Penentuan Objek Wisata yang Dikunjungi dengan Menggunakan Algoritma *Divide and Conquer* dan Program Dinamis

Ellen - 13516007
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ellennugroho9@gmail.com

***Abstraksi*—Seringkali manusia memiliki kepenatan akan kehidupan dan kesibukkannya, oleh karena itu manusia membutuhkan rekreasi. Salah satu kegiatan rekreasi yang dilakukan adalah berwisata pada saat liburan tiba. Saat merencanakan liburan, sering ditemukan masalah-masalah. Salah satunya adalah bagaimana cara memanfaatkan waktu yang tersedia untuk mengunjungi tempat-tempat wisata, ditilik dari lama waktu yang akan dihabiskan dan seberapa ingin mengunjungi tempat wisata tersebut. Pada makalah ini, akan dibahas mengenai salah satu penyelesaian masalah ini dengan menggunakan *quick sort* dari algoritma *Divide and Conquer* serta program dinamis dengan pendekatan masalah 0/1 *knapsack*.**

Kata kunci—wisata, waktu, prioritas,* quick sort, *program dinamis.

I. PENDAHULUAN

Liburan merupakan salah satu cara yang sangat membantu dalam menjernihkan pikiran, terlebih untuk melepas penat dari kesibukan padat yang kian mengintai. Saat ini, masyarakat Indonesia memang sedang giat-giatnya mencari liburan. Hal ini disebabkan liburan anak-anak sekolah yang sudah dekat dan juga libur lebaran telah menanti, oleh karena itu tentulah makin banyak orang yang akan bepergian, baik ke antarkota maupun antarnegara. Menurut laporan dari *Mastercard* pada tahun 2016, Indonesia masuk dalam 10 besar jumlah wisatawan ke luar negeri di Asia Pasifik pada tahun 2021, diperkirakan dengan jumlah wisatawan 10,6 juta orang, dengan laju pertumbuhan 8,6% per tahunnya. Menurut Dyah Permatasari, *Head of Public Relation* Astindo, banyak faktor yang memengaruhi perkembangan jumlah wisatawan Indonesia. Salah satunya adalah peningkatan penduduk ekonomi kelas menengah hingga kelas atas yang menjadikan *travelling* sebagai sebuah gaya hidup, ditambah lagi karena perkembangan teknologi yang pesat sehingga informasi mengenai tujuan wisata mudah diperoleh. Belum lagi banyaknya *travel fair* dan promosi-promosi yang disediakan oleh maskapai penerbangan dan tempat-tempat penginapan membuat peminat wisata cukup membludak.

Sayangnya, walaupun *traveling* merupakan sebuah hobi – bahkan sebagian orang menganggap *traveling* merupakan gaya hidup – waktu yang diperoleh untuk melakukan kegiatan jalan-jalan ini cukup terbatas. Umumnya, wisatawan lebih memilih untuk melakukan perjalanan ke destinasi wisata sebanyak mungkin dengan waktu terbatas yang dimilikinya. Seringkali wisatawan kebingungan dalam menentukan rencana pengunjungan destinasi wisata yang sesuai, oleh karena itu banyak wisatawan yang menggunakan jasa pariwisata. Namun, ada kalanya ketika wisatawan merasa bahwa rencana perjanalan yang dibuat oleh agen jasa pariwisata kurang sesuai atau ingin menghemat biaya yang dikeluarkan dengan tidak menggunakan agen *travel* pariwisata, sehingga ingin membuat rencana perjalanannya sendiri.

Untuk membuat rencana perjalanan sendiri, terdapat banyak faktor yang harus dipertimbangkan. Sebagian faktor tersebut adalah: (1) waktu serta (2) skala prioritas. Agar dapat dimaksimalkan waktu dan tenaga kerja yang dikeluarkan untuk mendapatkan rencana perjalanan ini dengan maksimal, maka kemampuan berkomputasi akan sangatlah membantu.

Dengan memanfaatkan algoritma *divide and conquer* untuk melakukan *quick sort* pada daftar tujuan wisata dengan *cost*-nya serta pendekatan masalah dengan menggunakan prinsip 0/1 *knapsack* melalui algoritma *dynamic programming* untuk menentukan destinasi wisata mana saja yang perlu dimasukkan dalam rencana perjalanan dan destinasi mana saja yang tidak perlu dimasukkan dalam rencana perjalanan, penulis berharap bahwa makalah ini dapat membantu wisatawan dalam menentukan rencana perjalanannya.

II. DASAR TEORI

A. Divide and Conquer

Divide and conquer dulunya adalah strategi militer yang berhasil. Jenderal perang mengamati bahwa lebih mudah mengalahkan satu peleton tentara yang berkekuatan 50,000 orang diikuti dengan mengalahjan 50,000 tentara lainnya, dibanding mengalahkan langsung 100,000 tentara langsung. Strategi ini pun pernah diterapkan pada saat Belanda menjajah Indonesia, namun lebih dikenal sebagai strategi *divide et impera*, yang berarti pecah belah dulu baru kuasai. Kini, strategi ini menjadi strategi fundamental dalam ilmu komputer

dengan nama *Divide and Conquer*. Teknik ini banyak melahirkan algoritma penting seperti algoritma pencarian biner, algoritma pengurutan *merge sort* dan *quick sort*, algoritma transformasi fourier cepat, algoritma perkalian matriks Strassen, dan sebagainya.

Divide and Conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah menjadi beberapa submasalah yang lebih kecil, lalu memisahkan submasalah tersebut secara independen, kemudian menggabungkan solusi-solusi submasalah tersebut sehingga menjadi solusi masalah semula. Persisnya, algoritma *Divide and Conquer* disusun menjadi tiga proses utama, yaitu:

1. *Divide*

   Pembagian masalah menjadi beberapa submasalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil. Diusahakan untuk membaginya sama rata.

2. *Conquer*

   Pemecahan masalah masing-masing submasalah secara rekursif.

3. *Combine*

   Penggabungan solusi masing-masing submasalah sehingga membentuk solusi masalah semua.

Obyek permasalahan yang dibagi adalah masukan (*input*) yang berukuran *n*. Masukkan dapat berupa tabel, matriks, dan sebagainya – bergantung pada masalah yang ingin dipecahkan. Tiap submasalah memiliki karakteristik yang sama sehingga dapat diselesaikan dengan fungsi rekursif. Umumnya, submasalah yang dipecahkan dengan algoritma ini tidak saling beririsan.

Keuntungan dalam penggunaan algoritma *Divide and Conquer* adalah metode ini memberikan pendekatan sederhana untuk memecahkan masalah yang secara konseptual sulit dengan mereduksi ukuran masalah secara rekursif sampai menjadi kasus basis. Keuntungan kedua adalah algoritma ini dapat secara substansial mengurangi biaya kompleksitas.

Bentuk rinci dari algoritma *Divide and Conquer* dicirikan oleh batas ambang masukkan n_0 sehingga masalah tidak perlu dibagi lagi, ukuran submasalah hasil pembagian, jumlah submasalah seluruhnya, serta algoritma yang digunakan untuk menggabungkan solusi setiap submasalah.

Skema umum algoritma *Divide and Conquer* adalah sebagai berikut:

```
procedure DnC (input n: integer)
{
    Menyelesaikan masalah dengan
Algoritma Divide and Conquer
    Masukan: masukan yang berukuran n
    Keluaran: solusi dari masalah
semula
}

Deklarasi
    r, k: integer

Algoritma
    if n <= n0 then {ukuran masalah
sudah cukup kecil}
        SOLVE submasalah berukuran n
ini
    else
        bagi menjadi dua submasalah,
masing-masing berukuran n/2
        DnC(submasalah pertama)
        DnC(submasalah kedua)
        COMBINE solusi dua submasalah
    endif
```

Kompleksitas waktu prosedur *Divide and Conquer* dinyatakan dalam relasi rekurens sebagai berikut:

$$T(n) = \begin{cases} 2T(n/2) + cn & , n > 2 \\ a & , n = 2 \end{cases}$$

B. Pengurutan Cepat dengan Menggunakan Algoritma Divide and Conquer

Misalkan elemen-elemen dalam larik a yang akan diurut telah terdefinisi sejumlah n elemen. Algoritma pengurutan dengan metode *Divide and Conquer* adalah:

1. Untuk kasus n = 1, maka larik a sudah terurut (*solve*)
2. Untuk kasus n > 1, maka
   a. *Divide*: bagi tabel a menjadi dua bagian (a1 dan a2)
   b. *Conquer*: secara rekursif menerapkan algoritma *Divide and Conquer* pada masing-masing bagian
   c. *Combine*: menggabungkan hasil pengurutan a1 dan a2 sehingga diperoleh tabel a yang terurut

Terdapat dua buah pendekatan pengurutan dengan menggunakan *Divide and Conquer*, yaitu mudah membagi namun sulit menggabung dan sulit membagi namun mudah menggabung. Contoh pendekatan pertama adalah *merge sort* dan *insertion sort*, sementara contoh pendekatan kedua adalah *quick sort* dan *selection sort*. Pada bagian ini hanya akan dibahas *quick sort*-nya saja.

Algoritma *quick sort* melakukan partisi pada lariknya. Penggabungan larik tidak dinyatakan secara eksplisit dalam sebuah prosedur, karena proses partisi yang dilakukan mengurutkan sekaligus menggabungkan.

Teknik partisi tabel pada algoritma *quick sort* adalah sebagai berikut:

1. pilih $x \in \{a_1, a_2, \ldots, a_n\}$ sebagai elemen *pivot*
2. cari elemen $a_p >= x$ dari bagian kiri larik
3. cari elemen $a_q <= x$ dari bagian kiri larik
4. tukar a_p dan a_q
5. ulangi langkah kedua dari posisi p + 1 dan langkah ketiga dari posisi q – 1 sampai kedua pemindaian bertemu di tengah tabel

Berikut ini adalah *pseudo code* dari algoritma *quick sort*:

```
procedure QuickSort (input/output a:
array[1..n] of integer, input i, j:
integer)
{
    Mengurutkan tabel a[i..j] dengan
Quick Sort.
    Masukan: Tabel a[i..j] yang sudah
terdefinisi
    Keluaran: Tabel a[i..j] yang terurut
menaik
}

Deklarasi
    k : integer

Algoritma
    if i < j then
        Partisi(a, i, j, k)
        QuickSort(a, i, k)
        QuickSort(a, k+1, j)
    endif
```

Berikut ini merupakan *pseudo code* dari partisi tabel:

```
procedure Partisi (input/output a: array
[1..n] of integer, input i, j: integer,
output q: integer)
{
    Membagi tabel a[i..j] menjadi
subtabel a[i..q] dan a[q+1..j]
    Masukan: Tabel a[i..j] yang sudah
terdefinisi harganya
    Keluaran: Subtabel a[i..q] dan
subtabel a[q+1..j]
}

Deklarasi
    pivot, temp: integer

Algoritma
    pivot <- a[(i+j) div 2]
    p <- i
    q <- j
    repeat
        while a[p] < pivot do
            p <- p + 1
        endwhile
        while a[q] > pivot do
            q <- q - 1
        endwhile
        if p < q then
            temp <- a[p]
            a[p] <- a[q]
            a[q] <- temp
            p <- p + 1
            q <- q - 1
        endif
    until p > q
```

C. *Program Dinamis (*Dynamic Programming*)*

Dynamic Programming atau program dinamis adalah sebuah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Dengan penyelesaian persoalan dengan metode ini:

1. Terdapat sejumlah berhingga pilihan yang mungkin
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap selanjutnya
3. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap

Penyelesaian masalah dengan program dinamis mirip dengan penyelesaian masalah dengan algoritma *Greedy* karena sama-sama menyelesaikan masalah dengan bertahap. Dengan algoritma *greedy*, dibuat keputusan pada setiap tahap dengan cara mengambil pilihan yang paling menarik yang juga memenuhi ukuran optimasi yang digunakan. Pengambilan keputusan pada setiap tahap berdasarkan hanya pada informasi lokal, dan pada setiap tahap tidak dibuat keputusan yang salah. Pada permasalahan-permasalahan tertentu, algoritma *greedy* bekerja dengan baik. Namun, kebanyakan persoalan tidak dapat diselesaikan dengan algoritma *greedy* secara optimal. Hal ini dikarenakan pengambilan keputusan pada setiap langkah *greedy* tidak pernah mempertimbangkan lebih jauh apakah pilihan tersebut pada langkah-langkah selanjutnya merupakan pilihan yang tepat. Dari sejumlah pilihan yang menarik, kita tidak tahu pilihan mana yang terbaik sampai kita menuju proses yang lebih jauh ke depan. Misalnya dalam mencari lintasan terpendek, dengan algoritma *greedy* akan dilakukan enumerasi semua rangkaian keputusan yang mungkin dan memilih rangkaian keputusan yang terbaik (*exhaustive search*). Untuk graf dengan jumlah lintasan yang banyak, tentu saja *exhaustive search* tidak mangkus. Namun, dengan program dinamis, dapat secara drastis mengurangi pengenumerasian rangkaian keputusan yang tidak mengarah ke solusi optimum. Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas. Prinsip ini berbunyi: jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Prinsip optimalitas ini berarrti jika bekerja dari tahap k ke tahap k+1, dapat digunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal. Jika pada setiap tahap dihitung ongkos (*cost*), maka dapat dirumuskan:

Ongkos pada tahap k + 1 = (ongkos tahap k) + (ongkos tahap k ke tahap k + 1)

Dengan prinsip optimalitas ini dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya. Jadi, perbedaan mendasar antara algoritma *greedy* dan algoritma program dinamis adalah pada algoritma *greedy* hanya satu rangkaian keputusan yang dihasilkan, sedangkan pada metode program dinamis lebih dari satu rangkaian keputusan.

Program dinamis mengacu pada perhitungan dengan menggunakan tabel. Seperti halnya algoritma *greedy*, program dinamis digunakan untuk memecahkan masalah optimasi, yaitu masalah yang memiliki banyak kemungkinan solusi, dan diminta mencari solusi yang memberikan nilai yang optimum.

Program dinamis diterapkan pada persoalan yang memiliki karakteristik sebagai berikut:

1. Persoalan dapat dibagi menjadi beberapa tahap yang dimana pada setiap tahap hanya diambil satu keputusan
2. Masing-masing tahap terdiri dari sejumlah status yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada tahap tersebut. Jumlah status dapat berhingga maupun tak berhingga
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya
4. Ongkos pada suatu tahap meningkat secara teratur 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 untuk persoalan tersebut

Dalam menyelesaikan persoalan dengan program dinamis, dapat digunakan dua ancangan berbeda: maju dan mundur Penyelesaian secara maju atau mundur keduanya ekivalen dan menghasilkan solusi optimum yang sama. Namun kebanyakan pada contoh kasus, algoritma program dinamis secara mundur lebih mangkus.

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

1. Karakteristikan struktur solusi optimal
2. Definisikan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Kontruksikan solusi optimal

D. Penyelesaian Persoalan 0/1 Knapsack dengan Program Dinamis

0/1 *knapsack* merupakan suatu permasalahan bagaimana memilih objek dari sekian banyak dan berapa besar objek tersebut akan disimpan sehingga diperoleh suatu penyimpanan yang optimal dengan konstrain total berat objek tidak boleh melebih berat tertentu yang telah ditetapkan sebelumnya.

Misalkan terdapat sebuah contoh kasus dimana harus dipilih barang yang akan dibawa pada sebuah karung. Tahap (k) adalah proses memasukkan barang ke dalam karung, dan status (y) menyatakan kapasitas muat karung yang tersisa setelah memasukkan barang pada tahap sebelumnya.

Dari tahap ke-1, masukkan objek pertama ke dalam karung untuk setiap satuan kapasitas karung sampai batas kapasitas maksimumnya. Misalkan ketika memasukkan objek pada tahap k, kapasitas muat karung sekarang adalah $y - w_k$. Untuk mengisi kapasitas sisanya, diterapkan prinsip optimalitas dengan mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas sisa $y - w_k$ (yaitu $f_{k-1}(y - w_k)$). Selanjutnya, bandingkan nilai keuntungan dari objek pada tahap k (yaitu p_k) plus nilai $f_{k-1}(y - w_k)$ dengan keuntungan pengisian hanya k – 1 macam objek, $f_{k-1}(y)$. Jika $p_k + f_{k-1}(y - w_k)$ lebih kecil dari $f_{k-1}(y)$, maka objek yang ke-k tidak dimasukkan ke dalam karung, tetapi jika lebih besar, maka objek yang ke-k dimasukkan.

Relasi rekurens untuk persoalan ini adalah:

$$f_0(y) = 0, \quad y = 0, 1, 2, \ldots, M \qquad \text{(basis)}$$
$$f_k(y) = -\infty, \quad y < 0 \qquad \text{(basis)}$$

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

yang dalam hal ini:

1. $f_k(y)$ adalah kentungan optimum dari persoalan 0/1 *knapsack* pada tahap k untuk kapasitas karung sebesar y
2. $f_0(y) = 0$ adalah nilai dari persoalan *knapsack* kosong dengan kapasitas y
3. $f_k(y) = -\infty$ adalah nilai dari persoalan *knapsack* untuk kapasitas negatif. Solusi optimum dari persoalan 0/1 *knapsack* adalah $f_n(M)$

Contoh: n = 4 dan M = 5. Masing-masing objek memiliki berat dan profit sebagai berikut:

No.	**w_i**	**p_i**
1.	2	18
2.	1	15
3.	3	40
4.	2	20

Berikut ini merupakan tahapan-tahapan penyelesaian persoalan 0/1 *knapsack* menggunakan program dinamis maju:

1. Tahap 1

$f_1(y) = \max\{f_0(y), p_1 + f_0(y - w_1)\}$
$= \max\{f_0(y), 18 + f_0(y - 2)\}$

y	$f_0(y)$	$18 + f_0(y-2)$	$f_1(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0,0,0,0)
1	0	-∞	0	(0,0,0,0)
2	0	18	18	(1,0,0,0)
3	0	18	18	(1,0,0,0)
4	0	18	18	(1,0,0,0)
5	0	18	18	(1,0,0,0)

2. Tahap 2

$f_2(y) = \max\{f_1(y), p_2 + f_1(y - w_2)\}$
$= \max\{f_1(y), 15 + f_1(y - 1)\}$

y	$f_1(y)$	$15 + f_1(y - 1)$	$f_2(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0,0,0,0)
1	0	15	15	(0,1,0,0)
2	18	15	18	(1,0,0,0)
3	18	33	33	(1,1,0,0)

4	18	33	33	(1,1,0,0)
5	18	33	33	(1,1,0,0)

3. Tahap 3

$f_3(y) = \max\{f_2(y), p_3 + f_2(y - w_3)\}$
$= \max\{f_2(y), 40 + f_2(y - 3)\}$

y	$f_2(y)$	$40 + f_2(y-3)$	$f_3(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0,0,0,0)
1	15	-∞	15	(0,1,0,0)
2	18	-∞	18	(1,0,0,0)
3	33	40	40	(0,0,1,0)
4	33	55	55	(0,1,1,0)
5	33	58	58	(1,0,1,0)

4. Tahap 4

$f_4(y) = \max\{f_3(y), p_4 + f_3(y - w_4)\}$
$= \max\{f_3(y), 20 + f_3(y - 2)\}$

y	$f_3(y)$	$20 + f_3(y-2)$	$f_4(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0,0,0,0)
1	15	-∞	15	(0,1,0,0)
2	18	20	20	(0,0,0,1)
3	40	35	40	(0,0,1,0)
4	55	38	55	(0,1,1,0)
5	58	60	60	(0,0,1,1)

Melalui tahapan-tahapan di atas, dapat diketahui bahwa solusi optimum 0/1 *knapsack* yaitu pengambilan kombinasi barang ketiga dan keempat yang menghasilkan keuntungan sebesar 60.

III. Implementasi Algoritma

Dalam menentukan destinasi wisata yang dikunjungi dengan menggunakan algoritma *quick sort* dan program dinamis, diperlukan prasyarat yang harus dipenuhi, yaitu: (1) daftar wisata yang ingin dikunjungi, (2) waktu yang diperlukan untuk mengunjungi tempat wisata tersebut, serta (3) skala prioritas dengan jangkauan satu sampai sepuluh; semakin besar angka prioritas maka semakin diinginkan tempat wisata tersebut untuk dikunjungi. Diasumsikan bahwa tempat-tempat wisata yang dimasukkan oleh pengguna berada dalam satu wilayah. Selanjutnya, diperlukan juga masukan dari pengguna berupa waktu total yang dimilikinya.

Setelah itu, akan dilakukan pengurutan tempat tujuan wisata berdasarkan prioritas dengan *quick sort*. Akan dilakukan pengurutan terurut mengecil. Kemudian, hasil pengurutan tempat wisata akan diproses dengan program dinamis dengan melakukan pendekatan permasalahan menyerupai 0/1 *knapsack problem*.

Berikut ini merupakan *pseudo code* dari implementasi pemecahan masalah penentuan tempat-tempat tujuan wisata:

```
{
    Memproses daftar wisata pengguna
menjadi daftar wisata yang sudah
terseleksi
    Masukan: daftar wisata, kriteria
(waktu yang diperlukan dan yang
tersedia, skala prioritas)
    Keluaran: daftar wisata yang sudah
terseleksi
}

a1= array of tuple [name, timeneeded,
priority]
time = waktu tersedia
count = len(a1) {panjang larik a1}

QuickSort(a1, 0, count - 1) {quick sort
berdasarkan prioritas, terurut mengecil}

for (int j=0; j<=time; j++) do
    a2[0,j] = 0
end for

for (int i=1; i<=count; i++) do
    for (int j=0; j<=time; j++) do
        if a1[i][1] <= j then
            a2[i,j] = max(a2[i-1,j],
a2[i-1, j-a1[i][1]] + a1[i][2]])
        else
            a2[i,j] = a2[i-1,j]
    end for
end for

maxcapacity = a2[count][time]
i = count
w = time
a3 = array of integer {menyimpan index
daftar wisata}

while i and w>0 do
    if a2[i,w] != a2[i-1,w] then
        a3.add(i)
        w = w - a1[i][1]
        i = i-1
    else
        i = i-1
    endif
endwhile
```

Program akan menghasilkan tujuan wisata mana saja yang layak dikunjungi. Waktu yang diperlukan untuk mengunjungi tujuan wisata tersebut tidak akan melebihi dari kapasitas waktu tersedia.

Kompleksitas algoritma *quick sort* adalah sebesar O(n log n), dan kompleksitas algoritma 0/1 *knapsack* yang diterapkan adalah O(nW), dimana n adalah jumlah tujuan wisata dan W adalah waktu yang dimiliki.

IV. Studi Kasus

Diberikan daftar tujuan wisata Lembang, waktu yang dibutuhkan (dalam jam) untuk mengunjungi tujuan wisata tersebut, serta skala prioritas tujuan wisata sebagai berikut.

No.	Nama Wisata	Waktu Dibutuhkan (jam)	Prioritas
1.	Farmhouse	3	7

2.	Curug Tilu Leuwi Opat	4	4
3.	Gunung Tangkuban Perahu	6	9
4.	Maribaya Bandung	2	4
5.	Observatorium Bosscha	1	3
6.	Dusun Bambu	4	6
7.	Floating Market	3	5

Sementara itu, untuk perjalanan ini, *client* hanya memiliki waktu selama total 12 jam. Langkah pertama yang diambil adalah mengurutkan daerah tujuan wisata berdasarkan prioritas dengan algoritma *quick sort*. Berikut ini merupakan hasil dari pengurutan:

No.	Nama Wisata	Waktu Dibutuhkan (jam)	Prioritas
1.	Gunung Tangkuban Perahu	6	9
2.	Farmhouse	3	7
3.	Dusun Bambu	4	6
4.	Floating Market	3	5
5.	Curug Tilu Leuwi Opat	4	4
6.	Maribaya Bandung	2	4
7.	Observatorium Bosscha	1	3

Dengan menggunakan algoritma program dinamis, akan dilakukan aksi pertahap, dengan total tujuh tahap. Waktu yang dibutuhkan untuk mengunjungi tujuan wisata dianggap sebagai *cost* dan prioritas dianggap sebagai keuntungan. Berikut merupakan langkah-langkahnya:

1. Tahap 1

   $f_1(y) = \max\{f_0(y), p_1 + f_0(y - w_1)\}$
   $= \max\{f_0(y), 9 + f_0(y - 6)\}$

y	$f_0(y)$	$9 + f_0(y-6)$	$f_1(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	-∞	0	(0,0,0,0,0,0,0)
2	0	-∞	0	(0,0,0,0,0,0,0)
3	0	-∞	0	(0,0,0,0,0,0,0)
4	0	-∞	0	(0,0,0,0,0,0,0)
5	0	-∞	0	(0,0,0,0,0,0,0)
6	0	-∞	0	(0,0,0,0,0,0,0)
7	0	9	9	(1,0,0,0,0,0,0)
8	0	9	9	(1,0,0,0,0,0,0)
9	0	9	9	(1,0,0,0,0,0,0)
10	0	9	9	(1,0,0,0,0,0,0)
11	0	9	9	(1,0,0,0,0,0,0)
12	0	9	9	(1,0,0,0,0,0,0)

2. Tahap 2

   $f_2(y) = \max\{f_1(y), p_2 + f_1(y - w_2)\}$
   $= \max\{f_1(y), 7 + f_1(y - 3)\}$

y	$f_1(y)$	$7 + f_1(y-3)$	$f_2(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	-∞	0	(0,0,0,0,0,0,0)
2	0	-∞	0	(0,0,0,0,0,0,0)
3	0	7	7	(0,1,0,0,0,0,0)
4	0	7	7	(0,1,0,0,0,0,0)
5	0	7	7	(0,1,0,0,0,0,0)
6	0	7	7	(0,1,0,0,0,0,0)
7	9	7	9	(1,0,0,0,0,0,0)
8	9	7	9	(1,0,0,0,0,0,0)
9	9	7	9	(1,0,0,0,0,0,0)
10	9	16	16	(1,1,0,0,0,0,0)
11	9	16	16	(1,1,0,0,0,0,0)
12	9	16	16	(1,1,0,0,0,0,0)

3. Tahap 3

   $f_3(y) = \max\{f_2(y), p_3 + f_2(y - w_3)\}$
   $= \max\{f_2(y), 6 + f_2(y - 4)\}$

y	$f_2(y)$	$6 + f_2(y - 4)$	$f_3(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	-∞	0	(0,0,0,0,0,0,0)
2	0	-∞	0	(0,0,0,0,0,0,0)
3	7	-∞	7	(0,1,0,0,0,0,0)
4	7	6	7	(0,1,0,0,0,0,0)
5	7	6	7	(0,1,0,0,0,0,0)
6	7	6	7	(0,1,0,0,0,0,0)
7	9	13	13	(0,1,1,0,0,0,0)
8	9	13	13	(0,1,1,0,0,0,0)
9	9	13	13	(0,1,1,0,0,0,0)
10	16	13	16	(1,1,0,0,0,0,0)
11	16	15	16	(1,1,0,0,0,0,0)
12	16	15	16	(1,1,0,0,0,0,0)

4. Tahap 4

   $f_4(y) = \max\{f_3(y), p_4 + f_3(y - w_4)\}$
   $= \max\{f_3(y), 5 + f_3(y - 3)\}$

y	$f_3(y)$	$5 + f_3(y - 3)$	$f_4(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	-∞	0	(0,0,0,0,0,0,0)
2	0	-∞	0	(0,0,0,0,0,0,0)
3	7	5	7	(0,1,0,0,0,0,0)
4	7	5	7	(0,1,0,0,0,0,0)
5	7	5	7	(0,1,0,0,0,0,0)
6	7	12	12	(0,1,0,1,0,0,0)
7	13	12	13	(0,1,1,0,0,0,0)
8	13	12	13	(0,1,1,0,0,0,0)
9	13	12	13	(0,1,1,0,0,0,0)
10	16	18	18	(0,1,1,1,0,0,0)
11	16	18	18	(0,1,1,1,0,0,0)

12	16	18	18	(0,1,1,1,0,0,0)

5. Tahap 5

$f_5(y) = \max\{f_4(y), p_5 + f_4(y - w_5)\}$
$= \max\{f_4(y), 4 + f_4(y - 4)\}$

y	$f_4(y)$	$4 + f_4(y - 4)$	$f_5(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	-∞	0	(0,0,0,0,0,0,0)
2	0	-∞	0	(0,0,0,0,0,0,0)
3	7	-∞	7	(0,1,0,0,0,0,0)
4	7	4	7	(0,1,0,0,0,0,0)
5	7	4	7	(0,1,0,0,0,0,0)
6	12	4	12	(0,1,0,1,0,0,0)
7	13	11	13	(0,1,1,0,0,0,0)
8	13	11	13	(0,1,1,0,0,0,0)
9	13	11	13	(0,1,1,0,0,0,0)
10	18	16	18	(0,1,1,1,0,0,0)
11	18	17	18	(0,1,1,1,0,0,0)
12	18	17	18	(0,1,1,1,0,0,0)

6. Tahap 6

$f_6(y) = \max\{f_5(y), p_6 + f_5(y - w_6)\}$
$= \max\{f_5(y), 4 + f_5(y - 2)\}$

y	$f_5(y)$	$4 + f_5(y - 2)$	$f_6(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	-∞	0	(0,0,0,0,0,0,0)
2	0	4	4	(0,0,0,0,0,1,0)
3	7	4	7	(0,1,0,0,0,0,0)
4	7	4	7	(0,1,0,0,0,0,0)
5	7	11	11	(0,1,0,0,0,1,0)
6	12	11	12	(0,1,0,1,0,0,0)
7	13	11	13	(0,1,1,0,0,0,0)
8	13	16	16	(0,1,0,1,0,1,0)
9	13	17	17	(0,1,1,0,0,1,0)
10	18	17	18	(0,1,1,1,0,0,0)
11	18	17	18	(0,1,1,1,0,0,0)
12	18	22	22	(0,1,1,1,0,1,0)

7. Tahap 7

$f_7(y) = \max\{f_6(y), p_7 + f_6(y - w_7)\}$
$= \max\{f_6(y), 3 + f_6(y - 1)\}$

y	$f_6(y)$	$3 + f_6(y-1)$	$f_7(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*, x_5^*, x_6^*, x_7^*)$
0	0	-∞	0	(0,0,0,0,0,0,0)
1	0	3	3	(0,0,0,0,0,0,1)
2	4	3	4	(0,0,0,0,0,1,0)
3	7	7	7	(0,1,0,0,0,0,0)
4	7	10	10	(0,1,0,0,0,0,1)
5	11	10	11	(0,1,0,0,0,1,0)
6	12	14	14	(0,1,0,0,0,1,1)
7	13	15	15	(0,1,0,1,0,0,1)
8	16	16	16	(0,1,0,1,0,1,0)
9	17	19	19	(0,1,0,1,0,1,1)
10	18	20	20	(0,1,1,0,0,1,1)
11	18	21	21	(0,1,1,1,0,0,1)
12	22	21	22	(0,1,1,1,0,1,0)

Melalui tahapan-tahapan di atas, dapat diketahui bahwa solusi optimum dalam pemilihan tujuan destinasi wisata adalah dengan memilih Farmhouse, Dusun Bambu, Floating Market, serta Maribaya Bandung. Waktu total wisata yang diambil adalah 3 + 4 + 3 + 2 = 12 jam, tidak melebihi dari waktu yang tersedia.

V. KESIMPULAN

Algoritma *Divide and Conquer* merupakan algoritma yang memecah masalah menjadi beberapa submasalah untuk dipecahkan secara masing-masing submasalah, lalu baru menggabungkan solusi submasalah-submasalah tersebut menjadi solusi umum permasalahan. Algoritma ini memiliki keuntungan yaitu menyelesaikan permasalahan kompleks dengan langkah sederhana dan memiliki keunggulan dalam kecepatan kerja program.

Algoritma program dinamis merupakan sebuah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Algoritma ini memiliki keuntungan yaitu dapat menghasilkan lebih dari satu solusi secara mangkus dan sangkil dan relatif lebih efisien dibandingkan dengan algoritma *Greedy*.

Dengan menggabungkan kedua algoritma ini, maka dapat ditemukan solusi-solusi dari permasalahan yang ada, termasuk permasalahan pemilihan daerah destinasi tempat wisata yang dikunjungi. Prasyarat yang harus dipenuhi adalah daftar tempat wisata yang ingin dikunjungi beserta waktu yang akan dihabiskan dan skala prioritas (satu sampai sepuluh) per tempat, serta waktu total yang dimiliki. Setelah diproses lebih lanjut dengan menggunakan kedua algoritma ini, akan dihasilkan daftar wisata yang layak dikunjungi dengan waktu yang tidak akan melebihi waktu yang dimiliki.

UCAPAN TERIMA KASIH

Ucapan terima kasih penulis haturkan kepada Tuhan Yang Maha Esa atas berkat dan anugerah-Nya penulis dapat menyelesaikan makalah ini dengan tepat waktu, terlebih dengan segala kekurangan dari penulis. Penulis juga ingin mengucapkan terima kasih kepada Ibu Masayu Leylia Khodra selaku dosen pengajar penulis yang telah menurunkan ilmunya kepada kelas pengajar dengan penuh semangat sehingga dapat dimengerti oleh penulis. Tak lupa juga penulis mengucapkan terima kasih kepada keluarga serta teman-teman penulis yang tetap menyemangati dan membantu penulis dalam menentukan ide yang cocok.

REFERENSI

[1] Munir, Rinaldi. 2009. "Strategi Algoritma". Bandung: Informatika.

[2] Martello, Silvano; Toth, Paolo (1990). Knapsack problems: Algorithms and computer implementations. Inggris: University of Bologna.

[3] Anon. 2018. "33 Tempat Wisata di Bandung yang Asyik Buat Liburan". https://www.nativeindonesia.com/tempat-wisata-di-bandung/. Diakses pada tanggal 10 Mei 2018.

[4] Anon. 2016. "Hal yang Perlu Diperhatikan dalam Menyusun *Itinerary*". https://wisataweb.wordpress.com/2016/02/23/hal-yang-perlu-diperhatikan-dalam-menyusun-itinerary/. Diakses pada tanggal 10 Mei 2018.

[5] Susanti, Inda. 2017. "Orang Indonesia Wisata ke Luar Negeri Diprediksi *Tembus 10, 6 Juta*". https://ekbis.sindonews.com/read/1179640/34/orang indonesia-wisata-ke-luar-negeri-diprediksi-tembus-106-juta-1486999440. Diakses pada tanggal 11 Mei 2018.

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, 13 Mei 2018

Ellen
13516007