Penerapan Algoritme *Greedy* untuk Menyelesaikan *Knapsack Problem* pada Program *Budget Manager*

Muhammad Naufal Izza Fikry - 13519088
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519088@std.stei.itb.ac.id

***Abstract*—Manajemen keungan merupakan kegiatan yang sangat bermanfaat, namun tidak semua orang dapat melakukannya dengan baik. Di makalah ini, penulis membuat sebuah program yang mengimplementasikan algoritme *greedy* untuk menyelesaikan masalah manajemen keuangan yang direpresentasikan sebagai *Knapsack Problem*.**

Keywords—Manajemen Keuangan, Algoritme, Greedy, Knapsack Problem.

I. Pendahuluan

Manjemen keuangan merupakan kegiatan yang perlu dilakukan banyak orang. Dengan melakukan manajemen keuangan, kita dapat memaksimalkan nilai (*value*) dari pemasukan yang kita dapatkan. Sering kali pengeluaran yang kita lakukan kurang tercatat dan terstruktur, sehingga ketika ada masa dimana kita memerlukan uang dengan jumlah tertentu, kebutuhan tersebut tidak dapat / sulit terpenuhi dengan kondisi keuangan kita saat itu. Ini dapat menjadi masalah besar, tergantung kondisi yang tengah dihadapi.

Di lain sisi, seseorang – pada umumnya – tentu ingin menikmati penghasilan – dalam hal ini, uang – yang ia dapatkan. Namun demikian, pengeluaran juga harus diperhatikan agar hal di atas sebisa mungkin dapat dihindari. Untuk menyelesaikan permasalahan ini, salah satu alternatif solusi yang dapat digunakan adalah dengan memanfaatkan aplikasi manajemen keuangan untuk memudahkan proses manajemen kita.

Pada makalah ini, penulis membuat sebuah program Manajemen Keuangan bulanan berbasis web dengan memanfaatkan Algoritme Greedy. Program ini memodelkan masalah Manajemen Keuangan di atas dengan *Knapsack Problem*, dimana kapasitas *knapsack* berupa jumlah hari dalam satu bulan (30 hari) dan item *knapsack* berupa jenis pengeluaran harian yang memiliki bobot 1 (representasi 1 hari) dan *value* berupa pengeluaran pada hari itu. Program ini ditujukan untuk umum, khususnya mahasiswa, yang mungkin mengalami kesulitan dalam mengatur keuangan agar dapat menikmati value pemasukan sebanyak mungkin dengan jumlah uang yang terbatas.

II. Landasan Teori

A. Permasalahan Optimasi

Permasalahan / persoalan optimasi merupakan persoalan yang bertujuan untuk mencari solusi optimal dari suatu masalah. Permasalahan optimasi dapat dibagi menjadi 2, yakni persoalan maksimasi dan minimasi. Persoalan maksimasi adalah persoalan yang bertujuan mencari nilai maksimal dari kumpulan nilai yang diberikan dengan batasan tertentu. Sementara persoalan minimasi, sebaliknya, bertujuan mencari nilai minimal dari kumpulan nilai yang diberikan dengan batasan tertentu. Pada pembuatan program, optimasi sangat diperlukan untuk membuat suatu program yang mangkus dan dapat memaksimalkan *resource* yang ada.

B. Algoritme Greedy

Algoritme greedy merupakan salah satu algoritme yang paling popular untuk menyelesaikan permasalahan optimasi. Seperti Namanya yang berarti rakus, algoritme ini membagi permasalahan menjadi beberapa *state* yang di setiap statenya, algoritme greedy akan mengambil solusi bagian yang diannggap paling menguntungkan saat itu, dengan harapan jika hal ini terus dilakukan pada semua *state*, dapat diperoleh solusi global yang paling optimal.

Akan tetapi, algoritme ini tidak selalu menjamin akan memberikan solusi terbaik / paling optimum. Hal ini dikarenakan sifat algoritme ini yang pada suatu waktu hanya memerhatikan suatu bagian tertentu dari permasalahan dan tidak memerhatikan permasalahan tersebut secara keseluruhan. Walau demikian, ada beberapa permasalahan yang akan selalu menghasilkan solusi optimum dengan menerapkan algoritme greedy, seperti kasus permasalahan penukaran uang pada mata uang euro.

Walau demikian, kelebihan algoritme ini terdapat alur penyelesaiannya yang relatif mudah sehingga tidak memerlukan logika yang rumit saat melakukan implementasi pada program. Singkat kata, algoritme ini relatif mudah diimplementasikan.

Algoritme *greedy* memiliki beberapa komponen, diantaranya:

1. Himpunan Kandidat (C): berisi kandidat yang akan dipilih pada setiap langkah / *state*

2. Himpunan Solusi (S): berisi kandidat yang sudah dipilih
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi Seleksi: memilih kandidat berdasarkan strategi *greedy* tertentu yang bersifat heuristik.
5. Fungsi Kelayakan: memeriksa apakah kandidat pada himpunan kandidat layak untuk masuk himpunan solusi atau tidak
6. Fungsi Objektif: berisi tujuan, yakni untuk memaksimumkan atau meminimumkan solusi.

Berikut merupakan skema umum dari algoritme *greedy*:

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
  x : kandidat
  S : himpunan_solusi

Algoritma:
  S ← {}      { inisialisasi S dengan kosong }
  while (not SOLUSI(S)) and (C ≠ {} ) do
      x ← SELEKSI(C)     { pilih sebuah kandidat dari C}
      C ← C – {x}        { buang x dari C karena sudah dipilih }
      if LAYAK(S ∪ {x}) then      { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
          S ← S ∪ {x}     { masukkan x ke dalam himpunan solusi }
      endif
  endwhile
  {SOLUSI(S) or C = {} }

  if SOLUSI(S) then    { solusi sudah lengkap }
    return S
  else
    write('tidak ada solusi')
  endif
```

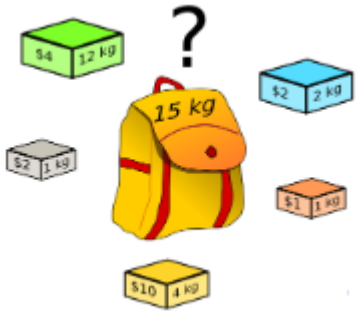
Gambar 1. skema umum algoritme *greedy*

(*sumber: slide kuliah IF2211 2021 oleh Dr. Rinaldi Munir*)

C. *Knapsack Problem*

Knapsack Problem adalah suatu permasalahan dimana kita diberi suatu *Knapsack* atau tas dengan kapasitas tertentu dan sekumpulan item yang memiliki kapasitas dan *value* masing-masing. Tujuan penyelesaian masalah ini adalah untuk menentukan item-item yang akan dimasukkan ke dalam *knapsack* agar diperoleh keuntungan yang maksimum. dapat dilihat bahwa permsalahan ini merupakan permasalahan maksimasi nilai.

Berikut merupakan ilustrasi dari *knapsack problem*:



Gambar 2. ilustrasi *knapsack problem*

(*sumber: slide kuliah IF2211 2021 oleh Dr. Rinaldi Munir*)

D. *Penyelesaian Knapsack Problem mengunakan Algoritme Greedy*

Seperti yang telah disebutkan sebelumnya, Algoritme *Greedy* dapat digunakan untuk menyelesaikan permasalahan optimasi. telah disebutkan bahwa permasalahan *knapsack* merupakan permasalahan optimasi, lebih tepatnya, optimasi maksimum. Itu artinya, algoritme *greedy* dapat dijadikan salah satu solusi untuk menyelesaikan permasalahan ini.

Algoritme *Greedy* membagi proses penyelesaian menjadi beberapa *state*, atau langkah, yang pada setiap *state*-nya akan diambil solusi optimum yang *feasible* untuk dimasukkan ke himpunan solusi.

Pada kasus ini, misalkan terdapat *knapsack* dengan kapasitas 15 kg dan beberapa barang seperti yang diilustrasikan pada Gambar 2. Mula mula, algoritme greedy akan mengambil barang dengan *value* paling tinggi. kemudian melakukan pengecekan apakah kapasitas barang tersebut melebihi sisa kapasitas knapsack. Apabila *feasible*, maka barang tersebut akan dimasukkan ke dalam knapsack (masuk himpunan solusi) dan tidak akan masuk apabila tidak *feasible*. Pengecekan dilanjutkan dengan barang selanjutnya yang memiliki *value* paling tinggi, selama masih ada sisa kapasitas pada knapsack.

E. *Manajemen Keuangan*

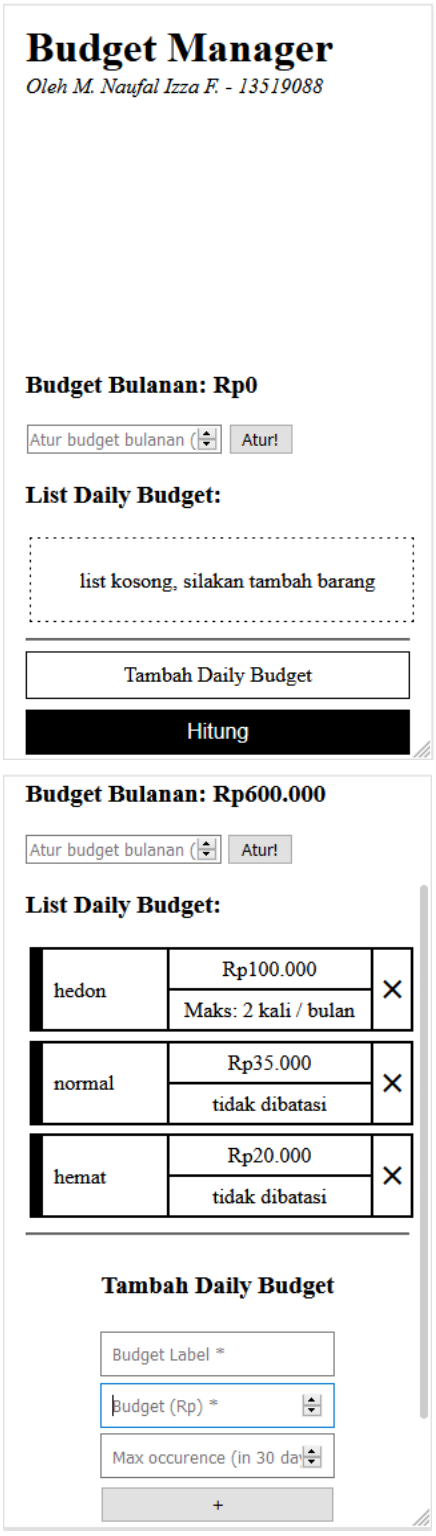
Manajemen keuangan adalah proses penganggaran, tabungan, investasi, pengeluaran, atau pengawasan penggunaan uang. Pada makalah ini, program yang dibuat penulis dapat menyelesaikan masalah manajemen keuangan penggunannya selama berada pada lingkup yang diliputi program dan program berjalan dengan semestinya.

III. LANGKAH-LANGKAH PENYELESAIAN

Pada saat tulisan ini dibuat, program telah dimuat pada *github page* penulis pada pranala berikut https://naufalizza.github.io/budget-manager/ sehingga beberapa tangkapan layar pada makalah ini akan diambil dari laman tersebut. Perlu penulis sampaikan bahwa di kemudian hari, pranala tersebut mungkin memuat hasil yang berbeda dari yang ditampilkan pada makalah ini. Oleh karena itu, harap dimaklumi apabila ada perbedaan konten pada pranala tersebut di kemudian hari.

A. *Tampilan Program dan Komponen Utamanya*

Berikut adalah tampilan program saat tulisan ini dibuat



Gambar 3. Tampilan layar utama program

Dapat dilihat pada gambar, beberapa komponen utama pada program ini adalah sebagai berikut:

1. budget bulanan
2. list of daily budget (budget harian)
3. budget harian yang berisi:
    a. label
    b. nominal budget
    c. kemunculan maksimal, dengan kasus khusus 0 yang berarti jumlah kemunculan tidak secara langsung dibatasi oleh pengguna

Pada program, komponen 1 dan 2 direpresentasikan oleh kedua variabel berikut, yakni monthly_budget dan items.

```
25  let monthly_budget = 0;
26  let items = [];
```

Gambar 4. dua varibel utama pendefinisi *state* program

Sementara komponen 3 merupakan objek dari *class* Card yang strukturnya sebagai berikut:

```
class Card{
    constructor(id, name, weight, profit){
        this.id = id;
        this.name = name;
        this.weight = weight;
        this.profit = profit;
        this.get_html_tag = function() {
            const formatted_money = Number(this.profit).toLocaleString("id-ID");
            let max_occurence = "";
            if (this.weight<=0){
                max_occurence = "tidak dibatasi";
            }
            else {
                max_occurence = "Maks: " + String(this.weight) + " kali / bulan";
            }
            const tag = `
            <div class="item_card">
                <div class="item_card_name">${this.name}</div>
                <div class="item_card_info">
                    <div class="item_card_profit">Rp${formatted_money}</div>
                    <div class="item_card_weight">${max_occurence}</div>
                </div>
                <input id="${this.id}" type="checkbox" class="discard_checkbox" style="display: none;">
                <label for="${this.id}" class="item_card_discard">x</label>
            </div>`;
            return tag;
        }
    }
}
```

Gambar 5. *Class* Card

B. *Langkah Penyelesaian*

Budget bulanan dan list of daily budget mula-mula di-*set* menjadi 0 dan *array* kosong pada program. Kemudian, program mulai membaca berbagai input dari pengguna, seperti saat pengguna mengatur budget bulanan, hingga menambah budget harian baru.

Setiap kali input dilakukan, program mengupdate variable yang bersangkutan dari kedua variable di atas, kemudian mengupdate tampilan dari program.

Saat menekan tombol hitung, program akan memanggil fungsi process() yang merupakan *entry point* dari fungsi utama program. Fungsi ini berisi penerapan algoritme *greedy* yang digunakan untuk menyelesaikan knapsack problem.

```
 40  function process(){…
128  }
```

Gambar 6. Fungsi process()

Fungsi process() dibagi menjadi 5 tahap, yakni:

1. Mengurutkan array ‘items’ berdasarkan budget masing-masing itemnya

2. Memasukkan daily budget ke knapsack dengan strategi *greedy*
3. Melakukan cek apakah budget yang dimiliki dan budget harian tersebut, semua hari memiliki alokasi dana.
4. menampilkan hasil ke layar

Berikut merupakan rincian untuk masing masing Langkah:

a) Langkah 1: Mengurutkan array 'items'

```
    // Step 1: sort items array by budget
  items.sort(function(a,b){
      return (b.profit - a.profit);
  });
```

Gambar 7. Langkah pertama penyelesaian

pada langkah ini, program memanfaatkan *method* sort() untuk melakukan sortir objek pada array items yang diurutkan berdasarkan budget, atau profit pada program, dari objek-objeknya.

b) Langkah 2: Memasukkan daily budget ke knapsack

```
// Step 2: 'memasukkan' daily_budgets ke knapsack
dengan strategi greedy
let budget_dif = 0;
let this_max_oc = 0
for (var i=0; i<items.length; i++){
    this_max_oc = 0;
    budget_dif = items[i].profit - cheapest_budget;
    if (budget_dif == 0){
        selected_items.push({obj: items[items.
        length-1], occurence: cheapest_occurence});
        knapsack_capacity -= cheapest_occurence;
        break;
    }
    this_max_oc = Math.floor(remaining_budget/
    budget_dif);
    if (items[i].weight !== 0){
        if (this_max_oc > items[i].weight){
            this_max_oc = items[i].weight;
        }
    }
    if (this_max_oc > knapsack_capacity){
        this_max_oc = knapsack_capacity;
    }
    if (this_max_oc > 0){
        selected_items.push({obj: items[i],
        occurence: this_max_oc})
        remaining_budget -= this_max_oc*(budget_dif);
        knapsack_capacity -= this_max_oc;
        cheapest_occurence -= this_max_oc;
    }
}
```

Gambar 8. Langkah kedua penyelesaian

Pada langkah ini, program mula-mula mengisi knapsack (semua hari dalam 1 bulan) dengan budget harian yang nilainya paling rendah, dengan asumsi budget paling rendah selalu tidak memiliki batas maksimal kemunculan.

Setelah itu, program mulai menerapkan strategi greedy dari budget harian yang nominalnya paling tinggi. max occurrence awalnya di-*set* maksimal. Jika melebihi batas kemunculan, ganti dengan batas kemunculan budget harian tersebut. Kemudian, jika melebihi sisa kapasitas knapsack, yakni jumlah hari dalam 1 bulan yang belum dialokasi budget harian, ganti dengan sisa kapasitas tersebut. Lalu tambahkan *item* tersebut dengan jumlah sebanyak max occurrence dan keluarkan item yang digunakan untuk memopulasi knapsack pada awal langkah ini sebanyak max occurrence juga.

Proses ini terus dilakukan hingga semua budget harian telah dicek atau ditemui budget harian yang memiliki budget sama dengan budget harian dengan budget terendah yang digunakan untuk memopulasi knapsack di awal langkah ini.

Budget-budget yang dipilih untuk masuk ke knapsack, dan budget minimal pada program ini, disimpan pada suatu list baru yang bernama 'selected_items'. list ini nantinya digunakan pada saat proses penampilan hasil ke layar.

c) Langkah 3: Melakukan cek budget

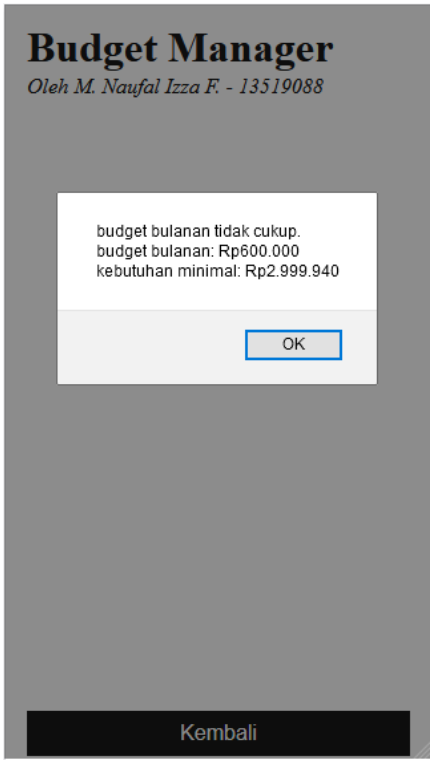
```
// Step 3: cek apakah knapsack_capacity masih > 0
//          dengan kata lain, akan ada hari yang
belum mendapat alokasi budget harian
if (knapsack_capacity > 0){
    alert("Alokasi budget < 30 hari, pastikan budget
    harian sudah di-set dengan benar.");
}
```

Gambar 9. Langkah ketiga penyelesaian

Program melakukan cek pada sisa kapasitas knapsack.

Apabila belum penuh (masih lebih dari 0), program akan memberikan alert berisi pesan bahwa alokasi budget masih kurang dari 30 hari.

Ini merepresentasikan budget bulanan yang lebih sedikit nilainya dari batas minimal budget harian yang digunakan selama 30 hari.



Gambar 10. Kasus budget bulanan tidak cukup

d) Langkah 4: menampilkan hasil ke layar

```
// Step 4: menampilkan hasil ke layar
let total = 0;
let result_tags = "";
selected_items.forEach(item => {
    total += item.obj.profit * item.occurence;
    result_tags += `
    <div class="result_item">
        <p><strong>${item.obj.name}: ${item.occurence}
        </strong> kali</p>
    </div>
    `
});
let pre_text = `
        <h3>Hasil Perhitungan:</h3>
        <p>Budget: Rp${Number(monthly_budget).
        toLocaleString("id-ID")}</p>
        <p>Digunakan: Rp${Number(total).toLocaleString
        ("id-ID")}</p>
        <p>Sisa: Rp${Number(monthly_budget-total).
        toLocaleString("id-ID")}</p>
        <p><strong>Rincian:</strong></p>
`;
result_tags = pre_text.concat(result_tags);
result_content.innerHTML = result_tags;
```

Gambar 11. Langkah keempat penyelesaian

Proses ini memanfaatkan beberapa data, yakni selected_items dan monthly_budget.

Program kemudian membuat suatu tag HTML yang berisi budget bulanan, budget yang digunakan, sisa budget, dan detail komponen dari list selected_items yang berisi nama dan kemunculan.

berikut adalah contoh kasusnya

Gambar 12. Salah satu kasus yang memberikan solusi

Kasus di atas berarti, dengan budget Rp700.000 per bulan, kita bisa belanja ‘hedon’ 1 kali, ‘normal’ 1 kali, dan belanja ‘hemat’ 28 kali dengan rincian:

hedon: Rp100.000 per hari

normal: Rp35.000 per hari

hemat: Rp20.000 per hari

serta masih menyisakan Rp5.000.

IV. KESIMPULAN

Penggunaan Algoritme *Greedy* untuk menyelesaikan permasalahan *Knapsack* atau *Knapsack Problem* dapat diimplementasikan pada program Manajemen Keuangan dan dapat memberi hasil yang cukup baik.

Program ini didesain untuk dapat digunakan oleh umum, namun lebih dikhususkan untuk digunakan mahasiswa untuk mengatur *flow* keuangan bulanan agar dapat memaksimalkan pendapatan bulanan yang mungkin terbatas.

Penulis merasa cukup puas dengan hasil program yang memanfaatkan algoritme greedy yang menurut hemat penulis, dapat menyelesaikan sebagian besar, bila tidak semua, permasalahan yang berkaitan dengan manajemen keuangan yang didefinisikan seperti pada program

PRANALA VIDEO DEMO (YOUTUBE)

https://youtu.be/fr4y_zmXSNU

UCAPAN TERIMAKASIH

Penulis mengucapkan terimakasih kepada Allah S. W. T. yang karena izin-Nya, penulis dapat menyelesaikan makalah ini. Kemudian, terimakasih juga penulis ucapkan bapak-ibu dosen, khususnya Dr. Ir. Rinaldi Munir yang telah menyediakan referensi dan Dr. Nur Ulfa Maulidevi selaku dosen pengajar untuk matakuliah IF2211 yang diambil penulis semester ini. Terakhir, penulis juga mengucapkan terimakasih kepada semua pihak yang baik secara langsung maupun tidak langsung telah membantu dan/atau memberi inspirasi penulis untuk menyelesaikan program dan makalah ini.

REFERENSI

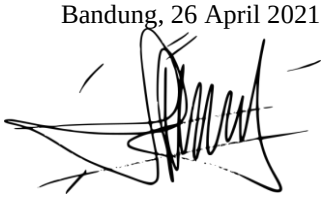
[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

[2] https://www.investopedia.com/terms/m/moneymanagement.asp diakses pada tanggal 11 Mei 2021 pukul 22:20

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, 26 April 2021



Muhammad Naufal Izza Fikry, 13519088