Algoritma *Greedy* dan *Branch and Bound* dalam Menentukan Daftar Belanja

Bryan Cornelius Lauwrence - 13522033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): bryan.cornelius.lauw@gmail.com

***Abstract*—Berbelanja adalah aktivitas sehari-hari manusia. Namun, berbelanja memerlukan biaya anggaran supaya kegiatan berbelanja tidak menghamburkan uang. Kegiatan berbelanja merupakan salah satu masalah *knapsack* apabila ingin memaksimalkan banyak barang yang dibeli dengan harga tertentu. Prioritas barang yang dibeli pun menjadi pertimbangan supaya dapat mengkategorikan barang yang benar benar diperlukan. Masalah *knapsack* dapat diselesaikan dengan algoritma *greedy* ataupun *branch and bound*. Namun, masalah berbelanja tidak tentu menggunakan algoritma yang mana karena barang yang dipilih dapat dibeli secara pecahan ataupun satuan. Oleh karena itu, masing-masing algoritma akan diimplementasikan pada kasus berbelanja tergantung barang yang ingin dibeli.**

Keywords—knapsack problem, greedy algorithm, branch and bound algorithm

I. PENDAHULUAN

Pada waktu berbelanja, seseorang akan mendata keperluan-keperluan yang akan dibeli. Selain daftar belanja, besar biaya yang dikeluarkan juga penting untuk diperhatikan supaya tidak mengeluarkan uang secara berlebihan. Dengan mempertimbangkan biaya, mungkin saja barang-barang yang ada di daftar belanja tidak dapat dibeli seluruhnya.

Dengan mempertimbangkan bahwa prioritas barang-barang di daftar belanja berbeda, tujuan utamanya adalah membeli barang sebanyak mungkin tanpa melebihi biaya maksimal. Barang yang dibeli pun sebisa mungkin barang yang lebih diperlukan. Kasus ini sama seperti *knapsack problem*. Pada *knapsack problem*, terdapat tas dengan kapasitas bobot tertentu dan beberapa barang yang memiliki besar keuntungan tertentu serta bobot tertentu. Pada kasus berbelanja, kapasitas bobot maksimal adalah biaya maksimal, harga dan jumlah barang belanja adalah bobot, serta prioritas adalah keuntungan yang didapatkan. Yang membedakan dengan *knapsack* pada umumnya adalah barang belanja yang dibutuhkan bisa lebih dari satu dan terdapat barang yang tidak dibeli satuan, misalnya dibeli kiloan.

Untuk menentukan daftar belanja yang terbaik, tidak mungkin digunakan algoritma *brute force* karena ada barang yang bersifat *fractional* sehingga pada kasus *fractional* daftar belanja hanya dapat ditentukan melalui algoritma *greedy*. Namun, apabila tidak ada barang *fractional*, dapat digunakan algoritma lainnya, seperti algoritma *brute force*, algoritma *branch and bound*, dan pemrograman dinamis. Algoritma *greedy* mungkin tidak menghasilkan daftar belanja yang optimal, algoritma ini hanya akan menghasilkan solusi optimal apabila seluruh barang di daftar belanja bersifat *fractional*.

Untuk memperoleh hasil yang optimal pada daftar belanja yang seluruh barangnya hanya dapat dibeli satuan, algoritma *brute force* tidak dipakai karena memerlukan waktu yang lama. Algoritma yang dipakai adalah algoritma *branch and bound* karena penerapannya lebih sederhana daripada program dinamis, ditambah bobotnya adalah nominal uang sehingga memerlukan banyak memori.

Pada makalah ini penulis akan mengaplikasikan algoritma *branch and bound* serta algoritma *greedy* untuk menentukan daftar belanja. Algoritma *branch and bound* akan digunakan apabila seluruh barang pada daftar belanja hanya dapat dibeli satuan. Algoritma *greedy* digunakan apabila terdapat barang yang dapat dibeli dengan jumlah pecahan. Satuan uang yang digunakan adalah Rupiah Indonesia.

II. TEORI DASAR

A. Algoritma Greedy

Algortima *greedy* adalah metode penyelesaian masalah yang sederhana. Prinsip *greedy* adalah mengambil yang terbaik pada suatu saat. Algoritma ini membentuk solusi langkah demi langkah. Pada setiap langkah, terdapat banyak pilihan yang dievaluasi [1]. Oleh karena itu, pada setiap langkah akan diambil pilihan yang sekiranya dapat mengoptimalkan solusi dengan harapan solusi tersebut merupakan bagian dari solusi optimal.

Elemen algoritma *greedy* adalah himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Himpunan kandidat adalah kandidat yang akan dipilih pada setiap langkah, himpunan solusi adalah kandidat yang sudah dipilih, fungsi solusi untuk memenentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi, fungsi seleksi menyeleksi kandidat berdasarkan strategi yang diterapkan, fungsi kelayakan memeriksa kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi atau tidak, dan fungsi objektif menentukan tujuan memaksimalkan atau meminimalkan.

Terdapat tiga strategi untuk persoalan *knapsack*, yaitu *greedy by profit*, *greedy by weight*, dan *greedy by density* [1].

Greedy by profit akan mengambil objek dengan keuntungan terbesar pada setiap langkahnya. *Greedy by weight* akan mengambil objek dengan bobot terkecil pada setiap langkahnya. *Greedy by density* akan mengambil objek dengan keuntungan per unit berat terbesar pada setiap langkahnya. Pada *fractional knapsack*, strategi terbaik yang pasti memberikan solusi optimal adalah *greedy by density* sehingga strategi tersebut akan dipilih untuk kasus ini.

Himpunan kandidat pada persoalan ini adalah barang-barang yang ingin dibeli. Himpunan solusi berisi barang-barang yang dibeli beserta jumlahnya. Fungsi solusi untuk memberikan daftar barang beserta biayanya. Fungsi seleksi menggunakan *greedy by density*. Fungsi kelayakan dengan memeriksa harga barang-barang yang dibeli tidak melebihi uang yang dibawa. Objektif dari kasus ini adalah memaksimumkan jumlah barang yang dibeli.

B. Algoritma Branch and Bound

Algoritma *branch and bound* merupakan salah satu algoritma untuk persoalan optimasi. Pada dasarnya, algoritma ini mengimplementasikan *BFS* dengan tambahan biaya pada setiap simpulnya serta batasan pembangkitan simpul. Simpul yang dibangkitkan adalah simpul yang biayanya optimal [2]. Simpul pun tidak akan dibangkitkan jika simpul tersebut melanggar batasan yang telah ditetapkan (bila ada). Apabila sudah sampai simpul daun dan biaya simpul daun tersebut sudah optimal, pencarian akan dihentikan.

Seperti yang sudah disebutkan, setiap simpul memiliki biaya yang dihitung dengan rumus tertentu. Oleh karena itu, struktur data yang digunakan adalah *priority queue*. Urutan pembangkitan simpul akan mengikuti *priority queue* yang digunakan. Simpul yang melanggat batasan juga tidak akan dibangkitkan karena tidak mengarah ke solusi.

Pada persoalan ini, batasan yang diberlakukan adalah jumlah barang yang dibeli tidak melebihi biaya belanja maksimal. Langkah *branch and bound* pada kasus ini sebagai berikut:

1. Hitung *density* setiap barang
2. Urutkan barang dari *density* yang lebih besar
3. Pada setiap langkah, evaluasi barang dimasukkan ke dalam daftar dan tidak lalu hitung *cost* simpul yang dibangkitkan
4. Masukkan simpul yang sudah dibangkitkan ke *priority queue* dengan memprioritaskan *cost* terbesar, tidak perlu masukkan ke dalam *priority queue* apabila melanggar batasan
5. Bangkitkan anak-anak dari simpul
6. Apabila seluruh barang sudah dievaluasi dan terdapat simpul hidup dengan *cost* lebih besar, lanjutkan pencarian
7. Hentikan pencarian ketika seluruh barang sudah dievaluasi dan *cost* simpul tersebut maksimal

Rumus untuk menghitung *cost* suatu simpul sebagai berikut:

$$\hat{c}(i) = F + (K - W)p_{i+1}/w_{i+1} \quad (1)$$

dengan F adalah keuntungan yang sudah diperolah, K adalah biaya maksimal, W adalah biaya yang dikeluarkan p_{i+1}/w_{i+1} adalah *density* barang berikutnya [3].

III. IMPELEMENTASI

Pengimplementasian pada algoritma akan dibedakan menjadi dua, yaitu dengan algoritma *greedy* dan algoritma *branch and bound*. Kedua algoritma digunakan secara terpisah. Jika daftar belanja tidak ada barang yang dapat dibeli dalam jumlah pecahan, maka digunakan algoritma *branch and bound*. Jika terdapat satu saja barang yang dapat dibeli dalam jumlah pecahan, maka digunakan algoritma *greedy*. Pengguna dapat memilih algoritma berdasarkan kasus yang diperlukan.

Variabel utama yang dibutuhkan sebagai berikut:

- nBarang: menandakan jumlah jenis barang yang ingin dibeli
- budget: besar anggaran yang dialokasikan untuk belanja
- barang: Barang yang ingin dibeli meliputi nama barang, harga satuan barang, prioritas barang, dan jumlah barang yang ingin dibeli.

Barang dapat berupa pecahan maupun tidak sehingga pembuatan kedua barang dipisahkan. Setiap barang juga memiliki densitasnya, yaitu nilai prioritas dibagi harga satuan barang. Untuk memudahkan, prioritas barang dipisahkan menjadi 1, 2, dan 3. Makin besar prioritasnya, makin penting barang tersebut. Berikut merupakan implementasi barang untuk algoritma *greedy*:

```
class BarangFractional(object):
    def __init__(self, nama: str, hargaSatuan:
int, prioritas: int, jumlah: float, pecahan:
bool):
        self.__nama = nama
        self.__hargaSatuan = hargaSatuan
        self.__prioritas = prioritas
        self.__jumlah = jumlah
        self.__pecahan = pecahan
        self.__density = prioritas/hargaSatuan

    def getNama(self) -> str:
        return self.__nama

    def getHargaSatuan(self) -> int:
        return self.__hargaSatuan

    def getPrioritas(self) -> int:
        return self.__prioritas

    def getJumlah(self) -> float:
        return self.  jumlah
```

```
    def getDensity(self) -> float:
        return self.__density

    def isPecahan(self) -> bool:
        return self.__pecahan
```

Dan berikut adalah implementasi barang untuk algoritma *branch and bound*:

```
class BarangInteger(object):
    def __init__(self, nama: str, harga: int,
prioritas: int):
        self.__nama = nama
        self.__harga = harga
        self.__prioritas = prioritas
        self.__density = prioritas/harga

    def getNama(self) -> str:
        return self.__nama

    def getHarga(self) -> int:
        return self.__harga

    def getPrioritas(self) -> int:
        return self.__prioritas

    def getDensity(self) -> float:
        return self.__density
```

A. *Knapsack dengan Branch and Bound*

Untuk proses *branch and bound*, barang pada daftar belanja akan diurutkan menurun berdasarkan densitasnya. Jika terdapat barang yang dibeli lebih dari satu, barang tersebut akan dipisah sebanyak jumlah yang ingin dibeli sehingga setiap langkah akan mengevaluasi satu barang tertentu akan dibeli atau tidak. Daftar belanja juga diberikan sebuah sentinel di akhir daftar belanja yang densitasnya bernilai nol supaya simpul daun hanya menghitung total biaya yang dikeluarkan untuk solusi yang telah ditemukan. Berikut adalah implementsai daftar belanja *branch and bound*:

```
class DaftarBelanjaInteger(object):
    def __init__(self, biayaMaks: int):
        sentinel = BarangInteger("sentinel", 10,
0)
        self.__listBelanja = [sentinel]
        self.__banyakBarang = 0
        self.__biayaMaks = biayaMaks

    def tambahBarang(self, b: BarangInteger,
jumlah: int):
        while (jumlah >= 1):
            self.__listBelanja.append(b)
            self.__banyakBarang += 1
            jumlah -= 1

    def urutkanBarang(self):
        self.__listBelanja.sort(key=lambda
x:x.getDensity(), reverse=True)

    def getBanyakBarang(self) -> int:
        return self.__banyakBarang

    def getListBelanja(self) ->
list[BarangInteger]:
        return self.__listBelanja

    def getBarangBelanja(self, i: int) ->
BarangInteger:
        return self.__listBelanja[i]

    def getBiayaMaks(self) -> int:
        return self.__biayaMaks
```

Pada setiap langkah, dibangkitkan sebuah simpul. Simpul berisi *list* dari 0 atau 1. Nilai 0 menandakan barang pada indeks tersebut tidak diambil dan nilai 1 menandakan barang pada indeks tersebut diambil. Setiap kali simpul dibangkitkan, *cost* dari simpul tersebut dihitung dengan rumus yang ada pada dasar teori. Kemudian, terdapat kendala sebagai berikut:

$$\sum_{i=1}^{n} w_i x_i \leq K \quad (2)$$

dengan w_i adalah harga barang pada indeks ke-i dan x_i adalah nilai 0 atau 1 yang telah dijelaskan sebelumnya.

Simpul yang sudah dibangkitkan akan dimasukkan ke dalam *priority queue* yang sudah tersedia. Makin besar *cost* simpul, makin besar juga prioritasnya. Pencarian dihentikan ketika tidak ada lagi simpul hidup atau simpul daun memiliki *cost* yang lebih besar daripada seluruh simpul hidup lainnya. Berikut merupakan implementasi simpul:

```
class Node(object):
    def __init__(self, listSolusi: list[int], F:
int, K: int, W: int, density: float):
        self.__listSolusi = listSolusi
        self.__profit = F
        self.__weight = W
        self.__cost = F + (K - W) * density

    def getListSolusi(self) -> list[int]:
        return self.__listSolusi

    def getCost(self) -> int:
```

```
        return self.__cost

    def getProfit(self) -> int:
        return self.__profit

    def getWeight(self) -> int:
        return self.__weight

    def isNodeValid(self, K) -> bool:
        return K >= self.__weight
```

Dan berikut adalah implementasi algoritma *branch and bound* secara keseluruhan:

```
class BranchNBound(object):
    def __init__(self, daftarBelanja:
DaftarBelanjaInteger):
        self.__daftarBelanja = daftarBelanja
        self.__daftarBelanja.urutkanBarang()
        self.__priorityQueue = []

    def head(self) -> Node:
        return self.__priorityQueue[0]

    def insertQueue(self, n: Node):
        i = 0
        found = False

        while (i < len(self.__priorityQueue) and
(not found)):
            if (self.__priorityQueue[i].getCost()
< n.getCost()):
                found = True
            else: i += 1

        self.__priorityQueue.insert(i, n)

    def generateRoot(self):
        barang1 =
self.__daftarBelanja.getBarangBelanja(0)
        self.  priorityQueue.append(Node([], 0,
self.__daftarBelanja.getBiayaMaks(), 0,
barang1.getDensity()))

    def generateChild(self):
        headNode = self.head()
        self.__priorityQueue.pop(0)
        currentSolution = headNode.getListSolusi()
        barangSekarang =
self.__daftarBelanja.getBarangBelanja(len(currentS
olution))
        barangNext =
self.__daftarBelanja.getBarangBelanja(len(currentS
olution)+1)

        leftNode = Node(currentSolution + [0],
headNode.getProfit(),
self.__daftarBelanja.getBiayaMaks(),
                        headNode.getWeight(),
barangNext.getDensity())
        self.insertQueue(leftNode)

        rightNode = Node(currentSolution + [1],
headNode.getProfit()+barangSekarang.getPrioritas()
, self.__daftarBelanja.getBiayaMaks(),
headNode.getWeight()+barangSekarang.getHarga(),
barangNext.getDensity())
        if
(rightNode.isNodeValid(self.__daftarBelanja.getBia
yaMaks())): self.insertQueue(rightNode)

    def finish(self) -> bool:
        headNode = self.head()
        if len(headNode.getListSolusi()) ==
self.  daftarBelanja.getBanyakBarang():
            for i in range(1,
len(self.__priorityQueue)):
                if
(self.  priorityQueue[i].getCost() >
headNode.getCost()):
                    return False
            return True
        return False

    def process(self) -> tuple[list[tuple[int,
str]], int]:
        self.generateRoot()

        while not self.finish():
            self.generateChild()

        temp = []
        hargaAkhir = 0
        finalSolution = self.__priorityQueue[0]
```

```
        listSolution =
finalSolution.getListSolusi()

        for i in range(len(listSolution)):
            if (listSolution[i] == 1):
                temp.append(self.__daftarBelanja.g
etBarangBelanja(i).getNama())
                hargaAkhir +=
self.__daftarBelanja.getBarangBelanja(i).getHarga(
)

        ret = []
        while (len(temp) > 0):
            strHead = temp[0]
            ret.append((temp.count(strHead),
strHead))
            temp = [item for item in temp if item
!= strHead]

        return (ret, hargaAkhir)
```

Program akan mengembalikan total biaya yang dikeluarkan dan daftar belanja yang disarankan oleh algoritma.

B. Knapsack dengan Greedy

Untuk proses *greedy*, barang pun akan diurutkan berdasarkan densitasnya secara menurun untuk mempermudah perhitungan. Perbedaan dengan *branch and bound*, barang pada algoritma *greedy* tidak akan dipisah menjadi satuan karena barang sudah terurut dan akan diambil sebanyak mungkin berdasarkan urutan yang sudah tersedia. Berikut merupakan implementasi barang *fractional*:

```
class BarangFractional(object):
    def __init__(self, nama: str, hargaSatuan:
int, prioritas: int, jumlah: float, pecahan:
bool):
        self.__nama = nama
        self.__hargaSatuan = hargaSatuan
        self.__prioritas = prioritas
        self.__jumlah = jumlah
        self.__pecahan = pecahan
        self.__density = prioritas/hargaSatuan

    def getNama(self) -> str:
        return self.__nama

    def getHargaSatuan(self) -> int:
        return self.__hargaSatuan

    def getPrioritas(self) -> int:
        return self.__prioritas

    def getJumlah(self) -> float:
        return self.__jumlah

    def getDensity(self) -> float:
        return self.__density

    def isPecahan(self) -> bool:
        return self.__pecahan
```

Boolean pecahan menandakan apakah barang tersebut dapat diambil sebagian atau tidak. Jika pecahan bernilai benar, artinya barang tersebut dapat diambil sebagian dan sebaliknya.

Daftar belanja yang diterapkan kurang lebih mirip dengan yang digunakan pada daftar belanja *branch and bound*. Berikut adalah implementasi daftar belanja pada algoritma *greedy*:

```
class DaftarBelanjaFractional(object):
    def __init__(self, biayaMaks: int):
        self.__listBelanja = []
        self.__biayaMaks = biayaMaks

    def tambahBarang(self, b: BarangFractional):
        self.__listBelanja.append(b)

    def urutkanBarang(self):
        self.__listBelanja.sort(key=lambda
x:x.getDensity(), reverse=True)

    def getListBelanja(self) ->
list[BarangFractional]:
        return self.__listBelanja

    def getBarangBelanja(self, i: int) ->
BarangFractional:
        return self.__listBelanja[i]

    def getBiayaMaks(self) -> int:
        return self.__biayaMaks
```

Berikut adalah algoritma *greedy* yang diterapkan:

```
class Greedy(object):
    def __init__(self, daftarBelanja:
DaftarBelanjaFractional):
        self.__daftarBelanja = daftarBelanja
        self.__daftarBelanja.urutkanBarang()
```

```
    def process(self) -> tuple[list[tuple[float,
str]], int]:
        temp = []
        hargaAkhir = 0

        i = 0
        while (i <
len(self.__daftarBelanja.getListBelanja()) and
hargaAkhir <=
self.__daftarBelanja.getBiayaMaks()):
            currentBarang =
self.__daftarBelanja.getBarangBelanja(i)
            i += 1

            if (not currentBarang.isPecahan()):
                max =
(self.__daftarBelanja.getBiayaMaks() - hargaAkhir)
// currentBarang.getHargaSatuan()
            else:
                max =
(self.__daftarBelanja.getBiayaMaks() - hargaAkhir)
/ currentBarang.getHargaSatuan()
                max = round(max, 2)

            if (max > 0):
                if (max >
currentBarang.getJumlah()): max =
currentBarang.getJumlah()
                hargaAkhir += max *
currentBarang.getHargaSatuan()
                temp.append((max,
currentBarang.getNama()))
        return (temp, hargaAkhir)
```

Perbedaan barang *fractional* dan *integer* terdapat pada jumlah barang. Barang *integer* hanya dapat diambil dalam jumlah bilangan bulat dan barang *fractional* dapat diambil dalam jumlah pecahan.

IV. Hasil Pengujian

Pengujian untuk masing-masing algoritma dilakukan secara terpisah karena masing-masing mengimplementasikan cara yang berbeda. Hasil yang akan diamati adaah optimal tidaknya solusi yang diberikan.

A. *Pengujian Algoritma Branch and Bound*

Kasus uji untuk algoritma *branch and bound* sebagai berikut:

TABLE I. DAFTAR HARGA KASUS UJI 1

Nomor	Nama Barang	Harga Satuan	Jumlah Barang	Prioritas
1	Air minum	5000	2	2
2	Sabun	5000	3	3
3	Shampo	15000	1	2
Anggaran	Rp 30.000			

Maka akan diperoleh hasil sebagai berikut:

TABLE II. HASIL KASUS UJI 1

Nomor	Nama Barang	Jumlah Barang
1	Sabun	2
2	Air minum	3
3	Shampo	1
Biaya	Rp 25.000	

B. *Pengujian Algoritma Greedy*

Terdapat dua kasus uji untuk algoritma *greedy*, yaitu daftar belanja yang seluruhnya barang pecahan dan daftar belanja yang terdapat barang *integer* di dalamnya. Berikut merupakan kedua kasus uji:

TABLE III. DAFTAR HARGA KASUS UJI 2

Nomor	Nama Barang	Harga Satuan	Jumlah Barang	Pecahan	Prioritas
1	Telur	25000	0.5	Ya	1
2	Daging ayam	5000	1	Ya	3
3	Buah Pepaya	15000	1.5	Ya	2
Anggaran	Rp 30.000				

TABLE IV. DAFTAR HARGA KASUS UJI 3

Nomor	Nama Barang	Harga Satuan	Jumlah Barang	Pecahan	Prioritas
1	Air minum	5000	1	Tidak	1
2	Sabun	5000	2	Tidak	2
3	Telur	15000	1.5	Ya	3
Anggaran	Rp 30.000				

Untuk setiap kasus uji, akan diperoleh hasil sebagai berikut:

TABLE V. HASIL KASUS UJI 2

Nomor	Nama Barang	Jumlah Barang
1	Telur	0.1
2	Daging ayam	1
3	Buah Pepaya	1.5
Biaya	Rp 30.000	

TABLE VI. HASIL KASUS UJI 2

Nomor	Nama Barang	Jumlah Barang
1	Air Minum	1
2	Sabun	2
3	Telur	1
Biaya	Rp 30.000	

C. *Analisis*

Berdasarkan kasus uji yang telah dilakukan, algoritma *branch and bound* mengutamakan barang dengan prioritas tinggi dan harga yang tinggi. Namun, perbedaan yang kontras antara harga barang dengan prioritas menyebabkan algoritma lebih mengutamakan harga barang yang lebih murah. Pada jumlah barang yang sedikit efeknya tidak terlalu terlihat, tetapi makin banyak jumlah barang efeknya akan lebih terlihat. Jadi,

algoritma *branch and bound* akan mengusahakan sebanyak mungkin jumlah barang yang dipilih.

Di sisi lain, algoritma *knapsack* akan mengambil seluruh barang yang densitasnya tinggi. Algoritma *knapsack* mengurutkan barang dari densitas tinggi ke densitas yang lebih rendah sehingga barang yang prioritasnya tinggi dan murah pasti akan masuk ke daftar beli. Jadi, seperti algoritma *branch and bound*, algoritma *knapsack* pun mengutamakan barang yang lebih murah.

V. KESIMPULAN

Berdasarkan algoritma yang sudah dibuat dan kasus uji yang telah dilakukan, kedua algoritma lebih mengutamakan kuantitas barang yang dibeli daripada prioritasnya. Prioritas barang akan lebih diutamakan apabila harga barang relatif sama. Kesimpulannya, kedua algoritma berhasil memberikan solusi untuk pembuatan dan pengoptimalan daftar belanja.

Masing-masing algoritma memiliki kelebihan dan kekurangannya tersendiri. Algoritma *branch and bound* menjamin hasil optimal, tetapi hanya bisa diterapkan apabila barang yang ingin dibelanjakan hanya dapat dibeli satuan. Sedangkan algoritma *greedy* dapat diterapkan pada semua kasus, tetapi belum tentu memberikan hasil yang optimal, kecuali jika seluruh barang yang ingin dibelanjakan berupa barang pecahan. Saran penulis untuk pengembangan yang lebih baik adalah dengan memisahkan barang *integer* dan *fractional* pada satu daftar belanja kemudian menghitung algoritma yang sesuai dengan masing-masing kasus. Dengan begitu, dapat diperoleh hasil yang lebih optimal.

REPOSITORY GITHUB

https://github.com/BryanLauw/Makalah-Stima

LINK VIDEO YOUTUBE

https://youtu.be/x5F3-54YKbY?si=jvxVUk6xamkh2G18

UCAPAN TERIMA KASIH

Puji syukur dan terima kasih kepada Tuhan Yang Maha Kuasa sebab atas rahmat dan kasih karunia-Nya, makalah ini dapat selesai sesuai tujuan penulis tepat pada waktunya. Tidak lupa terima kasih kepada dosen-dosen pengajar mata kuliah IF2211 Strategi Algoritma tahun pelajaran 2023/2024, yaitu Dr. Rinaldi Munir, Dr. Nur Ulfa Maulidevi, Dr. Rila Mandala, dan Bapak Monterico Adrian, S.T, M.T atas bimbingannya selama proses perkuliahan Strategi Algoritma dalam satu semester ini. Tidak lupa penulis mengucapkan terima kasih kepada keluarga penulis,teman-teman penulis, dan para pembaca makalah ini atas dukungannya sehingga makalah ini dapat selesai dengan baik. Terakhir, penulis mengucapkan maaf dengan segenap hati apabila terdapat salah kata yang tanpa sengaja menyakiti pihak-pihak tertentu.

REFERENCES

[1] Munir, Rinaldi. (2021). "Algoritma Greedy (Bagian 1)". [Online]: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. Diakses pada 11 Juni 2024.

[2] Munir, Rinaldi. (2021). "Algoritma Branch & Bound (Bagian 1)". [Online]: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf. diakses pada 11 Juni 2024.

[3] Munir, Rinaldi. (2022). "Algoritma Branch & Bound (Bagian 4)" [Online]: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Branchand-Bound-2022-Bagian4.pdf. Diakses pada 12 Juni 2024.

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, 12 Juni 2024

Bryan Cornelius Lauwrence 13522033