Pemanfaatan Decision Tree dalam Pemecahan *Knapsack's Problem* dengan Metode Branch and Bound

Christian Albert Hasiholan - 13521078[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521078@std.stei.itb.ac.id

***Abstract*—Kombinatorial memiliki banyak penerapan pada dunia nyata. Salah satu cabang dari kombinatorial adalah optimasi kombinatorial, yaitu penentuan kombinasi paling optimal. Terdapat beberapa persoalan optimasi kombinatorial dan *Knapsack's Problem* merupakan salah satu yang cukup sering ditemukan. Oleh karena itu pada makalah ini akan dijelaskan tentang *Knapsack's Problem* serta metode yang cukup optimal untuk menyelesaikan *Knapsack's Problem*, yaitu menggunakan metode *Branch and Bound*.**

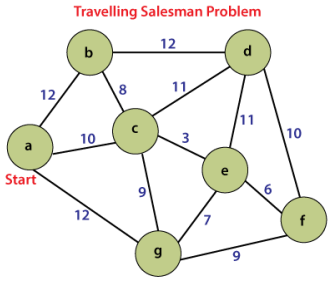
***Keywords—Branch and Bound*, Kombinatorial, Optimasi Kombinatorial, *Knapsack's Problem*.**

I. PENDAHULUAN

Kombinatorial merupakan salah satu area matematika yang cukup sering ditemui dalam masalah sehari-hari. Aplikasi kombinatorial di dunia nyata sendiri yang paling mudah diperhatikan adalah password yang ada pada seluruh media sosial. Selain itu kombinatorial memiliki banyak cabang ilmu yang berhubungan dengan area matematika lainnya. Salah satu cabang tersebut merupakan optimasi kombinatorial.

Kombinatorial sering kali digunakan untuk menghitung banyaknya kemungkinan yang ada. Namun terkadang dicari juga kemungkinan yang paling optimal. Oleh karena itulah terdapat salah satu cabang dari kombinatorial yang menangani masalah tersebut. Optimasi adalah proses atau metode yang digunakan untuk mencapai hasil yang optimal. Maka optimasi kombinatorial memiliki fungsi untuk mencari hasil yang optimal dari seluruh kombinasi yang ada.

Optimasi kombinatorial banyak digunakan dalam berbagai bidang di *computer science*, seperti *artificial intelligence*, *machine learning*, dan lain-lain. Beberapa persoalan yang termasuk dalam optimasi kombinatorial sebenarnya juga telah sering ditemui, seperti *travelling salesman problem*, *minimum spanning tree problem*, dan *knapsack problem*.



Gambar 1.1. Contoh Persoalan Optimasi Kombinatorial
Sumber: *Javapoint (https://www.javatpoint.com/travelling-salesman-problem-in-java)*

Pada makalah ini persoalan optimasi kombinatorial yang akan dibahas merupakan *knapsack's problem*. *Knapsack's problem* merupakan persoalan yang mana *knapsack* harus diisi seoptimal mungkin dibatasi kapasitas *knapsack* itu sendiri. Dalam memecahkan permasalahan ini ada beberapa metode yang bisa digunakan. Baik dari metode yang tidak efisien sampai metode yang kompleks namun efisien. Dan metode yang akan digunakan adalah *branch and bound* yang mengaplikasikan *decision tree* dalam pencarian solusi.

II. DASAR TEORI

A. *Kombinatorial*

Kombinatorial adalah cabang matematika untuk menghitung jumlah penyusunan objek-objek tanpa harus mengenumerasi semua kemungkinan susunannya.

1. Permutasi

   Permutasi adalah banyaknya penyusunan objek dengan urutan yang berbeda. Misal terdapat n buah objek, maka jumlah penyusunan objek tersebut dapat dihitung dengan rumus

   $$n(n-1)(n-2)..(2)(1) = n!$$

   Jika dari n objek hanya dipilih r objek, dimana $r \leq n$ maka dapat digunakan rumus berikut untuk memperoleh jumlah susunan yang mungkin.

   $$P(n,r) = n(n-1)(n-2)..\big(n-(r-1)\big) = \frac{n!}{(n-r)!}$$

2. Kombinasi

Kombinasi adalah sebuah bentuk khusus dari permutasi. Perbedaan kombinasi dengan permutasi terdapat pada urutan kemunculan objeknya. Pada permutasi urutan kemunculan objek diperhitungkan, sedangkan pada kombinasi urutan kemunculan objek diabaikan, sehingga bila urutannya berbeda namun banyak kemunculan objek sama maka dianggap sebagai susunan yang sama.

Pada kombinasi, untuk menentukan jumlah pemilihan r elemen dari n buah elemen maka dapat digunakan rumus berikut

$$C(n,r) = \frac{n(n-1)(n-2)\dots(n-(r-1))}{r!} = \frac{n!}{r!\,(n-r)!}$$

B. Graf

Graf adalah kumpulan simpul yang dihubungkan oleh sisi. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Secara formal, graf (G) didefinisikan sebagai $G = (V, E)$ yang dalam hal ini:

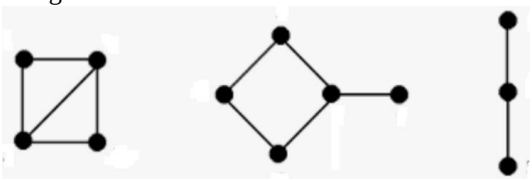
V = himpunan tidak kosong dari simpul-simpul (*vertices*)
$= \{v_1, v_2, \dots, v_n\}$

E = himpunan sisi (*edges*) yang menghubungkan simpul
$= \{e_1, e_2, \dots, e_n\}$

Berdasarkan ada dan tidaknya gelang atau sisi ganda pada suatu graf, maka graf dapat digolongkan menjadi 2 jenis, yaitu :

1. Graf sederhana (*simple graph*)

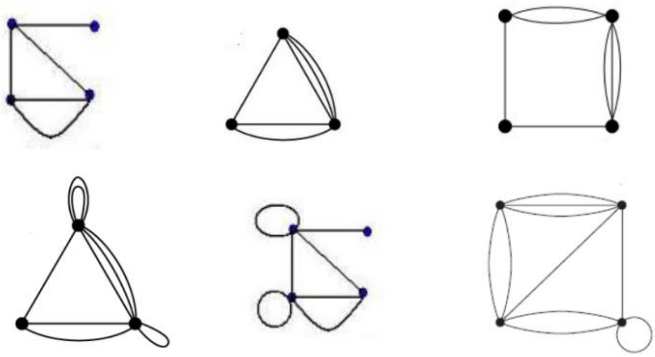
Graf sederhana adalah graf yang tidak memiliki gelang maupun sisi ganda.



Gambar 2.1. Graf Sederhana
Sumber: Slide Bahan Kuliah Graf (Bag 1)

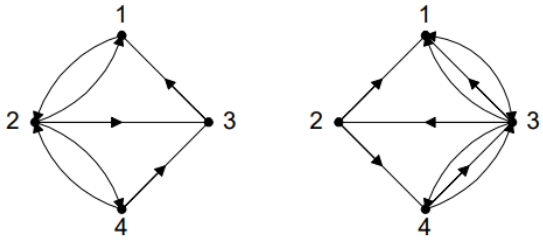
2. Graf tak-sederhana (*unsimple graph*)

Kebalikannya dari graf sederahana, graf tak sederhana memiliki sisi ganda atau sisi gelang. Graf tak sederhana juga dapat digolongkan kembali berdasarkan apakah graf tersebut memiliki sisi ganda atau sisi gelang. Graf yang mengandung sisi ganda disebut graf ganda (*multi-graph*), sedangkan graf yang mengandung sisi gelang disebut graf semu (*pseudo-graph*).



Gambar 2.2. Graf Tak Sederhana
Sumber: Slide Bahan Kuliah Graf (Bag 1)

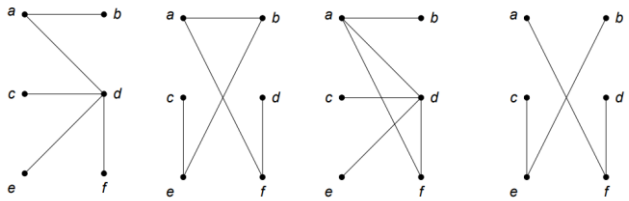
Berdasarkan orientasi arahnya, graf juga dapat dibedakan menjadi dua jenis, yaitu graf tak berarah (*undirected graph*) dan graf berarah (*directed graph* atau *digraph*). Contoh dari graf berarah adalah sebagai berikut :



Gambar 2.3. Graf Berarah
Sumber: Slide Bahan Kuliah Graf (Bag 1)

C. Tree

Pohon adalah graf tak-berarah yang terhubung yang tidak mengandung sirkuit.



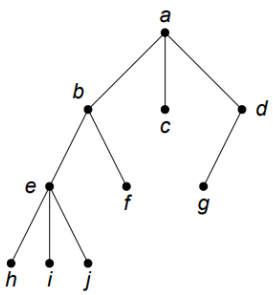
Gambar 3.4. Pohon
Sumber: Slide Bahan Kuliah Pohon (Bag 1)

Jika $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya adalah n. Maka, pernyataan dibawah ini ekuivalen :

1. G adalah pohon.
2. Setiap pasang simpul di G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.

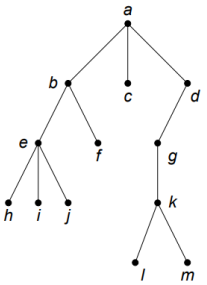
D. Pohon berakar

Pohon berakar adalah pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya dirberi arah sehingga menjadi graf berarah dinamakan pohon berakar (*rooted tree*).



Gambar 3.5. Pohon Berakar
Sumber: Slide Bahan Kuliah Pohon (Bag 2)

Dalam pohon berakar terdapat beberapa terminologi dan akan ditunjukkan pada gambar di bawah ini



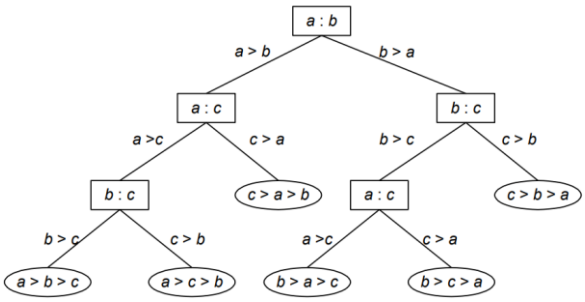
Gambar 3.6. Contoh Terminologi Pohon Berakar
Sumber: Slide Bahan Kuliah Pohon (Bag 2)

Terminologi yang terdapat pada pohon berakar yaitu:

1. Anak (*child*) dan Orangtua (*parent*)
   Jika terdapat dua buah simpul yang terhubung maka simpul asal disebut orangtua dan simpul yang dituju disebut sebagai anak. Pada gambar b,c, dan d adalah anak-anak dari simpul a, sedangkan a adalah adalah orang tua dari simpul tersebut.
2. Lintasan (*path*)
   Lintasan adalah jumlah sisi yang dilewati dari suatu simpul ke simpul lainnya yang terhubung. Contoh lintasan pada gambar adalah lintasan dari a ke j yang melewati simpul a,b,e,j dan panjangnya 3.
3. Saudara kandung (*sibling*)
   Simpul anak yang memiliki simpul parent yang sama. Contohnya f dan e adalah *sibling*, tetapi g dan e bukan karena memiliki simpul parent yang berbeda.
4. Upapohon (*subtree*)
   Sebuah pohon berkar yang merupkan bagian dari pohon awal.
5. Derajat (*degree*)
   Derajat merupakan jumlah upapohon atau jumlah simpul anak pada suatu simpul. Derajat maksimun dari semua simpul merupakan derajat pohon tersebut. Pada gambar, derajat a adalah 3, b adalah 2, sedangkan derajat pohon adalah 3 karena maksimum derajat pada pohon ada pada simpul e.
6. Daun
   Daun adalah simpul yang tidak memiliki anak atau simpul berderajat 0. Simpul h,i,j,f,c,l, dan m adalah daun.
7. Simpul dalam (*internal nodes*)
   Simpul dalam adalah simpul yang memilik anak, namun *root* tidak termasuk simpul dalam. Contoh simpul dalam adalah simpul b,d,e,g, dan k.
8. Aras (*level*) atau Tingkat
   Panjang lintasan yang dilalui dari akar menuju suatu simpul. *Level* dimulai dari root yang berlevel 0.
9. Tinggi (*height*) atau Kedalaman (*depth*)
   Aras maksimun dari sutu pohon. Pohon pada gambar memiliki tinggi 4.

E. Pohon Keputusan

Pohon keputusan merupakan salah satu aplikasi dari pohon berakar. Pohon keputusan digunakan untuk mengilustrasikan persoalan yang terdiri dari pemilihan keputusan dan dari keputusan yang dipilih akan sampai pada sebuah solusi. Pohon keputusan dapat digunakan untuk memilih keputusan yang akan diambil dengan mempertimbangkan kondisi yang ada. Akar pada pohon keputusan menyatakan sebuah perbandingan dan sisi melambangkan keputusan yang dipilih. Daun-daun menyatakan solusi akhir dari keputusan yang telah dipilih.

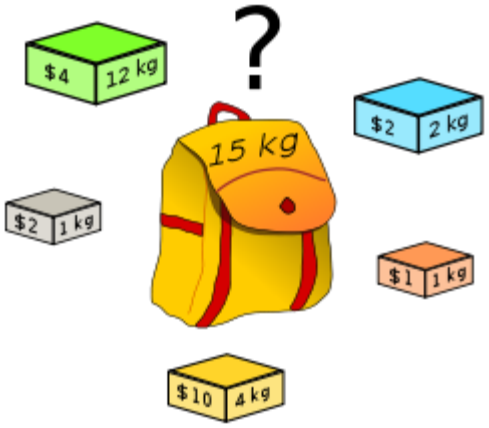


Gambar 3.7. Contoh Pohon Keputusan
Sumber: Slide Bahan Kuliah Pohon (Bag 2)

F. Knapsack's Problem

1. Pengertian

Knapsack's problem adalah salah satu *combinatorial optimization problem*. Misalkan terdapat n buah objek, tiap objek memiliki *value* dan *weight*nya masing-masing, maka tentukan kombinasi objek yang menghasilkan *value* semaksimal mungkin namun dengan total *weight* yang kurang atau sama dengan kapsitas *knapsack* yang ada.



Gambar 3.8. *Knapsack's Problem*
Sumber: Wikipedia
(*https://en.wikipedia.org/wiki/Knapsack_problem*)

2. Tipe-tipe *Knapsack's Problem*

Knapsack's problem dapat dibagi kembali menjadi beberapa tipe, yaitu *0/1 knapsack*, *fractional knapsack*, *bounded* dan *unbounded knapsack*.

a. 0/1 Knapsack

Pada *0/1 knapsack*, kemunculan tiap objek dilambangkan dengan x_i, yang bernilai 0 atau 1. 0 menandakan bahwa objek tersebut tidak masuk dalam kombinasi, sedangkan 1

menandakan bahwa objek tersebut masuk. Jika terdapat n buah objek, dimana tiap objek memiliki value v_i dan weight w_i, dengan kapasitas maksimum *weight* adalah W, maka

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

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

b. Fractional Knapsack

Fractional knapsack cukup mirip dengan *0/1 knapsack*. Namun pada *fractional knapsack*, tiap objek dapat dipecah menjadi *value* dan *weight* yang lebih kecil. Tujuan hal ini dilakukan adalah untuk memaksimalkan *value* yang diperoleh dengan total *weight* yang sama dengan batas *weight* maksimal. Oleh karena itu pemecahan masalah ini seringkali digunakan dengan mencari rasio dari tiap objek terlebih dahulu.

c. Bounded and ubounded Knapsack

Bounded dan unbounded knapsack problem juga cukup mirip dengan 2 tipe *knapsack* sebelumnya. Pada *knapsack* ini, objek tidak bisa dipecah, tetapi jumlah objek yang diambil bisa lebih dari 1. Lalu perbedaan antara *bounded* dan *unbounded* terletak pada batas tiap objek yang dapat diambil, pada *bounded* terdapat non-negatif integer c yang melambangkan batas maksimalnya, sedangkan pada *unbounded* tidak ada batas maksimalnya.

$$subject\ to \sum_{i=1}^{n} w_i x_i \leq W\,, \qquad x_i \in \{0,1,2,\dots,c\}$$

$$subject\ to \sum_{i=1}^{n} w_i x_i \leq W\,, \qquad x_i \in \mathbb{Z}, \qquad x_i \geq 0$$

3. Metode

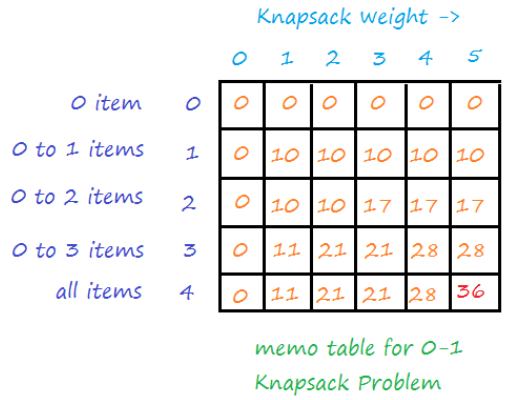
Terdapat beberapa metode untuk menyelesaikan suatu *knapsack's problem*, tetapi tidak semua metode bisa digunakan untuk menyelesaikan persoalan yang sama karena ada metode yang hanya bisa digunakan pada tipe *knapsack's problem* tertentu.

a. Greedy Approach

Greedy approach hanya dapat dilakukan pada *fractional knapsack's problem* sedangkan pada *0/1 knapsack's problem* dapat menghasilkan hasil yang tidak tepat. Ide dari metode ini adalah dengan menghitung terlebih dahulu rasio antara *value per weight* tiap objek. Setelah didapat seluruh rasionya dan diurutkan dari yang terbesar hingga terkecil, maka akan dipilih objek-objek dengan rasio terbesar hingga total *weight* seluruh objek mencapai batas maksimal.

b. Dynamic Programming

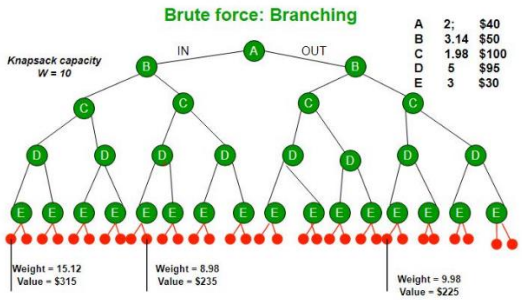
Dynamic programming dapat dilakukan pada *0/1 knapasack's problem*. Namun metode ini tidak bisa digunakan jika *weight* pada objek ada yang bukan integer. Pada metode ini dibuat sebuah tabel *DP* berukuran $(n+1)\ X\ W$. $DP[i][j]$ mengartikan maksimum *value* dari *j-weight* berdasarkan value objek 1 sampai i.



Gambar 3.9. *Knapsack's Problem with Dynamic Programming*
Sumber: PencilProgrammer
(*https://pencilprogrammer.com/algorithms/0-1-knapsack-problem-dynamic-programming/*)

c. Brute Force

Brute force adalah salah satu metode untuk mencari solusi dari *0/1 knapsack's problem*, yaitu dengan mencoba semua kemungkinan yang ada. Karena tiap objek hanya memiliki 2 pilihan maka untuk n buah objek terdapat 2^n kemungkinan solusi. Tentu untuk objek yang sangat banyak metode ini sangat tidak efisien karena kompleksitasnya yang eksponensial. Metode ini juga dapat digambarkan dengan pohon biner.



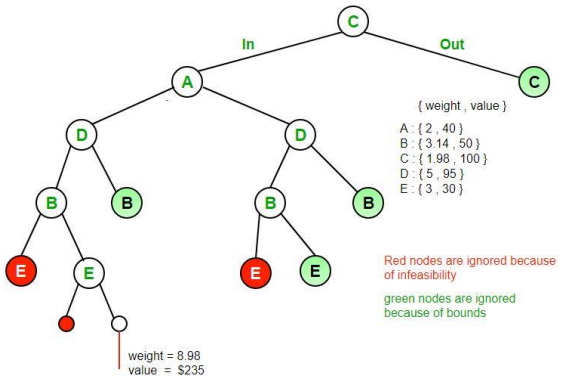
Gambar 3.10. Knapsack's Problem with Brute Force
Sumber: Geeksforgeeks
(*https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/*)

d. Backtracking

Backtracking adalah pengembangan dari metode *Brute Force*. Perbedaannya adalah dengan backtracking tidak perlu semua solusi dicari. Ketika sampai pada simpul yang melewati batas maksimal *weight* maka simpul tersebut diabaikan.

e. Branch and Bound

Branch and bound juga merupakan pengembangan dari metode *Brute Force* dan *Backtracking*. Pada metode ini perlu dicari terlebih dahulu batasan untuk tiap simpul sehingga bila melewati batasan yang ada, simpul tersebut dapat diabaikan. Batasan tersebut dapat dicari dengan *Greedy approach*.



Gambar 3.10. *Knapsack's Problem with Branch and Bound*
Sumber: Geeksforgeeks
(*https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/)*

III. Penerapan Branch and Bound Untuk Menentukan Solusi Knapsack's Problem

Misal terdapat 4 buah objek yang masing-masing memiliki value dan weight, yaitu

Value	12	10	10	18
Weight	6	4	2	9

Tabel 3.1. Tabel Contoh Objek

dan knapsack memiliki kapasitas $W = 15$, maka akan dipilih beberapa objek yang menghasilkan total value maksimal, namun dengan total *weight* yang kurang dari atau sama dengan kapasitas *knapsack*.

Karena metode yang digunakan adalah *Branch and Bound*, maka sama seperti *Greedy approach*, perlu dicari dahulu rasio tiap objek dan diurutkan dari yang terbesar hingga terkecil.

i	1	2	3	4
Value(v_i)	10	10	12	18
Weight(w_i)	2	4	6	9
Ratio(r_i)	5	2.5	2	2

Tabel 3.2. Tabel Objek Setelah Diurutkan

Setelah itu akan dibentuk decision tree dan pada tiap simpulnya akan dicari *upper* (u) dan *lower (c) bound*. Untuk mencari *upper bound* dapat dilakukan dengan rumus

$$u = -\sum_{i=1}^{n} v_i x_i \ , w_i \leq W$$

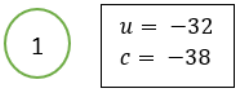
dimana $x_i$menentukan apakah objek tersebut dimasukan ke *knapsack* atau tidak. *Lower bound* dapat dicari dengan cara yang sama namun boleh dilakukan *fraction* untuk memaksimalkan hasilnya.

$$frac = (c - \sum_{i=1}^{n} w_i) * r_{n+1}$$
$$c = u - frac$$

Kemudian inisialisasi sebuah variabel, variabel tersebut akan menyimpan *upper bound* terbesar dan akan dibandingkan dengan tiap simpul. Nilai dari variabel tersebut dapat diperoleh dari *upper bound root*. Maka tentukan *upper* dan *lower bound* dari *root* dengan rumus sebelumnya.

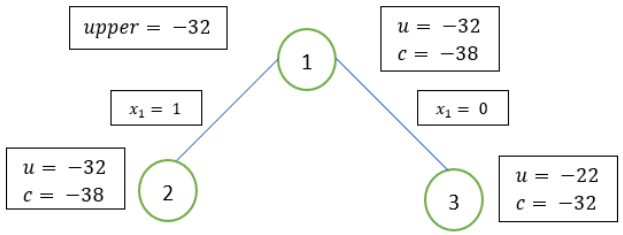
$$u = -(10 + 10 + 12) = -32$$
$$c = -(10 + 10 + 12 + (15 - 12) * 2) = -38$$

$$upper = -32$$



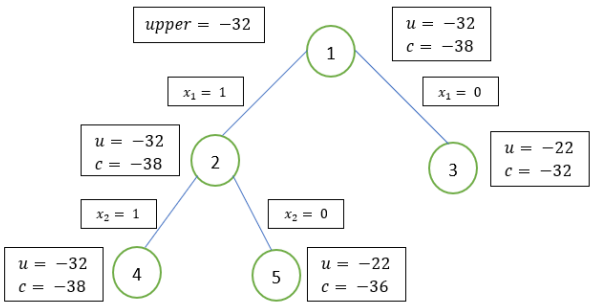
Gambar 3.1. Simpul Root dan State 0
Sumber: Dokumentasi Penulis

Selanjutnya buat simpul anak dari root, dimana simpul anak pertama merupakan state bila objek 1 dimasukan pada knapsack sedangkan simpul anak kedua merupakan state bila objek 1 tidak dimasukan. Tentukan juga upper dan lower bound dari kedua node. Jika pada suatu node ditemukan upper bound yang kurang dari variabel upper, maka nilai dari variabel upper perlu diperbaharui dengan nilai upper bound tadi.



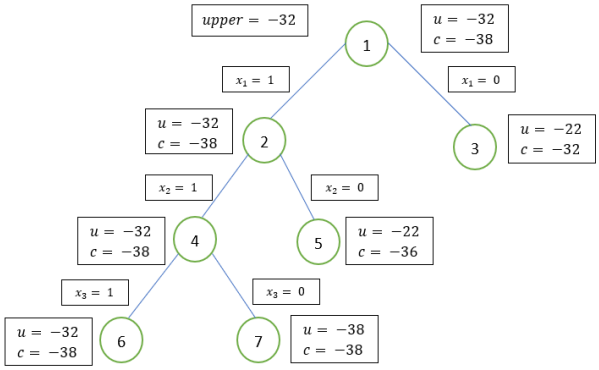
Gambar 3.2. State 1
Sumber: Dokumentasi Penulis

Terdapat beberapa cara untuk menentukan node mana yang selanjutnya dicabangkan, namun untuk optimasi metode branch-bound maka digunakanlah least cost, yaitu memilih node dengan cost atau lower bound yang lebih kecil. Dari gambar di atas maka node yang dipilih adalah node 2. Kemudian lakukan langkah yang sama seperti sebelumnya.



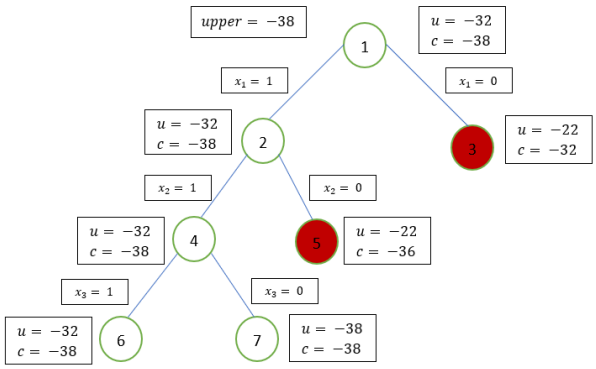
Gambar 3.3. State 2
Sumber: Dokumentasi Penulis

Pilih kembali simpul dengan least cost, lalu lakukan cara yang sama.



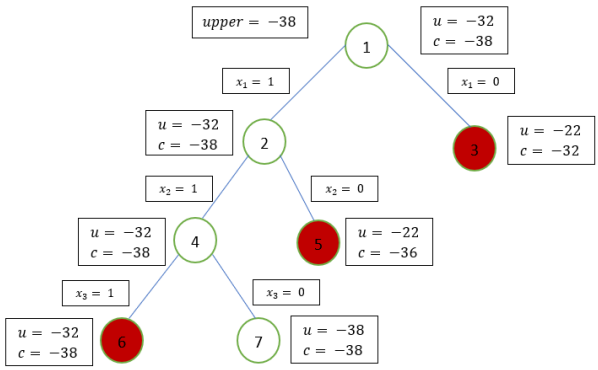
Gambar 3.4. State 3
Sumber: Dokumentasi Penulis

Lihat bahwa pada simpul 7 nilai upper boundnya lebih kecil dari variabel upper, maka variabel upper harus diperbaharui. Setelah diperbaharui maka bandingkan kembali dengan leaf yang masih aktif, jika lower bound dari leaf lebih besar dari variabel upper, maka leaf tersebut dapat diterminasi. Sehingga kondisi tree sekarang menjadi seperti ini karena simpul 3 dan 5 telah diterminasi.



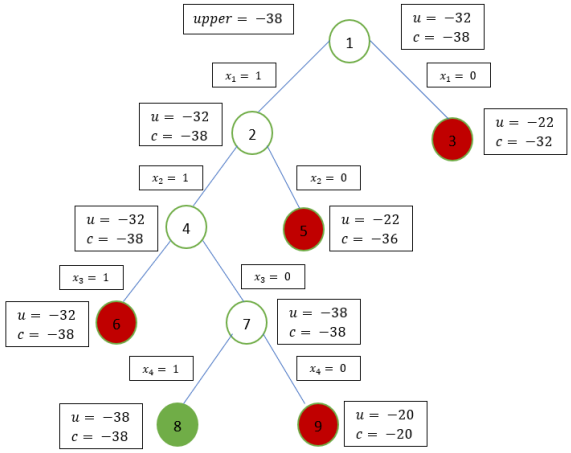
Gambar 3.5. State 3 Setelah Terminasi
Sumber: Dokumentasi Penulis

Ingat juga karena branch-bound merupakan metode pengembangan dari backtrace, maka ketika pada suatu simpul total weightnya telah melewati kapasitas maka simpul tersebut dapat diterminasi saja.



Gambar 3.6. State 4
Sumber: Dokumentasi Penulis

Maka percabangan selanjutnya dilakukan pada simpul 7, lakukan kembali seperti langkah-langkah sebelumnya.



Gambar 3.7. State 5 dan Solusi
Sumber: Dokumentasi Penulis

Karena pada simpul 9 lower boundnya lebih dari variabel upper maka simpul tersebut dapat diterminasi. Dapat diperhatikan juga bahwa tidak ada percabangan lagi yang dapat dilakukan pada simpul manapun, maka solusinya telah ditemukan. Solusinya terdapat pada simpul 8 dan kombinasi dari objek yang dimasukan pada knapsack dapat dicari dengan menelusuri simpul 8 dari root, yaitu

$$x_1 = 1; x_2 = 1; x_3 = 0; x_4 = 1$$
$$x = \{x_1, x_2, x_4\}$$

dengan total value adalah

$$Sum\ value = 10 + 10 + 18 = 38$$

IV. KESIMPULAN

Kombinatorial memiliki banyak jenis-jenis masalah, salah satunya adalah *Knapsack's problem*. *Knapsack's problem* memiliki beberapa jenis dan beberapa metode penyelesaian. Tiap metode tersebut memiliki kelebihan dan kekurangannya masing-masing dan juga bergantung pada tipe *knapsack's problem*. Salah satu metode penyelesaiannya adalah *Branch-Bound* yang merupakan penggabungan dari *decision tree*, *backtracking dan greedy approach*. Dari contoh persoalan yang telah diselesaikan, dapat diperoleh hasil yang maksimal berdasarkan kumpulan objek dan limit kapasitas *knapsack*. Penyelesain dengan metode branch-bound juga tidak terlalu kompleks dan cukup optimal.

V. PENUTUP

Puji syukur penulis panjatkan sebesar-besarnya pada Tuhan Yang Maha Esa, atas karuniaNya yang diberikan kepada penulis sehingga penulis dapat menyelesaikan makalah ini dengan tepat waktu. Penulis juga menyampaikan rasa terima kasih kepada seluruh dosen mata kuliah Matematika Diskrit khususnya Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen untuk kelas Matematika Diskrit K01, yaitu kelas tempat saya berkuliah.

Penulis juga mengucapkan terima kasih kepada kedua orang tua penulis yang selalu mendukung kegiatan perkuliahan penulis. Penulis juga ingin berterima kasih kepada teman-teman yang selalu membantu dan memberikan ide, masukan, dan semangat dalam mengerjakan makalah ini.

REFERENCES

[1] Geeksforgeeks. 2022. *0/1 Knapsack using Branch and Bound*. Diakses pada https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound/ pada tanggal 10 Desember 2022.

[2] Munir, Rinaldi. 2022. Kombinatorial (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kombinatorial-2020-Bagian1.pdf pada tanggal 10 Desember 2022.

[3] Munir, Rinaldi. 2022. Graf (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf pada tanggal 10 Desember 2022.

[4] Munir, Rinaldi. 2022. Pohon (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf pada tanggal 10 Desember 2022.

[5] Munir, Rinaldi. 2022. Pohon (Bagian 2). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf pada 10 Desember 2022.

[6] Geeksforgeeks. 2022. *Implementation of 0/1 Knapsack using Branch and Bound*. Diakses https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/ pada tanggal 12 Desember 2022.

[7] Brilliant. Maltby, Henry. Ross, Eli. 2022. *Combinatorial Optimization.* Diakses pada https://brilliant.org/wiki/combinatorial-optimization/ pada tanggal 10 Desember 2022.

[8] Pencil Prgrammer. 2022. 0-1 *Knapsack Problem using Dynamic Programming.* Diakses pada https://pencilprogrammer.com/algorithms/0-1-knapsack-problem-dynamic-programming/ pada tanggal 12 Desember 2022.

[9] CodeCrucks. 2022. *Knapsack Problem using Branch and Bound*. Diakses pada https://codecrucks.com/knapsack-problem-using-branch-and-bound/ pada tanggal 10 Desember 2022.

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 Desember 2022

Christian Albert Hasiholan
13521078