Pemanfaatan Permasalahan Knapsack pada Optimasi Diskon Belanja dengan Budget Minimum

Daru Bagus Dananjaya - 13519080
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519080@std.stei.itb.ac.id

***Abstrak*—Diskon yang semakin besar cenderung mempengaruhi perilaku belanja seseorang. Namun, harga dari sebuah produk juga memegang peranan penting dalam pengambilan keputusan dalam berbelanja karena setiap orang pasti memiliki modal yang terbatas. Dalam makalah ini, akan dibahas mengenai pemanfaatan persoalan 0/1 knapsack dalam mengoptimasi keputusan belanja seseorang dan perbandingan performa antara metode rekursif dan program dinamis dalam menangani berbagai ukuran permasalahan. Hasil dari perbandingan menunjukkan bahwa program dinamis memiliki performa yang lebih baik dibandingkan metode rekursif dalam hal kompleksitas waktu untuk ukuran permasalahan yang besar.**

Kata kunci—Rekursif; Program Dinamis; 0/1 knapsack; optimasi

I. PENDAHULUAN

Belanja merupakan kegiatan yang sering dilakukan oleh kebanyakan orang, banyak orang memiliki daftar keinginan yang sangat banyak, tetapi hanya memiliki dana yang terbatas. Apalagi, diskon yang ditawarkan oleh *merchant-merchant* melalui promosi, obral, dan kupon menjadikan aktivitas belanja menjadi semakin menarik bagi kebanyakan orang. Pada makalah ini, akan ditunjukkan model optimasi yang dapat dibuat untuk memilih barang yang ada di dalam *wishlist* belanjaan untuk mendapatkan diskon semaksimal mungkin namun dengan modal seminimal mungkin.

Ilmu komputasi dapat dimanfaatkan ke dalam bidang kehidupan manusia. Seiring dengan berkembangnya zaman, banyak algoritme pemecahan masalah baru yang ditemukan dan dikembangkan untuk mempermudah kehidupan manusia. Kemunculan metode-metode baru ini menjadi katalis untuk penemuan baru maupun pengembangan teknologi yang sudah ada.

Salah satu pemanfaatan ilmu komputasi dalam kehidupan sehari-hari dan dapat berhubungan langsung dengan aktivitas belanja adalah persoalan 0/1 *knapsack*. Dengan menggunakan persoalan *knapsack*, dapat dilakukan pemodelan supaya mengeluarkan modal seminimal mungkin tetapi mendapatkan diskon secara maksimal.

Dalam makalah ini, akan digunakan metode *rekursif* dan program dinamis dalam menyelesaikan persoalan 0/1 *knapsack* untuk mengoptimasi diskon belanja kemudian akan dibandingkan performa kedua metode tersebut ketika menangani berbagai ukuran persoalan.

II. LANDASAN TEORI

A. Program Dinamis

Program dinamis (*dynamic programming*) merupakan salah satu algoritme pemecah masalah yang memiliki ciri khas yaitu membagi sebuah masalah besar menjadi sejumlah upapersoalan (*subproblem*). Metode pemecah masalah ini ditemukan oleh seorang matematikawan Amerika bernama Richard Bellman sebagai sebuah solusi untuk optimalisasi proses pengambilan keputusan yang melibatkan berbagai tahap (*multistage*). Jenis persoalan yang dapat diselesaikan dengan program dinamis umumnya memiliki upapersoalan yang saling *overlapping* sehingga setiap upapersoalan hanya perlu diselesaikan sekali saja kemudian disimpan ke dalam suatu tabel. Solusi global diputuskan melalui koleksi solusi dari upapersoalan yang telah diselesaikan dan terdapat di tabel tersebut.[1]

Algoritme pada program dinamis ditandai dengan adanya tahapan-tahapan serta digunakannya prinsip optimalitas dalam proses pemecahan masalah. Dalam menyelesaikan persoalan optimasi, algoritme akan mencari solusi terbaik dari setiap upapersoalan yang ada. Prinsip optimalitas berarti dalam sebuah program dinamis, suatu solusi yang paling optimal akan menghasilkan solusi yang optimal pula untuk setiap upapersoalan pada setiap tahapannya.[1]

Perbedaan program dinamis dan algoritme *greedy* dalam menyelesaikan persoalan optimasi terletak pada upapersoalan yang menjadi perhatian. Pada algoritme *greedy*, pada satu tahap, hanya satu upapersoalan saja yang menjadi perhatian sebagai solusi yang optimal, sedangkan pada program dinamis, seluruh kemungkinan upapersoalan yang terjadi bisa menjadi perhatian dan komputasinya dilakukan secara rekursif.

Meskipun pada proses penyelesaiannya sama-sama menggunakan metode pemecahan persoalan besar menjadi sejumlah persoalan kecil secara rekursif, terdapat perbedaan antara program dinamis dan algoritme *divide and conquer*, yaitu pada jenis persoalan yang ditangani, pada program dinamis, upapersoalan yang ditangani program dinamis dapat saling *overlapping* antara satu dengan lainnya.

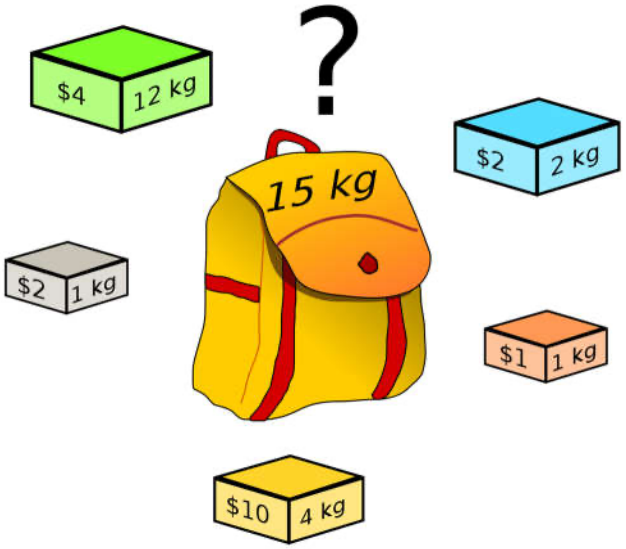
Dalam penyelesaian persoalan, terdapat dua pendekatan yang dapat digunakan dalam program dinamis, yaitu:

1. Program dinamis maju (*top-down*)
2. Program dinamis mundur (*bottom-up*)

Perbedaan dari dua pendekatan tersebut adalah pada tahap inisiasi algoritme, pada pendekatan maju, komputasi dilakukan mulai dari upapersoalan terkecil hingga persoalan globalnya, sedangkan pada pendekatan *bottom-up*, yang dilakukan adalah sebaliknya.

B. Persoalan Knapsack

Persoalan *Knapsack* merupakan sebuah persoalan optimasi. Permasalahannya didefinisikan dengan diberikan suatu himpunan barang, setiap barang dalam himpunan tersebut memiliki keuntungan dan bobot, Kemudian ditentukan kombinasi barang yang dapat dimasukkan ke dalam kontainer *(knapsack)* sehingga menghasilkan nilai maksimum dan bobotnya tidak melebihi kapasitas kontainer.



***Gambar 1** Ilustrasi Permasalahan Knapsack*

(Sumber :

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

Persoalan *knapsack* diselesaikan dengan maksimalisasi nilai serta Batasan pada bobot yang dapat diambil. Secara matematis dapat dituliskan sebagai berikut:

$$maksimasi : \sum_{i=1}^{n} v_i x_i$$

$$batasan : \sum_{i=1}^{n} w_i x_i \leq W$$

dengan v_i adalah profit dari barang ke-*i*, w_i adalah bobot dari barang ke-*i*, x_i adalah jumlah barang yang diambil pada tahap ke-*i*, dan W merupakan kapasitas dari kontainer.

Terdapat beberapa varian dari persoalan *knapsack*, yaitu :

1. Persoalan 0/1 *Knapsack*

   Pada persoalan *knapsack* jenis ini, jumlah barang maksimum yang dapat diambil untuk setiap jenis barang yang ada di dalam himpunan adalah satu.

2. Persoalan *Knapsack* Terbatas (*bounded knapsack problem*)

   Pada persoalan *knapsack* jenis ini, jumlah barang maksimum yang dapat diambil untuk setiap jenis barang yang ada adalah sebesar *c*. Dengan *c* adalah sebuah bilangan bulat yang ditentukan oleh pengguna.

3. Persoalan *Knapsack* Tanpa Batas (*unbounded knapsack problem*)

   Pada persoalan *knapsack* jenis ini, tidak terdapat jumlah maksimum yang dapat diambil untuk setiap jenis barang yang terdapat di himpunan.[2]

Perbedaan dari ketiga varian tersebut dapat dituliskan secara matematis sebagai berikut:

$$0/1\ knapsack: 0 \leq x_i \leq 1$$

$$bounded\ knapsack : 0 \leq x_i \leq c, c \in R$$

$$unbounded\ knapsack : 0 \leq x_i$$

III. IMPLEMENTASI

Pada bagian ini, akan ditunjukkan bagaimana persoalan 0/1 *knapsack* dalam hal ini, optimasi diskon belanja dapat diselesaikan menggunakan dua buah pendekatan, yaitu rekursif dan program dinamis.

A. Definisi Permasalahan

Misalkan terdapat *n* barang dalam sebuah himpunan barang dengan indeks *i* (*i* = 0, 1, 2, … , *n*-1). $Harga_i$ dan $Diskon_i$ merupakan harga dan diskon yang diperoleh jika barang ke-*i* dipilih dari himpunan. Seorang pembeli akan memiliki "modal", permasalahan *knapsack* diharapkan dapat memilih barang sehingga total harga yang harus dikeluarkan tidak lebih dari "modal" yang tersedia dan diskon yang diperoleh maksimum. Berdasarkan teori yang telah dijelaskan pada bagian sebelumnya, x_i akan menjadi keputusan apakah barang ke-*i* dipilih atau tidak. Jika barang ke-*i* dipilih, maka nilai x_i akan di-*assign* menjadi 1, dan 0 jika sebaliknya.

Dari definisi permasalahan di atas, permasalahan dapat dituliskan secara matematis menjadi:

$$Z = max \sum_{i=1}^{n} Diskon_i x_i$$

$$batasan : \sum_{i=1}^{n} Harga_i x_i \leq modal$$

$$x_i \in \{0,1\}$$

B. Pengumpulan Data

Sebanyak 23 data barang dikumpulkan dari *Amazon.com* kemudian diekstraksi ke dalam tabel di bawah. Setiap barang memiliki nama, harga setelah diskon, harga sebelum diskon, dan diskon yang diberikan.

No	Item	Harga	Harga Asli	Diskon
1	Robotics Vacuum Cleaner	$ 149,99	$ 229,99	$ 80,00
2	Wireless Earbuds	$ 39,99	$ 49,99	$ 10,00
3	Electric Scale	$ 10,98	$ 12,99	$ 2,01
4	Air Fryer	$ 99,98	$ 119,99	$ 20,01
5	WiFi Router	$ 89,99	$ 99,99	$ 10,00
6	Smart Door Lock	$ 115,59	$ 369,99	$ 254,40
7	Electric Piano	$ 199,99	$ 299,99	$ 100,00
8	Baby Bassinet	$ 178,49	$ 239,99	$ 61,50
9	Bike Helmet	$ 33,96	$ 38,99	$ 5,03
10	Power bank	$ 40,04	$ 46,98	$ 6,94
11	Smart Watch	$ 335,99	$ 399,99	$ 64,00
12	Playstation Controller	$ 575,99	$ 649,49	$ 73,50
13	Printer	$ 21,99	$ 59,99	$ 38,00
14	Sunglasses	$ 129,46	$ 149,99	$ 20,53
15	Karaoke Microphone	$ 14,44	$ 19,69	$ 5,25
16	Skateboard	$ 37,49	$ 59,99	$ 22,50
17	Dog food	$ 3,59	$ 10,88	$ 7,29
18	Face mask	$ 8,68	$ 14,99	$ 6,31
19	Smart Plug	$ 14,99	$ 19,99	$ 5,00
20	USB C Adapters	$ 25,49	$ 36,99	$ 11,50
21	Flashlight	$ 16,99	$ 22,99	$ 6,00
22	Microfiber Mop	$ 17,84	$ 24,99	$ 7,15
23	Phone case	$ 9,34	$ 13,99	$ 4,65

***Tabel 1** Daftar Barang*

(Sumber : *Dokumen Penulis*)

C. Pemecahan Masalah

1) Metode Rekursif

Metode rekursif merupakan algoritme paling tradisional yang dapat digunakan untuk menyelesaikan persoalan 0/1 *knapsack*. Pada metode ini, persoalan *knapsack* diselesaikan dengan menggunakan fungsi rekursif yang memanggil dirinya sendiri untuk menghitung sub-persoalan dan mengumpulkan hasilnya untuk menemukan solusi dari persoalan globalnya.

Perhitungan yang dilakukan oleh fungsi rekursif bersifat perhitungan-satu-kali, sehingga seluruh pemanggilan rekursif yang dibutuhkan untuk mencapai final *state* adalah independen antara satu dengan yang lainnya. Oleh karena itu, untuk persoalan dengan banyak *instance*, pendekatan rekursif ini cenderung melakukan perhitungan yang sama secara berkali-kali dan berakibat pada meningkatnya kompleksitas komputasi.

Dalam pendekatan rekursif yang digunakan pada persoalan ini, algoritme akan mulai memilih barang yang ada pada akhir indeks list barang kemudian bergerak ke depan. Algoritme rekursif melakukan pengecekan terhadap diskon maksimum yang mungkin kita dapatkan jika memilih suatu barang dengan menambahkan ke sebuah kontainer yang menampung *current discount* dan kemungkinan diskon dengan modal yang tersisa ketika harga barang yang dipilih telah mengurangi modal yang ada.

```
def recKnapsack(B,P,D,N):
    # Function untuk menyelesaikan persoalan
     knapsack
    # menggunakan pendekatan naive recursive
    # B : modal tersedia
    # P : list harga
    # D : list diskon
    # N : jumlah barang
    if (N==0 or B==0):
        return 0
    if P[N-1] > B:
        return recKnapsack(B,P,D,N-1)
    else:
        totalDiskon = max(
           D[N-1] + recKnapsack(B-P[N-1],P,D,N-1),
           recKnapsack(B,P,D,N-1))
```

```
        return totalDiskon
```

2) Metode Program Dinamis

Metode program dinamis dapat menjadi solusi ketika jumlah *instance* yang ada di dalam himpunan semakin banyak, karena berbeda dari metode rekursif, pendekatan dengan program dinamis menyimpan nilai dari sebuah barang pada sebuah iterasi yang akan digunakan pada iterasi-iterasi berikutnya sehingga menyingkirkan kebutuhan untuk menghitung hal yang sama secara berulang. Oleh karena itu, secara teori pendekatan program dinamis ini akan memberikan kompleksitas waktu yang lebih baik dibandingkan metode rekursif untuk jumlah *instance* yang lebih besar.

Pada implementasi persoalan *knapsack* menggunakan pendekatan program dinamis, dibuat sebuah tabel dua-dimensi dengan kolom yang merepresentasikan total modal yang tersedia (Modal = 0, 1, 2, 3, …, M) dan baris merepresentasikan barang yang dapat dipilih pada persoalan 0/1 *knapsack* (Item = 0, 1, 2, …, 23). Untuk setiap sel matriks dimana kolom adalah modal yang merepresentasikan diskon maksimum dan baris adalah sebuah item yang dapat dipilih. Keputusan apakah item di indeks berikutnya harus dipilih atau tidak, diperoleh dengan membandingkan dua hasil. Pertama, apakah dengan menambahkan item akan meningkatkan total diskon namun dengan tetap memenuhi batasan modal. Kedua, total diskon yang mungkin didapat untuk modal yang sema dengan memilih item pada indeks sebelumnya. Dari kedua keputusan tersebut, dipilih nilai yang terbesar kemudian simpan nilainya ke dalam tabel yang telah dibuat tadi.

Harga→ Item↓	0	1	….	M
0				
1				
…				
23				

***Tabel 2** Tabel Program Dinamis*

(Sumber : Dokumen penulis)

Nilai maksimum diskon yang mungkin diperoleh untuk seluruh sel pada Tabel 2 dipilih dan disimpan menggunakan algoritme berikut ini :

a) Mengambil diskon terbaik sebelumnya

```
prevBest = DP[item-1][budget]
```

blok program di atas mengambil nilai diskon dengan modal yang sama namun tanpa memilih *item* pada indeks ke-*i* kemudian nilainya disimpan pada DP[*item*-1][*budget*].

b) Menghitung nilai maksimum baru setelah item ke-i dipilih

```
newBest = D[item-1] + DP[item-1][budget-
(P[item-1])]
```

blok program di atas merepresentasikan total diskon setelah *item* indeks ke-*i* dimasukkan ke dalam kontainer *knapsack*.

c) Memilih nilai maksimum diskon yang akan dimasukkan ke tabel

```
DP[item][budget] = max(prevBest,newBest)
```

Pada tahap ini, akan dipilih nilai maksimum diskon yang mungkin diperoleh setelah memilih *i item* dengan batasan modal *j*. Apabila total diskon yang diperoleh pada tahap sebelumnya lebih besar, akan dipilih *prevBest*. Namun jika total diskon menjadi lebih besar ketika ditambahkan *item* baru, maka *newBest* dipilih. Proses penghitungan ini dilakukan secara terus menerus untuk seluruh sel hingga didapat nilai optimal pada tabel[*23*][*M*].

```
DP = [[0 for x in range(budget+1)]
for x in range(len(Harga)+1)]

def dynamicKnapsack(DP, B, P, D, N):
    # Function untuk menyelesaikan persoalan
knapsack
    # menggunakan pendekatan program dinamis
    # DP : tabel program dinamis
    # B : modal tersedia
    # P : list harga
    # D : list diskon
    # N : jumlah barang
    for item in range(N+1):
        for budget in range(B+1):
            if (item == 0 or budget == 0):
                DP[item][budget] = 0
            elif (P[item-1] > budget):
                DP[item][budget] = DP[item-
                    1][budget]
            else:
                prevBest = DP[item-1][budget]
                newBest = D[item-1] + DP[item-
                    1][budget-(P[item-1])]
                DP[item][budget] =
                    max(prevBest,newBest)
    return DP[N][budget], DP
```

IV. ANALISIS DAN PEMBAHASAN

Dalam implementasi persoalan *knapsack* pada kasus ini, digunakan beberapa asumsi untuk membuat kasus menjadi *solveable*. Asumsi yang digunakan yaitu membulatkan harga barang ke bilangan bulat terdekat, hal ini harus dilakukan karena apabila harga dari suatu barang dibiarkan dalam bilangan asli, maka persoalan menjadi NP-*hard*, atau biasa dikenali dengan *subset sum problem.* Tidak ada algoritme yang membutuhkan waktu polinomial untuk persoalan NP-*hard* kecuali P=NP. Kelemahan dari asumsi ini adalah, beberapa barang mungkin masuk ke dalam kategori produk yang sama, namun barang tersebut tetap tersedia untuk dipilih.

Algoritme persoalan *knapsack* diimplementasikan dalam Bahasa *Python* 3. Komputer yang digunakan untuk menjalankan program memiliki spesifikasi Intel i5-7th gen, 64-bit, dan 8GB RAM. Berdasarkan perhitungan yang dilakukan, didapatkan hasil sebagai berikut :

Jumlah *item* yang dipilih : 17 *item*

Jumlah modal yang dikeluarkan : $ 900

Jumlah diskon yang didapat : $ 626,59

No	Item	Harga	Diskon
1	Robotics Vacuum Cleaner	$ 149,99	$ 80,00
2	Wireless Earbuds	$ 39,99	$ 10,00
3	Electric Scale	$ 10,98	$ 2,01
6	Smart Door Lock	$ 115,59	$ 254,40
7	Electric Piano	$ 199,99	$ 100,00
8	Baby Bassinet	$ 178,49	$ 61,50
9	Bike Helmet	$ 33,96	$ 5,03
13	Printer	$ 21,99	$ 38,00
15	Karaoke Microphone	$ 14,44	$ 5,25
16	Skateboard	$ 37,49	$ 22,50
17	Dog food	$ 3,59	$ 7,29
18	Face mask	$ 8,68	$ 6,31
19	Smart Plug	$ 14,99	$ 5,00
20	USB C Adapters	$ 25,49	$ 11,50
21	Flashlight	$ 16,99	$ 6,00
22	Microfiber Mop	$ 17,84	$ 7,15
23	Phone case	$ 9,34	$ 4,65
	17	900	626,59

Tabel 3 *Hasil pemilihan barang*

(Sumber : Dokumen penulis)

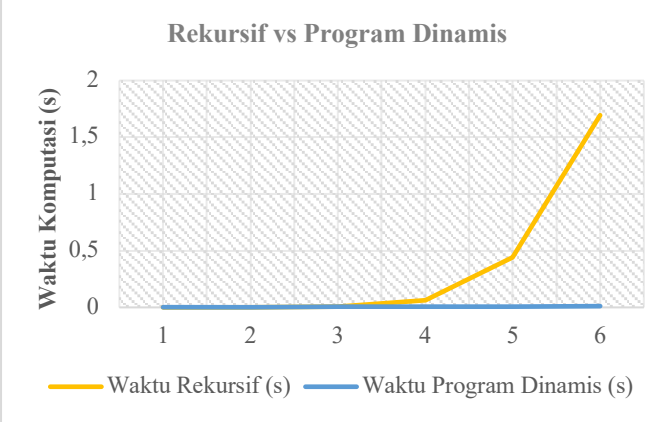
Berdasarkan percobaan yang dilakukan, waktu yang dibutuhkan oleh algoritme program dinamis sedikit lebih cepat, yaitu 0,011098 detik, sedangkan waktu yang dibutuhkan oleh fungsi rekursif untuk mendapatkan hasil adalah 1,694093 detik. Selain itu, penulis juga melakukan percobaan untuk beberapa jumlah *instance data.*

No	Jumlah Barang	Modal	Waktu	
			Rekursif (s)	Program Dinamis (s)
1	10	400	0,000263	0,001679
2	13	500	0,00125	0,00269
3	16	600	0,005784	0,004122
4	19	700	0,062379	0,006361
5	21	900	0,438759	0,008532
6	23	900	1,694093	0,011098

Tabel 4 *Hasil uji untuk beberapa jumlah instance*

(Sumber : Dokumen Penulis)

Berdasarkan data dari tabel, dapat dilihat bahwa untuk jumlah barang kurang dari 16, tidak terlihat perbedaan yang signifikan dalam hal waktu komputasi. Namun, untuk jumlah *instance* yang relatif besar, terlihat bahwa performa pendekatan rekursif kurang baik jika dibandingkan dengan pendekatan program dinamis. Argumen ini diperkuat dengan grafik berikut.



Grafik 1 *Perbandingan waktu komputasi fungsi rekursif dan program dinamis*

(Sumber : Dokumen Penulis)

Grafik di atas menunjukkan bahwa untuk jumlah *instance* yang relatif sedikit, kedua pendekatan memiliki waktu komputasi yang mirip, namun ketika jumlah *instance* banyak, waktu komputasi fungsi rekursif akan meningkat secara eksponensial, berbeda dari program dinamis yang relatif konstan berapapun jumlah *instance* yang diproses.

V. KESIMPULAN

Persoalan knapsack memiliki sangat banyak manfaat bagi kehidupan sehari-hari manusia, terutama untuk mengambil sebuah keputusan. Permasalahan optimasi diskon belanja memiliki objektif untuk menemukan kumpulan barang yang dapat dibeli dari *wishlist* dengan modal terbatas namun menghasilkan total diskon semaksimal mungkin. Selain itu, pada makalah ini juga dilakukan perbandingan performa antara pendekatan rekursif dan program dinamis untuk menyelesaikan persoalan 0/1 knapsack. Berdasarkan percobaan yang dilakukan, didapatkan hasil bahwa untuk ukuran permasalahan yang kecil, kedua pendekatan memiliki performa yang relatif sama. Namun ketika ukuran permasalahan besar, waktu komputasi yang dibutuhkan oleh pendekatan rekursif meningkat secara

eksponensial, sedangkan untuk pendekatan program dinamis relatif konstan. Meskipun performa algoritma program dinamis lebih baik, tetapi program dinamis lumayan mengkonsumsi memori, oleh karena itu, program dinamis yang diimplementasikan dalam makalah ini dapat diganti menggunakan *Approximate Dynamic Programming.*

UCAPAN TERIMAKASIH

Penulis mengucapkan terima kasih dan puji syukur kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat mengerjakan makalah ini dengan lancer dan tepat waktu. Penulis juga mengucapkan terimakasih kepada :

1. Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc. dan seluruh dosen pengajar atas bimbingan dan ilmu yang diajarkan selama perkuliahan Strategi Algoritma.
2. Teman-teman mahasiswa IF'19 yang bersedia menjadi tempat berkeluh kesah serta membantu saya dalam menyelesaikan tugas ini.

VIDEO LINK AT YOUTUBE

Video penjelasan mengenai makalah ini dapat disaksikan pada pranala YouTube di bawah ini.

https://youtu.be/vOVkm9-gac0

REFERENCES

[1] A. Levitin, *Introduction to the Design and Analysis of Algorithms, 3rd Edition*. 2011.

[2] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

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.

Semarang, 11 Mei 2021

Daru Bagus Dananjaya 13519080