Pengaplikasian *Dynamic Programming* dan Algoritma *Divide and Conquer* dalam Menentukan *Bucket List* untuk Penjelajah

Rachel Sidney Devianti - 13515124
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515124@std.stei.itb.ac.id

***Abstrak*—Kegiatan menjelajah atau biasa disebut dengan *travelling* merupakan suatu hal yang sudah tidak asing lagi di telinga kita. Ada yang melakukannya untuk hobi, pekerjaan, atau sekedar hiburan semata. Dalam melakukan penjelajahan, banyak hal yang perlu diperhitungkan, mulai dari biaya tiket, akomodasi, serta barang-barang yang perlu dipersiapkan untuk keperluan selama bepergian. Terkadang, kesulitan dalam menentukan barang-barang yang harus dibawa menjadi salah satu kendala utama dalam bepergian. Kapasitas tas yang tersedia tidak sebanding dengan banyaknya barang yang dialokasikan. Pada kesempatan ini, akan dibahas mengenai pengaplikasian *dynamic programming* dalam menentukan barang-barang yang harus dibawa dengan memperhitungkan kapasitas maksimum, berat masing-masing barang, dan skala prioritas barang dari sudut pandang penjelajah. Selain itu, algoritma *divide and conquer* juga akan digunakan untuk mengurutkan barang berdasarkan skala prioritasnya.**

***Kata kunci*—menjelajah; *dynamic programming*; *divide and conquer*; kapasitas maksimum; skala prioritas.**

I. PENDAHULUAN

Menurut Kamus Besar Bahasa Indonesia, jelajah atau menjelajah merupakah kegiatan bepergian ke mana-mana untuk menyelidiki dan sebagainya. Sedangkan, penjelajah merupakan orang yang menjelajah. Aktivitas menjelajah atau *travelling* ini telah ada sejak jaman dahulu. Dilihat dari sejarahnya, disebut-sebut bahwa perusahaan Inggris Cox and Kings merupakan biro perjalanan tertua yang berdiri pada tahun 1758. Kegiatan menjelajah atau *travelling* ini kemudian menjadi semakin populer seiring dengan mulai berkembangnya jasa penerbangan pada tahun 1920.

Travelling merupakan sebuah metode untuk memperluas wawasan dan pengetahuan dengan mengunjungi tempat-tempat yang baru atau yang sudah pernah dikunjungi sebelumnya. *Travelling* dapat dilakukan secara individu ataupun berkelompok dalam berbagai usia dan tidak terbatas pada kalangan tertentu. Tujuan seseorang melakukan kegiatan jelajah beraneka ragam, ada yang melakukannya untuk pekerjaan, hobi, atau sekedar hiburan untuk melepas penat.

Menurut www.dailymail.co.uk[5], ada dua macam tipe penjelajah atau *traveller*, yaitu *backpacker* dan *flashpacker*. *Backpacker* umumnya berorientasi pada minimisasi anggaran (*budget oriented*) sehingga dibutuhkan perencanaan yang matang sesuai dengan besarnya anggaran yang dialokasikan untuk melakukan penjelajahan. Sedangkan, *flashpacker* umumnya lebih berorientasi kepada tujuan perjalanan atau pengalaman yang ingin diperoleh (*experienced oriented*).

Baik penjelajah tipe *backpacker* maupun tipe *flashpacker*, keduanya sama-sama membutuhkan perencanaan yang matang untuk melakukan kegiatan jelajah atau *travelling*. Beberapa hal yang harus diperhatikan untuk dibawa pada saat melakukan kegiatan jelajah menurut www.eaglecreek.com[6], antara lain:

1. uang dan dokumen-dokumen penting

   Untuk melakukan kegiatan jelajah atau *travelling* sebaiknya tak lupa untuk mempersiapkan paspor, KTP/kartu pelajar, asuransi kesehatan, kontak hotel tempat menginap, kontak tur yang diikuti (apabila mengikuti tur), tiket transportasi, dan kontak darurat yang dapat dihubungi.

2. tas khusus untuk barang-barang personal

   Dalam mempersiapkan barang-barang personal, sebaiknya mempertimbangkan untuk membawa telepon genggam, pengisi daya atau *charger*, kamera, pelantang telinga atau *headphone*, adapter, buku, majalah, buku petunjuk, peta daerah yang dituju, pembersih tangan, kacamata, dan obat-obatan.

3. koper yang ringan dan mampu memuat barang-barang yang diperlukan

   Dalam menentukan koper yang akan dibawa harus mempertimbangkan kemudahan untuk dibawa ke mana-mana. Koper harus menjadi sarana penunjang kegiatan jelajah bukan dipandang sebagai beban yang harus dibawa ke mana-mana.

Seringkali penjelajah atau *traveller* menemukan kesulitan dalam menyusun dan memilah-milah barang-barang yang harus dibawa. Kapasitas tas yang tersedia tidak cukup menampung seluruh barang-barang yang dikehendaki untuk dibawa. Akibatnya, barang-barang yang tergolong vital riskan untuk tertinggal.

Oleh sebab itu, pada kesempatan ini, penulis akan mengaplikasikan *dynamic programming* dan algoritma *divide and conquer* dalam membantu penjelajah untuk mengidentifikasi barang-barang apa saja yang harus dibawa. Hal-hal yang akan diperhitungkan antara lain: kapasitas maksimum tas yang direpresentasikan dalam satuan berat (gram), berat masing-masing barang (gram), dan skala prioritas masing-masing barang dilihat dari sudut pandang penjelajah (0 – 5). Untuk menunjang penelitian, penulis juga membuat program aplikasi sederhana untuk mensimulasikan perhitungan tersebut yang diberi nama 'Traveler's Bucket List'.

II. DASAR TEORI

A. Program Dinamis atau *Dynamic Programming*

Program dinamis merupakan metode pemecahan masalah dengan menguraikan solusi menjadi sekumpulan langkah atau tahapan. Solusi dari persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan. Karakteristik dari program dinamis, yaitu: terdapat sejumlah pilihan yang mungkin, solusi pada setiap tahap dibangun menggunakan hasil solusi tahap sebelumnya, dan menggunakan persyaratan optimasi dan kendala untuk membatasi jumlah pilihan yang harus dipertimbangkan pada setiap tahap[1].

Pada umumnya, cara penyelesaian masalah dengan menggunakan metode program dinamis memiliki kemiripan dengan cara penyelesaian masalah dengan menggunakan algoritma *greedy*. Algoritma *greedy* juga membentuk solusi secara bertahap. Pada algoritma *greedy*, keputusan pada setiap tahap didasarkan pada pilihan yang paling memenuhi ukuran optimasi yang digunakan. Perbedaannya, pengambilan keputusan pada algoritma ini hanya didasarkan pada informasi lokal dan pada setiap tahap, keputusan yang diambil sudah bersifat final (tidak dapat berubah).

Pada kebanyakan persoalan, algoritma *greedy* tidak dapat memberikan solusi yang optimal. Hal ini dikarenakan pada algoritma *greedy*, pengambilan keputusan pada setiap langkah tidak pernah mempertimbangkan lebih jauh apakah pilihan tersebut pada langkah-langkah selanjutnya merupakan pilihan yang juga menghasilkan solusi yang optimal. Dari sejumlah pilihan yang ada, tidak akan pernah diketahui pilihan mana yang terbaik hingga menuju proses yang lebih jauh ke depan.

Algoritma lain yang dapat menutupi kelemahan algoritma *greedy* adalah algoritma *brute force*, atau lebih tepatnya *exhaustive search*. *Exhaustive search* akan mengenumerasikan semua rangkaian keputusan yang mungkin dan memilih rangkaian keputusan yang terbaik. Oleh sebab itu, solusi yang diberikan oleh *exhaustive search* akan selalu optimal. Namun, untuk persoalan dengan skala besar, *exhaustive search* ini cenderung tidak mangkus. Contohnya, pada persoalan *Travelling Salesman Problem*, algoritma ini memiliki kompleksitas waktu $\boldsymbol{O(n.n!)}$ dan pada persoalan *Integer Knapsack*, algoritma ini memiliki kompleksitas waktu $\boldsymbol{O(2^n)}$.

Dengan menggunakan program dinamis, enumerasi rangkaian keputusan yang tidak mengarah ke solusi optimum dapat dikurangi secara drastis. Dari sudut pandang ini, program dinamis tentu dapat menutupi kelemahan dari *exhaustive search*. Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan Prinsip Optimalitas, yang berbunyi, "jika solusi total optimal, maka bagian dari solusi sampai tahap ke-*k* juga merupakan solusi yang optimal." Prinsip ini berarti jika kita bekerja dari tahap *k* ke tahap *k + 1*, kita dapat menggunakan hasil optimal dari tahap *k* tanpa harus kembali ke tahap awal[1]. Dengan menggunakan prinsip ini, dapat dijamin bahwa pengambilan keputusan pada suatu tahap merupakan keputusan yang benar untuk tahap-tahap selanjutnya.

Berikut ini akan dijelaskan lebih detail mengenai karakteristik persoalan program dinamis[1].

1. Persoalan dibagi menjadi beberapa tahap, pada setiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status. Status merupakan berbagai macam kemungkinan masukan yang ada pada tahap tersebut.
3. Hasil keputusan yang diambil pada setiap tahap ditransformasi dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. *Cost* pada setiap tahap meningkat secara teratur dengan bertambahnya jumlah tahapan.
5. *Cost* pada setiap tahap bergantung pada tahap-tahap yang telah dilewati dan *cost* pada tahap itu sendiri.
6. Keputusan terbaik pada setiap tahap bersifat independen terhadap keputusan pada tahap sebelumnya.
7. Keputusan terbaik untuk setiap status pada tahap *k*, memberikan keputusan terbaik untuk setiap status pada tahap *k + 1*.
8. Prinsip Optimalitas berlaku.

Pada penyelesaian persoalan dengan program dinamis, dapat menggunakan pendekatan maju atau pendekatan mundur. Program dinamis maju bergerak dari tahap 1, 2, 3, dan sebagainya. Sedangkan program dinamis mundur bergerak dari tahap *n*, *n – 1*, *n – 2*, dan sebagainya. Kedua jenis pendekatan ini menghasilkan solusi optimum yang sama[3].

Program dinamis dapat digunakan untuk menyelesaikan persoalan-persoalan, seperti: 0/1 *knapsack*, *Travelling Salesman Problem*, lintasan terpendek, dan lain-lain. Namun, pada kesempatan ini hanya akan dibahas mengenai penyelesaian persoalan 0/1 *knapsack* dengan program dinamis.

0/1 *knapsack* merupakan persoalan untuk menentukan objek apa saja yang dapat dimuat untuk mendapatkan keuntungan yang paling maksimum dengan konstrain total berat objek tidak boleh melebihi berat tertentu yang telah ditetapkan sebelumnya. Tinjau persoalan 0/1 *knapsack* dengan *n* (jumlah objek) = 4 dan *W* (kapasitas *knapsack*) = 5. Masing-masing objek memiliki berat dan profit yang akan dijabarkan pada tabel berikut:

No.	w_i	p_i
1.	2	12
2.	1	15
3.	3	50

No.	w_i	p_i
4.	1	10

Metode penyelesaian persoalan dengan program dinamis dipilih menggunakan pendekatan maju. Kemudian, didefinisikan relasi rekurens untuk persoalan ini, yaitu[3]:

$f_0(y) = 0, \quad y = 0, 1, 2, ..., W$ (basis)

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

$f_k(y) = max \{f_{k-1}(y), p_k + f_{k-1}(y - w_k)\}, \quad k = 1, 2, 3, ..., n$ (rekurens)

Berikut ini adalah tabel tahapan-tahapan penyelesaian persoalan 0/1 *knapsack* menggunakan program dinamis[4].

Tahap 1

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

$= max (f_0(y), 12 + f_0(y - 2)\}$

y	$f_0(y)$	$12 + f_0(y - 2)$	$f_1(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0, 0, 0, 0)
1	0	-∞	0	(0, 0, 0, 0)
2	0	12	12	(1, 0, 0, 0)
3	0	12	12	(1, 0, 0, 0)
4	0	12	12	(1, 0, 0, 0)
5	0	12	12	(1, 0, 0, 0)

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	12	15	15	(0, 1, 0, 0)
3	12	27	27	(1, 1, 0, 0)
4	12	27	27	(1, 1, 0, 0)
5	12	27	27	(1, 1, 0, 0)

Tahap 3

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

$= max (f_2(y), 15 + f_2(y - 3)\}$

y	$f_2(y)$	$50 + f_2(y - 3)$	$f_3(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0, 0, 0, 0)
1	15	-∞	15	(0, 1, 0, 0)
2	15	-∞	15	(0, 1, 0, 0)
3	27	50	50	(0, 0, 1, 0)
4	27	65	65	(0, 1, 1, 0)
5	27	65	65	(0, 1, 1, 0)

Tahap 4

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

$= max (f_3(y), 10 + f_3(y - 1)\}$

y	$f_3(y)$	$10 + f_3(y - 1)$	$f_4(y)$	$(x_1^*, x_2^*, x_3^*, x_4^*)$
0	0	-∞	0	(0, 0, 0, 0)
1	15	10	15	(0, 1, 0, 0)
2	15	25	25	(0, 1, 0, 1)
3	50	25	50	(0, 0, 1, 0)
4	65	60	65	(0, 1, 1, 0)
5	65	75	75	(0, 1, 1, 1)

Melalui tahapan-tahapan yang dibangkitkan, diketahui bahwa solusi optimum diperoleh melalui kombinasi barang ke-2, 3, dan 4 yang menghasilkan keuntungan sebesar 75.

B. Algoritma *Divide and Conquer*

Algoritma *divide and conquer* merupakan metode pemecahan masalah yang dilakukan dengan cara membagi suatu masalah menjadi beberapa upamasalah yang berukuran lebih kecil. Masing-masing upamasalah dicari solusinya secara independen kemudian menggabungkan solusi-solusi tersebut sehingga menjadi solusi yang utuh untuk menyelesaikan permasalahan semula.

Lebih rincinya, algoritma *divide and conquer* tersusun oleh tiga proses utama[2]:

1. *Divide* : membagi masalah menjadi beberapa upamasalah yang memiliki kemiripan dengan masalah semula.
2. *Conquer* : menyelesaikan masing-masing upamasalah. Umumnya, masing-masing upamasalah diselesaikan secara rekursif.
3. *Combine* : menggabungkan solusi dari masing-masing upamasalah sehingga membentuk solusi untuk permasalahan semula.

Algoritma *divide and conquer* dapat digunakan untuk menyelesaikan berbagai jenis persoalan, seperti: mencari minimum/maksimum, mencari pasangan titik terdekat, dan melakukan pengurutan. Pada pengaplikasian algoritma *divide and conquer* untuk melakukan pengurutan, terbagi menjadi dua metode, yaitu: mudah dibagi/sulit digabung dan sulit dibagi/mudah digabung. Contoh pengurutan dengan metode yang mudah dibagi/sulit digabung adalah *merge sort* dan *insertion sort*. Sedangkan contoh pengurutan dengan metode sulit dibagi/mudah digabung adalah *quick sort* dan *selection sort*[2].

Dilihat dari kompleksitas waktunya, *quick sort/merge sort* lebih baik dibandingkan dengan *insertion sort/selection sort*. Pada kesempatan ini, penulis menggunakan algoritma *quick sort* untuk melakukan pengurutan.

Pada algoritma *quick sort*, proses pembagian tabel disebut partisi. Penggabungan tabel tidak dinyatakan secara eksplisit karena proses partisi sekaligus mengurutkan dan menggabungkan dengan menggunakan bantuan pivot (salah satu elemen pada tabel). Berikut ini adalah *screenshot* dari algoritma *quick sort* dalam bahasa Java.

```
public void QuickSort(int i, int j) {
  if (i < j) { /* ukuran T > 1 */
    Divide(i,j); /* partisi menjadi T[i..k] dan T[l..j] */
    QuickSort(i, pright);
    /* mengurutkan T[i..k] dengan Quick Sort left - pright*/
    QuickSort(pleft, j);
    /* mengurutkan T[l..j] dengan Quick Sort pleft - right*/
  }
}


public void Divide(int left, int right) {
  int pivot, tempInt;
  String tempString;
  pivot = things[(left+right)/2][1];
  /* pivot adalah elemen tengah */
  pleft = left;
  pright = right;
  do
  {
    while (things[pleft][1] > pivot) {
      pleft++;
    }
    /* things[pleft][1] <= pivot */
    while (things[pright][1] < pivot) {
      pright--;
    }
    /* things[pright][1] >= pivot */
    if (pleft <= pright) {
    /* menukar things[pleft][1] dengan things[pright][1] */
      for (int i = 0; i < 3; i++) {
        tempInt = things[pleft][i];
        things[pleft][i] = things[pright][i];
        things[pright][i] = tempInt;
      }
      tempString = nameThings[pleft];
      nameThings[pleft] = nameThings[pright];
      nameThings[pright] = tempString;
      /* menentukan awal pemindaian berikutnya */
      pleft++;
      pright--;
    }
  }
  while (pleft <= pright);
}
```

Gambar 1. Algoritma Quick Sort dalam Bahasa Java
Sumber: koleksi pribadi (dibuat 13 Mei 2017)

III. METODOLOGI

Program dinamis dan algoritma *quick sort* akan digunakan untuk membantu penjelajah dalam menentukan barang-barang yang layak untuk dibawa pada saat *travelling*. Beberapa hal yang perlu dipersiapkan sebelum melakukan perhitungan, yaitu:

1. Barang-barang apa saja yang menurut prakiraan dibutuhkan oleh penjelajah pada saat melakukan kegiatan jelajah atau *travelling*.
2. Kisaran berat dari masing-masing barang tersebut dalam satuan gram agar lebih akurat.

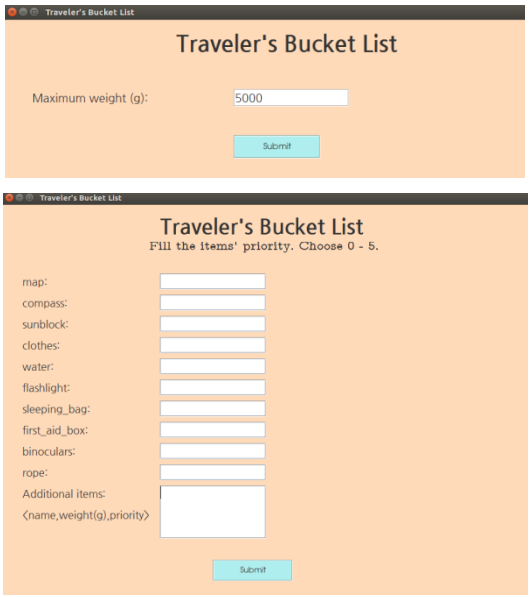
Berikut ini adalah *list* prakiraan barang-barang yang diperlukan untuk melakukan *travelling*:

1. peta : 100 gram
2. kompas : 500 gram
3. pelindung matahari : 100 gram
4. pakaian : 1100 gram
5. air minum : 1000 gram
6. senter : 400 gram
7. kantong tidur : 750 gram
8. P3K : 500 gram
9. teropong : 750 gram
10. tali : 600 gram

Selanjutnya, dibutuhkan input dari penjelajah berupa:

1. Kapasitas maksimum yang disediakan oleh penjelajah untuk melakukan kegiatan jelajah atau *travelling*.
   Kapasitas ini direpresentasikan dengan satuan berat (gram).
2. Input berupa skala 0 – 5 yang merepresentasikan prioritas masing-masing barang menurut sudut pandang penjelajah.
   Skala 0 menyatakan bahwa penjelajah tidak membutuhkan barang tersebut, sedangkan, skala 5 menyatakan bahwa penjelajah sangat membutuhkan barang tersebut. Skala 1 – 4 menyesuaikan dengan batas bawah dan batas atas yang telah didefinisikan sebelumnya.
3. Input tambahan barang-barang yang dianggap penting untuk dibawa.
   Penjelajah dapat melakukan *listing* terhadap barang-barang yang juga dianggap penting dengan format nama barang, berat barang dalam satuan gram, dan skala prioritas untuk masing-masing barang tersebut.

Berikut ini adalah layar penerimaan input dari penjelajah yang telah dibuat oleh penulis menggunakan Java Swing.



Gambar 2. Ilustrasi Penerimaan Input dari Penjelajah
Sumber: koleksi pribadi (dibuat 14 Mei 2017)

Selanjutnya, akan dilakukan pengurutan terhadap barang-barang yang telah dipilih oleh penjelajah berdasarkan skala prioritas. Pengurutan ini dilakukan mulai dari barang yang memiliki skala yang tertinggi hingga barang yang memiliki skala yang terendah. Barang yang memiliki skala prioritas bernilai 0 akan diabaikan karena dianggap tidak akan dibutuhkan oleh penjelajah. Adapun dalam melakukan pengurutan, digunakan algoritma *divide and conquer*, yaitu

quick sort dengan kompleksitas waktu pada kasus rata-rata adalah ***O(n log n)***.

Setelah dilakukan pengurutan, akan diaplikasikan program dinamis untuk melakukan perhitungan terhadap barang-barang yang harus dibawa dengan mempertimbangkan kapasitas maksimum yang tersedia (gram), skala prioritas masing-masing barang, dan berat masing-masing barang (gram). Tahapan-tahapan pada program dinamis dibangkitkan menggunakan rumus:

$f_0(y) = 0, \quad y = 0, 1, 2, ..., W$ (basis)
$f_k(y) = -\infty, \quad y < 0$ (basis)
$f_k(y) = max\{f_{k-1}(y), p_k + f_{k-1}(y - w_k)\}, \quad k = 1, 2, 3, ..., n$ (rekurens)

Keterangan:
p_k : skala prioritas barang pada tahap *k*
w_k : berat barang pada tahap *k* (gram)
y : besar kapasitas tertentu pada masing-masing tahap (gram)
W : kapasitas maksimum koper/tas (gram)
n : jumlah barang yang masuk ke dalam perhitungan

Berikut ini adalah kode program untuk membangkitkan tabel pada masing-masing tahapan solusi program dinamis.

```
//prosedur untuk membangkitkan tabel tahapan-tahapan dari persoalan
public void calculation() {
  System.out.println("Solution Table:");
  for (int i = 0; i < length; i++) {
    for (int j = 0; j < weightMax + 1; j++) {
      tableSolution[j][0] = optimumPrev[j][0];
      int temp = j - thingsProcess[i][0];
      if (temp < 0) {
        tableSolution[j][1] = -1;
      }
      else {
        tableSolution[j][1] = thingsProcess[i][1] + optimumPrev[temp][0];
      }

      int indexMax = maximum(tableSolution[j][0], tableSolution[j][1]);
      optimumNext[j][0] = tableSolution[j][indexMax];
      if (indexMax == 0) {
        for (int k = 1; k < length + 1; k++) {
          optimumNext[j][k] = optimumPrev[j][k];
        }
      }
      else {
        for (int k = 1; k < length + 1; k++) {
          if (k == i + 1) {
            optimumNext[j][k] = 1;
          }
          else {
            optimumNext[j][k] = optimumPrev[temp][k];
          }
        }
      }
    }

    for (int j = 0; j < weightMax + 1; j++) {
      for (int k = 0; k < length + 1; k++) {
        optimumPrev[j][k] = optimumNext[j][k];
      }
    }
  }
}
```

Gambar 3. Prosedur untuk Membangkitkan Tahapan Solusi Program Dinamis
Sumber: koleksi pribadi (dibuat 13 Mei 2017)

Setelah tabel tahapan-tahapan tersebut dibangkitkan, akan diperoleh solusi dari persoalan berupa kombinasi barang-barang yang jika dijumlahkan akan menghasilkan nilai skala prioritas yang terbesar, namun, total berat barang-barang tersebut tidak akan melebihi kapasitas maksimum yang tersedia. Program dinamis dapat menghasilkan lebih dari satu solusi persoalan apabila terdapat lebih dari satu kombinasi barang-barang yang menghasilkan nilai skala prioritas yang terbesar.

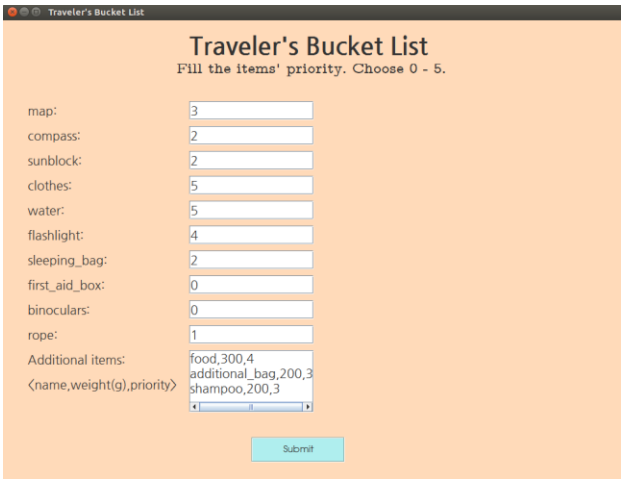
Pada bagian awal, telah dilakukan pengurutan dengan menggunakan algoritma *quick sort* sehingga penjelajah akan memperoleh solusi yang telah terurut berdasarkan tingkat kepentingan masing-masing barang. Berdasarkan solusi yang diberikan oleh program dinamis tersebut, penjelajah dapat mengetahui barang-barang apa saja yang layak untuk dibawa; barang-barang yang penting dan tidak melebihi kapasitas maksimum (kapasitas koper/tas) yang disediakan.

IV. Studi Kasus

4.1. Simulasi Penentuan Barang untuk Penjelajah

Pada upabab ini akan dibahas mengenai langkah-langkah pengaplikasian program dinamis dan algoritma *divide and conquer* dalam menentukan *bucket list* untuk penjelajah atau *traveller*. Seorang penjelajah ingin menjelajah dari suatu kota ke kota lain. Namun, kapasitas tas yang dialokasikan relatif sedikit dibandingkan dengan banyaknya barang yang harus dimuat. Oleh sebab itu, akan digunakan program dinamis dan algoritma *divide and conquer* untuk membantu melakukan pemilahan terhadap barang-barang tersebut. Berikut ini adalah langkah-langkah untuk memperoleh solusi:

1. Penjelajah mengukur kapasitas koper/tas yang dapat dibawa, yaitu sebesar 4300 g atau 4,3 kg.
2. Penjelajah mengisi skala prioritas masing-masing barang atau menambah *list* barang-barang yang dianggap penting. Berat masing-masing barang dapat dilihat pada bagian metodologi.



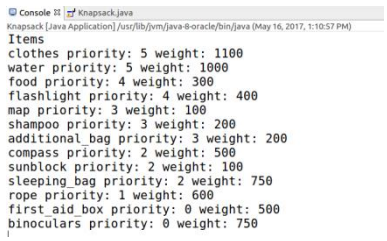
Gambar 4. Skala Prioritas untuk Masing-Masing Barang
Sumber: koleksi pribadi (dibuat 14 Mei 2017)

3. Pengurutan terhadap barang-barang yang memiliki skala prioritas 1 – 5 dengan terurut mengecil. Berikut ilustrasi pengurutan yang dilakukan oleh algoritma *quick sort*.

pivot=2												
3	2	2	5	5	4	2	0	0	1	4	3	3
p	p											q
3	3	2	5	5	4	2	0	0	1	4	3	2
		p									q	
3	3	3	5	5	4	2	0	0	1	4	2	2
			p	p	p	p				q		
3	3	3	5	5	4	4	0	0	1	2	2	2
						q	p,q	q	q			
pivot=5							pivot=1					
3	3	3	5	5	4	4	0	0	1	2	2	2
p				q	q	q	p					q
5	3	3	5	3	4	4	2	0	1	2	2	0
	p		q					p			q	
5	5	3	3	3	4	4	2	2	1	2	0	0
	q	p,q							p	q		
pivot=5		pivot=3					2	2	2	1	0	0

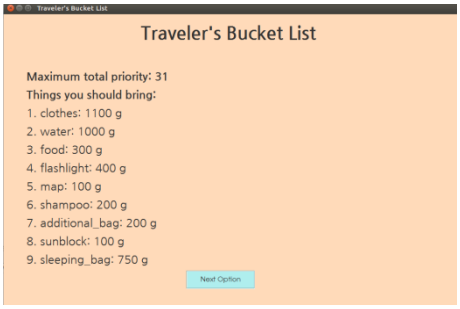
5	5	3	3	3	4	4			q	p		
p	q	p				q	pivot=2			pivot=0		
5	5	4	3	3	4	3	2	2	2	1	0	0
q	p		p		q		p		q	p	p	q
		4	4	3	3	3	2	2	2	1	0	0
				p,q				p,q			q	p
		4	4	3	3	3	2	2	2	pivot=1		
			q		p		q		p	1	0	
		pivot=4			pivot=3					p,q	q	
		4	4		3	3				1	0	
		p	q		p	q			q		p	
		4	4		3	3						
		q	p		q	p						

Hasil pengurutan dari algoritma *quick sort* untuk persoalan ini:



Gambar 5. Hasil Pengurutan Barang berdasarkan Skala Prioritas
Sumber: koleksi pribadi (dibuat 13 Mei 2017)

4. Pengaplikasian program dinamis dalam menentukan barang-barang apa saja yang perlu untuk dibawa dengan tidak melebihi kapasitas maksimum. Berikut ini adalah hasil dari pengaplikasian program dinamis yang menghasilkan dua macam solusi yang berbeda. Adapun untuk kedua solusi, total prioritas yang dihasilkan bernilai 31, nilai ini merupakan total prioritas maksimum untuk kapasitas 4300 gram.



Gambar 6. Solusi 1 Studi Kasus 1
Sumber: koleksi pribadi (dibuat 14 Mei 2017)

Gambar 7. Solusi 2 Studi Kasus 1
Sumber: koleksi pribadi (dibuat 14 Mei 2017)

Adapun pembangkitan tahapan pada program dinamis tidak dijelaskan untuk studi kasus ini dikarenakan kapasitas maksimum yang bernilai terlalu besar sehingga tabel tahapan sulit untuk diamati. Oleh sebab itu, pembangkitan tahapan pada program dinamis akan dijelaskan pada studi kasus selanjutnya.

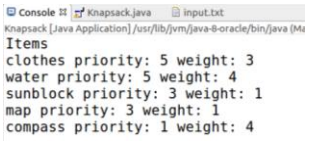
4.2. Simulasi Pembangkitan Tahapan-Tahapan Program Dinamis

Pada upabab ini penjelasan akan difokuskan pada pembangkitan tahapan-tahapan pada program dinamis yang direpresentasikan dengan tabel. Berikut ini akan dibahas kasus kedua persoalan pemilihan barang. Tinjau ilustrasi berikut.

1. Kapasitas maksimum bernilai $W = 8$.
2. Berikut ini adalah *list* barang-barang yang akan dimuat beserta beratnya, $n = 5$. Selain itu, juga ditentukan skala prioritas dari masing-masing barang yang direpresentasikan dengan angka 0 – 5.

Nama Barang	w_i	p_i
peta	1	3
kompas	4	1
pelindung matahari	1	3
pakaian	3	5
air minum	4	5

3. Barang-barang diurutkan dengan menggunakan algoritma *quick sort* berdasarkan skala prioritas seperti yang telah dijelaskan pada studi kasus sebelumnya.



Gambar 8. Hasil Pengurutan Barang berdasarkan Skala Prioritas
Sumber: koleksi pribadi (dibuat 13 Mei 2017)

4. Berikut ini adalah hasil kompilasi pada *Console* prosedur calculation() yang telah dijabarkan pada bagian metodologi untuk membangkitkan tahapan-tahapan. Total tahapan berjumlah 5 tahap sesuai dengan jumlah barang. Adapun penyelesaian persoalan dilakukan dengan pendekatan maju.

Tahap 1

y	$f_0(y)$	$5 + f_0(y-3)$	$f_1(y)$	x_1	x_2	x_3	x_4	x_5
0	0	-1	0	0	0	0	0	0
1	0	-1	0	0	0	0	0	0
2	0	-1	0	0	0	0	0	0
3	0	5	5	1	0	0	0	0
4	0	5	5	1	0	0	0	0
5	0	5	5	1	0	0	0	0
6	0	5	5	1	0	0	0	0
7	0	5	5	1	0	0	0	0
8	0	5	5	1	0	0	0	0

Tahap 2

y	$f_1(y)$	$5 + f_1(y-4)$	$f_2(y)$	x_1	x_2	x_3	x_4	x_5
0	0	-1	0	0	0	0	0	0
1	0	-1	0	0	0	0	0	0
2	0	-1	0	0	0	0	0	0
3	5	-1	5	1	0	0	0	0
4	5	5	5	0	1	0	0	0
5	5	5	5	0	1	0	0	0
6	5	5	5	0	1	0	0	0
7	5	10	10	1	1	0	0	0
8	5	10	10	1	1	0	0	0

Tahap 3

y	$f_2(y)$	$3 + f_2(y-1)$	$f_3(y)$	x_1	x_2	x_3	x_4	x_5
0	0	-1	0	0	0	0	0	0
1	0	3	3	0	0	1	0	0
2	0	3	3	0	0	1	0	0
3	5	3	5	1	0	0	0	0
4	5	8	8	1	0	1	0	0
5	5	8	8	0	1	1	0	0
6	5	8	8	0	1	1	0	0
7	10	8	10	1	1	0	0	0
8	10	13	13	1	1	1	0	0

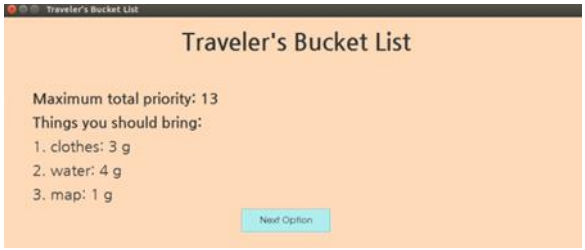
Tahap 4

y	$f_3(y)$	$3 + f_3(y-1)$	$f_4(y)$	x_1	x_2	x_3	x_4	x_5
0	0	-1	0	0	0	0	0	0
1	3	3	3	0	0	0	1	0
2	3	6	6	0	0	1	1	0
3	5	6	6	0	0	1	1	0
4	8	8	8	1	0	0	1	0
5	8	11	11	1	0	1	1	0
6	8	11	11	0	1	1	1	0
7	10	11	11	0	1	1	1	0
8	13	13	13	1	1	0	1	0

Tahap 5

y	$f_4(y)$	$1 + f_4(y-4)$	$f_5(y)$	x_1	x_2	x_3	x_4	x_5
0	0	-1	0	0	0	0	0	0
1	3	-1	3	0	0	0	1	0
2	6	-1	6	0	0	1	1	0
3	6	-1	6	0	0	1	1	0
4	8	1	8	1	0	0	1	0
5	11	4	11	1	0	1	1	0
6	11	7	11	0	1	1	1	0
7	11	7	11	0	1	1	1	0
8	13	9	13	1	1	0	1	0

5. Melalui pengaplikasian program dinamis seperti yang telah dijabarkan sebelumnya, dapat ditentukan barang-barang yang perlu dibawa dengan tidak melebihi kapasitas maksimum, yaitu: *clothes*, *water*, dan *map*. Total prioritas maksimum yang diperoleh adalah 13.



Gambar 9. Solusi Studi Kasus 2
Sumber: koleksi pribadi (dibuat 14 Mei 2017)

V. KESIMPULAN

Program dinamis memiliki karakteristik unik karena dapat memecahkan persoalan optimasi dengan menjamin keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya. Selain itu, program dinamis dapat menghasilkan lebih dari satu solusi untuk suatu persoalan seperti yang ditemukan pada studi kasus pertama. Algoritma *divide and conquer* (*quick sort*) yang digunakan juga efektif untuk melakukan pengurutan karena kompleksitas waktunya yang bersifat polinomial yaitu ***O(n log n)***.

Program dinamis dan algoritma *divide and conquer* (*quick sort*) ini terbukti dapat digunakan untuk membantu penjelajah melakukan pemilahan terhadap barang-barang yang harus dibawa. Pemilahan ini dilakukan dengan mempertimbangkan kapasitas maksimum yang tersedia, skala prioritas barang (0 – 5) berdasarkan input pengguna, serta berat masing-masing barang. Solusi dari persoalan akan memberikan daftar barang-barang yang dapat dibawa dengan kapasitas maksimum sebesar *W* dan menghasilkan total skala prioritas maksimum. Program dalam bahasa Java yang telah dibuat untuk mendukung pembuatan makalah ini dapat diakses pada *link* berikut ini: https://github.com/crahels/BucketListTraveler.

DAFTAR PUSTAKA

[1] R. Munir, "Program Dinamis," dalam *Strategi Algoritma*. Bandung: Informatika, 2009, hlm. 167-176.

[2] R. Munir, "Algoritma Divide and Conquer," dalam *Strategi Algoritma*. Bandung: Informatika, 2009, hlm. 70-93.

[3] M. Sniedovich, "The Rest is Mathematics and Experience," dalam *Dynamic Programming Foundation and Principles*, ed. 2. Melbourne: CRC Press, 2010, hlm 183-194.

[4] H. Kellerer, dkk, "The Subset Sum Problem," dalam *Knapsack Problems*, ed. 1. Berlin: Springer-Verlag, 2004, hlm 75-85.

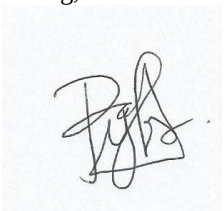
[5] Anon., *Are You a Backpacker or a Flashpacker?*. [*Online*] Tersedia dalam: http://www.dailymail.co.uk/travel/travel_news/article-2893401/Are-backpacker-flashpacker-Research-suggests-adventure-seekers-prefer-little-bit-extra-comfort.html. [diakses 15 Mei 2017 pukul 11.00 WIB].

[6] Anon., *What to Pack: The Ultimate Travel Packing Checklist*. [*Online*] Tersedia dalam: http://www.eaglecreek.com/blog/what-pack-ultimate-travel-packing-checklist. [diakses 15 Mei 2017 pukul 13.00 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, 18 Mei 2017



Rachel Sidney Devianti
13515124