Penerapan Algoritma Greedy dalam Pengambilan Keputusan dengan Eisenhower Decision Matrix

Muhammad Akram Al Bari - 13519142
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519142@std.stei.itb.ac.id

***Abstrak*—*Eisenhower Decision Matrix*, atau dikenal juga sebagai *Urgent-Important Matrix* adalah salah satu metode yang dapat diterapkan untuk membantu kita dalam melakukan *time management* terhadap aktivitas-aktivitas yang kita miliki. *Eisenhower Matrix* membagi aktivitas kita ke dalam empat kategori, yaitu: *Urgent-Important*, *Urgent-Not Important, Not Urgent-Important, Not Urgent - Not Important*. Algoritma Greedy dapat membantu kita untuk memutuskan kategori aktivitas mana saja yang masih memungkinkan untuk kita lakukan dengan mengacu pada *Eisenhower Matrix*. Secara konsep, persoalan menentukan pilihan dengan *Eisenhower Matrix* ini adalah bentuk aplikasi langsung dari persoalan *0-1 Integer Knapsack Problem*.**

Kata Kunci*—*algoritma; greedy; eisenhower; decision; matrix; integer; knapsack; problem

I. Pendahuluan

Matriks Eisenhower adalah salah satu alat bantu yang digunakan oleh banyak orang untuk mengatur waktu yang mereka miliki agar dapat melakukan aktivitas-aktivitas yang produktif dan memang bermanfaat. Metode Matriks Eisenhower ini juga biasa dikenal sebagai Matriks *Urgent-Important* diakrenakan dalam penggunaannya, matriks ini membagi aktivitas-aktivitas ke dalam empat kuadran berdasarkan kriteria *Urgent* dan *Important*.

Idealnya, dengan menggunakan matriks ini kita dapat dengan mudah memetakan aktivitas-aktivitas yang seharusnya kita lakukan, dan aktivitas mana saja yang sebaiknya kita tinggalkan. Namun, dalam realisasinya, tentunya banyak faktor X yang membuat pengaplikasian alat bantu ini tidak semudah yang diharapkan pada awalnya. Salah satunya adalah faktor waktu.

Tak jarang, suatu aktivitas yang memang *Urgent* dan *Important* ternyata mengharuskan kita untuk mengeluarkan lebih banyak waktu dari yang sebelumnya sudah kita rencanakan. Tentunya, hal ini akan membuat kita perlu untuk menyesuaikan kembali alokasi waktu terhadap aktivitas-aktivitas lain yang telah kita rencanakan. Berdasarkan hemat penulis, banyak pihak yang kadangkala merasa kesulitan untuk menata ulang aktivitas-aktivitas mereka ketika situasi seperti ini terjadi. Oleh karena itu, terlintas pikiran dalam benak penulis bahwa situasi-situasi yang demikian ini dapat diatasi dengan bantuan algoritma.

Algoritma Greedy adalah salah satu algoritma yang sering digunakan dalam kasus optimasi. Algoritma Greedy merupakan algoritma yang cukup sederhana dan mudah untuk diimplementasikan. Secara sekilas, cara kerja Algoritma Greedy adalah dengan mencari nilai "optimum lokal" pada setiap kesempatan yang ada, dengan harapan nantinya nilai-nilai "optimum lokal" ini dapat mengantarkan kepada hasil optimum yang sesungguhnya di kasus yang sedang ditangani.

Untuk itu, pada makalah ini akan coba untuk diulas terkait pemanfaatan Algoritma Greedy guna membantu menemukan solusi yang optimal terhadap persoalan yang berkaitan dengan pemilihan aktivitas pada Matriks Eisenhower. Secara sederhana, persoalan ini merupakan variasi daribentuk persoalan *0-1 Integer Knapsack Problem*.

II. Landasan Teori

A. Algoritma Greedy

Algoritma Greedy adalah segala macam algoritma yang mengikuti cara penyelesaian masalah heuristik dengan cara membuat pillihan yang "optimum lokal" pada setiap kemungkinan yang ada. Pada kebanyakan persoalan, Algoritma Greedy biasanya tidak menghasilkan solusi akhir yang optimal, namun meskipun begitu, Algoritma Greedy dapat menghasilkan solusi "optimum lokal" yang dapat mengaproksimasi solusi optimum sesungguhnya dari suatu persoalan dengan waktu komputasi yang baik.

Sebagai contoh, Algoritma Greedy pada persoalan *travelling salesman problem*, yang mana persoalan ini terkenal sebagai persoalan yang sangat kompleks secara komputasi, adalah dengan mengikuti heuristik sebagai berikut: "Pada setiap langkah dalam perjalanan, kunjungi kota terdekat yang belum pernah dikunjungi". Herustik ini tidak membawa kita untuk menemukan solusi yang terbaik, namun pendekatan ini akan membuat persoalan yang sedang dijalankan dapat berakhir dalam waktu atau langkah yang cukup cepat.

Dalam optimasi matematis, Algoritma

Greedy dapat dengan optimal menyelesaikan persoalan kombinatorial yang memiiki properti tertentu dan juga dapat memberikan faktor konstant terhadap aproksimasi suatu persoalan optimasi yang memiliki struktur tertentu.

Secara umum, Algoritma Greedy memiliki lima buah elemen sebagai berikut:

1. Himpunan kandidat (C). Merupakan himpunan yang berisi kandidat yang akan dipilih pada setiap langkah (misal: koin, grafik, pekerjaan, benda, dll.).
2. Himpunan solusi (S). Merupakan himpunan kandidat yang sudah dipilih (kumpulan solusi "optimum lokal").
3. Fungsi solusi. Merupakan fungsi yang menentukan apakah himpunan kandidat yang telah dipilih sudah membentuk solusi akhir yang diinginkan atau belum.
4. Fungsi seleksi. Fungsi yang digunakan untuk memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini seperti yang sudah dijelaskan diawal tadi, adalah strategi yang bersifat heuristik.
5. Fungsi kelayakan. Digunakan untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi atau tidak.
6. Fungsi obyektif. Berguna untuk memaksimumkan atau meminimumkan nilai-nilai yang tengah ditelusuri.

```
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 Algoritma Greedy

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

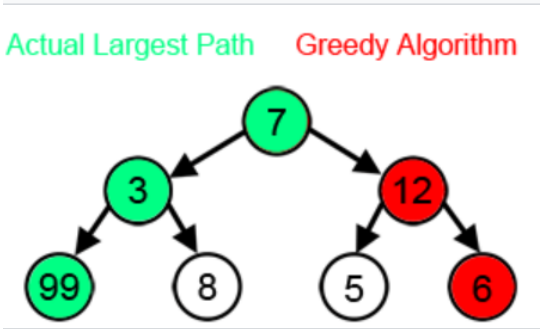
Algoritma Greedy menghasilkan solusi yang baik untuk sebagian persoalan matematis, tapi tidak untuk sebagian yang lain. Sebagian besar persoalan yang Algoritma Greedy dapat bekerja dengan baik adalah apabila persoalan tersebut memiliki dua hal sebagai berikut:

- *Greedy Choice Property*

Kita dapat memilih apapun pilihan yang terlihat terbaik pada suatu titik tertentu dan kemudian menyelesaikan subpersoalan yang muncul kemudian. Pilihan yang dibuat oleh algoritma greedy dapat bergantung kepada pilihan pilihan yang telah dipilih sejauh ini, tapi tidak terhadap pilihan pilihan yang dipilih kemudian atau semua solusi dari subpersoalan yang ada. Greedy secara iteratif membuat sebuah pilihan greedy secara suksesif, mengurangi persoalan yang ada menjadi persoalan yang lebih kecil. Atau dengan kata lain, algoritma greedy tidak pernah melakukan konsiderasi ulang terhadap pilihan yang teah diambilnya. Ini adalah perbedaan utama algoritma greedy dengan *dynamic programming*, di mana ia melakukan pencarian secara exhaustive dan dijamin akan menemukan solusi. Setelah setiap tahapan, *dynamic programming* memiliki pilihan berdasarkan seluruh pilihan yang telah dibuat pada tahap sebelumnya, dan juga melakukan konsiderasi ulang terhadap jalur algoritma pada tahap sebelumnya untuk sampai ke solusi yang optimal.

- *Optimal substructure*

Suatu persoalan dikatakan memiliki *optimal substructure* apabila suatu solusi optimal dari persoalan tersebut mengandung solusi-solusi optimal lain dari subpersoalannya.



Gambar 2. Persoalan Greedy yang "gagal"

Sumber:https://www.ics.uci.edu/~majumder/vispercep/ICIAP.pdf

Untuk banyak persoalan, algoritma greedy tidak dapat memberikan solusi optimal, dan bahkan dapat menghasilkan solusi unik yang justru buruk. Meskipun begitu, algoritma ini tetap baik digunakan untuk memberikan aproksimasi terhadap banyak persooalan yang ada, terutama persoalan-persoalan yang apabila ingin didapatkan hasil optimalnya memerlukan waktu komputasi yang sangat kompleks dan panjang.

B. Heuristik

Dalam konteks optimasi matematis dan juga ilmu komputer, herustik adalah sebuah teknik yang didesain untuk menyelesaikan suatu persoalan lebih cepat dibandingkan dengan metode klasik yang terlalu lambat, atau digunakan juga untuk menemukan aproksimasi terhadap suatu solusi yang metode klasik gagal menemukan solusi eksaknya. Hal ini didapatkan dengan cara mengorbankan *optimality*, *completeness, accuracy,* atau *precision* untuk mendapatkan waktu komputasi yang lebih baik. Dengan kata lain, heuristik dapat kita artikan sebagai *jalan pintas* untuk mendapatkan solusi/mengaproksimasi suatu solusi persoalan.

Fungsi heuristik, atau lebih sering disebut

sebagai heuristik, adalah sebuah fungsi yang melakukan proses pengurutan alternatif pada suatu algoritma pencarian pada setiap langkah yang terjadi, berdasarkan pada informasi yang tersedia untuk kemudian memilih lankah selanjutnya yang akan dijalankan. Pendekatan seperti ini dapat memberikan solusi optimal, atau setidaknya dapat melakukan aproksimasi terhadap solusi optimal yang sesungguhnya.

Tujuan dari heuristik adalah menghasilkan solusi yang cukup baik dalam kurun waktu yang juga baik untuk menyelesaikan suatu persoalan yang dimiliki. Solusi ini mungkin saja bukan merupakan solusi terbaik dari sekian banyak solusi-solusi yang dimiliki oleh suatu persoalan, atau bisa saja solusi yang didapatkan merupakan aproksimasi atau hampiran dari solusi terbaik. Meskipun begitu, pendekatan ini tetap berharga karena untuk mendapatkan solusi dari heuristik tidak membutuhkan waktu yang lama.

Heuristik dapat menghasilkan solusi secara independen maupun ketika digunakan bersamaan dengan algoritma optimasi yang lain untuk meningkatkan efisiensi. Heuristik banyak digunakan sebagai dasar dari dunia *Artificial Intelligence* (AI) karena heuristik dapat digunakan di situasi ketika tidak terdapat algoritma lain yang dapat menyelesaikan suatu persoalan tersebut.

Ada beberapa hal yang perlu dipertimbangkan ketika kita akan menggunakan heuristik untuk menyelesaikan suatu persoalan. Hal-hal tersebut ialah:

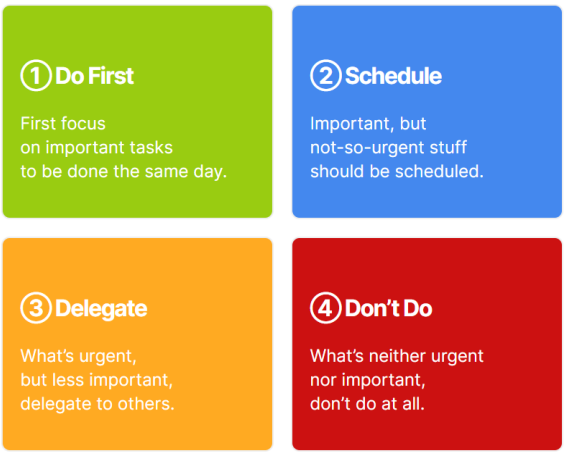
- *Optimality*. Ketika terdapat beberapa solusi terhadap suatu persoalan, kita perlu mempertimbangkan apakah heuristik akan menjamin terpilihnya solusi terbaik. Serta apakah memang diperlukan untuk menemukan solusi terbaik tersebut.
- *Completeness*. Ketika beberapa solusi terdapat pada suatu persoalan, kita perlu mempertimbangkan apakah heuristik mampu memberikan kita seluruh kemungkinan solusi tersebut. Dan apakah kita juga memang membutuhkan seluruh solusi yang ada. Hal ini dikarenakan biasanya heuristik diperuntukan untuk menemukan satu buah solusi saja.
- *Accuracy & Precision*. Kita perlu mempertimbangkan juga apakah ketika heuristik digunakan untuk menemukan suatu hampiran, apakah hampiran tersebut memiliki rentang error yang masih dapat ditoleransi dan diterima atau tidak.
- *Execution Time*. Kadangkala terdapat beberapa heuristik yang dapat digunakan untuk menyelesaikan suatu persoalan. Kita perlu memilih, apakah heuristik yang kita gunakan sudah memberikan kemungkinan penyelesaian yang terbaik, atau apakah masih terdapat heuristik lain yang lebih baik. Beberapa heuristik memiliki selisih kecepatan komputasi yang sedikit dibandingkan dengan metode klasik. Apabila hal ini yang terjadi, mmaka bisa jadi heuristik malah akan memberikan dampak negatif.

C. Matriks Eisenhower

Matriks Eisenhower atau biasa dikenal juga sebagai Matriks *Urgent-Important*, adalah suatu metode yang dapat digunakan untuk membantu dalam menentukan prioritas aktivitas yang didasarkan pada tingkat urgensi serta kepentingannya. Dengan matriks ini, kita akan melakukan pengurutan aktivitas, mulai dari yang tidak terlalu urgen dan penting, aktivitas yang lebih baik didelegasikan atau bahkan tidak dilakukan sama sekali, hingga aktivitas yang memang urgent dan penting sehingga harus segera diprioritaskan untuk dikerjakan.

Dwight D. Eisenhower adalah presiden ke-34 Amerika Serikat. Beliau telah banyak mengambil keputusan-keputusan yang berat dalam kurun waktu yang berkelanjutan, yang mana beliau dasarkan pilihan tersebut kepada apa yang seharusnya menjadi fokus pada tiap harinya. Inilah yang akhirnya menjadi dasar dari *Eisenhower Principle*, yang merupakan dasar dari matriks ini.

Melakukan urutan prioritas aktivitas berdasarkan urgensi dan kepentingannya akan menghasilkan 4 buah kuadran dengan label yang berbeda-beda.



Gambar 3. Kuadran Eisenhower
sumber: https://www.gloveworx.com/blog/using-eisenhower-decision-matrix/

Kuadran yang pertama biasa dikenal sebagai kuadran *Do First*, di mana aktivitas-aktivitas pada kuadran ini adalah aktivitas yang memang penting bagi kehidupan dan bahkan mungkin karir kita, dan memang perlu dikerjakan hari ini atau dalam waktu dekat. Kita perlu memberikan prioritas tertinggi terhadap aktivitas-aktivitas yang terdapat pada kuadran ini.

Kuadran yang kedua adalah *Schedule*. Kuadran ini berisi aktivitas-aktivitas yang penting namun belum ada di titik urgent, sehingga masih bisa

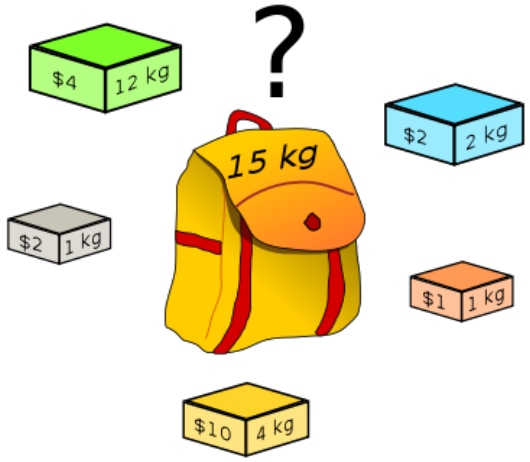
dikerjakan di hari-hari yang akan datang. Aktivitas yang masuk pada kuadran ini baiknya kita rencanakan untuk dikerjakan sejak jauh-jauh hari.

Kuadran selanjutnya adalah *Delegate*. Pada kuadran ini, berisi aktivitas-aktivitas yang sifatnya urgent, harus segera dikerjakan, namun ia tidak terlalu penting atau berdampak bagi diri kita pribadi. Sehingga aktivitas-aktivitas pada kuadran ini baiknya kita delegasikan atau kita pindahtangankan untuk dikerjakan oleh orang lain dan kita bisa fokus kepada aktivitas pada kuadran pertama.

Pada makalah ini, kasus yang ditinjau adalah kasus realisasi dari perencanaan menggunakan Eisenhower Matriks yang terkendala oleh faktor waktu yang tiba-tiba berubah secara dinamis. Oleh karena itu, persoalan yang dibahas pada makalah ini sangat dekat dengan persoalan *0-1 Integer Knapsack Problem*.

D. *0-1 Integer Knapsack Problem*

Persoalan Knapsack adalah persoalan optimasi kombinatorial. Secara sederhana, persoalan ini adalah persoalan yang apabila kita diberikan sekumpulan barang, masing-masing dengan *berat* dan *nilai*nya, maka tentukan nilai tiap-tiap barang yang bisa diambil sehingga sekumpulan barang yang diambil tersebut memiliki total *berat* kurang dari atau sama dengan batas berat yang dapat diambil dan memiliki total *nilai* yang terbesar yang mungkin.



Gambar 4. Ilustrasi Persoalan Knapsack
sumber:https://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/Knapsack.svg/1200px-Knapsack.svg.png

Persoalan Knapsack muncul diberbagai aspek kehidupan yang berkaitan dengan pengambilan keputusan. Salah satu persoalan yang paling umum untuk diselesaikan berkaitan dengan Knapsack ialah *0-1 Knapsack Problem*, di mana setiap barang hanya memiliki kemungkinan nilai 0 (tidak diambil) atau 1 (diambil).

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0,1\}.$$

Gambar 5. Formulasi Persoalan 0-1 Knapsack
sumber:https://media.geeksforgeeks.org/wp-content/cdn-uploads/knapsack-problem.png

Beberapa algoritma sering digunakan untuk menyelesaikan persoalan Knapsack. Diantara yang cukup sering digunakan ialah algoritma *Branch and Bound*. Namun pada makalah kali ini, pendekatan yang diambil adalah dengan menggunakan Algoritma Greedy.

III. PEMBAHASAN

Pada bagian ini, akan dijelaskan mengenai tahapan-tahapan implementasi Algoritma Greedy untuk menyelesaikan persoalan Matriks Eisenhower. Dalam implementasi ini, *constraint* persoalannya adalah waktu, di mana waktu pada konteks ini merupakan *weight* atau batas maksimum yang dapat ditoleransi oleh Algoritma Greedy

A. Algoritma

Pada persoalan kali ini, tahapan implementasi akan dibagi menjadi:

1. Setiap aktivitas akan memiliki nilai (w_i) dengan i melambangkan nomor aktivitas dan w adalah waktu yang dibutuhkan untuk menuntaskan aktivitas tersebut.
2. Setiap aktivitas akan memiliki nilai (v_i) dengan i melambangkan nomor aktivitas, dan v adalah prioritas dari aktivitas tersebut. Untuk menyederhanakan persoalan, nilai v hanya akan terdefinisi dari rentang [1,4]. Dengan niai 1 artinya aktivitas tersebut urgent dan penting, 2 artinya ia urgent tapi tidak penting, 3 artinya tidak urgent tapi penting, dan 4 artinya tidak urgent dan tidak penting.

Kemudian, akan diberikan w di mana w ini adalah batas waktu maksimum yang dimiliki oleh seseorang yang sedang menghadapi persoalan saat ini. Nantinya, total akumulasi dari nilai w_i haruslah kurang dari atau sama dengan nilai w. Secara langkah penyelesaian, kurang lebih sebagai berikut:

1. Buatlah daftar aktivitas dengan nilai w_i yang beragam, serta nilai v_i yang terdefinisi pada rentang [1,4].
2. Definisikan pula nilai w, yang akan menjadi ambang batas maksimal banyak waktu kegiatan yang dapat diterima. w pada aplikasinya di kehidupan sehari-hari bisa kita artikan sebagai sisa waktu yang kita miliki untuk melaksanakan aktivitas-aktivitas yang lain.
3. Lakukan proses looping sebanyak i_{max} , dengan terlebih dahulu memasukan aktivitas-aktivitas yang memiliki nilai v_i sama dengan 1. Algoritma greedy yang digunakan adalah dengan memasukan nilai v_i secara berurutan mulai dari 1 hingga maksimum 4, dengan memberikan prioritas kepada

nilai w_i yang lebih minimum untuk setiap aktivitas dengan nilai v_i yang sama.

4. Algoritma akan terus dijalankan sampai seluruh total penjumlahan w_i maksimal sama dengan nilai w.

5. Karena memanfaatkan Matriks Eisenhower, maka solusi yang diterima dapat dibagi ke dalam beberapa kasus berikut:

.a. Seminimal-minimalnya solusi diterima jika seluruh aktivitas dengan nilai v_i sama dengan 1 berhasil diambil

b. Sama dengan kasus poin a, namun nilai penjumlah w_i sampai dengan seluruh elemen $v_i$dengan nilai v_i sama dengan 1 habis, namun masih di bawah nilai w. Maka lanjutkan untuk memenuhi sampai dengan pindah ke aktivitas dengan nilai v yang lebih besar satu.

c. jika nilai w_i sudah sama dengan atau lebih dari w sebelum semua aktivitas dengan nilai v_i sama dengan 1 habis, maka tidak ada solusi yang ditemukan.

B. Studi Kasus

Misalnya diberikan daftar aktivitas dalam bentuk tabel sebagai berikut:

No.	Aktivitas	W(jam)	V
1	Aktivitas 1	1	1
2	Aktivitas 2	1	1
3	Aktivitas 3	0.5	3
4	Aktivitas 4	0.25	2
5	Aktivitas 5	1	4
6	Aktivitas 6	2	2
7	Aktivitas 7	0.25	1
8	Aktivitas 8	0.25	1
9	Aktivitas 9	1	2

Tabel 1. Daftar Aktivitas Kasus 1

Algoritma Greedy yang digunakan memiliki *source code* sebagai berikut:

```
def fungsiSeleksi(listOfActivites, v):
    result = listOfActivites[0]
    for activity in listOfActivites:
        result = min(result["v"],activity["v"])
    return result

def greedyEisenhowerMatriks(listOfActivities, wMaks):
    result = []
    currW = 0
    stop = False
    for i in range(1,4):
        for activiy in listOfActivies:
            if (currW+activity["w"]) > wMaks:
                stop = True
                break
            result.append(fungsiSeleksi(listOfActivites,i))
            currW += activity["w"]
        return result
    if stop:
        break
    return result
```

Gambar 6. Source Code Algoritma Greedy

sumber:dokumen penulis

Dengan menggunakan aktivitas pada tabel 1 sebagai aktivitas acuan, dan nilai w yang digunakan adalah 5, maka akan didapatkan hasil output sebagai berikut:

```
Waktu kamu yang tersisa adalah 5 jam
Kamu dapat mengerjakan aktivitas yang Urgent dan Penting: Aktivitas 7, 8, 1, dan 2
Kamu juga masih memiliki sisa waktu untuk mengerjakan aktivitas yang Penting: Aktivitas 9
Kamu masih memiliki sisa waktu 1.5 jam!
>>>
```

Gambar 7. Contoh hasil kasus

sumber: dokumen penulis

Berdasarkan uji kasus di atas, didapatkan bahwa algoritma mampu untuk menjalankan aktivitas-aktivitas sesuai ketentuan, yakni seluruh aktivitas yang memiliki nilai v_i sama dengan 1, baru kemudian lanjut ke aktivitas dengan nilai v_i yang lain. Kasus yang didapatkan di atas adalah kasus poin b, yakni ketika masih didapatkan sisa waktu dan bisa digunakan untuk menjalankan aktivitas-aktivitas yang lain.

Secara umum, cara kerja sehingga akhirnya hasil seperti pada output muncul adalah sesuai pada yang telah ditulis di bagian Algoritma. Dengan bantuan fungsiSeleksi, program akan mampu untuk memilah dan menandai mana saja aktivitas yang "terbaik" untuk diambil pada setiap waktunya. Selain digunakan untuk memilih, terdapat juga bagian pada code yang digunakan untuk membuang atau menandai bagian aktivitas yang telah terpilih agar tidak terpilih kembali.

Apabila nantinya Algoritma menemukan kasus lain, misal kasus pada poin C, maka program akan menolak dan memberikan peringatan bahwa waktu yang tersisa tidak mencukupi untuk melaksanakan segala aktivitas-aktivitas yang bersifat urgent dan penting. Semisal dengan menggunakan daftar aktivitas sebagai berikut dan nilai w yang sama dengan kasus sebelumnya,

No.	Aktivitas	W(jam)	V
1	Aktivitas 1	1	1
2	Aktivitas 2	1	1
3	Aktivitas 3	0.5	1
4	Aktivitas 4	0.25	1
5	Aktivitas 5	1	1
6	Aktivitas 6	2	1
7	Aktivitas 7	0.25	1
8	Aktivitas 8	0.25	1
9	Aktivitas 9	1	1

Tabel 2. Daftar Aktivitas Kasus 2

Tabel ini berisikan jumlah dan nilai w yang sama dengan tabel sebelumnya, hanya saja sekarang nilai v nya diubah menjadi 1 semua. Dan karena nlai w_{total} tidak dapat menampung semua aktivitas yang ada, maka Algoritma akan

melempar kasus ini kepada kasus nomor C, yakni tidak ditemukan solusi.

Sejauh pengujian yang dilakukan oleh penulis, belum ditemukan adanya kasus yang anomali atau tidak dapat ditangani sebagaimana seharusnya. Oleh karena itu, untuk saat ini penulis menganggap bahwa hasil eksekusi dari algoritma ini pada kasus Matriks Eusenhower sudah sangat baik dan dapat terus dikembangkan kedepannya untuk memberikan hasil yang lebih baik lagi.

IV. Kesimpulan

Meskipun telah dibahas bahwa Algoritma Greedy umumnya tidak dapat menemukan solusi optimal global untuk banyak persoalan, namun pada kasus yang diujikan oleh penulis pada makalah ini, Algoritma Greedy telah bekerja dengan sangat baik, baik itu dalam konteks menemukan solusi yang diinginkan, maupun dari segi kecepatan komputasinya. Hal ini dikarenakan kasus uji yang diberikan oleh penulis memang dirasa adalah salah satu kasus yang cocok untuk diselesaikan dengan Algoritma Greedy. Dengan ditulisnya makalah ini, penulis berharap bahwa kedepannya dapat dikembangkan kembali penggunaan Algoritma Greedy ataupun algoritma yang lainnya yang dapat membantu untuk menyelesaikan persoalan-persoalan yang berkaitan dengan Matriks Eusenhower.

V. Ucapan Terima Kasih

Pertama-tama, penulis mengucapkan puji dan syukur kepada Allah Swt. karena atas berkat rahmat dan hidayah-Nya, penulis dapat menyelesaikan penulisan makalah ini dengan cukup baik. Kedua, tak lupa juga penulis mengucapkan terima kasih dan hormat yang sebesar-besarnya kepada Prof. Dwi Hendratmo Widiyantoro, Ph.D selaku dosen pengampu mata kuliah Strategi Algoritma di Kelas 03, karena atas bimbingan dan ilmu dari beliaulah penulis dapat memiliki pemahaman yang cukup baik untuk menuliskan apa yang ada pada makalah ini. Tak lupa juga penulis ucapkan terima kasih atas bantuan dan dorongan dari teman-teman seperkuliahan, sehingga penulis terus bersemangat dan selalu berusaha untuk mengikuti perkuliahan yang ada dengan baik.

tautan Video Youtube

Berikut ini adalah tautan video YouTube yang berisi penjelasan singkat dari isi makalah

https://youtu.be/y6q1XLeoPHU

Referensi

[1]https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Diakses pada 10 Mei 2021 pukul 22.10 WIB

[2] https://brilliant.org/wiki/greedy-algorithm. Diakses pada 10 Mei 2021 pukul 22.25 WIB

[3] https://www.geeksforgeeks.org/greedy-algorithms/. Diakses pada 11 Mei 2021 pukul 09.13 WIB

[4]https://www.gloveworx.com/blog/using-eisenhower-decision-matrix/. Diakses pada 11 Mei 2021 pukul 09.51 WIB

[5] https://techterms.com/definition/heuristic. Diakses pada 11 Mei 2021 pukul 13.30 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, 11 Mei 2021

Muhammad Akram Al Bari
13519142