Penentuan Menu Makan dengan Pemrograman Dinamis

Jordhy Fernando – 13515004
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515004@std.stei.itb.ac.id

***Abstrak* — Penentuan menu makan yang benar akan meningkatkan kesehatan dan produktivitas orang-orang dengan memenuhi kebutuhan kalori dan gizi. Kebanyakan orang pada era modern ini, mengabaikan asupan kalori dan gizi yang diperoleh dari makanan yang dikonsumsi dikarenakan kesibukan mereka ataupun karena masalah ekonomi. Ada orang yang mengalami kelebihan kalori atau gizi dan juga ada orang mengalami kekurangan kalori atau gizi. Oleh karena itu, dibutuhkan suatu cara untuk menentukan menu makan yang tepat bagi orang tersebut untuk mengatur asupan kalori maupun gizinya. Salah satu penyelesaian masalah ini adalah dengan memodelkan permasalahan ini sebagai *knapsack problem* dan menyelesaikannya dengan pemrograman dinamis. Hasil yang didapatkan dijamin optimal, namun mungkin belum bisa diaplikasikan secara langsung karena belum mempertimbangkan parameter-parameter tertentu dan juga keterbatasan dari program yang dibuat.**

Kata Kunci — knapsack problem; menu makan; optimasi; pemrograman dinamis

I. PENDAHULUAN

Pada era modern ini, kebanyakan orang cenderung sibuk dengan pekerjaan mereka dan mereka sering mengabaikan kebutuhan kalori maupun asupan gizi yang mereka dapatkan dari makanan yang mereka konsumsi. Padahal untuk menjaga maupun meningkatkan produktivitas dan tenaga mereka harus menjaga kebutuhan kalori dan asupan gizi dari makanan yang mereka konsumsi.

Alasan lain yang menyebabkan kurangnya asupan kalori maupun gizi adalah masalah ekonomi. Permasalahan ini umumnya terjadi pada kalangan orang berpendapatan rendah yang membatasi pengeluaran untuk makan. Mereka cenderung membeli makanan hanya untuk mengenyangkan perut dengan pengeluaran seminimal mungkin. Mereka tidak memperhatikan kebutuhan kalori maupun asupan gizi yang mereka butuhkan dan akibatnya kesehatan mereka dapat terganggu.

Masalah pada pola makan tidak hanya pada kurangnya asupan kalori dan gizi. Pola makan yang tidak benar juga dapat menyebabkan asupan kalori maupun gizi yang berlebih. Asupan kalori maupun gizi yang berlebih ini dapat mengakibatkan obesitas (kegemukan) dan juga dapat menimbulkan berbagai penyakit, misalnya diabetes, penyakit jantung, dan hipertensi.

Kalori dan gizi yang dibutuhkan oleh setiap orang berbeda-beda dan terdapat beberapa faktor yang memengaruhi hal tersebut. Faktor-faktor tersebut diantaranya adalah usia, jenis kelamin, metabolisme, faktor genetik, dan tingkat keaftikan seseorang. Selain itu, mungkin masih terdapat faktor lain yang memengaruhi kebutuhan kalori dan gizi seseorang. Akibat dari perbedaan kebutuhan kalori dan gizi tersebut, pola makan setiap orang pun menjadi berbeda-beda.

Berdasarkan permasalahan-permasalahan tersebut, dibutuhkan pengaturan pola makan yang benar untuk memenuhi kebutuhan kalori dan gizi. Salah satu cara untuk mengatur pola makan adalah dengan menentukan menu makan yang cocok berdasarkan tujuan yang diinginkan, misalnya dengan membatasi kalori atau gizi tertentu dan memaksimalkan kalori atau gizi tertentu yang berbeda dengan yang dibatasi. Contohnya pada orang yang menderita diabetes tetapi membutuhkan kalori yang banyak, makanan yang dikonsumsi harus dibatasi kandungan karbohidratnya tetapi tetap harus memaksimalkan kalori yang didapatkan. Pengaturan menu makan ini juga terkadang perlu mempertimbangkan biaya diperlukan, terutama untuk pada orang berpendapatan rendah.

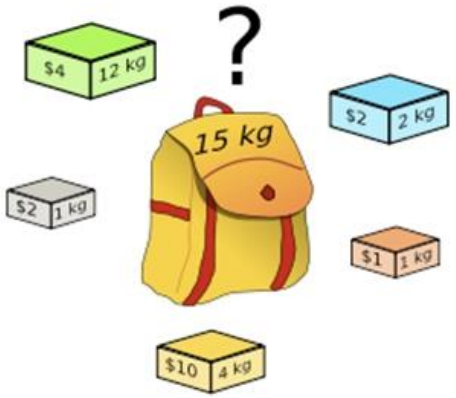
Makalah ini akan mengaplikasikan pemrograman dinamis (*dynamic programming*) yang merupakan salah satu topik mata kuliah "Strategi Algoritma" untuk menyelesaikan permasalahan penentuan menu makan yang memiliki batasan tertentu (kalori terbatas, gizi tertentu yang dibatasi, atau menu makan dengan anggaran tertentu) dan memaksimalkan kalori atau gizi tertentu yang didapatkan dari menu tersebut. Masalah ini dapat dimodelkan menjadi permasalahan *Integer (1/0) Knapsack* yang memiliki beberapa batasan (*constraint*). Makalah ini tidak akan dibahas masalah penentuan menu makan yang tidak memaksimalkan faktor tertentu.

II. DASAR TEORI

A. *Integer (1/0) Knapsack Problem*

Integer (1/0) Knapsack Problem adalah permasalahan memilih *n* buah objek untuk dimasukkan ke dalam sebuah *knapsack* (karung, tas, buntilan, dsb) yang memiliki kapasitas bobot *K* sedemikian rupa sehingga objek-objek yang dipilih tidak melebihi kapasitas *knapsack* namun keuntungan yang didapatkan maksimal. Setiap objek yang akan dipilih memiliki properti bobot (*weight*) w_i dan keuntungan atau nilai (*value*) v_i. Pada *Integer (1/0) Knapsack Problem*, setiap objek hanya dapat dipilih sekali. Secara matematis, *Integer (1/0) Knapsack Problem* dapat dituliskan sebagai berikut.

Maksimasi $\sum_{i=1}^{n} v_i x_i$
dengan batasan (*constraint*) $\sum_{i=1}^{n} w_i x_i \leq K$
dengan $x_i \in \{0, 1\}$
Solusi: $\{x_1, x_2, \ldots, x_n\}$



Gambar 1 – Ilustrasi *Integer (1/0) Knapsack Problem* [3]

Integer (1/0) Knapsack Problem dapat digunakan untuk memodelkan berbagai persoalan optimasi yang solusinya berupa *subset* yang memiliki batasan (*constraint*) tertentu. Contoh permasalahan yang dapat dimodelkan sebagai *Integer (1/0) Knapsack Problem* adalah penyimpanan barang di gudang dan pengaturan pengiriman barang seefisien mungkin untuk mendapatkan keuntungan terbesar.

Dalam teori komputasi, *Integer (1/0) Knapsack Problem* adalah satu persoalan yang termasuk ke dalam persoalan NP-Complete. Ini berarti belum ditemukan sebuah algoritma yang tepat dan mangkus (dapat menyelesaikan persoalan dalam waktu polinomial) untuk menyelesaikan *Integer (1/0) Knapsack Problem.* Walaupun belum ditemukan algoritma yang mangkus untuk menyelesaikan persoalan ini, terdapat algoritma yang berjalan dalam waktu *pseudo-polynomial* yang dapat menyelesaikan permasalahan ini, yaitu dengan menggunakan pemrograman dinamis (*dynamic programming)*. Selain dengan menggunakan pemrograman dinamis (*dynamic programming*), persoalan ini juga dapat diselesaikan dengan metode-metode lain, yaitu *exhaustive search* (mengenumerasi semua kemungkinan solusi), *branch and bound*, dan *greedy* (pendekatan dengan *greedy* tidak selalu menghasilkan solusi yang optimal, tetapi dapat digunakan mengaproksimasi bobot (*cost*) yang dapat digunakan untuk pendekatan *branch and bound*).

Terdapat beberapa variasi dari *Integer (1/0) Knapsack Problem* yang muncul dari berbagai aplikasi dari *Integer (1/0) Knapsack Problem.* Variasi-variasi ini muncul dengan mengganti parameter-parameter dari permasalahan semula, misalnya jumlah tujuan, jumlah *knapsack* yang digunakan, atau jumlah batasan (*constraint*). Variasi-variasi dari *Integer (1/0) Knapsack Problem* diantaranya adalah *Bounded Knapsack Problem* (setiap objek dapat dipilih sampai maksimal *c* kali), *Unbounded Knapsack Problem* (tidak ada batas pemilihan untuk setiap objek), *Multi-objective Knapsack Problem* (tujuan / hal yang ingin dioptimasi lebih dari 1), *Multi-dimensional Knapsack Problem* (batasan / *constraint* lebih dari 1), dan *Multiple Knapsack Problem* (jumlah *knapsack* lebih dari 1).

B. *Pemrograman Dinamis (Dynamic Programming)*

Pemrograman Dinamis (*Dynamic Programming*) adalah salah satu metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Istilah pemrograman dinamis pertama kali diperkenalkan oleh seorang professor dari Universitas Princeton yang juga bekerja di RAND *corporation*, bernama Richard Bellman, pada era tahun 1950-an. Istilah ini muncul karena perhitungan solusi dilakukan dengan menggunakan tabel-tabel. Pemrograman dinamis umumnya digunakan untuk menyelesaikan persoalan optimasi.

Pemecahan masalah dengan menggunakan metode pemrograman dinamis memiliki kemiripan dengan metode *greedy* yang juga membentuk solusi secara bertahap. Perbedaan kedua metode ini terletak pada rangkaian keputusan yang dipertimbangkan dalam penyelesaian masalah. Pada metode *greedy*, hanya satu rangkaian keputusan saja yang dipertimbangkan, yaitu rangkaian keputusan yang terdiri dari keputusan-keputusan yang memberikan hasil paling baik untuk setiap tahapnya (optimal lokal), yang harapannya dapat mengarah ke optimal global. Pada metode *greedy* keputusan yang dipilih pada tahap tertentu tidak mempertimbangkan rangkaian keputusan untuk tahapan berikutnya maupun rangkaian keputusan yang telah

dipilih pada tahap sebelumnya. Salah satu contoh permasalahan yang tidak dapat diselesaikan dengan metode *greedy* (tidak selalu benar) adalah *Integer (1/0) Knapsack Problem.*

Berbeda dengan metode *greedy*, metode pemrograman dinamis mempertimbangkan lebih dari satu rangkaian keputusan. Rangkaian-rangkaian keputusan tersebut dibuat dengan menggunakan prinsip optimalitas yang menyatakan bahwa jika solusi total optimal, maka bagian solusi sampai tahap ke-*k* juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap *k* ke tahap *k* + 1, kita dapat menggunakan hasil optimal dari tahap *k* tanpa harus kembali ke tahap awal. Berdasarkan prinsip optimalitas ini dapat dirumuskan bahwa ongkos pada tahap *k* + 1 adalah (ongkos yang dihasilkan pada tahap *k*) + (ongkos dari tahap *k* ke tahap *k* + 1). Pemilihan keputusan pada suatu tahap pada metode pemrograman dinamis dilakukan dengan menggunakan persyaratan optimasi dan kendala untuk membatasi pilihan yang harus dipertimbangkan.

Pemrograman dinamis dapat diterapkan pada persoalan yang memiliki karakterisitik sebagai berikut [2].

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Status merupakan kemungkinan masukan yang ada pada tahap tersebut.
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k* + 1.
8. Prinsip optimalitas berlaku pada persoalan tersebut.

Terdapat dua pendekatan yang dapat digunakan untuk menyelesaikan suatu persoalan dengan pemrograman dinamis yaitu sebagai berikut [2].

1. Pemrograman dinamis maju (*forward* atau *up-down*). Pemrograman dinamis bergerak mulai dari tahap 1, lalu ke tahap 2, 3, dan seterusnya sampai tahap ke-*n*. Rangkaian keputusannya adalah $x_1, x_2, \ldots, x_n$.
2. Pemrograman dinamis mundur (*backward* atau *bottom-up*). Pemrograman dinamis bergerak mulai tahap *n*, lalu mundur ke tahap n - 1, n – 2, dan seterusnya sampai tahap ke 1. Rangkaian keputusannya adalah $x_n, x_{n-1}, \ldots, x_1$.

Kedua pendekatan ini ekivalen dan sama-sama menghasilkan solusi yang optimum.

Pada umumnya untuk mengembangkan algoritma pemrograman dinamis terdapat empat langkah yaitu sebagai berikut [2].

1. Karakteristikkan struktur solusi optimum.
2. Definisikan secara rekursif nilai solusi optimal.
3. Hitung nilai solusi optimal secara maju atau mundur.
4. Konstruksi solusi optimal.

Pada implementasinya, pemrograman dinamis dapat diimplementasikan secara rekursif maupun secara iteratif dengan memanfaatkan larik (*array*) multi dimensi (tergantung banyaknya parameter yang merepresentasikan status untuk suatu tahap) untuk melakukan memorisasi. Implementasi secara iteratif akan membangun solusi dari bawah ke atas (*bottom-up*) dan menelusuri semua status yang mungkin untuk setiap tahap, sedangkan implemetasi secara rekursi akan membangun solusi dari atas ke bawah (*top-down*) membentuk sebuah pohon dan hanya akan mengunjungi status yang dibutuhkan pada setiap tahapnya untuk mencapai solusi optimal (tidak menelusuri semua status yang mungkin untuk setiap tahapnya).

III. Penyelesaian Multi-dimensional knapsack problem dengan Menggunakan Pemrograman Dinamis

Penyelesaian *Multi-dimensional Knapsack Problem* dengan menggunakan pemrograman dinamis akan dilakukan dengan mengikuti empat langkah yang telah disebutkan sebelumnya. Pada persoalan ini, tahap (*k*) adalah proses memasukkan objek ke-*k* ke dalam *knapsack* dan status (*y*) menyatakan kapasitas masing-masing batasan pada *knapsack* yang tersisa setelah memasukkan objek pada tahap sebelumnya. Status (*y*) dan kapasitas *knapsack* (*M)* untuk permasalahan ini dapat dinyatakan dalam bentuk vektor multidimensi tergantung banyaknya batasan (*constraint*) yang ada. Misalkan terdapat *m* batasan (*constraint*) maka $y = (y_1, y_2, \ldots, y_m)$.

Struktur solusi optimum untuk persoalan ini adalah sebagai berikut. Misalkan ketika memasukkan objek pada

tahap *k*, kapasitas *knapsack* menjadi ($y - w_k$, dengan w_k adalah bobot untuk setiap batasan yang ada). Untuk mengisi kapasitas yang tersisa akan digunakan prinsip optimalitas yang mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas yang tersisa, yaitu $f_{k-1}(y - w_k)$. Setelah itu, nilai keuntungan dari pemilihan objek pada tahap ke-*k* (p_k) ditambah dengan nilai $f_{k-1}(y - w_k)$ dibandingkan dengan keuntungan tanpa memilih objek pada tahap ke-*k* ($f_{k-1}(y)$) dan dipilih keputusan yang menghasilkan keuntungan yang lebih besar (mengambil objek ke-*k* atau tidak).

Relasi rekurens untuk persoalan ini adalah sebagai berikut.

$f_0(y) = 0, y_i = 0, 1, 2, \ldots, M_i$
untuk $1 \le i \le m$ (basis)
$f_k(y) = -\infty$, jika terdapat $y_i < 0$
untuk $1 \le i \le m$ (basis)
$f_k(y) = \max \{ f_{k-1}(y), p_k + f_{k-1}(y - w_k)\}$, (rekurens)
$k = 1, 2, \ldots, n$ dan $y_i \ge 0$ untuk $1 \le i \le m$

$f_k(y)$ adalah keuntungan optimum dari *Multi-dimensional Knapsack Problem* pada tahap *k* untuk kapasitas sebesar *y*, $f_0(y) = 0$ adalah nilai dari nilai persoalan *knapsack* kosong (tidak ada objek yang ingin dimasukkan ke dalam *knapsack*) dnegan kapasitas *y*, dan $f_k(y) = -\infty$ adalah nilai dari persoalan *knapsack* untuk kapasitas negatif (melanggar batasan / *constraint*). Solusi optimum dari dari *Multi-dimensional Knapsack Problem* adalah $f_n(M)$ dengan waktu komplesitas algoritmanya adalah $O(nM)$.

Berikut adalah contoh penerapan pemrograman dinamis untuk menyelesaikan *Multi-dimensional Knapsack Problem*. Misalkan batasan yang digunakan adalah kapasitas bobot dan volume maksimal yang dapat ditampung oleh *knapsack* ($m = 2$) dengan bobot dan volume maksimal masing-masing adalah 3 dan 2($M_1 = 3$, $M_2 = 2$) dan jumlah objek yang ingin dimasukkan ada 3 dengan rincian sebagai berikut.

Tabel I Rincian persoalan *knapsack* dengan 3 objek

Objek ke-*i*	bobot (w_i)	volume (v_i)	nilai (p_i)
1	2	1	65
2	3	2	80
3	1	1	30

Berikut adalah perhitungan untuk setiap tahapnya.

1. Tahap 1:
$f_1(y_1, y_2) = \max \{f_0(y_1, y_2), p_1 + f_0(y_1 - w_1, y_2 - v_1)\}$
$= \max \{f_0(y_1, y_2), 65 + f_0(y_1 - 2, y_2 - 1)\}$

Tabel II Tahap 1 penyelesaian persoalan *knapsack*

y_1	y_2	$f_0(y1, y2)$	$65 + f_0(y_1 - 2, y_2 - 1)$	$f_1(y_1, y_2)$	(x_1^*, x_2^*, x_3^*)
0	0	0	-∞	0	(0, 0, 0)
0	1	0	-∞	0	(0, 0, 0)
0	2	0	-∞	0	(0, 0, 0)
1	0	0	-∞	0	(0, 0, 0)
1	1	0	-∞	0	(0, 0, 0)
1	2	0	-∞	0	(0, 0, 0)
2	0	0	-∞	0	(0, 0, 0)
2	1	0	65	65	(1, 0, 0)
2	2	0	65	65	(1, 0, 0)
3	0	0	-∞	0	(0, 0, 0)
3	1	0	65	65	(1, 0, 0)
3	2	0	65	65	(1, 0, 0)

2. Tahap 2:
$f_2(y_1, y_2) = \max \{f_1(y_1, y_2), p_2 + f_1(y_1 - w_2, y_2 - v_2)\}$
$= \max \{f_1(y_1, y_2), 80 + f_1(y_1 - 3, y_2 - 2)\}$

Tabel III Tahap 2 penyelesaian persoalan *knapsack*

y_1	y_2	$f_1(y1, y2)$	$80 + f_1(y_1 - 3, y_2 - 2)$	$f_2(y_1, y_2)$	(x_1^*, x_2^*, x_3^*)
0	0	0	80 - ∞ = -∞	0	(0, 0, 0)
0	1	0	80 - ∞ = -∞	0	(0, 0, 0)
0	2	0	80 - ∞ = -∞	0	(0, 0, 0)
1	0	0	80 - ∞ = -∞	0	(0, 0, 0)
1	1	0	80 - ∞ = -∞	0	(0, 0, 0)
1	2	0	80 - ∞ = -∞	0	(0, 0, 0)
2	0	0	80 - ∞ = -∞	0	(0, 0, 0)
2	1	65	80 - ∞ = -∞	65	(1, 0, 0)
2	2	65	80 - ∞ = -∞	65	(1, 0, 0)
3	0	0	80 - ∞ = -∞	0	(0, 0, 0)
3	1	65	80 - ∞ = -∞	65	(1, 0, 0)
3	2	65	80 + 0 = 80	80	(0, 1, 0)

3. Tahap 3:
$f_3(y_1, y_2) = \max \{f_2(y_1, y_2), p_3 + f_2(y_1 - w_3, y_2 - v_3)\}$
$= \max \{f_2(y_1, y_2), 30 + f_2(y_1 - 1, y_2 - 1)\}$

Tabel IV Tahap 3 penyelesaian persoalan *knapsack*

y_1	y_2	$f_2(y1, y2)$	$30 + f_2(y_1 - 1, y_2 - 1)$	$f_3(y_1, y_2)$	(x_1^*, x_2^*, x_3^*)
0	0	0	30 - ∞ = -∞	0	(0, 0, 0)
0	1	0	30 - ∞ = -∞	0	(0, 0, 0)
0	2	0	30 - ∞ = -∞	0	(0, 0, 0)
1	0	0	30 - ∞ = -∞	0	(0, 0, 0)
1	1	0	30 + 0 = 30	30	(0, 0, 1)
1	2	0	30 + 0 = 30	30	(0, 0, 1)
2	0	0	30 -∞ = -∞	0	(0, 0, 0)
2	1	65	30 + 0 = 30	65	(1, 0, 0)
2	2	65	30 + 0 = 30	65	(1, 0, 0)
3	0	0	30 - ∞ = -∞	0	(0, 0, 0)
3	1	65	30 + 0 = 30	65	(1, 0, 0)
3	2	80	30 + 65 = 95	95	(1, 0, 1)

Solusi optimum dari persoalan ini adalah $X = (1, 0, 1)$ dengan $\sum p = f_3(3, 2) = 95$.

IV. Penerapan Pemrograman Dinamis Untuk Menentukan Menu Makan

Misalkan suatu rumah makan X menjual makanan dan minuman sebagai berikut beserta rincian kalori dan zat gizi lainnya, serta harganya.

Tabel V Rincian menu di rumah makan X

Menu	Kalori	Karbo (g)	Lemak (g)	Serat (g)	Protein (g)	Harga
Nasi Putih	129	27.9	0.3	0.4	2.7	4000
Nasi Merah	110	22.8	0.9	0.2	2.6	5000
Cumi Goreng	106	8.4	1.8	0.4	12.9	12000
Ayam Goreng	221	0	13.1	0	24.3	10000
Telur Dadar	98	1.2	7.1	0	6.8	5000
Sayur Campur	76	12	2	4	2.6	6000
Bayam	37	3.4	2.1	2.1	2.7	6000
Mie Goreng	221	40.3	3.3	0.2	7.3	14000
Jus Tomat	41	10.3	0.1	1	1.9	7000
Jus Jeruk	112	25.8	0.5	0.5	1.7	7000

Dari menu tersebut ingin dipilih serangkaian makanan dan atau minuman dengan tujuan mengoptimalkan kalori atau zat gizi tertentu yang didapat dengan batasan tertentu, misalnya batasan anggaran yang dimiliki, maksimal kalori yang didapat, atau batasan terhadap zat gizi tertentu. Persoalan ini dapat dimodelkan sebagai *Multi-dimensional Knapsack Problem* dengan asumsi setiap jenis makanan atau minuman hanya dapat dipilih satu kali saja. Yang menjadi status (*y*) dalam permasalahan ini adalah jumlah kalori atau zat gizi yang ingin dibatasi (bisa lebih dari 1) dengan tahapan sebanyak 10 (ada 10 buah objek yang dapat dipilih) dengan tujuan untuk mengoptimalkan suatu kandungan tertentu yang diperoleh dari makanan atau minuman yang dipilih.

Persoalan ini akan diselesaikan dengan menggunakan program dengan metode pemrograman dinamis yang telah dijelaskan pada bagian II. Program dibuat dengan menggunakan algoritma yang telah dibahas pada bagian III dengan pendekatan rekursif (*top-down*). Pengguna akan memasukkan batasan yang diinginkan (maksimal 2 batasan pada program) serta hal yang ingin dioptimasi kemudian program akan mengeluarkan rekomendasi menu makan yang sesuai dengan batasan-batasan yang diberikan. Untuk bilangan desimal akan dikonvesikan menjadi bilangan bulat dengan cara dikalikan dengan 10 saat perhitungannya. Berikut adalah hasil eksperimen yang telah dilakukan oleh penulis dengan tiga buah kasus uji (Program hanya menampilkan 1 solusi saja).

1. Kasus Uji 1

   Batasan pertama: harga ≤ Rp 15000
   Batasan kedua: tidak ada
   Optimasi: kalori

```
Menu Makan yang direkomendasikan:
Nasi Putih
Ayam Goreng
karbohidrat total(g): 27.9
kalori total(kal): 350
lemak total(g): 13.4
serat total(g): 0.4
protein total(g): 27
harga total: Rp 14000
Waktu eksekusi: 6.000 ms
```

Gambar 2 – Hasil Pengujian 1

2. Kasus Uji 2

   Batasan pertama: karbohidrat ≤ 30 g
   Batasan kedua: lemak ≤ 13 g
   Optimasi: protein

```
Menu Makan yang direkomendasikan:
Cumi Goreng
Telur Dadar
Sayur Campur
Bayam
karbohidrat total(g): 25
kalori total(kal): 317
lemak total(g): 13
serat total(g): 6.5
protein total(g): 25
harga total: Rp 29000
Waktu eksekusi: 7.000 ms
```

Gambar 3 – Hasil Pengujian 2

3. Kasus Uji 3

   Batasan pertama: kalori ≤ 250 kal
   Batasan kedua: protein ≤ 15 g
   Optimasi: serat

```
Menu Makan yang direkomendasikan:
Sayur Campur
Bayam
Jus Tomat
karbohidrat total(g): 25.7
kalori total(kal): 154
lemak total(g): 4.2
serat total(g): 7.1
protein total(g): 7.2
harga total: Rp 19000
Waktu eksekusi: 6.000 ms
```

Gambar 4 – Hasil Pengujian 3

Berdasarkan pengujian yang telah dilakukan, dapat dilihat bahwa hasil menu makan yang didapatkan dengan menggunakan pemrograman dinamis merupakan hasil yang paling optimal dari semua kemungkinan solusi yang

ada dengan waktu pemrosesan yang cukup cepat yaitu sekitar 6 ms. Namun, hasil yang didapatkan ini belum mempertimbangkan kategori menu yang dipilih, misalnya harus terdiri dari makanan 4 sehat 5 sempurna, dan hasil yang didapatkan mungkin akan lebih baik jika jumlah objek (menu) yang disediakan lebih banyak lagi. Selain itu, terdapat juga permasalahan pada batasan memori yang dapat digunakan oleh program sehingga program yang diimplementasikan dibatasi jumlah batasan (*constraint*) sebanyak semaksimal 2 dan juga dengan semakin banyaknya objek yang tersedia waktu pemrosesannya akan semakin lama. Untuk perkembangan lebih lanjut masalah pemilihan menu makan ini dapat dimodelkan menjadi persoalan *knapsack* yang *multi-objective* (lebih dari satu tujuan yang dioptimasi), *multi-dimensional* (lebih dari satu batasan), dan juga *unbounded* (tidak ada batasan pada jumlah setiap objek) serta membagi objek-objek menjadi kelompok-kelompok tertentu (*multiple choice knapsack*). Persoalan itu tentunya akan membutuhan algoritma yang berbeda untuk menyelesaikannya dan hal tersebut tidak akan dibahas di makalah ini. Studi lain yang juga menarik untuk dibahas adalah studi terhadap teknik-teknik optimasi lain yang dapat digunakan untuk menyelesaikan permasalahan ini, misalnya *branch and bound*. Untuk kode sumber program dapat dilihat di http://ideone.com/wZ04a0.

V. KESIMPULAN

Algoritma pemrograman dinamis dapat digunakan untuk menyelesaikan persoalan optimasi dikarenakan adanya salah satu prinsip yang digunakan dalam pemrograman dinamis yaitu prinsip optimalitas. Salah satu aplikasi dari pemrograman dinamis adalah untuk menyelesaikan persoalan *knapsack* dalam waktu *pseudo polynomial time* yang lebih baik dibandingkan dengan metode-metode lain, misalnya *bruteforce* dan *branch and bound*. Namun, kelemahannya terletak pada keterbatasan memori yang dimiliki untuk memorisasi setiap tahap yang ada.

Persoalan *knapsack* yang diselesaikan dengan pemrograman dinamis dapat digunakan untuk memodelkan berbagai persoalan yang berhubungan dengan pemilihan objek-objek tertentu dengan tujuan optimasi yang memiliki batasan tertentu. Salah satunya adalah untuk memilih menu makan yang memenuhi tujuan tertentu, misalnya mendapatkan kalori semaksimal mungkin, dengan batasan-batasan tertentu. Dari hasil eksperimen dapat dilihat bahwa hasil yang didapatkan merupakan hasil yang optimal. Namun, hasil tersebut belum bisa digunakan pada dunia nyata karena implementasi dari program belum mempertimbangkan berbagai faktor lainnya, seperti jenis makanan dan jumlah yang dapat dipilih untuk setiap makanannya. Terdapat juga faktor-faktor lain yang mungkin juga dapat memengaruhi pemilihan menu makan, seperti alergi terhadap makanan tertentu.

VI. UCAPAN TERIMA KASIH

Pertama penulis ingin mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena dengan rahmat dan karunai-Nya penulis dapat menyelesaikan makalah dengan judul "Penentuan Menu Makan dengan Pemrograman Dinamis" ini dengan baik. Penulis juga berterima kasih kepada para dosen pengajar mata kuliah IF2211 Strategi Algoritma, Dr. Ir. Rinaldi Munir, M.T.,Masayu Leylia Khodra, S.T., M.T., dan Dr. Nur Ulfa Maulidevi, S.T., M. Sc., atas bimbingan mereka selama ini dalam mengajar dan memberikan ilmu sehingga penulis mampu membuat makalah ini. Penulis juga berterima kasih kepada rekan-rekan yang telah memberikan semangat dan dorongan kepada penulis.

REFERENSI

[1] Kellerer, et al., "Knapsacks Problems", Springer, 2004.

[2] R. Munir, *Program Dinamis (Dynamic Programming).* 2015. [*Online*] Tersedia dalam http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Program%20Dinamis%20(2015).pdf. [diakses 17 Mei 2017 pukul 15.30 WIB].

[3] R. Munir, *Algoritma Brute Force*, 2016. [*Online*] Tersedia dalam http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2015-2016/Algoritma-Brute-Force-(2016).ppt. [diakses 17 Mei 2017 pukul 15.00 WIB].

[4] https://www.fatsecret.co.id/kalori-gizi/ [diakses 17 Mei 2017 pukul 16.00 WIB].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Mei 2017

Jordhy Fernando – 13515004