652000	652000	653500
664000	665000	665000	665000	666500	666500	666500	666500	666500	666500
685000	685000	685000	685000	685500	685500	686000	686000	686000	687500

Columns 481 through 490

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
495500	495500	495500	495500	495500	495500	495500	495500	495500	495500
509500	509500	509500	509500	509500	509500	509500	509500	509500	509500

553500	553500	553500	553500	553500	553500	553500	553500	553500	553500
569500	569500	569500	569500	569500	569500	569500	569500	569500	569500
589500	589500	589500	589500	589500	589500	589500	589500	589500	589500
589500	589500	589500	589500	589500	589500	589500	589500	589500	589500
591500	591500	591500	591500	591500	591500	591500	591500	591500	591500
607500	607500	607500	607500	607500	607500	610000	610000	610000	611500
613000	613000	613000	613000	613000	613000	613000	613000	613000	613000
614000	614000	614000	614000	614000	614000	614000	614000	614000	614000
621500	621500	621500	621500	621500	621500	621500	621500	621500	621500
625500	625500	625500	625500	625500	625500	625500	625500	625500	626500
625500	625500	625500	625500	625500	625500	626500	626500	626500	628000
634000	634000	634000	634000	635500	635500	635500	635500	635500	635500
643000	643000	643000	643000	643000	643000	643500	643500	643500	645000
653500	653500	653500	653500	653500	653500	654000	654000	654000	655500
666500	667000	667000	667000	668500	668500	668500	668500	668500	668500
687500	687500	687500	687500	687500	687500	687500	687500	687500	689000

Columns 491 through 500

0	0	0	0	0	0	0	0	0	0
7500	7500	7500	7500	7500	7500	7500	7500	7500	7500
32500	32500	32500	32500	32500	32500	32500	32500	32500	32500
67500	67500	67500	67500	67500	67500	67500	67500	67500	67500
82500	82500	82500	82500	82500	82500	82500	82500	82500	82500
132500	132500	132500	132500	132500	132500	132500	132500	132500	132500
207500	207500	207500	207500	207500	207500	207500	207500	207500	207500
210500	210500	210500	210500	210500	210500	210500	210500	210500	210500
235500	235500	235500	235500	235500	235500	235500	235500	235500	235500
258000	258000	258000	258000	258000	258000	258000	258000	258000	258000
268000	268000	268000	268000	268000	268000	268000	268000	268000	268000
278000	278000	278000	278000	278000	278000	278000	278000	278000	278000
288000	288000	288000	288000	288000	288000	288000	288000	288000	288000
304000	304000	304000	304000	304000	304000	304000	304000	304000	304000
394000	394000	394000	394000	394000	394000	394000	394000	394000	394000
398000	398000	398000	398000	398000	398000	398000	398000	398000	398000
438000	438000	438000	438000	438000	438000	438000	438000	438000	438000
473000	473000	473000	473000	473000	473000	473000	473000	473000	473000
488000	488000	488000	488000	488000	488000	488000	488000	488000	488000
489500	489500	489500	489500	489500	489500	489500	489500	489500	489500
495500	495500	495500	495500	495500	495500	495500	495500	495500	495500
509500	509500	509500	509500	509500	509500	509500	509500	509500	509500
553500	553500	553500	553500	553500	553500	553500	553500	553500	553500
569500	569500	569500	569500	569500	569500	572000	572000	572000	573500
589500	589500	589500	589500	589500	589500	589500	589500	589500	589500
589500	589500	589500	589500	589500	589500	592000	592000	592000	593500
591500	591500	591500	591500	591500	591500	592000	592000	592000	593500
611500	611500	611500	611500	611500	611500	611500	611500	611500	611500
613000	613000	613000	613000	613000	613000	615500	615500	615500	617000
614000	614000	614000	614000	614000	614000	615500	615500	615500	617000
621500	621500	621500	621500	621500	621500	621500	621500	621500	621500
626500	626500	626500	626500	626500	626500	626500	626500	626500	626500
628000	628000	628000	628000	628000	628000	628000	628000	628000	629000
635500	636500	636500	636500	638000	638000	638000	638000	638000	638000
645000	645000	645000	645000	645000	645000	645000	645000	645000	646500
655500	655500	655500	655500	655500	655500	656000	656000	656000	657500

```
    668500    669000    669000    669000    670500    670500    670500    670500    670500    670500
    689000    689000    689000    689000    689000    689000    690000    690000    690000    691500
```

Keuntungan =

```
    691500
```

```
  Columns 1 through 20

    1    1    1    1    1    1    1    1    1    1    0    1    1    1    0    1    1    1    1    0

  Columns 21 through 37

    1    1    1    1    1    0    1    1    0    1    1    1    1    1    1    1    1
```

Jumlah beban yang di angkut

beban =

```
   499
```