Solusi Optimal *Coin Change Problem* dengan Algoritma *Greedy* dan *Dynamic Programming*

Indah Purwitasari Ihsan
Teknik Informatika
Universitas Sains dan Teknologi Jayapura
indah.ihsan1@gmail.com

Evanita V. Manullang
Teknik Informatika
Universitas Sains dan Teknologi Jayapura
eva.manullang@gmail.com

***Abstrak* – Masalah optimasi merupakan masalah untuk mencari solusi optimum. Manusia selalu berinovasi untuk mencari algoritma yang tepat agar bisa menyelesaikan masalah tersebut. Algoritma yang dicari adalah yang memiliki ketepatan solusi yang cukup optimal dan kecepatan yang cukup tinggi. Persoalan optimasi yang dibahas dalam penelitian ini adalah persoalan penukaran uang. Bagaimana cara agar dapat menukar uang dengan nilai yang sama namun dengan jumlah uang yang lebih sedikit. Persoalan tersebut diselesaikan dengan mengimplementasikan algoritma *greedy* dan algoritma *dynamic programming*. Dari penelitian yang dilakukan, telah dapat diketahui dengan jelas kelebihan dan kekurangan masing-masing algoritma tersebut serta telah dibuat program aplikasi dengan menggunakan bahasa pemograman delphi 7 dan java.**

***Kata Kunci** :greedy, dynamic programming, penukaran uang*

I. PENDAHULUAN

Masalah optimasi merupakan masalah untuk mencari solusi optimum. Manusia selalu berinovasi untuk mencari algoritma yang tepat agar bisa menyelesaikan masalah optimasi tersebut. Salah satu dari masalah optimasi adalah persoalan penukaran uang(coin change). Dimana persoalan ini membahas tentang bagaimana cara agar dapat menukar uang dengan nilai yang sama namun dengan jumlah uang yang lebih sedikit dari berbagai satuan. Persoalan ini merupakan persoalan umum yang biasa terjadi dalam kehidupan sehari-hari umat manusia. Contohnya ketika kita ingin menukarkan uang Rp. 100.000 dengan uang 50.000 dan 10.000, dalam hal ini banyak kombinasi dan kemungkinan yang bisa terjadi. Maka dari itu dibutuhkan suatu algortima yang tepat[9].

Dalam persoalan diatas, algoritma yang dicari adalah yang memiliki ketepatan solusi yang cukup optimal dan kecepatan yang cukup tinggi.. Untuk menyelesaikan persoalan penukaran uang dapat digunakan beberapa algoritma Untuk mengetahui algoritma yang paling baik, dilakukan analisis terhadap dua algoritma pemecahan masalah yaitu *Greedy*, dan *Dynamic Programming*.

Algoritma *greedy* merupakan algoritma penyelesaian dengan mengutamakan kecepatan menemukan satu solusi, yang dianggap optimal, sedangkan *dynamic programming* mengutamakan kemangkusan algoritma tetapi memiliki tingkat kerumitan cukup tinggi.

II. LANDASAN TEORI

A. Persoalan Penukaran Uang

Persoalan penukaran uang dapat dijabarkan sebagai berikut: Misalkan kita ingin menukarkan uang senilai *V* dengan sekumpulan uang koin *N* dari berbagai satuan (dengan asumsi tersedia banyak koin untuk setiap satuan, sehingga tidak mungkin terjadi kekurangan koin). Berapa jumlah minimum koin yang diperlukan tersebut?

Contoh: koin-koin yang tersedia bernilai 4, 3, dan 1 (dalam jumlah yang banyak untuk tiap satuan). Bagaimana cara menukarkan uang sebanyak 6 dengan koin-koin tersebut?

Dengan sedikit pengetahuan, dapat diketahui bahwa jumlah koin yang diperlukan tidak akan lebih dari jumlah uang dibagi dengan pecahan terkecil koin dan tidak akan kurang dari jumlah uang dibagi dengan pecahan terbesar koin. Jadi bisa dipastikan bahwa jumlah koin yang diperlukan tidak akan lebih dari 6.

B. Algoritma *Greedy*

Secara harfiah, *greedy* berarti rakus atau tamak. Algoritma *Greedy* merupakan algoritma sederhana yang paling populer untuk pemecahan persoalan optimasi seperti halnya persoalan coin change. Prinsip *greedy* adalah: "*take what you can get now!*" (mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan).

Sebuah algoritma dikatakan *greedy* apabila membuat pilihan optimum local pada setiap langkah dengan harapan akan menemukan solusi yang optimal, Untuk beberapa kasus, algoritma *greedy* mudah diimplementasikan dan memiliki kecepatan program yang cukup tinggi [3].

Dengan kata lain algoritma *greedy* membentuk solusi langkah per langkah (*step by step*) dimana pada setiap langkah terdapat banyak pilihan yang perlu di eksplorasi, dan solusi yang paling bagus yang akan diambil sebagai solusi optimum.

Sebuah masalah harus mengandung 2 unsur sebagai berikut agar algoritma *greedy* dapat bekerja[3] :

1. Memiliki sub-struktur optimal
   Solusi optimal masalah mengandung solusi optimal pada sub masalah.

2. Memiliki property *greedy*

   Jika kita membuat sebuah pilihan terbaik pada saat itu dan memecahkan sub masalahnya kemudian, kita masih mendekati solusi optimal. Kita tidak akan pernah mempertimbangkan kembali pilihan yang sebelumnya kita ambil.

C. Algoritma *Dymanic Programming*

Program Dinamis (*dynamic programming*) adalah 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[5].

Pada penyelesaian persoalan menggunakan algoritma ini, persoalan dibagi – bagi menjadi beberapa tahap (*stage*), yang pada setiap tahap diambil keputusan yang terbaik. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.

Pada metode *greedy* hanya satu rangkaian keputusan yang pernah dihasilkan, sedangkan pada metode program dinamis lebih dari satu rangkaian keputusan. Hanya rangkaian keputusan yang memenuhi prinsip optimalitas yang akan dihasilkan.

III. IMPLEMENTASI ALGORITMA *GREEDY* DAN ALGOTIMA *DYNAMIC PROGRAMMING*

Pada bagian ini akan dibahas mengenai menyelesaikan persoalan penukaran uang menggunakan algoritma *greedy* dan algoritma *dynmic programming*.

A. Persoalan Penukaran Uang - Versi *Greedy*

Diberikan sebuah target jumlah *V* sen dan sebuah list satuan uang *N* koin. Kita mempunyai nilaiKoin[i] (dalam mata uang sen) untuk tipe koin i ∈ [0 . . N-1], Berapa jumlah minimum koin yang dibutuhkan untuk memperoleh jumlah *V*. diaumsikan kita memiliki banyak koin untuk tiap satuan.

Contoh 1:

Diketahui : $V = 10$, $N = 2$, nilaiKoin = {1,5}

Penyelesaian : Dengan menggunakan algoritma greedy, pilih satu koin terbesar dan jumlahnya dengan koin sebelumnya yang telah diambil tidak lebih dari 10 (pada kondisi awal, jumlah koin sebelumnya adalah 0).

Langkah-langkah algoritma *greedy* :

1. Pilih 1 buah koin 5 sen; total = 5 sen
2. pilih 5 buah koin 1 sen; total = 5+1+1+1+1+1 = 10

Jadi, total koin yang dipakai adalah 6 koin.

Dengan menggunakan algoritma greedy didapatkan solusi menggunakan semua nilaiKoin yang tersedia.

Masalah penukaran uang dinyatakan sebagai himpunan : Nilai uang yang ditukar: *V*, Himpunan koin (*multiset*): $\{i_1, i_2, \ldots, i_n\}$. Himpunan solusi: $X = \{x_1, x_2, \ldots, x_n\}$, $x_i = 1$ jika i_i dipilih, $x_i = 0$ jika i_i tidak dipilih.

algoritma *greedy* memiliki komponen sebagai berikut:

- *Himpunan kandidat(i)*: himpunan koin yang merepresentasikan nilai 1, 5, paling sedikit mengandung satu koin untuk setiap nilai.
- *Himpunan solusi(x)*: total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
- *Fungsi seleksi*: pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
- *Fungsi layak*: memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
- *Fungsi obyektif*: jumlah koin yang digunakan minimum.

Skema Algoritma *greedy* :

```
function CoinExchange(input i : himpunan_koin, V
: integer) → himpunan_koin
{ mengembalikan koin-koin yang total nilainya =
  v, tetapi jumlah koinnya minimum,
  Masukan:  himpunan  kandidat  i,  Keluaran:
  himpunan solusi yang bertipe himpunan_kandidat}

Deklarasi
   S : himpunan_koin
   x : koin

Algoritma
  S ← {}
  while (∑(nilai semua koin di dalam S) ≠ V) and
        (i ≠ {} ) do
     x ← koin yang mempunyai nilai terbesar
     i ← i - {x}
     if  (∑(nilai semua koin di dalam S) + nilai
          koin x ≤ V then
       S ← S ∪ {x}
     endif
  endwhile

  if (∑(nilai semua koin di dalam S) = V then
     return S
  else
     write('tidak ada solusi')
  endif
```

Di sini, pengulangan akan berhenti bila V yang dicari ditemukan dalam senarai atau seluruh senarai sudah di bandingkan. Jika jumlah elemen senarai adalah *n*, maka kompleksitas waktu terburuknya adalah *O*(n), yaitu V tidak ditemukan. kita andaikan V terdapat pada elemen terakhir atau V tidak ditemukan, dengan kata lain kompleksitas waktunya adalah *O*(n). Pemilihan bilangan terbesar yang memenuhi

tidak akan memakan waktu lebih dari n jadi kompleksitasnya *O*(n).

B. Persoalan Penukaran Uang - Versi *General*

Algoritma *greedy* tidak selalu menghasilkan solusi yang optimal. Jika sebuah persoalan memenuhi persyaratan *greedy* maka kita dapat menggunakan algoritma *greedy* namun untuk persoalan penukaran uang dalam versi *general*, kita memerlukan algoritma yang lebih mangkus yaitu algoritma *dynamic programming*.

Perhatikan contoh berikut ini :

Diberikan $V = 7$, $N = 4$, nilaiKoin = {1,3,4,5}

Jika menggunakan *greedy* tentunya pertama kita akan memilih koin yang terbesar lebih dulu yaitu 5, sehingga solusi yang terbentuk adalah 5+1+1 = 7, algoritma *greedy* menemukan solusi dengan menggunakan 3 koin. Solusi ini tidaklah optimal. solusi optimalnya adalah hanya dengan menggunakan 2 koin saja yaitu koin 4 + koin 3 = 7.

Perhatikan lagi pengertian algoritma *dynamic programming*, dimana cara menyelesaikan persoalan dengan membagi langkah menjadi tahap – tahap, dan setiap tahap saling berkaitan.

Solusi : Dalam algoritma ini gunakan pencarian rekurensi change(value) dimana value merupakan sisa koin yang kita gunakan untuk :

1. Change(0) = 0; dibutuhkan 0 koin untuk memproduksi 0 sen.
2. Change(<0) = ∞ ; dalam praktek, hanya digunakan nilai positif yang lebih besar dari 0.
3. Change(1) = 1 + min(change(value-nilaiKoin[i])); $\forall\, i \in [0\,..\,N-1]$; coba semua kemungkinan.

Jadi, solusinya berada dalam change(v).

Langkah – langkah solusi algoritma *dynamic programming* :

1. Karakterisasikan struktur dari solusi penukaran uang
   - Deinisikan C(v) menjadi angka minimum dari koin yang kita butuhkan untuk menukar v sen.
   - Jika kita mengetahui bahwa solusi optimal untuk menukar v sen menggunakan sebuah koin dari himpunan koin yang tersedia [i], maka :

     C[v] = 1+C[$v - i_k$]

2. Tentukan nilai rekursif dari solusi optimal
   Berdasarkan teori solusi algoritma *dynamic programming* :

$$C[v] = \begin{cases} 0 & jika\ v = 0, \\ \infty & jika\ v < 0, \\ 1 + \min\{C[v - i_k]\} & jika\ v \geq 1 \end{cases}$$

3. Hitung nilai dengan cara bottom-up dan buat solusi optimal
   Hindari menguji C[v] untuk v < 0 dengan memastikan bahwa $v \geq i_k$ sebelum mengerjakan $C[v - i_k]$.

   Kita menambahkan array *denom*[*1 . . n*], dimana *denom*[v] adalah satuan dari koin yang digunakan dalam solusi optimal untuk masalah penukaran uang v sen.

Skema :

```
COMPUTE-CHANGE(n, d, k)
1  C[0] := 0
2  for v := 1 to n do
3      C[v] := ∞
4      for j := 1 to k do
5          if v ≥ i_k i and 1+C[v − i_k] < C[v] then
6          C[v] := 1+C[v − i_k]
7          denom[v] := i_k
8  return c
```

4. Cetak solusi optimal

```
PRINT-COINS(denom, v)
  if  v > 0
     PRINT-COINS(denom, v − denomv])
     print denom[v]
```

Kita akan menyelesaikan permasalahan pada Contoh 1 menggunakan *dynamic programming*:

Diketahui : $V = 10$, $N = 2$, nilaiKoin = {1,5}

Maka :

C[0] = 0;

$$C[1] = min\begin{cases} 1 + C[1-1] = 1 \\ 1 + C[1-5] = \infty \end{cases}$$

$$C[2] = min\begin{cases} 1 + C[2-1] = 2 \\ 1 + C[2-5] = \infty \end{cases}$$

$$C[3] = min\begin{cases} 1 + C[3-1] = 3 \\ 1 + C[3-5] = \infty \end{cases}$$

$$C[4] = min\begin{cases} 1 + C[4-1] = 4 \\ 1 + C[4-5] = \infty \end{cases}$$

$$C[5] = min\begin{cases} 1 + C[5-1] = 5 \\ 1 + C[5-5] = 1 \end{cases}$$

$$C[6] = min\begin{cases} 1 + C[6-1] = 6 \\ 1 + C[6-5] = 2 \end{cases}$$

$$C[7] = min\begin{cases} 1 + C[7-1] = 7 \\ 1 + C[7-5] = 3 \end{cases}$$

$$C[8] = min\begin{cases} 1 + C[8-1] = 8 \\ 1 + C[8-5] = 1 + C[3] = 4 \end{cases}$$

$$C[9] = min\begin{cases} 1 + C[9-1] = 9 \\ 1 + C[9-5] = 1 + C[4] = 5 \end{cases}$$

$$C[10] = min\begin{cases} 1 + C[10-1] = 10 \\ 1 + C[10-5] = 1 + C[5] = 2\,; \quad \{5,5\} \end{cases}$$

Solusi ditemukan pada tahap ke-10 yaitu pada C[10] dimana terdapat 2 koin 5 sen, 2x5 = 10.

Hasil dari tahap – tahap tersebut dapat dibuat ke dalam tabel :

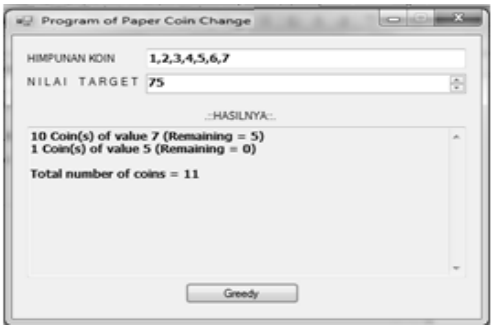
<0	0	1	2	3	4	5	6	7	8	9	10
∞	0	1	2	3	4	1	2	3	4	5	2

Tabel 1. Tabel solusi

Dapat dilihat bahwa solusi ini lebih optimal dibandingkan solusi pada algoritma *greedy* karena pada *greedy* solusinya menggunakan 6 koin sedangkan *dynamic programming* hanya menggunakan 2 koin. Namun membutuhkan waktu yang lebih lama dan perhitungan yang lebih rumit dan harus teliti.

Dalam algoritma *dynamic programming* terdapat dua perulangan tersarang, yaitu pada baris 2 dan 4. Ada *O*(V) kemungkinan berbeda di setiap state. Dan akan dicoba N tipe dari koin di setiap statenya. Kompleksitas waktu seluruhnya dari *dynamic programming* untuk persoalan ini adalah *O*(NV).

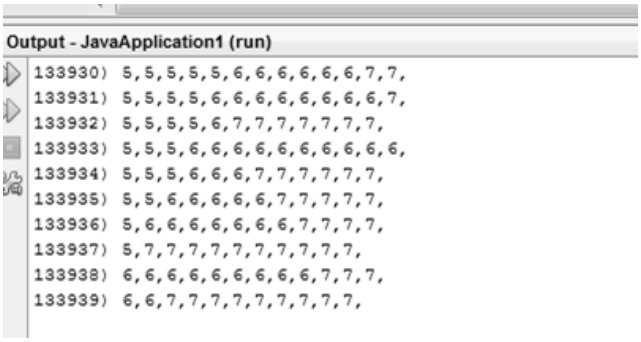
IV. PROGRAM SEDERHANA



Gambar 4.1. Penyelesaian menggunakan greedy

Dalam implementasi program untuk algoritma greedy, pada gambar 4.1 akan disimulasikan dengan memasukkan nilai target 75 dan himpunan koin yang tersedia yaitu A={1,2,3,4,5,6,7}, kemudian dengan klik tombol greedy akan ditampilkan hasil berupa solusi dengan total koin 11, dimana 10 koin 7 dan 1 koin 5, tetapi jika menggunakan dynamic programming seperti pada gambar 4.2, maka solusi yang ditemukan yaitu dua solusi optimal dengan jumlah koin yang juga sama yaitu 11, dimana :

1. 10 koin 7 dan 1 koin 5
2. 9 koin 7 dan 2 koin 6



Gambar 4.2. Penyelesaian menggunakan dynamic programming

V. KESIMPULAN

Dengan menggunakan algoritma *greedy* untuk persoalan penukaran uang kita dapat langsung menemukan solusi tanpa mempertimbangkan kemungkinan solusi yang lain yang mungkin lebih optimal. Algoritma *greedy* merupakan algoritma yang mudah diimplementasikan, dan memiliki kompleksitas yang cukup baik dalam hal ini hanya memerlukan sedikit waktu dan sedikit ruang (*space*), namun solusi yang di hasilkan tidaklah selalu optimal.

Sedangkan algoritma *dynamic Programming* menghasilkan solusi yang paling optimal, namun di tinjau dari segi implementasi ke dalam bahasa pemgraman terasa rumit. Algoritma ini juga membutuhkan waktu yang lebih lama dibandingkan algoritma greedy dan *space* yang lebih besar. Dapat dilihat bahwa untuk menyelesaikan soal yang sama algoritma *dynamic programming* membutuhkan 10 tahap pengerjaan dan pembandingan, ini membuktikan bahwa dibutuhkan ruang yang besar yang waktu yang lebih lama

Jika kita ingin menyelesaikan persoalan dengan cepat maka tidak ada salahnya menggunakan algoritma *greedy*, namun jika kita membutuhkan suatu solusi yang optimal yang bisa menghasilkan keuntungan lebih baik maka gunakan algoritma *dynamic progrramming*.

REFERENSI

[1] "*Coin Changing Revisited*", http://condor.depaul.edu/rjohnson/algorithm/coins.pdf. diakses tanggal 28 oktober 2011.

[2] Cormen Thomas H.,Leiserson Charles E., Rivest Ronald L., Stein Clifford, "*Introduction to Algorithms Second Edition*", 2001, The MIT Press, Cambridge.

[3] "*Dynamic Programming Solution to the Coin Changing Problem*", http://condor.depaul.edu/rjohnson/algorithm/coins.pdf. diakses pada tanggal 28 oktober 2011.

[4] Halim Steven,dan Halim Felix, "*Competitive Programming 2*," Hanbook or ACMI ICPC AND IOI CONTESTANTS, 2011, hal. 51-65,. (references).

[5] Karamana, "*Coin Change Problem(Using Dynamic Programming*", http://www.codeproject.com/KB/recipes/coinChangeProblem.aspx#, diakses pada tanggal 13 november 2011.

[6] MADCOMS,"*Pemograman Borland Delph 7 jilid 1*", 2003, Andi, Yogyakarta.

[7] Munir Rinaldi, "Algoritma *Greedy*", http://www.informatika.org/~rinaldi/Stmik/2011-2012/Algoritma%20Greedy.ppt. diakses tanggal 30 oktober 2011.

[8] Munir Rinaldi, "Program dinamis", http://www.informatika.org/~rinaldi/Stmik/Program%20Dinamis.ppt. diakses tanggal 30 oktober 2011.

[9] Sugianto Anggriawan, Susanto David, Muhammad Zakka Fauzan, "*Penyelesaian Berbagai Permasalahan Algoritma Dengan Kombinasi Algoritma Brute Force dan Greedy*" , http://www.informatika.org/~rinaldi/Stmik/2005-2006/Makalah2006/MakalahStmik2006-02.pdf. diakses pada tanggal 7 november 2011.

[10] "*The Coin Changing Problem*", http://ace.cs.ohiou.edu/~razvan/courses/cs404/lecture19.pdf. diakses pada tanggal 28 oktober 2011.