PENYELESAIAN MASALAH *INTEGER KNAPSACK* DENGAN ALGORITMA *DYNAMIC PROGRAMMING* DAN ALGORITMA *BRANCH AND BOUND*

SKRIPSI

oleh

Ahmad Budi Prasetya
NIM 091810101019

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2015

PENYELESAIAN MASALAH *INTEGER KNAPSACK* DENGAN ALGORITMA *DYNAMIC PROGRAMMING* DAN ALGORITMA *BRANCH AND BOUND*

SKRIPSI

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

Oleh :

Ahmad Budi Prasetya
NIM 091810101019

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2015

PERSEMBAHAN

Skripsi ini saya persembahkan untuk:

1. Almarhum Abah Syufa'at dan Ibuk Siti Munawaroh yang senantiasa memberi doa, semangat, dan kasih sayang;
2. semua kakak dan ipar serta keponakan yang selalu memberi semangat;
3. seluruh guru dan dosen sejak taman kanak-kanak hingga perguruan tinggi yang telah membimbing dan membagi ilmu dengan tulus;
4. almamater Jurusan Matematika FMIPA Universitas Jember, SMA Negeri 2 Jember, SMP Negeri 1 Ambulu, SD Negeri Karanganyar 1, TK Dharma Wanita Karanganyar;

MOTTO

"Bacalah dan Tuhanmulah yang Maha Pemurah. Yang mengajar (manusia) dengan perantara pena. Dia mengajar pada manusia apa yang tidak ia ketahui"
(terjemahan Q.S. Al-Alaq: 3-5)[*)]

"Seorang yang terpelajar harus juga berlaku adil sudah sejak dalam pikiran apalagi dalam perbuatan"
(Pramoedya Ananta Toer)[**)]

[*)] Departemen Agama Republik Indonesia. 2005. Alhikmah, Al-Qur'an dan Terjemahannya. Bandung : CV Penerbit Diponegoro.

[**)] Toer, P.A. 2011. *Bumi Manusia*. Jakarta : Lentera Dipantara.

PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama : Ahmad Budi Prasetya

NIM : 091810101019

menyatakan dengan sesungguhnya bahwa karya ilmiah yang berjudul "Penyelesaian Masalah *Integer Knapsack* dengan Algoritma *Dynamic Programming* dan Algoritma *Branch and Bound*" adalah benar-benar hasil karya sendiri, kecuali kutipan yang sudah saya sebutkan sumbernya, belum pernah diajukan dalam institusi manapun, 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, September 2015
Yang menyatakan,

Ahmad Budi Prasetya

NIM. 091810101019

SKRIPSI

PENYELESAIAN MASALAH *INTEGER KNAPSACK*
DENGAN ALGORITMA *DYNAMIC PROGRAMMING*
DAN ALGORITMA *BRANCH AND BOUND*

Oleh

Ahmad Budi Prasetya
NIM 091810101019

Pembimbing

Dosen Pembimbing Utama : Ahmad Kamsyakawuni, S.Si., M.Kom

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

PENGESAHAN

Skripsi berjudul "Penyelesaian Masalah *Integer Knapsack* dengan Algoritma *Dynamic Programming* dan Algoritma *Branch and Bound*" telah diuji dan disahkan pada:

Hari, tanggal :

Tempat : Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember

Tim penguji

Dosen Pembimbing Utama,	Dosen Pembimbing Anggota,
Ahmad Kamsyakawuni, S.Si., M.Kom NIP. 197211291998021001	Kusbudiono, S.Si., M.Si. NIP. 197211291998021001
Penguji I,	Penguji II,
Kosala Dwidja Purnomo, S.Si., M.Si. NIP. 196908281998021001	Drs. Rusli Hidayat, M.Sc NIP. 1966101219930310

Mengesahkan,
Dekan

Prof. Drs. Kusno, DEA, Ph.D.
NIP 196101081986021001

RINGKASAN

Penyelesaian Masalah *Integer Knapsack* dengan Algoritma *Dynamic Programming* dan Algoritma *Branch and Bound*; Ahmad Budi Prasetya; 091810101019; 2015; 45 halaman; Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember.

Permasalahan *Knapsack* merupakan suatu permasalahan dalam menentukan pemilihan obyek yang masing-masing mempunyai bobot atau berat (*weight*) untuk dimuat dalam sebuah media penyimpanan tanpa melebihi kapasitas media penyimpanan tersebut sehingga diperoleh hasil yang optimal dan keuntungan maksimum (*profit*). Permasalahan *Integer Knapsack* adalah menentukan objek mana saja yang harus dimuat atau tidak dimuat sama sekali dalam media penyimpanan. Objek yang dimasukkan kedalam media pengangkutan dalam satu item dimensinya harus dimasukkan semua atau tidak sama sekali. Objek dari permasalahan ini adalah bagaimana memilih objek-objek yang dimasukkan ke dalam media pengangkutan sehingga tidak melebihi kapasitas dari media pengangkutan namun memaksimalkan total keuntungan yang diperoleh.

Algoritma yang digunakan untuk menyelesaiakan permasalahan *Integer Knapsack* tersebut, diantaranya adalah algoritma *Dynamic Programming* dan algoritma *Branch and Bound*. Oleh karena itu penulis tertarik untuk menerapkan algoritma tersebut untuk menentukan barang yang akan diangkut serta mencari nilai keuntungan maksimum dengan berat total barang tidak melebihi kapasitas berat media pengangkutan. Kompleksitas waktu dari kedua algoritma tersebut juga akan menjadi bahan pertimbangan ketika menentukan algoritma mana yang lebih baik untuk diterapkan pada masalah *Integer Knapsack.*

Pada penelitian ini dilakukan di toko bangunan UD. Toda yang terletak di Kecamatan Ambulu Kabupaten Jember. Pengambilan data dilakukan dengan metode wawancara dan data yang diambil barupa data harga beli, harga jual, dan banyaknya barang. Data tersebut kemudian diolah menggunakan algoritma *Dynamic Programming* dan algoritma *Branch and Bound* dengan bantuan program *software* matematika yaitu MATLAB. Dari hasil penelitian dengan bantuan program menunjukkan: (1) Keuntungan maksimum dihasilkan menggunakan algoritma *Dynamic Programming* adalah sebesar Rp. 13.387.500,- dengan berat barang yang diangkut 8.000 kg dan waktu komputasi 11,752 detik. (2) Keuntungan maksimum dihasilkan menggunakan algoritma *Branch and Bound* adalah sebesar Rp. 10.410.000,- dengan berat barang yang diangkut 7.990 kg dan waktu komputasi 0,035285 detik. Berdasarkan kompleksitas waktu pada pengerjaan mengunakan algoritma *Dynamic Programming* adalah $O(n^3)$ yang berarti kelompok kubik. Sedangkan algoritma *Branch and Bound* adalah $O(n^2)$ yang berarti kuadratik. Sehingga algoritma *Branch and Bound* lebih efisien dalam penyelesaian masalah *Integer knapsack*.

Dari hasil diatas dapat disimpulkan bahwa algoritma *Dynamic Programming* lebih optimal untuk mencari keuntungan dari pada algoritma *Branch and Bound* untuk menyelesaikan masalah *Integer Knapsack*. Serta waktu komputasi dan kompleksitas waktu yang dibutuhkan lebih efisien. Karena algoritma *Dynamic Programming* bersifat global untuk melakukan pencarian solusinya berbeda dengan algoritma *Branch and Bound* pencariannya bersifat lokal.

PRAKATA

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik, dan karuniaNya sehingga skripsi yang berjudul "Penyelesaian Masalah *Integer Knapsack* dengan Algoritma *Dynamic Programming* dan Algoritma *Branch and Bound*" dapat terselesaikan. Skripsi ini disusun untuk memenuhi salah satu syarat dalam menyelesaikan pendidikan strata 1 (S1) di Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember. Sholawat dan salam semoga tercurahkan keharibaan beliau nabi Muhammad SAW yang telah menjadi pembawa rahmatan lil'alamin.

Penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak, baik secara langsung maupun tidak langsung. Oleh karena itu, penulis menyampaikan terima kasih kepada:

1. Prof. Drs. Kusno, DEA., Ph.D. selaku Dekan Fakultas MIPA Universitas Jember dan Kosala Dwidja Purnomo, S.Si., M.Si. selaku Ketua Jurusan Matematika Fakultas MIPA Universitas Jember yang telah memberikan fasilitas-fasilitas dalam tahap perkuliahan;
2. Ahmad Kamsyakawuni, S.Si., M.Kom. selaku Dosen Pembimbing Utama dan Kusbudiono, S.Si., M.Si. selaku Dosen Pembimbing Anggota yang telah memberikan bimbingan dan bantuan untuk pengerjaan skripsi ini;
3. Kosala Dwidja Purnomo, S.Si., M.Si. selaku Dosen Penguji I dan Drs. Rusli Hidayat, M.Sc. selaku Dosen Penguji II yang telah memberikan kritik dan saran yang membangun untuk penyempurnaan skripsi ini;
4. Kiswara Agung Santoso, S.Si., M.Kom. dan Mohammad Ziaul Arif, S.Si., M.Si. selaku Dosen Pembimbing Akademik yang telah membimbing dalam pemilihan matakuliah;

5. seluruh dosen dan karyawan Jurusan Matematika Fakultas MIPA Universitas Jember yang telah memberikan ilmu serta membantu selama proses perkuliahan berlangsung;
6. Almarhum Abah Syufa'at dan Ibuk Siti Munawaroh yang selalu memberi doa dan dukungan baik lahir maupun batin;
7. Kakak-kakak dan ipar serta keponakan yang sudah memberi semangat dan dukungan;
8. Zaidar Rahmi Martiningrum yang telah sabar dan memberi semangat, dukungan, perhatian, dan bantuan moral serta materi selama proses perkuliahan hingga selesai;
9. Rekan Sri Astutik, Jauharin Insiyah, Hadi Siswanto, M. Arif Riyanto, dan Zainul Anwar yang senantiasa memberi bantuan dan saran dalam proses menyelesaikan tugas akhir;
10. Kawan Cahyo Hadi S., Medhy Amalia, Laily Fauziah, dan Armada Eka F. yang senantiasa memberi dukungan dalam proses menyelesaikan tugas akhir;
11. Keluarga besar Bani Anshori dan Keluarga besar warung Al-Qahwah yang senantiasa menemani dan memberi dukungan;
12. Teman-teman MALINC'09, MATHGIC'10 dan BATHICS'12 yang selalu memberikan dukungan dalam hal pembelajaran selama kuliah;
13. Kawan LPMM ALPHA serta seluruh LPM dalam naungan PPMI Kota Jember yang sudah memberikan wawasan dan menjadi teman diskusi;
14. Semua pihak yang membantu terselesainya skripsi ini.

Penulis menyadari bahwa dalam menyusun skripsi ini masih terdapat kekurangan baik isi maupun susunannya. Oleh karena itu, penulis mengharapkan saran dan kritik demi penyempurnaan skripsi ini. Akhirnya penulis berharap semoga skripsi ini dapat memberi manfaat dan sumbangan bagi pembaca.

DAFTAR ISI

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Suatu permasalahan biasanya menuntut solusi yang optimum agar dapat diperoleh keuntungan yang sebesar besarnya. Salah satu model untuk merepresentasikan suatu permasalahan adalah Program Linier (*Linear Programming*). Program linier itu sendiri merupakan suatu metode yang dapat digunakan untuk menyelesaikan permasalahan pengalokasian sumber daya yang terbatas. Masalah pengalokasian sumber daya ini akan muncul apabila seseorang harus menentukan tingkat kegiatan-kegiatan yang akan dilakukan, dimana masing-masing kegiatan membutuhkan sumber daya yang sama sedangkan jumlahnya terbatas (Ridho, 2014). Masalah pengalokasian sumber daya merupakan hal yang sering dijumpai dalam pekerjaan sehari-hari, seperti optimasi, pemilihan proyek, penjadwalan, pendistribusian dan lain-lain.

Permasalahan *Knapsack* merupakan suatu permasalahan dalam menentukan pemilihan obyek yang masing-masing mempunyai bobot atau berat (*weight*) untuk dimuat dalam sebuah media pengangkutan tanpa melebihi kapasitas media pengangkutan tersebut sehingga diperoleh hasil yang optimal dan keuntungan maksimum (*profit*). *Knapsack* dapat diilustrasikan seperti cara memasukkan beberapa barang ke dalam suatu kantong (*Knapsack*). Kapasitas kantong ini memiliki keterbatasan bobot sehingga barang yang dimasukkan kedalam kantong tidak melebihi batas dan diperoleh keuntungan yang optimal. Permasalahan *knapsack* terbagi menjadi tiga, yaitu *Integer Knapsack, Bounded Knapsack,* dan *Unbounded Knapsack*. Permasalahan *Integer Knapsack* adalah menentukan objek mana saja yang harus dimuat atau tidak dimuat sama sekali dalam satu item pada media pengangkutan. Permasalahan *Bounded Knapsack* adalah menentukan berapa bagian dari masing-masing objek yang akan dimuat dalam media pengangkutan, sedangkan pada

Unbounded Knapsack persoalannya adalah penentuan untuk barang yang tidak terbatas (Martello, 2006).

Penelitian tentang masalah *Knapsack* pernah dilakukan oleh Paryati (2009) khususnya permasalahan *Integer Knapsack* yang diselesaikan dengan menggunakan algoritma *Greedy*. 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. Penelitian lain dengan membandingkan algoritma pernah dilakukan yaitu penerapan algoritma *Greedy* dan *Dynamic Programming* pada permasalahan *Integer Knapsack* (Arista, 2013). Dari penelitian tersebut disimpulkan bahwa algoritma D*ynamic Programming* lebih efisien dalam perhitungan dari pada algoritma *Greedy*. Pada penelitian lain algoritma *Dynamic Programming* dibandingkan dengan algoritma *Harmony Search* oleh Ridho (2014). Pada penelitian tersebut disimpulkan bahwa algoritma *Dynamic Programming* mempunyai langkah perhitungan lebih sederhana. Sedangkan pemecahan masalah *Knapsack* dengan menggunakan algoritma *Branch and Bound* pernah dilakukan oleh Permata (2007). Algoritma ini mampu mencapai solusi yang optimum pada masalah *Knapsack*. Selain itu Muthohar (2008) juga pernah menyelesaikan untuk masalah *Integer Knapsack* dengan algoritma *Branch And Bound*.

Pada skripsi ini penulis tertarik untuk membandingkan algortima *Dynamic Programming* dengan algoritma *Branch and Bound* dalam menyelesaikan permasalahan *Knapsack*, khususnya *Integer Knapsack.* Berdasarkan penelitian sebelumnya, penulis mengangkat tema yang akan diteliti yaitu "Penyelesaian masalah *Integer Knapsack* menggunakan algoritma *Dynamic Programming* dan algoritma *Branch And Bound*".

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang diatas, maka dapat dirumuskan permasalahan yang akan dibahas dalam skripsi ini adalah :

a. bagaimana penyelesaian permasalahan *Integer Knapsack* menggunakan Algoritma *Dynamic Programming*?
b. bagaimana penyelesaian permasalahan *Integer Knapsack* menggunakan Algoritma *Branch and Bound*?
c. bagaimana perbandingan kedua algoritma tersebut berdasarkan hasil maupun langkah perhitungan yang dibutuhkan?

1.3 Batasan Masalah

Adapun batasan masalah dalam skripsi ini adalah:

a. tingkat permintaan konsumen untuk masing-masing barang dalam satu item diasumsikan sama.
b. barang yang dibeli dalam satu item diangkut semua atau tidak sama sekali.
c. volume barang yang akan diteliti disortir terlebih dahulu.
d. hanya barang yang mencantumkan nilai berat (*weight*) yang akan diteliti.

1.4 Tujuan

Berdasarkan rumusan masalah diatas, maka didapat tujuan sebagai berikut:

a. mengetahui penyelesaian optimum pada permasalahan *Integer Knapsack* menggunakan algoritma *Dynamic Programming*.
b. mengetahui penyelesaian optimum pada permasalahan *Integer Knapsack* menggunakan algoritma *Branch and Bound*.
c. mengetahui kompleksitas waktu pada algoritma *Dynamic Programming* dan algoritma *Branch and Bound.*
d. membuat program penyelesaian masalah tersebut menggunakan software Matlab.
e. membandingkan algoritma *Dynamic Programming* dan algoritma *Branch and Bound* dalam penyelesaian permasalahan *Integer Knapsack.*

1.5 Manfaat

Manfaat yang diperoleh dari penelitian ini adalah memberikan pertimbangan dan solusi menentukan barang yang akan diangkut pada kapasitas bobot yang terbatas agar memperoleh hasil yang maksimum.

BAB 2. TINJAUAN PUSTAKA

2.1 Program Linier

Program linier dengan teknik optimasi linier adalah upaya menyelesaikan suatu masalah, dimana semua fungsi matematika yang digunakan dalam program linier merupakan fungsi linier. Program linier adalah salah satu teknik dalam riset operasi untuk memecahkan persoalan optimasi dengan menggunakan persamaan dan ketidaksamaan linier dalam rangka untuk mencari pemecahan yang optimum dengan memperhatikan pembatasan-pembatasan yang ada.

Optimasi merupakan suatu proses pencapaian yang ideal atau optimal pada suatu permasalahan yang berhubungan dengan keputusan yang terbaik, maksimum, minimum serta memberikan cara penentuan solusi yang memuaskan. Nilai optimal yang didapat dalam suatu optimasi dapat berupa besaran waktu, panjang, suhu, dan lain-lain. Berikut ini adalah beberapa persoalan yang memerlukan optimasi, adalah:

a. penentuan pemilihan barang pada masalah *knapsack*;
b. penentuan lintasan terpendek pada masalah *travelling salesman problem*;
c. penentuan jalur kendaraan umum agar semua lokasi dapat dijangkau;
d. penentuan jumlah pekerja seminimal mugkin pada suatu proses produksi.

2.2 Definisi *Knapsack*

Knapsack merupakan suatu permasalahan bagaimana memilih objek dari sekian banyak objek dan berapa besar bobot objek tersebut akan diangkut sehingga diperoleh suatu pengangkutan yang optimal. Dari situ *knapsack* dapat diilustrasikan sebagai suatu kantong atau media pengangkutan. Selain itu masalah *knapsack* dapat digambarkan sebuah wadah dengan kapasitas M akan diisi dengan benda sebanyak n masing-masing benda memiliki keuntungan berupa *profit* (p) dan berat berupa *weight*

(w) (Kellerer, 2004). Sehingga dapat diambil kesimpulan bahwa masalah *knapsack* dapat digambarkan sebagai proses pemilihan dari berbagai macam barang yang bernilai mengingat bahwa wadah memiliki kapasitas berat yang terbatas.

Permasalahan *knapsack* adalah permasalahan optimasi yang mendasar. Dalam kehidupan sehari-hari permasalahan ini dapat dijumpai ketika hendak memilih sebuah solusi atas suatu permasalahan optimasi kombinatorik. Sehingga permasalahan *knapsack* itu merupakan permasalahan optimasi kombinatorial dimana satu set item, masing–masing memiliki beban dan nilai, menentukan jumlah dari setiap item yang akan disertakan dalam koleksi sehingga beban total item kurang dari atau sama dengan beban kapasitas dan dengan nilai total sebesar mungkin.

Permasalahan *knapsack* biasanya dihadapi pada perusahaan yang bergerak dibidang jasa pengangkutan (pengiriman) barang serta media pengangkutan memiliki kapasitas total bobot barang yang diangkut terbatas. Dari permasalahan tersebut dituntut untuk memperoleh keuntungan maksimal dalam mengangkut barang dengan tidak melebihi kapasitas yang ada.

2.2.1 Macam-macam masalah *Knapsack*

Knapsack memiliki beberapa jenis persoalan, yaitu:

a. *Integer knapsack* (*Knapsack* 0-1)

   Objek yang dimasukkan ke dalam media pengangkutan dimensinya harus dimasukkan semua atau tidak sama sekali.

b. *Bounded knapsack* (*Knapsack* terbatas)

   Objek yang dimasukkan ke dalam media pengangkutan dimensinya bisa di masukkan sebagian atau seluruhnya.

c. *Unbounded knapsack* (*Knapsack* tak terbatas)

   Jumlah objek yang dimasukkan ke dalam media pengangkutan macamnya tidak terbatas.

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

2.2.2 *Integer Knapsack*

Permasalahan *integer knapsack*, objek yang dimasukkan kedalam media pengangkutan dalam satu item dimensinya harus dimasukkan semua atau tidak sama sekali. Sehingga dapat disimpulkan bahwa *integer knapsack* merupakan permasalahan program bilangan bulat yang memiliki satu kendala tunggal, sehingga pada modelnya ditambahkan batasan untuk variabel keputusan yang dihasilkan harus bernilai bulat. Diberikan n buah objek dan sebuah media pengangkutan yang memiliki daya tampung maksimal senilai M. Setiap benda memiliki bobot (w_i) dengan nilai keuntungan *profit* (p_i). Objek dari permasalahan ini adalah bagaimana memilih objek-objek yang dimasukkan ke dalam media pengangkutan sehingga tidak melebihi kapasitas dari media pengangkutan namun memaksimalkan total keuntungan yang diperoleh (Ridho, 2014).

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

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

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

Dari penjelasan diatas permasalahan *integer knapsack* dalam model matematika dapat dirumuskan sebagai berikut:

Fungsi tujuan Maksimal/Minimal:

$$Z = \sum_{i=1}^{n} p_i \, x_i \quad (2.1)$$

Kendala:

$$z = \sum_{i=1}^{n} w_i x_i \leq M \quad (2.2)$$

dimana x_i= 0 atau 1, i = 1, 2, …,n,

dengan,

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 pengangkutan (*knapsack*),

x_i = banyaknya barang jenis ke-i.

2.3 Algoritma

Algoritma adalah suatu urutan langkah-langkah penyelesaian masalah yang disusun secara logis dan sistematis. Algoritma ditulis dengan notasi khusus yang mudah dimengerti dan notasi dapat diterjemahkan menjadi bahasa pemrograman. Algoritma akan memerlukan masukan (*input*) tertentu untuk memulainya dan akan menghasilkan keluaran (*output*) tertentu pada akhirnya. Suatu algoritma harus menghasilkan *output* yang efektif dalam waktu yang relatif singkat dan penggunaan memori yang relatif sedikit dengan langkah yang berhingga dan prosesnya berakhir baik dalam keadaan diperoleh suatu solusi.

2.3.1 Efisiensi Algoritma

Secara umum algoritma yang dapat menyelesaikan permasalahan dalam waktu yang singkat memiliki tingkat kompleksitas yang rendah. Algoritma yang

membutuhkan waktu yang lama dalam penyelesaian permasalahan mempunyai tingkat kompleksitas yang tinggi (Nurhayati, 2009)

Kemangkusan (efisiensi) sebuah algoritma dapat digunakan untuk menilai algoritma terbaik yang dapat ditentukan dengan menggunakan:

a. Kompleksitas waktu $T(n)$ adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari ukuran masukan n.
b. Kompleksitas ruang $S(n)$ adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari ukuran masukan n.

(Munir, 2010)

Algoritma yang bagus adalah algoritma yang mangkus (efisien). Algoritma dapat dikatakan mangkus jika kebutuhan waktu dan ruang untuk tahapan komputasinya berjumlah sedikit. Kemangkusan algoritma juga berguna dalam membandingkan algoritma yang lebih baik untuk menyelesaikan permasalahan yang sama.

2.3.2 Notasi *Big-O*

Notasi *Big-O* adalah notasi matematika yang digunakan untuk menggambarkan tingkah laku asimtotik dari fungsi. Notasi ini berguna untuk menganalisa kompleksitas waktu dari suatu algoritma. Terdapat beberapa kelompok algoritma berdasarkan notasi *Big-O* yang dihasilkan dari perhitungan kompleksitas algoritma dapat dilihat pada Tabel 2.1.

Tabel 2.1 Kelompok Algoritma Berdasarkan Notasi *Big-O*

Kelompok Algoritma	Nama
$O(1)$	Konstan
$O(\log n)$	Logaritmik
$O(n)$	Linear
$O(n \log n)$	$n \log n$
$O(n^2)$	Kuadratik
$O(n^3)$	Kubik

Kelompok Algoritma	Nama
O(2^n)	Eksponensial
O($n!$)	Faktorial

Definisi:

$T(n) = O(f(n))$, artinya $T(n)$ berorde paling besar $f(n)$ bila terdapat konstanta C dari n_0 sehingga $T(n) \leq C(f(n))$. Untuk $n = n_0$.

Jika sebuah algoritma mempunyai kompleksitas waktu $O(f(n))$, maka jika n dibuat semakin besar waktu yang dibutuhkan tidak akan pernah melebihi satu tahapan C dikali $f(n)$ (Wibowo, 2011).

2.4 Pengertian *Dynamic Programming*

Dynamic programming yang ditemukan oleh Richard Bellman pada tahun 1953 merupakan suatu metode penyelesaian masalah dimana solusi persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan. *Dynamic programming* merupakan metode yang secara umum memecahkan masalah optimasi yang melibatkan urutan keputusan, solusi optimal dari masalah yang asli dapat ditemukan dari solusi optimal subproblem (Lew, 2007).

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).

2.4.1 Unsur-unsur *Dynamic Programming*

Proses *dynamic programming* harus dipahami terlebih dahulu unsur-unsur yang membentuknya agar dapat dimengerti. Menurut Taha (1996) ada 5 unsur yang dapat membentuk *dynamic programming*, yaitu:

a. Tahap

   Tahap dapat diartikan sebagai bagian dari masalah pada *dynamic programming* yang mengandung variabel keputusan yang akan dipilih salah satu atau lebih yang paling optimal.

b. Keadaan

   Keadaan merupakan kondisi yang memungkinkan pada suatu tahap tertentu yang menunjukan kaitan antara tahap satu dengan tahap berikutnya.

c. Variabel keputusan

   Variabel keputusan adalah variabel yang menguraikan secara lengkap keputusan-keputusan yang dibuat.

d. Nilai fungsi

   Nilai fungsi adalah nilai yang ditentukan dari variabel keputusan.

e. Keputusan optimal

   Keputusan optimal adalah keputusan yang merupakan solusi terbaik dari masalah.

2.4.2 Prosedur Perhitungan Algoritma *Dynamic Programming* pada Permasalahan *Integer Knapsack*

Teknik perhitungan *dynamic programming* terutama didasarkan pada prinsip optimasi rekursi (bersifat pengulangan) yang diketahui sebagai prinsip optimalisasi. Prinsip ini mengandung arti bahwa bila dibuat keputusan multi tahap mulai pada tahan tertentu, kebijaksanaan optimal untuk tahap-tahap selanjtunya tergantung pada

ketetapan tertentu tersebut. Jika pada suatu ketika proses menghitung perolehan optimal sampai pada tahap-n maka selesailah prosedur perhitungan berdasarkan pendekatan *dynamic programming*. Selanjutnya tinggal menentukan keputusan optimal untuk seluruh persoalan. Untuk itu, dimulai pada keputusan optimal tahap-n dan kemudian menelusuri keputusan optimal pada tahap-tahap sebelumnya.

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 rekursif maju seperti pada Gambar 2.1.

Tahap 1 → Tahap 2 → Tahap 3 → ⋯ → Tahap n

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 rekursif mundur seperti pada Gambar 2.2.

Tahap 1 ← ⋯ ← Tahap n-2 ← Tahap n-1← Tahap n

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.

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 (Munir, 2004).

Permasalahan *integer knapsack* dengan menggunakan strategi *dynamic programming* meliputi beberapa tahap:

a. tahap (k) adalah proses memasukkan barang ke dalam media pengangkutan (M)
b. status (y) menyatakan kapasitas muat media pengangkutan (M) yang tersisa setelah memasukkan barang pada tahap sebelumnya.

Dari tahap pertama dimasukkan objek ke-1 ke dalam media pengangkutan (M) untuk setiap satuan kapasitas media pengangkutan (M) sampai batas kapasitas maksimum. Karena kapasitas media pengangkutan (M) adalah bilangan bulat, maka pendekatan ini praktis.

Berikut ini contoh langkah *dynamic programming* pada masalah *integer knapsack*:

a. misalkan ketika memasukkan objek pada tahap k, kapasitas muat M sekarang adalah $y - w_k$,
b. untuk mengisi kapasitas sisanya, kita menerapkan prinsip optimalitas dengan mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas sisa $y - w_k$ yaitu $f_{k-1}(y - w_k)$,

c. 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,$f_{k-1}(y)$,

d. 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, maka objek yang ke-k dimasukkan.

Relasi rekurens untuk persoalan ini adalah

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

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

$f_k(y) = \max\{f_{k-1}(y), p_k + f_{(k-1)}(y-w_k)\}, k = 1, 2, \dots, n$ (2.3)

dengan,

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

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

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

2.5 Pengertian *Branch and Bound*

Branch and bound adalah metode algoritmik general untuk menemukan solusi optimal dari berbagai masalah optimasi, khususnya pada diskrit dan optimasi kombinatorial. Dasarnya adalah pendekatan enumerasi dengan cara mematikan (*pruning*) *search space* yang tidak mengarah ke solusi. Metode ini pertama kali diperkenalkan oleh A.H. Land dan A.G. Doig pada tahun 1960. (Munawar, 2007). *Branch and bound* prosedur memerlukan dua alat. Yang pertama adalah cara untuk men-*cover* regional *feasible* dengan beberapa subregional *feasible* yang lebih kecil. Ini disebut *branching*. Karena pemanggilan prosedur dilakukan berulang-ulang secara rekursif**,** maka semua subregional akan secara alami membentuk sturktur pohon. Yang

kedua adalah *bounding*, yaitu cara untuk mencari nilai batas atas atau batas bawah untuk solusi optimal di dalam subregional *feasible*.

Algoritma *Branch and Bound* merupakan metode pencarian di dalam ruang solusi secara sistematis. Ruang solusi diorganisasikan ke dalam pohon ruang status. Pohon ruang status tersebut dibangun dengan skema BFS (*Breadth First Search*). Untuk mempercepat pencarian ke simpul solusi, maka setiap simpul diberi sebuah nilai ongkos (*cost*). Simpul berikutnya yang akan diekspansi adalah simpul yang memiliki ongkos paling kecil diantara simpul-simpul hidup lainnya. Sedangkan simpul lainnya dimatikan.

2.5.1 Unsur-unsur *Branch and Bound*

Branch and Bound juga merupakan salah satu strategi yang dapat digunakan dalam pencarian solusi optimum dari permasalahan *knapsack*. Dengan penentuan keuntungan maksimal pada tiap simpulnya, proses pencarian akan membawa pada solusi yang optimum. Permasalahan ini akan sampai pada keadaan dimana tidak ada lagi simpul yang dapat dibangkitkan kerena telah melewati batas kapasitas daya angkut membuat kita harus menentukan solusi optimum dengan membandingkan lintasan-lintasan mana yang berakhir didaun pada pohon yang akan manghasilkan keuntungan paling besar maka objek-objek tersebutlah yang akan dipilih.

Ada tiga unsur dalam algoritma ini, yaitu:

a. Fungsi Pembatas (*Bounding*): fungsi yang disediakan *subspace* dari ruang solusi dengan batas rendah untuk nilai solusi terbaik yang diperoleh dalam *subspace*.
b. Strategi Pencarian: suatu strategi untuk menyeleksi tiap-tiap node yang dihasilkan dan mendapatkan node yang optimum.
c. Metode Percabangan (*Branching*): suatu metode yang diaplikasikan jika *subspace* setelah diperiksa tidak dapat dibatalkan, karena itu pembagian *subspace* kedalam dua atau lebih *subspace* untuk diperiksa dalam sub rangkaian iterasi.

2.5.2 Prosedur Perhitungan Algoritma *Branch and Bound* Pada Permasalahan *Integer Knapsack*

Proses pencarian pada metode ini menggunakan skema BFS. BFS dikenal sebagai pencarian melebar dalam pohon. Pada skema BFS simpul yang dibangkitkan terlebih dahulu adalah simpul yang bertetanggaan dengan simpul akar.

Proses pemilihan simpul-E, simpul yang akan di*expand*, pada algoritma *branch and bound* tidak seperti pada metode BFS murni. Pada BFS murni pemilihan simpul-E berdasarkan urutan pembangkitan sedangkan pada algoritma *branch and bound* simpul-E dipilih berdasarkan niliai ongkos. Setiap simpul hidup diasosiasikan dengan sebuah ongkos yang menyatakan nilai batas (*bound*).

Pada algoritma *branch and bound*, pencarian ke simpul solusi dapat dipercepat dengan memilih simpul hidup berdasarkan nilai ongkos (*cost*). Setiap simpul hidup diasosiasikan dengan sebuah ongkos yang menyatakan nilai batas (*bound*) (Permata, 2007). Pada prakteknya, nilai batas untuk setiap simpul umumnya berupa taksiran atau perkiraan. Fungsi heuristik untuk menghitung taksiran nilai tersebut dinyatakan secara umum sebagai berikut :

$$c(i) = f(i) + g(i) \qquad (2.4)$$

dengan,

$c(i)$ = ongkos untuk simpul *i*

$f(i)$ = ongkos mencapai simpul *i* dari akar

$g(i)$ = ongkos mencapai simpul tujuan dari simpul akar *i* (perkiraan)

Nilai digunakan untuk mengurutkan pencarian. Simpul berikutnya yang dipilih untuk diekspansi adalah simpul yang memiliki minimum (Simpul-E). Strategi memilih simpul-E seperti ini dinamakan strategi pencarian berdasarkan biaya terkecil (*least cost search*).

Prinsip dari algoritma *branch and bound* ini adalah :

a. masukkan simpul akar ke dalam antrian *Q*. Jika simpul akar adalah simpul solusi (*goal node*), maka solusi telah ditemukan. *Stop*.

b. jika *Q* kosong, tidak ada solusi . *Stop*.

c. jika *Q* tidak kosong, pilih dari antrian *Q* simpul *i* yang mempunyai *cost* paling kecil. Jika terdapat beberapa simpul *i* yang memenuhi, pilih satu secara sembarang.

d. jika simpul *i* adalah simpul solusi, berarti solusi sudah ditemukan, *stop*. Jika simpul *i* bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika *i* tidak mempunyai anak, kembali ke langkah b.

e. untuk setiap anak *j* dari simpul *i*, hitung (*j*), dan masukkan semua anak-anak tersebut ke dalam antrian *Q*.

f. kembali ke langkah b.

BAB 3. METODE PENELITIAN

Penelitian pada tugas akhir dengan menggunakan algoritma *Dynamic Programming* dan *Branch and Bound* yang diterapkan pada toko bangunan UD. Toda yang berlokasi di Kecamatan Ambulu Kabupaten Jember. Adapun langkah-langkah penelitian akan dipaparkan pada bab ini.

3.1 Data Penelitian

Data yang digunakan untuk penelitian pada tugas akhir ini adalah sejumlah data barang yang dijual oleh UD. Toda kepada para pelanggan. UD. Toda adalah toko bahan bangunan yang terletak di Kecamatan Ambulu Kabupaten Jember. Pengambilan data dilakukan dengan cara wawancara langsung kepada pemilik UD. Toda. Data yang diambil menyangkut tentang barang-barang yang dibeli untuk dijual kembali kepada para pelanggan.

Adapun beberapa variabel yang mempengaruhi pada penelitian ini adalah keuntungan (*profit*) dan berat (*weight*) pada masing-masing barang, banyaknya barang serta kapasitas berat daya angkut maksimal. Sehingga peneliti perlu melakukan pengidentifikasian terlebih dahulu terhadap data mentah yang diterima.

Estimasi waktu pembelian barang dilakukan secara serentak dalam waktu yang sama dengan mengabaikan barang yang sudah habis terlebih dahulu. Sehingga proses pembelian barang dilakukan bersama-sama dengan menggunakan satu media pengangkutan dengan kapasitas daya angkut maksimum (M) adalah 8.000 kg.

3.2 Langkah-langkah Penelitian

Pada tugas akhir kali ini, langkah-langkah yang harus dilakukan untuk penyelesaian masalah *Integer Knapsack* dengan algoritma *Dynamic Programming* dan *Branch and Bound* adalah sebagai berikut.

a. Studi literatur

Langkah awal pada tugas akhir ini adalah melakukan studi dari berbagai literatur mengenai algoritma *Dynamic Programming* dan *Branch and Bound* serta permasalahan *Integer Knapsack.*

b. Pengumpulan data

Pada tugas akhir ini, data yang diperoleh dari toko bangunan UD. Toda yang berlokasi di Kecamatan Ambulu Kabupaten Jember.

c. Pengolahan data

1) Untuk mencari nilai keuntungan (p_i) dari masing-masing barang diperoleh dari selisih harga beli dan harga jual yang telah ditetapkan dengan menggunakan persamaan 2.1.
2) Untuk mencari berat (w_i) dari masing-masing barang diperoleh dengan cara mengalikan bobot per satuan dengan banyaknya barang dengan menggunakan persamaan 2.2.

d. Penerapan algoritma pada data yang sudah diolah.

1) Algoritma *Dynamic Programming*
   a) Menentukan struktur dari masalah, pada tahap ini menentukan variabel dari data kemudian diaplikasikan pada *Dynamic Programming.* Banyaknya barang *n* kemudian didefinisikan sebagai *i*, dengan kapasitas maksimum dinotasikan *M* dan tiap benda mempunyai variabel keputusan yang dinotasikan x_i.
   b) Menggunakan persamaan rekursif seperti persamaan 2.3. Dimana persamaan ini adalah untuk mencari kemungkinan pemilihan barang. Nilai *k* bergerak dari 1 sampai *n*, *y* bergerak dari 0 sampai kapasitas muat maksimal, dan *w* adalah berat barang ke-*k*.
   c) Menghitung nilai dari solusi optimal dengan memilih prosedur rekursif maju, yaitu menghitung nilai keuntungan barang pertama sampai terakhir.
   d) Menentukan keputusan optimal dengan cara memilih keuntungan yang paling maksimum pada setiap barang. Pada langkah ini nilai keputusan optimal

didapat dari perhitungan prosedur rekursif sehingga dapat dipilih nilai yang paling maksimum dari perhitungan yang dilakukan.

2) Algoritma *Branch and Bound*

a) Menentukan struktur dari masalah, pada tahap ini menentukan variabel dari data kemudian diaplikasikan pada *Branch and Bound*. Banyaknya barang *n* kemudian didefinisikan sebagai *i*, dengan kapasitas maksimum dinotasikan *M* dan tiap benda mempunyai variabel keputusan yang dinotasikan x_i.

b) Menggunakan persamaan heuristik seperti persamaan 2.4. Dimana persamaan ini adalah untuk mencari kemungkinan pemilihan barang. Nilai *i* bergerak dari 1 sampai *n*.

c) Menghitung nilai dari solusi optimal dengan cara menjumlahkan keuntungan dari akar sampai simpul tujuan.

e. Penentuan kompleksitas waktu dari algoritma *Dynamic Programming* dan algoritma *Branch and Bound*.

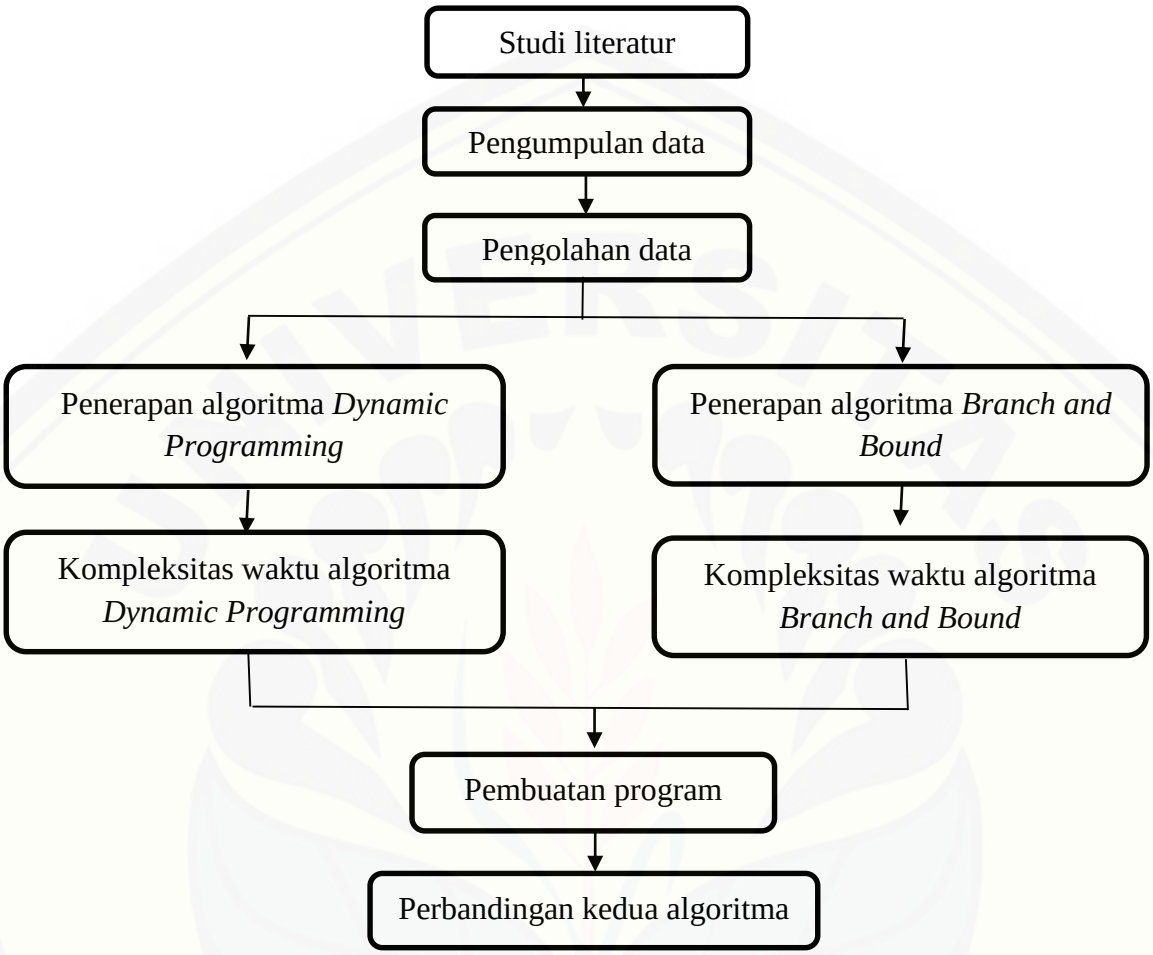
f. Pembuatan program

Sesuai algoritma yang digunakan pada langkah d, pembuatan program menggunakan bahasa pemrograman MATLAB. Program yang telah dibuat dijalankan menggunakan GUI.

g. Perbandingan kedua algoritma

Membandingkan hasil optimum dari perhitungan algoritma *Dynamic Programming* dan *Branch and Bound* dan kompleksitas waktu yang dibutuhkan pada masing-masing algoritma. Sehingga dapat dipilih algoritma mana yang lebih baik untuk menyelesaikan pada permasalahan yang sama.

Secara sistematis penjelasan dari setiap langkah penelitian tersebut ditunjukkan pada gambar 3.1.



Gambar 3.1 Skema Langkah-langkah Penelitian

BAB 4. HASIL DAN PEMBAHASAN

Pada bab ini akan dilakukan pembahasan masalah *Integer Knapsack* pada toko bangunan UD. Toda dengan menggunakan algoritma *Dynamic Programming* dan algoritma *Branch and Bound*. Pemilihan barang yang akan diangkut pada media pengangkutan yang terbatas agar mendapatkan keuntungan optimal. Dengan harapan UD. Toda tidak salah dalam pemilihan barang yang akan diangkut pada media pengangkutan yang terbatas sehingga dapat memperoleh keuntungan optimal. Algoritma yang digunakan untuk membantu proses perhitungan adalah algoritma *Dynamic Programming* dan algoritma *Branch and Bound* dengan bantuan *software* MATLAB.

4.1 Penerapan Algoritma pada Permasalahan *Integer Knapsack*

Penyelesaian optimasi pada permaslahan *Integer knapsack* menggunakan algoritma *Dinamic Programming* dan algoritma *Branch and Bound* 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.

Misalkan 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 4.1 Data berat dan keuntungan barang

Barang ke- i	w_i (Satuan Berat)	p_i (Satuan Harga)
1	2	80
2	1	60
3	1	75
4	2	40

4.1.1 Penyelesaian menggunakan algoritma *Dynamic Programing*

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.

- 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 4.2 Tahap 1 pencarian dengan *Dynamic Programming*

y	$f_0(y)$	$80 + f_0(y-2)$	$f_1(y)$	(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 4.3 Tahap 2 pencarian dengan *Dynamic Programming*

y	$f_1(y)$	$60 + f_1(y-1)$	$f_2(y)$	(x_1, x_2, x_3, x_4)
0	0	-∞	0	(0, 0, 0, 0)
1	0	60	60	(0, 1, 0, 0)
2	80	60	80	(1, 0, 0, 0)
3	80	140	140	(1, 1, 0, 0)
4	80	140	140	(1, 1, 0, 0)
5	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 4.4 Tahap 3 pencarian dengan *Dynamic Programming*

y	$f_2(y)$	$75 + f_2(y-1)$	$f_3(y)$	(x_1, x_2, x_3, x_4)
0	0	-∞	0	(0, 0, 0, 0)
1	60	75	75	(0, 0, 1, 0)
2	80	135	135	(0, 1, 1, 0)
3	140	155	155	(1, 0, 1, 0)
4	140	215	215	(1, 1, 1, 0)
5	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 4.5 Tahap 4 pencarian dengan *Dynamic Programming*

y	$f_3(y)$	$40 + f_3(y-2)$	$f_4(y)$	(x_1, x_2, x_3, x_4)
0	0	-∞	0	(0, 0, 0, 0)
1	75	-∞	75	(0, 0, 1, 0)
2	135	40	135	(0, 1, 1, 0)
3	155	155	155	(1, 0, 1, 0)
4	215	175	215	(1, 1, 1, 0)
5	215	195	215	(1, 1, 1, 0)

Dari langkah-langkah menentukan barang yang masuk solusi optimal di atas diperoleh nilai keuntungan terbesar adalah 215 dan barang yang diangkut adalah x_1, x_2, x_3 dengan total berat 5 (Satuan Berat).

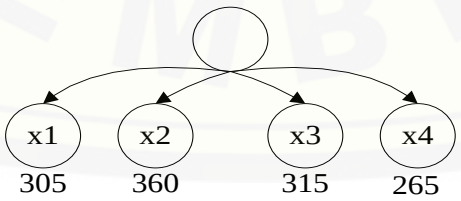
4.1.2 Penyelesaian menggunakan algoritma *Branch and Bound*

Algoritma *Branch and Bound* merupakan metode pencarian di dalam ruang solusi secara sistematis. Ruang solusi diorganisasikan ke dalam pohon ruang status. Untuk mempercepat pencarian ke simpul solusi, maka setiap simpul diberi sebuah nilai ongkos (*cost*). Simpul berikutnya yang akan diekspansi adalah simpul yang memiliki ongkos paling kecil diantara simpul-simpul hidup lainnya. Sedangkan simpul lainnya dimatikan.

$$C(i) = \{f(i) + (\frac{p_i}{w_i} \text{ max} * M \text{ sisa})\}$$

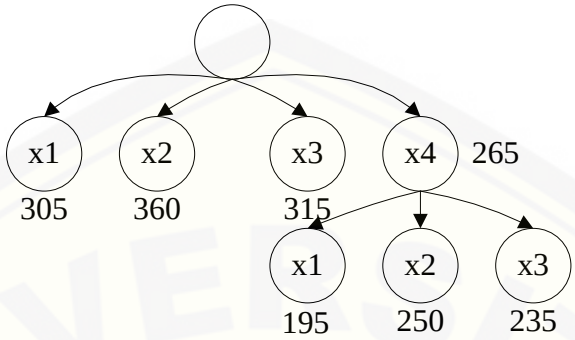
$$= \{(p_i \text{ lama} + p_i \text{ baru}) + \frac{p_i}{w_i} \text{ max} * (5 - w_i)\}$$

- Tahap 1



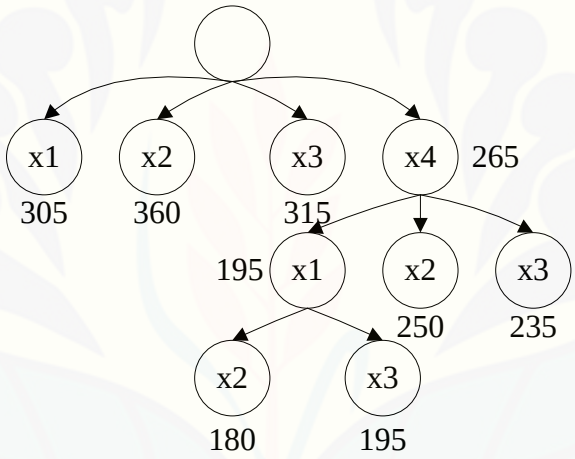
Gambar 4.1 Graf pohon algoritma *Branch and Bound* tahap 1

- Tahap 2



Gambar 4.2 Graf pohon algoritma *Branch and Bound* tahap 2

- Tahap 3



Gambar 4.3 Graf pohon algoritma *Branch and Bound* tahap 3

Karena jika simpul diteruskan akan melebihi kapasitas maksimum, maka simpul dihentikan (*goal node*). Dari langkah-langkah menentukan barang yang masuk solusi optimal di atas diperoleh nilai keuntungan terbesar adalah 180 dan barang yang diangkut adalah x_1, x_2, x_4 dengan total berat 5 (Satuan Berat).

Berdasarkan hasil dari kedua algoritma di atas, dapat disimpulkan jika algoritma *Dynamic Programming* lebih baik digunakan pada permasalahan *Integer knapsack* dapat dilihat dari besarnya keuntungan.

4.2 Hasil

Untuk membantu UD. Toda dalam pengambilan keputusan pemilihan barang yang akan diangkut atau tidak, akan dilakukan pengambilan data yang kemudian diidentifikasi sesuai langkah dalam pengerjaan algoritma yang digunakan. Data mentah yang sudah diambil dari UD. Toda meliputi jenis/merk barang, jumlah barang, harga beli, dan harga jual seperti yang terlampir pada lampiran A dengan kapasitas media pengangkutan maksimal (M) = 8.000 kg. Sehingga perlu dilakukan identifikasi guna memperoleh berat barang (w_i) dan keuntungan (p_i). Untuk memperoleh berat barang (w_i) pengidentifikasian dengan cara mengalikan bobot tiap satu item dengan banyaknya barang dalam satu item yang harus diangkut. Sedangkan untuk memperoleh keuntungan (p_i) dilakukan pengidentifikasian dengan cara menghitung selisih dari harga beli dengan harga jual. Sehingga diperoleh data yang sudah siap untuk diolah seperti pada Tabel 4.6.

Tabel 4.6 Data Identifikasi dari Lampiran A

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
1.	Semen	Puger 40Kg A	1600	60000
2.		Puger 40Kg B	400	15000
3.		Gresik 40Kg A	1400	52500
4.		Gresik 40Kg B	600	22500
5.	Lem (plastik)	Rajawali 1Kg	60	60000
6.		Rajawali ½Kg	60	120000
7.		Fox 1Kg	60	90000
8.		Fox ½Kg	60	60000
9.	Lem (Kaleng)	Rajawali ½ Kg	60	240000
10.		Fox ½Kg	60	240000
11.		Fox ¼Kg	30	240000
12.		Fox 100gr	20	400000

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
13.	Lem (Kaleng)	Epoxy 1Kg	30	600000
14.		Epoxy ¼ Kg	60	600000
15.	Paku	Ukuran ¾ dim 30Kg	30	60000
16.		Ukuran 1 dim 30Kg	30	60000
17.		Ukuran 1½ dim 30Kg	30	60000
18.		Ukuran 1¾ dim 30Kg	30	60000
19.		Ukuran 2 dim 30Kg	30	60000
20.		Ukuran 2½ dim 30Kg	30	60000
21.		Ukuran 3 dim 30Kg	30	60000
22.		Ukuran 3½ dim 30Kg	30	30000
23.		Ukuran 4 dim 30Kg	30	30000
24.		Ukuran 5 dim 30Kg	30	30000
25.		Ukuran 6 dim 30Kg	30	30000
26.		Payung 30Kg	30	160000
27.		GRC 30Kg	30	60000
28.	Kawat	Type 16 50Kg	50	50000
29.		Bendrat 20Kg	40	80000
30.	Cat Kayu	Glotex 1Kg	120	1200000
31.		Sendai 1Kg	120	600000
32.		Avian 1Kg	120	1200000
33.		Avian ½Kg	60	480000
34.	Kalsium	Gunung 40Kg	40	120000
35.		Matahari 40Kg	40	160000
36.	Cat Tembok	Paragon 1kg	120	240000

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
37.	Cat Tembok	Paragon 5kg	200	80000
38.		Paragon 20kg	200	100000
39.		Maritex 1kg	120	480000
40.		Maritex 5kg	200	200000
41.		Maritex 20kg	200	200000
42.		Altex 1kg	120	240000
43.		Altex 5kg	200	200000
44.		Altex 20kg	200	100000
45.		Metrolit 5kg	200	800000
46.		Metrolit 20kg	200	200000
47.		Finatex 5kg	200	400000
48.		Finatex 20kg	200	200000
49.		Rajacat 5kg	200	200000
50.		Inulex 5kg	200	400000
51.	Plitur	Mowilex 1Kg	120	600000
52.		Propan 1Kg	120	600000
53.		Impra 1Kg	120	600000
		Jumlah	**8600**	**13410000**

4.2.1 Penyelesaian masalah *Integer Knapsack* menggunakan algoritma *Dynamic Programming*

Data pengamatan yang sudah diidentifikasi seperti pada Tabel 4.6 selanjutnya diolah menggunakan algoritma *Dynamic Programming.* Langkah-langkah penyelesaiannya sebagai berikut.

a. Menentukan struktur dari masalah

Dari masalah tersebut diketahui n = 53 jenis barang yang dinotasikan i, i = 1, 2, …, 53, $i \epsilon N$ dengan bobot barang i dinotasikan dengan w_i sehingga w_i= [1400, 600, 1400, 600, 60, 60, 60, 60, 60, 60, 30, 20, 60, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 50, 40, 120, 120, 120, 60, 40, 40, 120, 200, 200, 120, 200, 200, 120, 200, 200, 200, 200, 200, 200, 200, 200, 120, 120, 120]. Kapasitas media pengangkutan dinotasikan dengan M = 8000. Dimana $M \epsilon N, w_i \leq M, i \epsilon N$, dengan batasan $\sum_{i=1}^{n} w_i x_i \leq M$. Tujuan dari *Dynamic Programming* yaitu mencari keuntungan optimal (dinotasikan dengan Z) dari barang yang akan diangkut dengan keuntungan dari tiap barang. Keuntungan dari tiap barang dinotasikan dengan p_i sehingga nilai p_i = [60000, 15000, 52500, 22500, 60000, 120000, 90000, 60000, 240000, 240000, 240000, 400000, 1200000, 600000, 60000, 60000, 60000, 60000, 60000, 60000, 60000, 30000, 30000, 30000, 160000, 60000, 50000, 80000, 1200000, 600000, 1200000, 480000, 120000, 160000, 240000, 80000, 100000, 480000, 200000, 200000, 240000, 200000, 100000, 800000, 200000, 400000, 200000, 200000, 400000, 600000, 600000, 600000], sehingga dapat ditulis $Z = \sum_{i=1}^{n} p_i x_i$.

Untuk memperoleh nilai keuntungan optimal dilakukan perhitungan dengan prosedur rekursif maju. Perhitungan dimulai dengan mencari nilai keuntungan barang ke-1 lalu dilanjutkan mencari keuntungan barang ke-2 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 keuntungan optimal untuk masalah tersebut. Nilai keuntungan yang optimal inilah yang dimasukkna ke dalam $z(i, j)$, sehingga dapat digunakan persamaan (2.3).

c. Menghitung nilai dari solusi optimal

Untuk menghitung nilai dari solusi optimal pada masalah media pengangkutan ini prosedur perhitungan yang digunakan adalah prosedur rekursif maju. Perhitungan

dimulai dari mencari keuntungan untuk barang ke-1 lalu dilanjutkan mencari nilai keuntungan barang ke-2 sampai pada tahap ke-53.

d. Menentukan keputusan optimal

Dari langkah c diperoleh matriks yang berisi keputusan-keputusan optimal yaitu matriks *z*. Keuntungan optimal dari masalah di atas adalah nilai dari $z(n, m)$

Langkah a dan b pada penyelesaian di atas dilakukan untuk memperoleh bentuk dari permasalahan yang akan diselesaikan menggunakan algoritma *Dynamic Programming* dan menentukan persamaan rekursif yang akan digunakan. Pada langkah c yaitu menghitung nilai solusi optimal tiap tahap. Langkah d adalah mencari nilai solusi optimal atau jenis barang yang akan diangkut atau tidak. Untuk hasil perhitungan perhitungan dengan algoritma *Dynamic Programming* pada Tabel 4.7 dipergunakan program karena keterbatasan dalam melakukan perhitungan manual untuk n = 53 dan dengan kapasitas maksimal media pengangkutan (M) = 8000 kg.

Hasil perhitungan masalah *Integer Knapsack* dengan menggunakan algoritma *Dynamic Programming* dapat dilihat pada Tabel 4.7. Hasil perhitungan tersebut merupakan *output* program yang menyajikan barang yang memungkinan untuk diangkut dari beberapa kemungkinan yang lain. Sesuai tahapan perhitungn manual menggunakan algoritma *Dynamic Programming* maka dihasilkan barang-barang akan diangkut dengan keuntungan optimal yang diperoleh sebagai berikut, X = {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,1, 1, 1, 1, 1}. Sesuai dengan kaidah pada *Integer Knapsack* bahwa barang yang dinotasikan dengan 1 maka barang tersebut diangkut, sedangkan jika dinotasikan dengan 0 maka barang tersebut tidak diangkut. Sehingga barang 4 tidak diangkut, sedangkan barang yang lain dinotasikan dengan 1 berarti barang tersebut diangkut, yaitu barang 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53.

Perolehan hasil optimal algoritma *Dynamic Prgramming* dapat dilihat pada Tabel 4.7 berikut.

Tabel 4.7 Hasil pilihan barang dengan algoritma *Dynamic Programming*

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
1.	Semen	Puger 40Kg A	1600	60000
2.		Puger 40Kg B	400	15000
3.		Gresik 40Kg A	1400	52500
5.	Lem (plastik)	Rajawali 1Kg	60	60000
6.		Rajawali ½Kg	60	120000
7.		Fox 1Kg	60	90000
8.		Fox ½Kg	60	60000
9.	Lem (Kaleng)	Rajawali ½ Kg	60	240000
10.		Fox ½Kg	60	240000
11.		Fox ¼Kg	30	240000
12.		Fox 100gr	20	400000
13.		Epoxy 1Kg	30	600000
14.		Epoxy ¼ Kg	60	600000
15.	Paku	Ukuran ¾ dim 30Kg	30	60000
16.		Ukuran 1 dim 30Kg	30	60000
17.		Ukuran 1½ dim 30Kg	30	60000
18.		Ukuran 1¾ dim 30Kg	30	60000
19.		Ukuran 2 dim 30Kg	30	60000
20.		Ukuran 2½ dim 30Kg	30	60000
21.		Ukuran 3 dim 30Kg	30	60000
22.		Ukuran 3½ dim 30Kg	30	30000
23.		Ukuran 4 dim 30Kg	30	30000
24.		Ukuran 5 dim 30Kg	30	30000
25.		Ukuran 6 dim 30Kg	30	30000

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
26.	Paku	Payung 30Kg	30	160000
27.		GRC 30Kg	30	60000
28.	Kawat	Type 16 50Kg	50	50000
29.		Bendrat 20Kg	40	80000
30.	Cat Kayu	Glotex 1Kg	120	1200000
31.		Sendai 1Kg	120	600000
32.		Avian 1Kg	120	1200000
33.		Avian ½Kg	60	480000
34.	Kalsium	Gunung 40Kg	40	120000
35.		Matahari 40Kg	40	160000
36.	Cat Tembok	Paragon 1kg	120	240000
37.		Paragon 5kg	200	80000
38.		Paragon 20kg	200	100000
39.		Maritex 1kg	120	480000
40.		Maritex 5kg	200	200000
41.		Maritex 20kg	200	200000
42.		Altex 1kg	120	240000
43.		Altex 5kg	200	200000
44.		Altex 20kg	200	100000
45.		Metrolit 5kg	200	800000
46.		Metrolit 20kg	200	200000
47.		Finatex 5kg	200	400000
48.		Finatex 20kg	200	200000
49.		Rajacat 5kg	200	200000
50.		Inulex 5kg	200	400000

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
51.		Mowilex 1Kg	120	600000
52.	Plitur	Propan 1Kg	120	600000
53.		Impra 1Kg	120	600000
		Jumlah	**8000**	**13387500**

Dari keterangan diatas didapat hasil maksimal dengan keuntungan optimal sebesar Rp. 13.387.500,- dengan berat 8.000 kg.

4.2.2 Penyelesaian masalah *Integer Knapsack* menggunakan algoritma *Branch and Bound*

Data pengamatan yang sudah diidentifikasi seperti pada table 4.6 selanjutnya diolah menggunakan algoritma *Branch and Bound.* Langkah-langkah penyelesaiannya sebagai berikut.

a. Menentukan struktur dari masalah

Dari masalah tersebut diketahui n = 53 jenis barang yang dinotasikan i, i = 1, 2, …, 53, $i \epsilon N$ dengan berat barang i dinotasikan dengan w_i sehingga w_i= [1600, 400, 1400, 600, 60, 60, 60, 60, 60, 60, 30, 20, 120, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 50, 40, 120, 120, 120, 60, 40, 40, 120, 200, 200, 120, 200, 200, 120, 200, 200, 200, 200, 200, 200, 200, 200, 120, 120, 120]. Kapasitas media pengangkutan dinotasikan dengan M = 8000. Dimana $M \epsilon N, w_i \leq M, i \epsilon N$, dengan batasan $\sum_{i=1}^{n} w_i x_i \leq M$. Tujuan dari *Branch and Bound* yaitu mencari keuntungan optimal (dinotasikan dengan Z) dari barang yang akan diangkut dengan keuntungan dari tiap barang. Keuntungan dari tiap barang dinotasikan dengan p_i sehingga nilai p_i = [60000, 15000, 52500, 22500, 60000, 120000, 90000, 60000, 240000, 240000, 240000, 400000, 1200000, 600000, 60000, 60000, 60000, 60000, 60000, 60000, 60000, 30000, 30000, 30000, 160000, 60000, 50000, 80000, 1200000, 600000, 1200000, 480000,

120000, 160000, 240000, 80000, 100000, 480000, 200000, 200000, 240000, 200000, 100000, 800000, 200000, 400000, 200000, 200000, 400000, 600000, 600000, 600000], sehingga dapat ditulis $Z = \sum_{i=1}^{n} p_i x_i$. Untuk mencari nilai *cost* dari simpul *i* untuk mencapai simpul tujuan diperlukan nilai $p_i \ / \ w_i$ sehingga nilai $p_i \ / \ w_i$ = [37.5, 37.5, 37.5, 37.5, 1000, 2000, 1500, 1000, 4000, 4000, 8000, 20000, 10000, 20000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 1000, 1000, 1000, 1000, 5333.3, 2000, 1000, 2000, 10000, 5000, 10000, 8000, 3000, 4000, 2000, 400, 500, 4000, 1000, 1000, 2000, 1000, 500, 4000, 1000, 2000, 1000, 1000, 2000, 5000, 5000, 5000].

b. Menentukan nilai *cost* dari tiap simpul

Perhitugan dimulai dengan mencari nilai *cost* awal sebagai simpul akar menggunakan persamaan (2.4). Lalu dilanjutkan mencari nilai *cost* dari semua simpul sebanyak jumlah barang (*n*) yang kemudian dicari nilai *cost* terkecil untuk dibangkitkan hingga sampai pada simpul tujuan dengan nilai keuntungan yang optimal. Keuntungan optimal didapat dari simpul-simpul yang dibangkitkan dengan memilih nilai *cost* terkecil.

c. Menentukan solusi optimal

Dari langkah b kemudian menentukan barang-barang yang akan diangkut serta menghitung nilai dari keuntungan yang optimal dari simpul awal (akar) hingga simpul tujuan (akhir).

Dengan menggunakan bantuan komputer untuk menyelesaikan perhitungan manual algoritma *Branch and Bound* maka dihasilkan barang-barang yang akan diangkut sebagai berikut, *X* = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,1, 1, 1, 1, 1, 1, 1}. Sehingga barang-barang yang diangkut setelah dikerjakan menggunakan algoritma *Branch and Bound*, yaitu barang 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 28, 29, 30, 31, 32, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53. Sedangkan barang yang tidak diangkut yaitu barang 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 33, 34, 35.

Hasil perhitungan algoritma *Branch and Bound* untuk barang-barang yang dipilih dapat dilihat pada Tabel 4.8.

Tabel 4.8 Hasil pilihan barang dengan algoritma *Branch and Bound*

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
1.	Semen	Puger 40Kg A	1600	60000
2.		Puger 40Kg B	400	15000
3.		Gresik 40Kg A	1400	52500
4.		Gresik 40Kg B	600	22500
5.	Lem (plastik)	Rajawali 1Kg	60	60000
6.		Rajawali ½Kg	60	120000
7.		Fox 1Kg	60	90000
8.		Fox ½Kg	60	60000
9.	Lem (Kaleng)	Rajawali ½ Kg	60	240000
10.		Fox ½Kg	60	240000
28.	Kawat	Type 16 50Kg	50	50000
29.		Bendrat 20Kg	40	80000
30.	Cat Kayu	Glotex 1Kg	120	1200000
31.		Sendai 1Kg	120	600000
32.		Avian 1Kg	120	1200000
36.	Cat Tembok	Paragon 1kg	120	240000
37.		Paragon 5kg	200	80000
38.		Paragon 20kg	200	100000
39.		Maritex 1kg	120	480000
40.		Maritex 5kg	200	200000
41.		Maritex 20kg	200	200000
42.		Altex 1kg	120	240000

No.	Jenis Produk	Nama Barang (i)	Bobot Barang (w_i)	Profit Barang (p_i)
43.	Cat Tembok	Altex 5kg	200	200000
44.		Altex 20kg	200	100000
45.		Metrolit 5kg	200	800000
46.		Metrolit 20kg	200	200000
47.		Finatex 5kg	200	400000
48.		Finatex 20kg	200	200000
49.		Rajacat 5kg	200	200000
50.		Inulex 5kg	200	400000
51.	Plitur	Mowilex 1Kg	120	600000
52.		Propan 1Kg	120	600000
53.		Impra 1Kg	120	600000
		Jumlah	**7990**	**10410000**

Dari keterangan diatas didapat hasil maksimal dengan keuntungan optimal sebesar Rp. 10.410.000,- dengan berat 7.990 kg.

4.2.3 Perhitungan kompleksitas waktu algoritma *Dynamic Programming.*

Berikut adalah perhitungan kompleksitas waktu dari semua tahapan algoritma *Dynamic Programming* sesuai dengan flowchart pada Lampiran B:

$T_{DP1} = m + 9n + 4$

$T_{DP2} = 3m^3 + 4mn^2 + 6n + 1$

$T(n) = T_{DP1} + T_{DP2}$

$= (m + 9n + 4) + (3m^3 + 4mn^2 + 6n + 1)$

$= 3m^3 + m + 4mn^2 + 15n + 5$

Karena $3m^3 + m + 4mn^2 + 15n + 5$, maka kompleksitas waktu algoritma *Dynamic Programming* adalah $O(n^3)$ atau kubik.

4.2.4 Perhitungan kompleksitas waktu algoritma *Branch and Bound.*

Berikut adalah perhitungan kompleksitas waktu dari semua tahapan algoritma *Branch and Bound* sesuai dengan flowchart pada Lampiran C:

T_{BB1} = $8n + 5$

T_{BB2} = $8n + 5$

T_{BB3} = $10n^2 + 8n$

T_{BB4} = $19n^2 + 1$

$T(n)$ = $T_{BB1} + T_{BB2} + T_{BB3} + T_{BB4}$

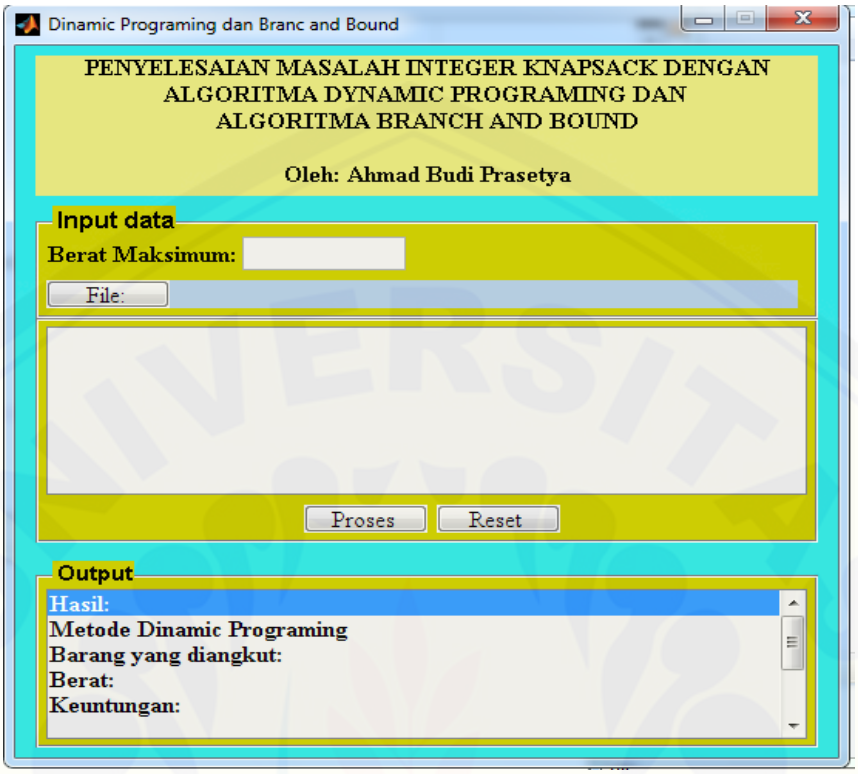
= $(8n + 5) + (8n + 5) + (10n^2 + 8n) + (19n^2 + 1)$

= $29n^2 + 24n + 11$

Karena $29n^2 + 24n + 11$, maka kompleksitas waktu algoritma *Branch and Bound* adalah $O(n^2)$ atau kuadratik.

4.2.5 Langkah-langkah menjalankan program

Pada langkah ini, dibuat sebuah perhitungan menggunakan program dengan bantuan *software* MATLAB. Program ini dibuat dengan tampilan GUI yang dapat dilihat pada gambar 4.4. Program ini dibuat dengan tujuan untuk mempermudah dan mempercepat perhitungan pada permasalahan *Integer Knapsack* dengan menggunakan algoritma *Dynamic Programming* dan algoritma *Branch and Bound*. Untuk memulai menjalankan program, langkah awal adalah membuka tampilan awal program GUI.

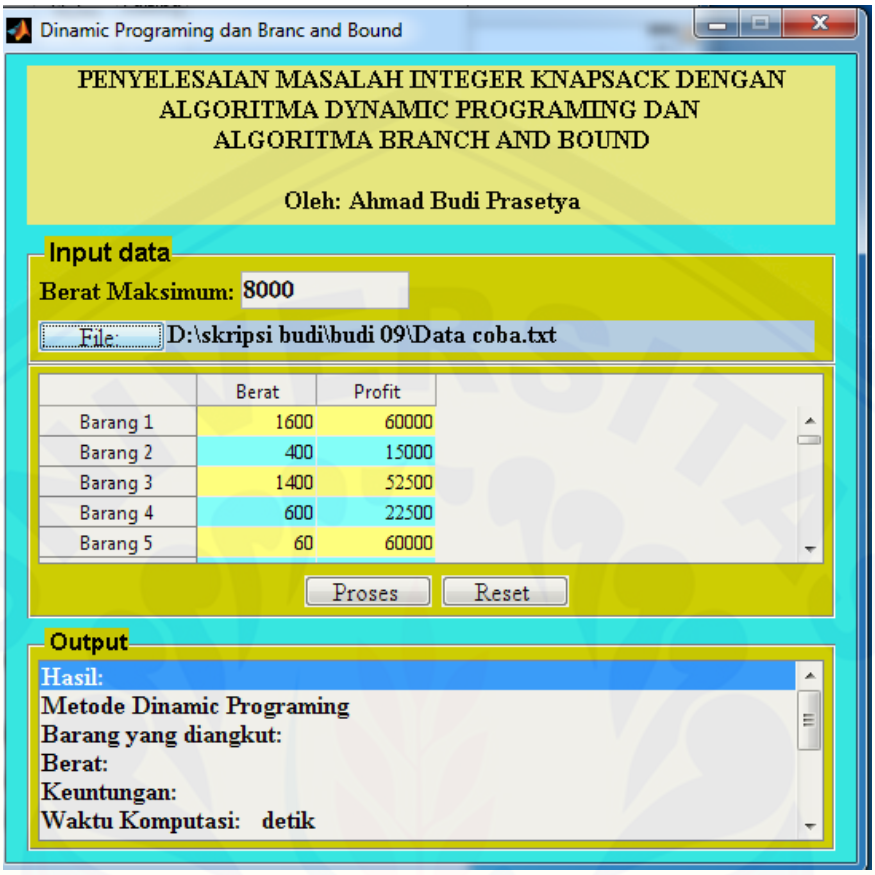


Gambar 4.4 Tampilan awal program

Menu yang terdapat pada Gambar 4.4 adalah sebagai berikut:

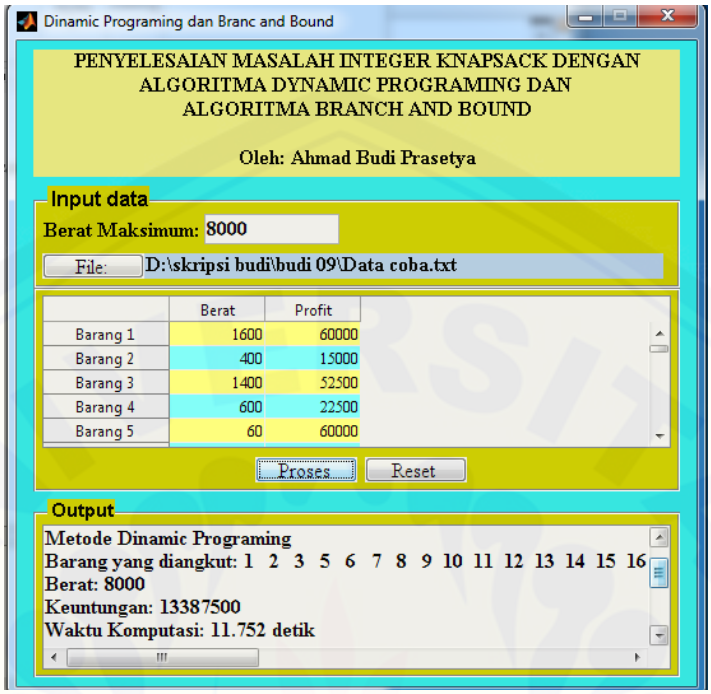
a. Berat Maksimum menunjukkan daya angkut maksimal untuk mengangkut barang.
b. *File*, digunakan untuk menginput berat tiap barang dan keuntungan tiap barang yang sudah disimpan dalam bentuk txt.
c. Proses, digunakan untuk memulai proses metode menggunakan algoritma *Dynamic Programming* dan *Branch and Bound*.
d. *Reset*, digunakan untuk menghapus proses metode yang telah dijalankan dan kembali pada tampilan awal program dengan input data sebelumnya.

Langkah awal yang digunakan untuk menjalankan program adalah menentukan nilai berat maksimum, berat tiap barang, dan keuntungan tiap barang pada kolom input data. Selanjutnya akan muncul tampilan program untuk pengisian data penelitian seperti yang ditujukkan pada Gambar 4.5

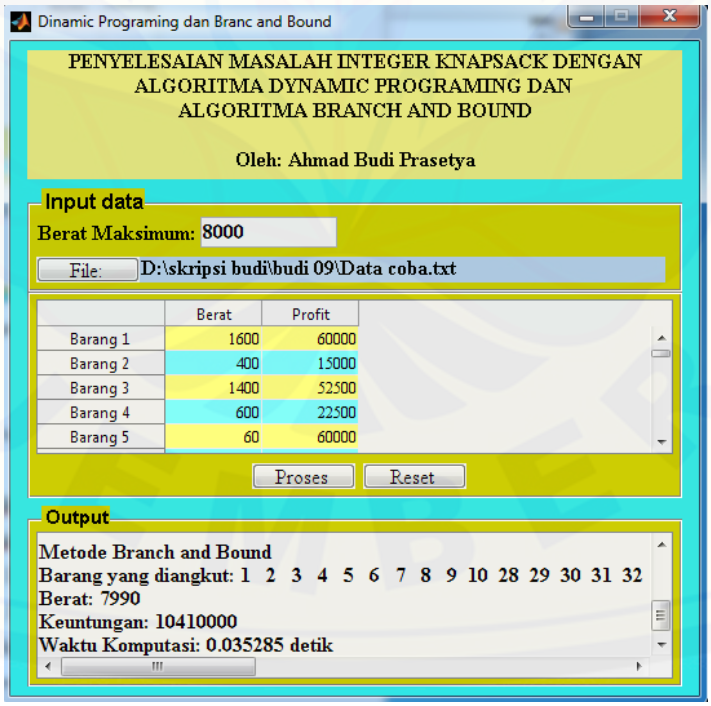


Gambar 4.5 Tampilan setelah data diisi

Pada tampilan akan muncul nilai dari berat dan keuntungan tiap barang yang akan segera diproses. Setelah itu *klik* tombol proses untuk memunculkan tampilan hasil perhitungan algoritma *Dynamic Programming* dan *Branch and Bound* seperti ditunjukkan pada Gambar 4.6 dan Gambar 4.7



Gambar 4.6 Hasil perhitungan algoritma *Dynamic Programming*



Gambar 4.7 Hasil perhitungan algoritma *Branch and Bound*

Berdasarkan program hasil perhitungan yang ditampilkan menggunakan algoritma *Dynamic Programming* seperti pada Gambar 4.6, dijelaskan bahwa barang yang dapat diangkut adalah barang 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53 dengan total berat 8.000 kg dan total keuntungan yang didapat Rp. 13.387.500,- serta waktu komputasi 11,752 detik. Sedangkan hasil dari program yang ditampilkan pada Gambar 4.7 menggunakan algoritma *Branch and Bound*, dijelaskan bahwa barang yang dapat diangkut adalah barang 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 28, 29, 30, 31, 32, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, dengan total berat 7.990 kg dan total keuntungan yang didapat Rp. 10.410.000,-serta waktu komputasi yang dibutuhkan 0,035285 detik.

4.3 Pembahasan

Berdasarkan hasil percobaan yang sudah dilakukan pada subbab sebelumnya, perhitungan menggunakan algoritma *Dynamic Programming* menghasilkan berat total yang diangkut mencapai maksimal kapasitas yang disediakan ,yaitu 8.000 kg dengan keuntungan yang didapat Rp. 13.387.500,-. Barang yang tidak diangkut hanya barang ke-4 yaitu Semen Gresik 40 kg B yang memiliki berat 600 kg. Karena jika diangkut akan melebihi kapasitas maksimal media pengakutan. Keuntungan dari keempat jenis semen diatas adalah sama besar yaitu Rp. 1.500,- per 40 kg (1 karung). Jadi sebenarnya kemungkinan dari keempat jenis semen memiliki kesempatan sama untuk diangkut atau tidak jika memiliki berat yang sama. Dan jika dibandingkan dengan barang yang lain, semen memiliki bobot yang besar tetapi keuntungan yang kecil sehingga kesempatan untuk diangkut lebih kecil. Sedangkan perhitungan menggunakan algoritma *Branch and Bound* menghasilkan berat total yang diangkut mencapai 7.990 kg dengan keuntungan yang didapat Rp. 10.410.000,-. Barang-barang yang tidak diangkut pada perhitungan dengan algoritma *Branch and Bound* cenderung lebih banyak dari pada algoritma *Dynamic Programming* karena pada algoritma *Branch and*

Bound pencarian solusi optimumnya dipengaruhi oleh nilai $\frac{p_i}{w_i}$ dan pengambilan keputusan untuk barang diangkut atau tidak dicari yang memiliki nilai *cost* terkecil. Sehingga menurut hasil optimasi dari kedua algoritma tersebut dapat dilihat bahwa algoritma *Dynamic Programming* lebih optimum jika dibandingkan dengan algoritma *Branch and Bound* untuk diaplikasikan pada permasalahan *Integer Knapsack*.

Berbeda halnya jika ditinjau dari perhitungan kompleksitas waktu dan atau waktu komputasi yang dibutuhkan pada kedua algoritma tersebut. Algoritma *Dynamic Programming* memiliki kompleksitas waktu yang dalam notasi *Big-O* dilambangkan dengan $O(n^3)$ yaitu kubik. Sedangkan algoritma *Branch and Bound* dalam perhitungan kompleksitas waktu menghasilkan $O(n^2)$ yaitu kuadratik. Dalam kemangkusan algoritma menggunakan notasi *Big-O* bahwa kudratik lebih mangkus (efisien) dari pada kubik. Jadi algoritma *Branch and Bound* lebih bagus jika diterapkan pada permasalahan *Integer Knapsack* dari segi kompleksitas waktu. Untuk waktu komputasi yang dibutuhkan kedua algoritma tersebut untuk menyelesaikan dengan jumlah data sebanyak n = 53 dan kapasitas berat maksimal M = 8000. Algoritma *Dynamic Programming* menghasilkan waktu komputasi sebesar 11,752 detik sedangkan algoritma *Branch and Bound* memiliki waktu komputasi jauh lebih cepat yaitu 0,035285 detik. Dengan kata lain, algoritma *Branch and Bound* lebih efektif untuk menyelasikan permasalahan *Integer Knapsack*.

Berdasarkan kedua hasil yang sudah dibahas diatas, menunjukkan bahwa algoritma *Dynamic Programming* lebih optimal dalam memperoleh keuntungan. Sedangkan jika ditinjau dari waktu pengerjaannya, algoritma *Branch and Bound* memiliki kompleksitas waktu yang lebih efisien. Perbedaaan mendasar dari kedua algoritma tersebut adalah pada algoritma *Dynamic Programming* solusi pencariannya bersifat global sedangkan algoritma *Branch and Bound* solusi pencariaannya bersifat lokal.

BAB 5. PENUTUP

5.1 Kesimpulan

Berdasarkan data, hasil analisis, dan pembahasan yang telah dilakukan pada bab sebelumnya, maka dapat diperoleh kesimpulan dari penyelesaian masalah *Integer Knapsack* dengan algoritma *Dynamic Programming* dan algoritma *Branch and Bound* sebagai berikut.

a. Penerapan algoritma *Dynamic Programming* pada masalah *Integer Knapsack* menghasilkan keuntungan Rp. 13.387.500,- dengan berat total 8.000 kg sedangkan waktu komputasi yang dibutuhkan 11,752 detik dan kompleksitas waktu yang dihasilkan adalah $O(n^3)$ yaitu kubik.
b. Penerapan algoritma *Branch and Bound* pada masalah *Integer Knapsack* menghasilkan keuntungan Rp. 10.410.000,- dengan berat total 7.990 kg sedangkan waktu komputasi yang dibutuhkan 0,035285 detik dan kompleksitas waktu yang dihasilkan adalah $O(n^2)$ yaitu kuadratik.
c. Berdasarkan hasil perhitungan kedua algoritma tersebut, algoritma *Dynamic Programming* menghasilkan keuntungan yang lebih besar tetapi memiliki kompleksitas waktu dan atau waktu komputasi lebih besar dari pada algoritma *Branch and Bound* dalam penyelesaian masalah *Integer Knapsack*. Karena algoritma *Dynamic Programming* pencariaannya bersifat global sedangkan algoritma *Branch and Bound* pencariaannya bersifat lokal.

5.2 Saran

Selain algoritma *Dynamic Programming* dan algoritma *Branch and Bound* penelitian ini dapat dikembangkan dengan menggunakan algoritma lain yang dapat digunakan untuk menyelesaikan masalah *Integer Knapsack*. Peneliti selanjutnya juga

dapat menambah atau merubah variabel sesuai permasalahan yang ada. Atau bisa menggunakan permasalahan yang disudah dikembangkan seperti *Bounded Knapsack dan Unbounded Knapsack* guna menyesuaikan permasalahan pada realita yang ada.

DAFTAR PUSTAKA

Arista, W. M. 2013. *Penerapan Algoritma Greedy dan Dynamic Programming Pada Permasalahan Integer Knapsack*. Jember: Jurusan Matematika FMIPA Universitas Jember.

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

Kellerer, H., et all. 2004. *Knapsack Problem*. Berlin: Springer.

Lew, H., Mauch, H. 2007. *Dynamic Programming a Computational Tool*. Berlin: Springer.

Martello, S. 2006. *New Trends in Exact Algorithms for the 0-1 Knapsack Problem*. http://www.cise.ufl.edu/~sahni/papers/knap.pdf. [5 Januari 2015]

Munawar, H. 2007. *Penerapan Algoritma Branch and Bound Dalam Assigment Problem*. Bandung: Teknik Informatika STEI-ITB.

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

Muthohar, M. F. 2008. *Penerapan Algoritma Branch and Bound Pada Masalah Integer Knapsack*. Bandung: Teknik Informatika STEI-ITB.

Nurhayati, O. D. 2009. *Dasar Algoritma. Program Studi Sistem Komputer*. Semarang: Universitas Diponegoro.

Paryati. 2009. *Strategi Algoritma Greedy Untuk Menyelesaikan Permasalahan Knapsack 0-1*. Yogyakarta: Teknik Informatika UPN Veteran Yogyakarta.

Permata, A. S. 2007. *Pemecahan Masalah Knapsack dengan Menggunakan Algoritma Branch and Bound*. Bandung: Teknik Informatika STEI-ITB.

Ridho, M. 2014. *Penyelesaian Masalah Knapsack 0-1 Dengan Algoritma Harmony Search Dan Dynamic Programming.* Jember: Jurusan Matematika FMIPA Universitas Jember.

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

Wibowo, F. 2011. *Matematika Diskrit: Kompleksitas Algoritma*. Yogyakarta: STMIK Amikom.

LAMPIRAN

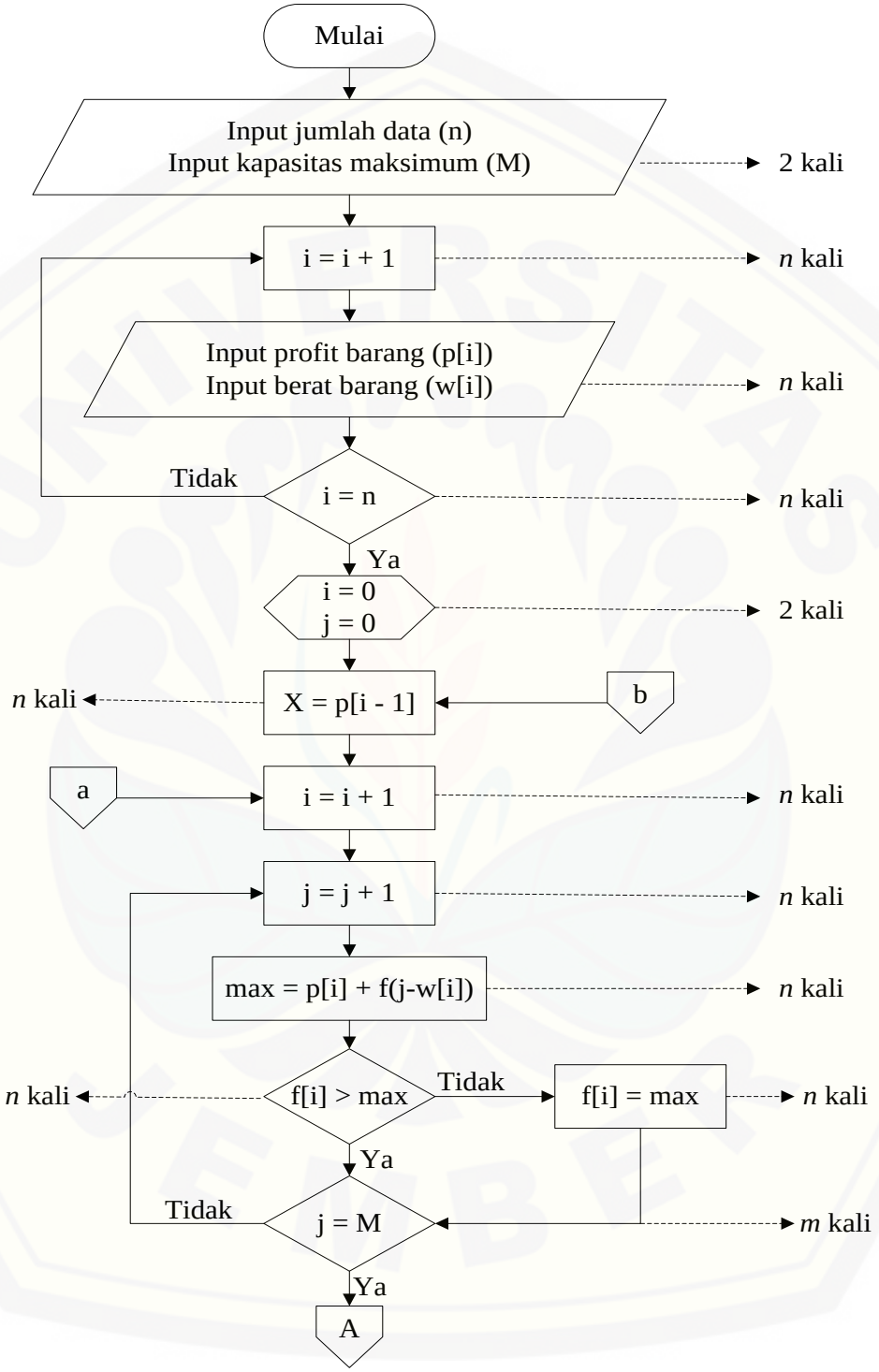
A. Daftar barang pada UD. Toda yang akan diteliti.

No.	Jenis Produk	Nama Barang	Jumlah	Harga Beli (Rp)	Harga Jual (Rp)
1.	Semen	Puger 40Kg A	40	46.500	48.000
2.		Puger 40Kg B	10	46.500	48.000
3.		Gresik 40Kg A	35	56.000	57.500
4.		Gresik 40Kg B	15	56.000	57.500
5.	Lem (Plastik)	Rajawali 1Kg	60	11.000	12.000
6.		Rajawali ½Kg	120	5.000	6.000
7.		Fox 1Kg	60	11.500	13.000
8.		Fox ½Kg	120	6.500	7.000
9.	Lem (Kaleng)	Rajawali ½ Kg	120	22.000	24.000
10.		Fox ½Kg	120	40.000	42.000
11.		Fox ¼Kg	120	13.000	15.000
12.		Fox 100gr	200	5.000	7.000
13.		Epoxy 1Kg	120	110.000	120.000
14.		Epoxy ¼ Kg	120	30.000	35.000
15.	Paku	Ukuran ¾ dim 30Kg	1	390.000	450.000
16.		Ukuran 1 dim 30Kg	1	375.000	435.000
17.		Ukuran 1½ dim 30Kg	1	360.000	420.000
18.		Ukuran 1¾ dim 30Kg	1	360.000	420.000
19.		Ukuran 2 dim 30Kg	1	360.000	420.000
20.		Ukuran 2½ dim 30Kg	1	345.000	405.000

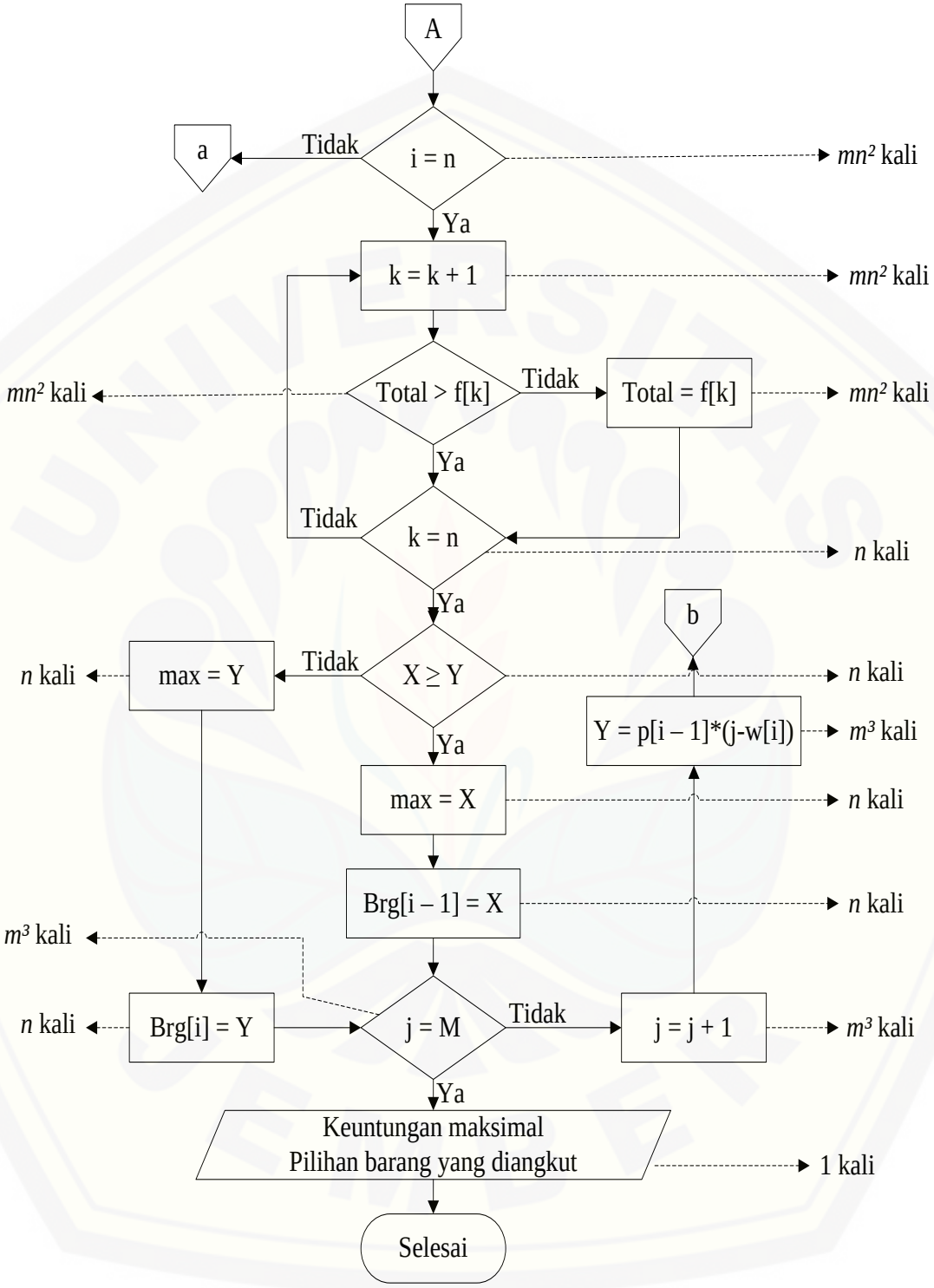
No.	Jenis Produk	Nama Barang	Jumlah	Harga Beli (Rp)	Harga Jual (Rp)
21.	Paku	Ukuran 3 dim 30Kg	1	330.000	390.000
22.		Ukuran 3½ dim 30Kg	1	330.000	360.000
23.		Ukuran 4 dim 30Kg	1	330.000	360.000
24.		Ukuran 5 dim 30Kg	1	315.000	345.000
25.		Ukuran 6 dim 30Kg	1	300.000	330.000
26.		Payung 30Kg	1	420.000	580.000
27.		GRC 30Kg	1	540.000	600.000
28.	Kawat	Type 16 50Kg	1	650.000	700.000
29.		Bendrat 20Kg	2	260.000	300.000
30.	Cat Kayu	Glotex 1Kg	120	50.000	60.000
31.		Sendai 1Kg	120	30.000	35.000
32.		Avian 1Kg	120	47.000	57.000
33.		Avian ½Kg	120	26.000	30.000
34.	Kalsium	Gunung 40Kg	1	1.280.000	1.400.000
35.		Matahari 40Kg	1	1.040.000	1.200.000
36.	Cat Tembok	Paragon 1kg	120	20.000	22.000
37.		Paragon 5kg	40	80.000	82.000
38.		Paragon 20kg	10	275.000	285.000
39.		Maritex 1kg	120	18.000	22.000
40.		Maritex 5kg	40	70.000	75.000
41.		Maritex 20kg	10	240.000	260.000
42.		Altex 1kg	120	20.000	22.000
43.		Altex 5kg	40	80.000	85.000
44.		Altex 20kg	10	280.000	290.000
45.		Metrolit 5kg	40	90.000	110.000

No.	Jenis Produk	Nama Barang	Jumlah	Harga Beli (Rp)	Harga Jual (Rp)
46.		Metrolit 20kg	10	480.000	500.000
47.		Finatex 5kg	40	65.000	75.000
48.	Cat Tembok	Finatex 20kg	10	230.000	250.000
49.		Rajacat 5kg	40	35.000	40.000
50.		Inulex 5kg	40	40.000	50.000
51.		Mowilex 1Kg	120	60.000	65.000
52.	Plitur	Propan 1Kg	120	65.000	70.000
53.		Impra 1Kg	120	50.000	55.000

B. *Flowchart* algoritma *Dynamic Programming*

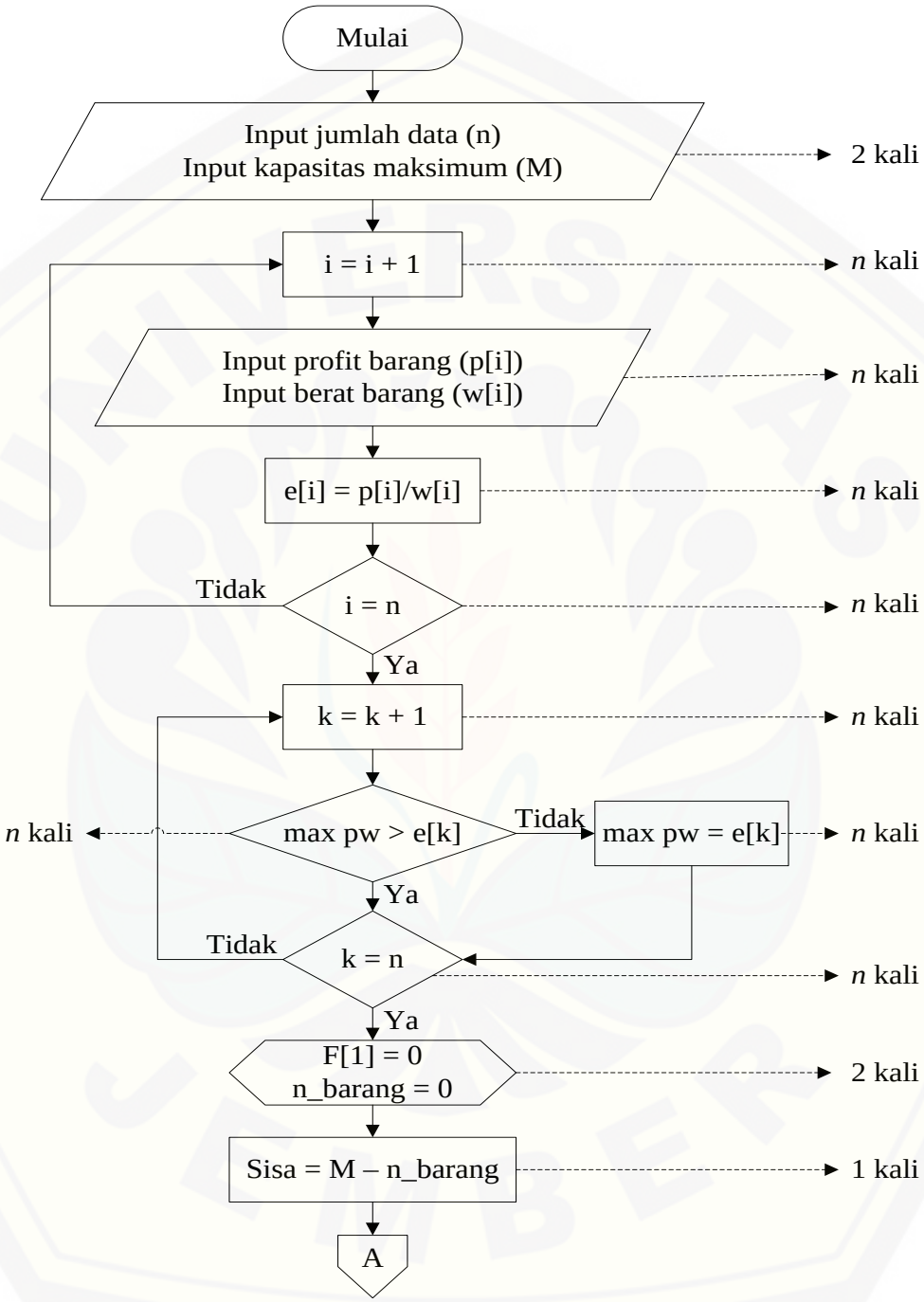


Jumlah perhitungan $T_{DP1} = m + 9n + 4$

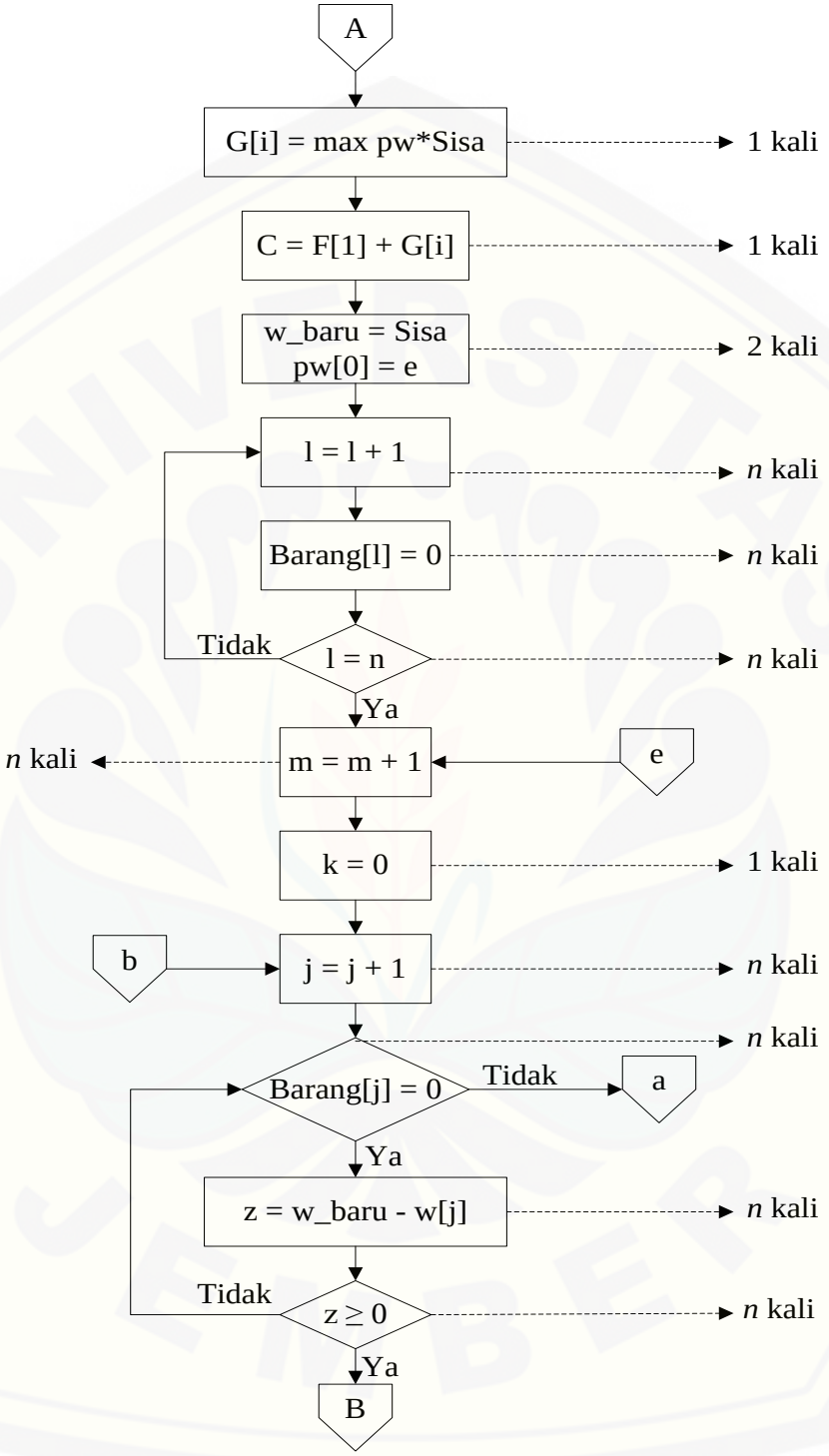


Jumlah perhitungan $T_{DP2} = 3m^3 + 4mn^2 + 6n + 1$

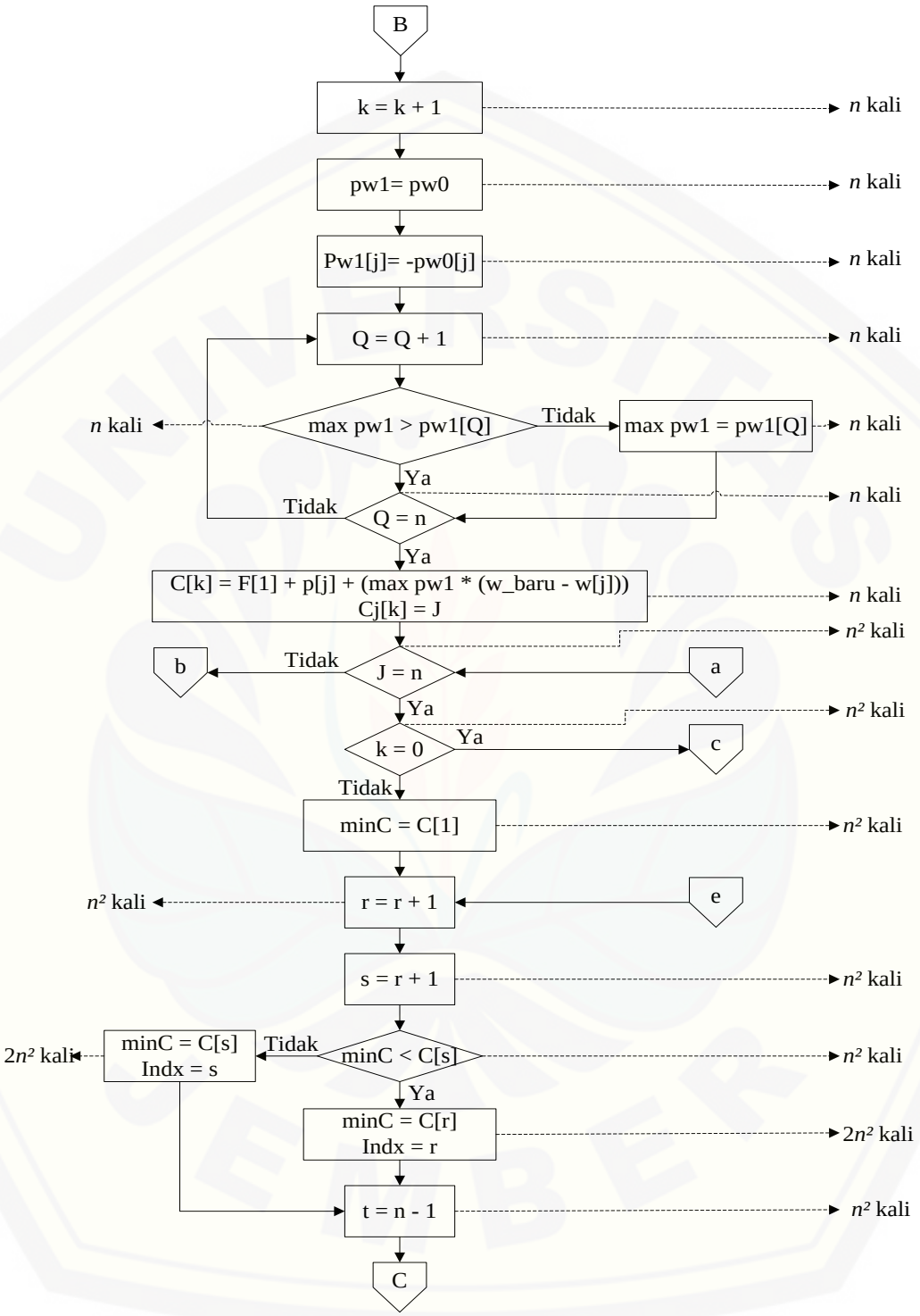
C. *Flowchart* algoritma *Branch and Bound*



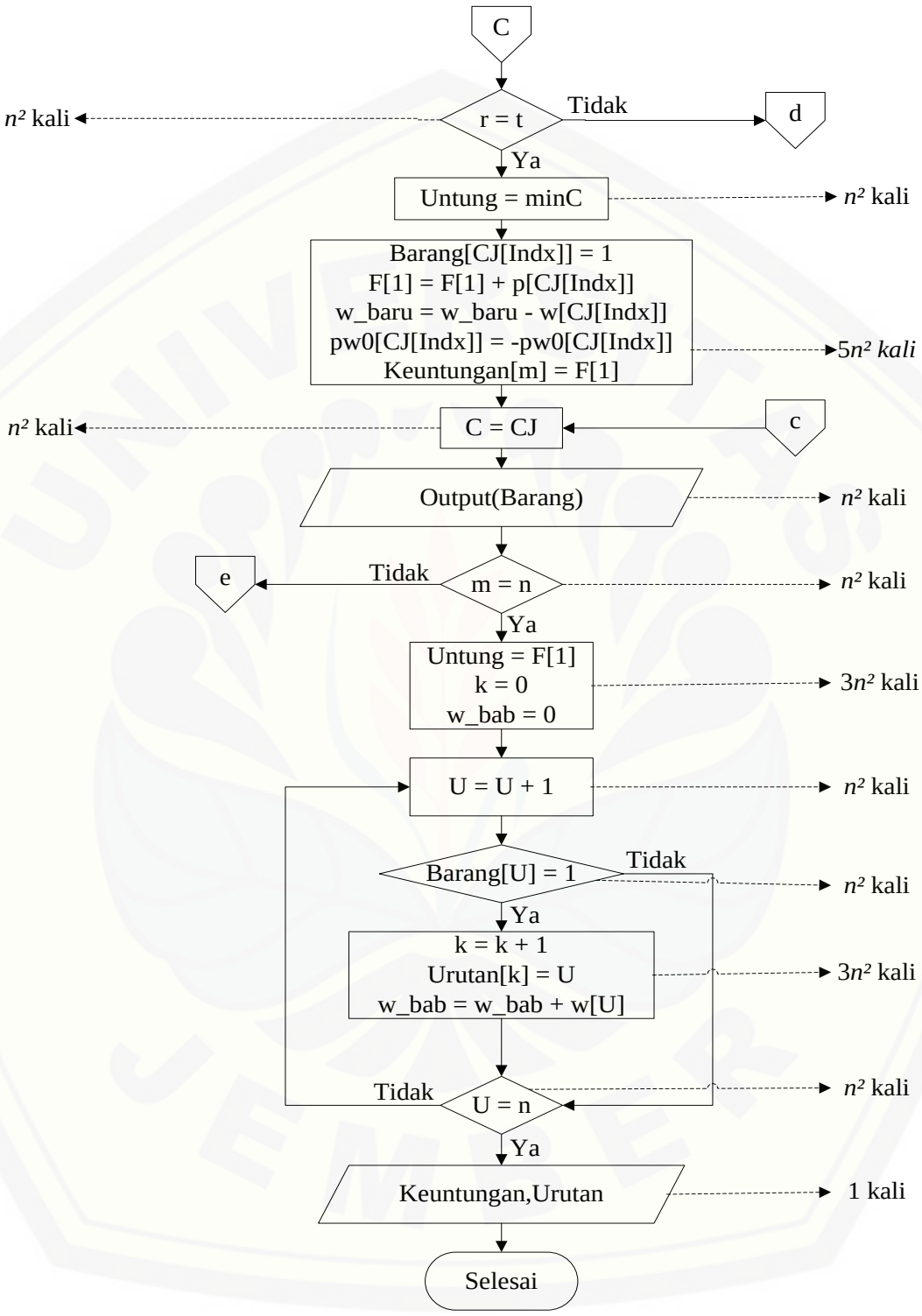
Jumlah perhitungan $T_{BB1} = 8n + 5$



Jumlah perhitungan $T_{BB2} = 8n + 5$



Jumlah perhitungan $T_{BB3} = 10n^2 + 8n$



Jumlah perhitungan $T_{BB4} = 19n^2 + 1$

D. Script Program algoritma *Dynamic Programming*

```
% clc; clear all; close all;
function [urutan hasil_untung_DP
berat_DP]=Dinamic_Programing(data,W)
% data=[2 65;3 80; 1 30];
berat=data(:,1);
P=data(:,2);
% W=5;
n=length(data);

%inisialisasi
y=0:W;
fy0=zeros(1,W+1);
barang0=zeros(W+1,n);

for i=1:n % Tahap i

    for j=1:W+1
        sisa=y(j)-berat(i);
        if sisa<0 % f1(y-Wi) negatif
            f1y(j)=fy0(j);
            barang1(j,:)=barang0(j,:);
        else% f1(y-Wi) positif
            [f1y(j)
indx]=max([fy0(j),P(i)+fy0(sisa+1)]);
            if indx(1)==1
            barang1(j,:)=barang0(j,:);

            else
            barang1(j,:)=barang0(sisa+1,:);
            barang1(j,i)=1;
            end
        end
    end
    fy0=f1y;
    barang0=barang1;
end

[hasil_untung_DP indx]=max(f1y);
barang_DP(1,:)=barang1(indx(1),:);
urutan=[]; k=0; berat_DP=0;
for i=1:n
```

```
    if barang_DP(i)==1
        k=k+1;
        urutan(k)=i;
        berat_DP=berat_DP+berat(i);
    end
end
%======================
clc;
disp(['Hasil Dinamic Programing Tahap ke '
num2str(n)]);
% disp('Barang:'); barang1
% disp('Keuntungan: '); f1y'
untung_ADP=f1y';
barang_ADP=barang1;
save('barang_DP0.mat','barang_ADP');
save('keuntungan_DP0.mat','untung_ADP');
```

E. Script Program algoritma *Branch and Bound*

```
% clc; clear all; close all;
% data=[2 65;3 80; 1 30];
function [urutan untung
berat_bab]=Branch_and_Bound(data,W)
berat=data(:,1);
P=data(:,2);
Pw=P./berat;
% W=5;
n=length(data);
disp('Branch and Bound');
disp('barang');

%inisialisasi
f1=0; n_barang=0;
sisa=W-n_barang;
Gi=max(Pw)*(sisa);
C=f1+Gi;

Wbaru=sisa;
Pw0=Pw;
barang=zeros(1,n);
for i=1:n
    k=0;
    for j=1:n
        if barang(j)==0

            if (Wbaru-berat(j))>=0
                k=k+1;Pw1=Pw0;
                Pw1(j)=-Pw0(j);
                c(k)=(f1+P(j))+(max(Pw1)*(Wbaru-
berat(j)));
                Cj(k)=j;
            end

        end

    end
%      c
    if k==0
        break
    else
```

```
    [untung indx]=min(c);
    barang(Cj(indx))=1;
    f1=f1+P(Cj(indx));
    Wbaru=Wbaru-berat(Cj(indx));
    Pw0(Cj(indx))=-Pw0(Cj(indx));
    keuntungan(i)=f1;
    barang_bab1(i,:)=barang;
    end
    c=[];
%     disp(num2str(barang));
end
% barang
untung=f1;k=0; berat_bab=0;
for i=1:n
    if barang(i)==1
        k=k+1;
        urutan(k)=i;
        berat_bab=berat_bab+berat(i);
    end


end
% barang
% disp('Keuntungan: ');
% keuntungan'
untung_BB=keuntungan';
barang_BB=barang_bab1;
save('keuntungan_BaB.mat','untung_BB');
save('barang_bab.mat','barang_BB');
```