Penerapan Algoritma Branch and Bound dalam Pemilihan Makanan untuk Pencegahan Diabetes

Pendekatan Masalah Knapsack 0/1

Shabrina Maharani - 13522134
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13522134@std.stei.itb.ac.id

***Abstrak*— Diabetes melitus adalah penyakit kronis yang disebabkan oleh tingginya kadar gula dalam darah akibat gangguan pada produksi atau fungsi insulin, dengan pola makan tidak sehat sebagai faktor utama. Penyakit diabetes dapat menyebabkan komplikasi penyakit pada pengidapnya. Oleh karena itu, penting untuk memilih makanan yang dapat memenuhi kebutuhan nutrisi harian tanpa meningkatkan kadar gula darah secara signifikan. Pemilihan makanan yang tepat dapat dilakukan dengan memanfaatkan algoritma Branch and Bound dengan pendekatan *Knapsack* 0/1. Algoritma ini memungkinkan pemilihan kombinasi makanan optimal berdasarkan batasan gula dan karbohidrat harian, sehingga kebutuhan gula tubuh tercukupi tanpa berlebihan. Penggunaan algoritma Branch and Bound dalam pemilihan makanan dapat membantu menjaga kadar gula darah tetap stabil dan mencegah diabetes.**

Kata Kunci—Diabetes; *Branch and Bound*; *Knapsack* 0/1 Problem

I. PENDAHULUAN

Diabetes melitus merupakan penyakit dengan kondisi seseorang mengalami gangguan dalam memproduksi atau memanfaatkan hormon insulin. Hormon insulin dan glukagon adalah dua hormon yang saling bekerja sama untuk menjaga kadar glukosa dan asam lemak dalam darah. Hormon insulin berperan untuk menurunkan kadar gula dalam darah dengan membantu sel-sel tubuh menyerap glukosa dari darah untuk digunakan sebagai energi atau disimpan sebagai glikogen di hati dan otot, sehingga kadar gula darah menurun setelah makan. Sebaliknya, hormon glukagon berperan meningkatkan kadar glukosa dalam darah dengan memicu pemecahan glikogen yang disimpan di hati menjadi glukosa. Hormon tersebut dilepaskan ke dalam darah ketika kadar glukosa darah rendah seperti saat berpuasa atau antara waktu makan. [1]

Diabetes melitus terbagi menjadi diabetes tipe 1 dan tipe 2. Diabetes tipe 1, juga dikenal sebagai diabetes yang tergantung insulin atau diabetes yang muncul pada masa kanak-kanak, ditandai dengan kurangnya produksi insulin karena sel beta pankreas tidak menghasilkan insulin. Penderita diabetes tipe 1 memerlukan suntikan insulin dari luar untuk bertahan hidup. Sedangkan, untuk diabetes tipe 2, atau dapat disebut sebagai diabetes yang tidak tergantung insulin ini ditandai dengan penurunan sensitivitas tubuh terhadap hormon insulin ketika kondisi sekresi insulin normal atau mengalami peningkatan, sehingga kadar gula dalam darah tetap tinggi bahkan setelah makan. Diabetes tipe 2 sering dikaitkan dengan obesitas dan sindrom metabolik, yang mencakup sejumlah faktor risiko seperti lingkar pinggang yang besar, kadar trigliserida yang tinggi, tingkat kolesterol baik yang rendah, tekanan darah tinggi, dan kadar glukosa darah yang tinggi. [1]

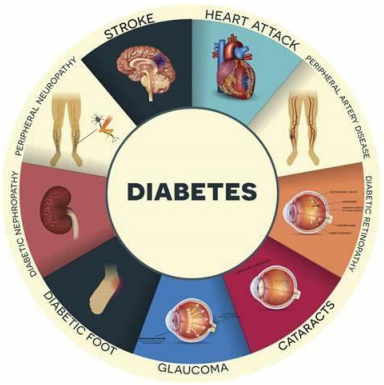

Fig. 1. Komplikasi akibat Diabetes

Sumber : https://th.bing.com/th/id/OIP.BxiTgG_ew7qzPnFIBhScLAHaHa?rs=1&pid=ImgDetMain

Berdasarkan data yang diperoleh dari International Diabetes Federation, diperkirakan akan terjadi peningkatan pengidap diabetes di Indonesia pada tahun 2030 sekitar 23.328.000 orang. [2] Selain itu, survei yang dilakukan oleh organisasi kesehatan dunia juga memperkirakan jumlah penderita diabetes melitus di Indonesia akan meningkat pada tahun 2030 yaitu sebesar 21,3 juta orang. [3] Angka tersebut sangat mengkhawatirkan karena jika tidak ditangani dengan baik, diabetes melitus dapat berujung pada kematian akibat komplikasi yang ditimbulkannya. World Health Organization (Organisasi Kesehatan Dunia) juga mengonfirmasi bahwa diabetes berisiko menjadi penyebab utama kebutaan, gagal ginjal, serangan jantung, stroke, dan amputasi anggota tubuh bagian bawah. [4] Pola makan yang sehat, aktivitas fisik yang

teratur, menjaga berat badan normal, dan menghindari penggunaan tembakau adalah cara untuk mencegah atau menunda timbulnya diabetes tipe 2. Pengelolaan diabetes tipe 1 melibatkan suntikan insulin secara teratur, pengaturan pola makan, dan olahraga. Sebaliknya, pengelolaan diabetes tipe 2 sering kali melibatkan perubahan gaya hidup seperti penurunan berat badan, pengaturan pola makan, olahraga, serta penggunaan obat penurun glukosa darah jika diperlukan.

Langkah awal untuk mencegah diabetes melitus dapat dimulai dengan mengatur pola makan. Pemilihan makanan dengan kadar gula rendah serta kandungan protein tinggi dapat membantu menjaga tubuh dalam keadaan normal. Menurut sebuah artikel yang ditulis oleh Courtney Southwick, seorang ahli di bidang kesehatan, makanan dengan protein yang baik dapat membantu dalam menjaga keseimbangan dalam darah dan membuat tubuh merasa kenyang lebih lama, sehingga mengurangi kemungkinan makan berlebihan yang dapat menyebabkan kelebihan kadar gula.[5] Karbohidrat yang kita konsumsi akan dipecah menjadi glukosa yang digunakan sebagai energi, tetapi jika tidak digunakan, glukosa tersebut akan menumpuk dan meningkatkan kadar gula dalam darah, yang dapat merusak sistem hormon dalam tubuh. Oleh karena itu, memilih makanan rendah gula, dengan kandungan karbohidrat yang cukup, serta tinggi protein namun tetap memuaskan untuk dikonsumsi, dapat menjadi pertimbangan penting dalam pemilihan makanan. Algoritma *branch and bound* dapat dimanfaatkan untuk memilih makanan sesuai kapasitas gula dan karbohidrat, dengan mempertimbangkan kadar protein dan kepuasan yang optimal.

II. Landasan Teori

A. *Knapsack Problem*

Knapsack Problem merupakan contoh dari masalah optimasi kombinatorial. Problem ini juga biasa dikenal sebagai *Rucksack Problem*. *Knapsack Problem* dipopulerkan pertama kali oleh seorang ahli matematika Bernama Tobias Dantzig pada tahun 1897. *Knapsack Problem* sejatinya adalah persoalan tentang maksimasi dengan mengutamakan nilai keuntungan yang optimal dan jumlah objek yang banyak tetapi tetap tidak melebihi kapasitas maksimum nya. [6]

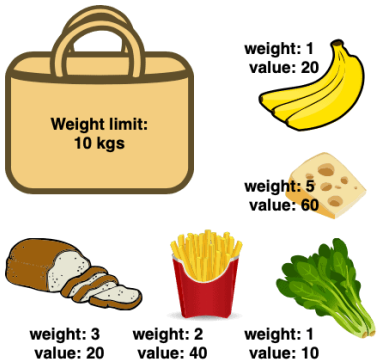

Fig. 2. Knapsack Problem

Sumber : https://copyprogramming.com/code/time-complexity-of-0-1-knapsack-using-dynamic-programming-code-example (diakses pada 10 Juni 2024 pukul 04.16 WIB)

Misalkan, terdapat n buah objek dengan sebuah knapsack yang memiliki kapasitas bobot *K*. Setiap objek memiliki bobot (w_i) dan *profit* atau keuntungan (p_i). *Knapsack Problem* dapat diselesaikan dengan memilih objek-objek yang akan dimasukkan ke dalam sebuah knapsack sehingga didapatkan total keuntungan yang maksimal dengan bobot yang tidak melebihi kapasitas. [7] Berikut adalah bentuk matematis dari ilustrasi sebuah *knapsack problem*.

Maksimasi $F = \sum_{i=1}^{n} p_i x_i$

dengan kendala (*constraint*)

$\sum_{i=1}^{n} w_i x_i \le K$

yang dalam hal ini, $x_i = 0$ atau $1, \quad i = 1, 2, \ldots, n$

Fig. 3. Formulasi Matematis dari Knapsack Problem

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Branchand-Bound-2022-Bagian4.pdf (diakses pada 10 Juni 2024 pukul 06.14 WIB)

Knapsack Problem terbagi menjadi empat klasifikasi diantaranya, yaitu *Fractional Knapsack Problem*, *0/1 Knapsack Problem*, *Bounded Knapsack Problem*, *dan Unbounded Knapsack Problem*. Berikut adalah beberapa penjelasan singkat terkait empat klasifikasi tersebut.

1. Fractional Knapsack Problem

   Fractional Knapsack Problem memiliki konsep yang hampir mirip dengan konsep *Knapsack Problem* pada umumnya, tetapi item-item dalam Fractional *Knapsack Problem* dapat dipecah agar dapat memaksimalkan nilai total yang diperoleh. Dengan kata lain, jika ada item yang tidak dapat dimasukkan sepenuhnya karena keterbatasan kapasitas, kita dapat memasukkan sebagian dari item tersebut untuk tetap menambah nilai total dalam *knapsack*. Pendekatan ini memberikan fleksibilitas lebih dalam optimisasi dan sering digunakan dalam situasi di mana item dapat dibagi tanpa kehilangan nilai proporsionalnya. Berikut adalah bentuk matematis dari *Fractional Knapsack Problem*.

Maksimasi $F = \sum_{i=1}^{n} p_i x_i$

dengan kendala (*constraint*)

$\sum_{i=1}^{n} w_i x_i \le K$

yang dalam hal ini, $0 \le x_i \le 1, \quad i = 1, 2, \ldots, n$

Fig. 4. Formulasi Matematis Fractional Knapsack Problem

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf (diakses pada 10 Juni 2024 pukul 06.43 WIB)

Properti objek				*Greedy by*		
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*
1	18	25	1,4	1	0	0
2	15	24	1,6	2/15	2/3	1
3	10	15	1,5	0	1	1/2
Total bobot				20	20	20
Total keuntungan				28,2	31,0	31,5

Fig. 5. Contoh Penyelesaian Fractional Knapsack Problem

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf (diakses pada 10 Juni 2024 pukul 06.43 WIB)

2. 0/1 Knapsack Problem

Dalam 0/1 *Knapsack Problem*, item tidak dapat dimasukkan secara sebagian. Hanya terdapat dua pilihan yaitu memasukkan seluruhnya untuk suatu item tertentu (1) atau tidak memasukkan sama sekali (0).

Properti objek				*Greedy by*			Solusi
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	Optimal
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

Fig. 6. Contoh Penyelesaian 0/1 Knapsack Problem

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf (diakses pada 10 Juni 2024 pukul 06.52 WIB)

Masalah *Knapsack* 0/1 dapat diselesaikan tidak hanya menggunakan algoritma *Branch and Bound*, tetapi juga beberapa algoritma lainnya, antara lain sebagai berikut:

a. Algoritma *Brute Force*, yang memeriksa setiap kemungkinan kombinasi satu per satu dengan kompleksitas waktu sebesar $O(2^n)$.

b. Algoritma *Greedy*, yang memilih solusi optimum lokal pada setiap langkahnya dengan ekspetasi mendapatkan solusi optimum global pada akhir penyelesaian. Namun, algoritma ini tidak selalu memberikan solusi optimum global karena total dari solusi optimum lokal belum tentu menghasilkan solusi global yang optimum. Kompleksitas algoritma *greedy* ini adalah $O(n^2)$.

c. Algoritma *Dynamic Programming*, yang memecah masalah menjadi beberapa tahap, memungkinkan solusi dianggap sebagai rangkaian keputusan yang saling terkait. Solusi akhir dibentuk dari solusi-solusi sebelumnya, sehingga memberikan hasil yang optimal.

3. Bounded Knapsack Problem

Bounded Knapsack adalah varian dari masalah knapsack di mana setiap item memiliki batasan jumlah tertentu yang dapat dimasukkan ke dalam tas. Perbedaannya dengan masalah knapsack 0/1 adalah bahwa setiap jenis item dapat dipilih lebih dari satu kali, namun dibatasi oleh jumlah maksimum yang ditentukan untuk masing-masing item.

4. Unbounded Knapsack Problem

Unbounded Knapsack adalah varian dari masalah knapsack di mana tidak ada batasan jumlah untuk setiap item yang dapat dimasukkan ke dalam tas. Artinya, dapat memasukkan suatu item ke dalam tas sebanyak yang diinginkan, asalkan total bobot item-item tersebut tidak melebihi kapasitas tas. Tujuan dari masalah ini adalah untuk memaksimalkan nilai total item yang dimasukkan ke dalam tas. Dalam masalah ini, karena tidak ada batasan jumlah, solusi sering kali melibatkan memasukkan item dengan rasio nilai terhadap bobot tertinggi sebanyak mungkin untuk mencapai nilai maksimum.

B. *Struktur Data Pohon*

Pohon dapat diartikan sebagai graf tidak berarah yang terhubung setiap simpulnya dan tidak mengandung sirkuit. Kumpulan pohon yang tidak saling terkait atau saling lepas dapat disebut dengan hutan. Pohon memiliki beberapa sifat atau properti yang dapat menjadi definisi dari pohon itu sendiri. Misalkan terdapat sebuah graf G dengan simpul *V* dan sisi *E*. Graf G merupakan graf yang tak berarah sederhana dengan jumlah simpul *n*. Maka, pernyataan di bawah ini dapat sepadan :

1. G merupakan pohon.
2. Setiap simpul berpasangan di dalam G akan terhubung dengan sebuah lintasan tunggal.
3. G terhubung antar simpulnya dan memiliki *n-1* buah sisi *E*.
4. G tidak mengandung sirkuit dan jika ditambahkan satu sisi tambahan pada graf G hanya akan membentuk satu sirkuit.
5. Semua sisi *E* pada G adalah jembatan.

Jika sebuah graf memenuhi kelima pernyataan di atas, graf tersebut dapat dipastikan adalah sebuah pohon. [8] Ruang solusi untuk algoritma seperti algoritma runut balik, *branch and bound*, algoritma-algoritma *route planning*, dan *dynamic programming* direpresentasikan ke dalam sebuah struktur pohon berakar yaitu pohon yang memiliki satu buah simpul yang diperlakukan sebagai akar. Tiap simpul pada pohon tersebut akan merepresentasikan status persoalan yang sedang dihadapi, sedangkan sisi *E* akan diberi label berupa nilai-nilai x_i. Jalur yang melintang dari akar ke daun dapat merepresentasikan kandidat solusi dan berkemungkinan membentuk ruang solusi. Pembentukan pohon ruang solusi ini dapat disebut juga dengan pohon ruang status (*state space tree)*. [9] Selain itu, algoritma-algoritma tersebut membentuk pohon secara dinamis di mana proses konstruksinya dan pemeliharaan struktur data pohon yang dapat berubah-ubah seiring waktu, tergantung pada operasi yang dilakukan. Dalam konteks algoritma dan struktur data, pohon dinamis memungkinkan penambahan, penghapusan, atau modifikasi simpul (*nodes*)

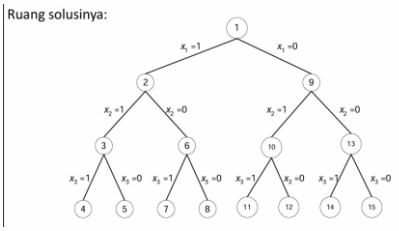

Fig. 7. Contoh Pembentukan Ruang Solusi dengan Struktur Pohon

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf (diakses pada 10 Juni 07.49 WIB)

C. Algoritma Branch and Bound

Algoritma *Branch and Bound* (B&B) pertama kali ditemukan pada tahun 1960 dan dipopulerkan oleh A.H. Land dan A. G. Doig. Algoritma ini digunakan untuk menyelesaikan masalah optimisisasi dengan tujuan meminimalkan ataupun memaksimalkan fungsi objektif tanpa melanggar batasan yang ada dalam persoalan. Metode ini menggabungkan pencarian menggunakan algoritma BFS dengan pencarian berdasarkan biaya terendah (least cost search), di mana simpul berikutnya yang akan dieksplorasi adalah simpul dengan biaya terendah. Pada algoritma *Branch and Bound*, pembangkitan simpul nya didasari oleh beberapa aturan tertentu. Selain itu, setiap simpul diberi nilai biaya atau biasa disebut *cost* (ĉ(i)), yang merupakan taksiran jalur dengan biaya terendah dari simpul tersebut ke tujuan. Simpul yang akan dieksplorasi selanjutnya tidak dipilih berdasarkan urutan pembangkitan, tetapi berdasarkan simpul dengan biaya / *cost* paling kecil. B&B juga menggunakan teknik pemangkasan untuk mengeliminasi jalur yang tidak mungkin menghasilkan solusi optimal. Pemangkasan dilakukan jika nilai simpul tidak lebih baik dari solusi terbaik yang ditemukan sejauh ini, jika simpul tersebut melanggar batasan yang ada, atau jika simpul tersebut tidak memiliki pilihan lain yang lebih baik. Dengan pendekatan ini, B&B mampu mengeksplorasi ruang solusi secara efektif dan efisien, memastikan hanya jalur yang berpotensi menghasilkan solusi optimal yang dieksplorasi lebih lanjut.

Tujuan utama algoritma *branch and bound* adalah menemukan nilai x sehingga fungsi *F(x)* dapat menghasilkan nilai maksimal atau nilai minimalnya dari hasil perhitungan. Untuk mencapai tujuan ini, algoritma ini menggunakan dua prinsip utama, yaitu melakukan rekursi untuk memecah ruang solusi menjadi bagian-bagian lebih kecil yang dikenal dengan istilah percabangan (*branch)*, dan mencari fungsi batasan atau estimasi minimum, kemudian melakukan percabangan pada simpul tersebut (*bound)*. Tanpa prinsip kedua, *branch and bound* akan serupa dengan algoritma *brute force* atau pencarian menyeluruh. Algoritma *Branch and Bound* dapat digunakan untuk menyelesaikan permasalahan seperti 0/1 *Knapsack Problem*, Persoalan N-Ratu, Permainan 15-Puzzle, dan sebagainya.

D. Pemanfaatan Algoritma Branch and Bound pada Masalah Knapsack 0/1

Berbeda dengan pendekatan *least cost search* yang *cost* atau biaya setiap simpulnya merepresentasikan batas bawah (lower bound), pada pemanfaatan algoritma *Branch and Bound* untuk menyelesaikan 0/1 *Knapsack Problem*, *cost* atau biaya dari masing-masing simpul merepresentasikan batas atas (*upper bound*) dari solusi yang optimum. Pada persoalan yang berkaitan dengan maksimasi, simpul yang akan diekspansi selanjutnya adalah simpul yang hidup dan memiliki *cost* yang paling besar. Untuk membuat pencarian solusi yang lebih efektif, objek-objek akan diurutkan berdasarkan p_i/w_i dari mulai yang paling besar sampai yang terkecil ($p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$). [4]

Dalam algoritma *branch and bound*, ruang status diwakili oleh struktur pohon biner. Setiap cabang kiri menyatakan bahwa objek *i* dipilih ($x_i = 1$), sedangkan cabang yang berada di kanan menyatakan bahwa objek *i* tidak dipilih ($x_i = 0$). Setiap simpul pada level *i* tertentu dalam pohon biner tersebut, menggambarkan himpunan bagian dari *n* objek yang dimasukkan ke dalam *knapsack*, yang dipilih dari *i* objek pertama. Setiap simpul akan diisi dengan total bobot *knapsack* yang sudah terpakai yang dinotasikan dengan huruf *W* dan total keuntungan yang telah dicapai yang dinotasikan dengan huruf *F*.

Biaya (*cost)* atau batas atas (upper bound) dari setiap simpul dihitung sebagai penjumlahan total keuntungan yang telah dicapai (*F*) ditambah dengan hasil kali antara sisa kapasitas maksimum *knapsack* (*K – W)* dengan rasio keuntungan per bobot dari objek yang tersisa berikutnya (p_{i+1}/w_{i+1}).

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

Fig. 8. Rumus *Cost* atau Biaya dari 0/1 Knapsack Problem dengan Algoritma Branch and Bound

Sumber : Algoritma-Branchand-Bound-2022-Bagian4.pdf (itb.ac.id) (diakses pada 10 Juni 08.48 WIB)

E. Kompleksitas Algoritma

Kompleksitas algoritma adalah faktor penting dalam menentukan makanan yang optimal. Saat mencari solusi terbaik untuk pemilihan makanan, penting untuk mempertimbangkan kompleksitas waktu dan ruang dari algoritma yang digunakan. Kompleksitas waktu mengukur efisiensi algoritma dalam menyelesaikan masalah berdasarkan ukuran inputnya, sedangkan kompleksitas ruang mengukur jumlah memori yang diperlukan oleh algoritma saat berjalan.

Dalam pemilihan makanan, kompleksitas waktu sangat penting karena efisiensi waktu komputasi diperlukan untuk mempercepat pencarian solusi makanan yang optimal. Masalah pemilihan makanan seringkali rumit dan memiliki banyak kombinasi, terutama jika melibatkan berbagai nutrisi dan preferensi individu. Oleh karena itu, algoritma dengan kompleksitas waktu yang tinggi dapat menghambat efisiensi pencarian solusi optimal. Algoritma dengan kompleksitas

waktu yang lebih rendah, seperti algoritma *branch and bound* yang dioptimalkan, dapat menjadi pilihan yang baik untuk mencapai solusi dalam waktu yang wajar.

Kompleksitas algoritma *branch and bound* dapat bervariasi tergantung pada masalah yang dihadapi. Secara umum, algoritma *branch and bound* memiliki kompleksitas waktu yang dapat disesuaikan dengan strategi pemotongan cabang yang digunakan. Biasanya, kompleksitas waktu algoritma ini bergantung pada jumlah *node* yang harus diproses dan dapat dioptimalkan untuk kasus-kasus tertentu. Selain itu, kompleksitas ruang juga perlu diperhatikan dalam pemilihan makanan. Memori yang cukup besar mungkin diperlukan saat algoritma bekerja dengan dataset besar atau melakukan operasi perhitungan yang kompleks. Dalam beberapa kasus, algoritma yang membutuhkan memori besar mungkin tidak efisien dalam penggunaan sumber daya.

Faktor lain seperti tingkat keakuratan dan kualitas solusi yang dihasilkan oleh algoritma perlu dipertimbangkan. Dalam pemilihan makanan, tujuan utamanya adalah mencapai solusi yang optimal dengan mempertimbangkan kebutuhan nutrisi harian, batasan gula, dan karbohidrat. Oleh karena itu, algoritma dengan kompleksitas waktu dan ruang yang sedang namun mampu memberikan solusi makanan yang optimal dan berkualitas tinggi menjadi pilihan yang lebih baik. Memilih algoritma yang sesuai dan menerapkan teknik optimisasi dengan teliti akan sangat berpengaruh dalam mendapatkan pilihan makanan yang efisien, memenuhi kebutuhan gizi, dan mengoptimalkan kesehatan.

III. Implementasi Algoritma Branch and Bound

A. *Langkah Penyelesaian 0/1 Knapsack Problem dengan Algoritma Branch and Bound*

Misalkan, diberikan 3 makanan diantaranya, yaitu rendang, bubur ayam, dan sup ayam dengan kadar karbohidrat, gula, protein, dan tingkat kepuasan sebagai berikut.

TABLE I. KADAR DALAM MAKANAN

i	Nama Makanan	Karbohidrat (gram)	Gula (gram)	Protein (gram)	Tingkat Kepuasan (1-5)
1	Rendang	24.0	1.5	30.0	5.0
2	Bubur Ayam	47.7	0.3	34.5	4.5
3	Sup Ayam	30.3	0.9	12.3	3.5

Kemudian, kapasitas gula dan karbohidrat seseorang dengan pilihan makanan tersebut adalah 80. Berikut adalah langkah dasar penyelesaian 0/1 *Knapsack Problem* dengan menggunakan algoritma *Branch and Bound*:

1. Menghitung p_i / w_i *(density)* untuk setiap objek

TABLE II. PERHITUNGAN *DENSITY* UNTUK SETIAP OBJEK

i	p_i (protein + tingkat kepuasan)	w_i (karbo + surga)	p_i/w_i
1	35.0	25.5	1.3725490196
2	39.0	48.0	0.8125
3	15.8	31.2	0.5064102564

2. Mengurutkan objek-objek berdasarkan p_i / w_i *(density)* dari yang terbesar ke yang terkecil. Kebetulan pada tabel di atas sudah terurut secara menurun.

3. Membangkitkan simpul *root* (simpul 0), dengan $W = 0$ dan $F = 0$ karena belum ada objek yang dipilih dan menghitung biaya atau *cost* untuk simpul 0 dengan $K = 80$.

$\hat{c}(0) = F + (K - W)\ p_1 / w_1 = 0 + (80 - 0)\ (1.3725490196) = 109.8039216$

4. Membangkitkan *left child node* dan *right child node* dari simpul akar

- Simpul 1 (makanan rendang dipilih) dengan W = 0 + 25.5 = 25.5 dan F = 0 + 35.0 = 35.0

  $\hat{c}(1) = F + (K - W)\ p_2 / w_2 = 35.0 + (80 - 25.5)\ (0.8125) = 79.28125$

- Simpul 2 (makanan rendang tidak dipilih) dengan W = 0 dan F = 0

  $\hat{c}(2) = F + (K - W)\ p_2 / w_2 = 0 + (80 - 0)\ (0.8125) = 65$

  Karena simpul 1 menghasilkan *cost* paling besar dan bobot nya belum melebihi kapasitas, untuk mendapatkan keuntungan yang optimum, maka simpul 1 selanjutnya akan di ekspansi.

5. Membangkitkan *left child node* dan *right child node* dari simpul 1

- Simpul 3 (makanan bubur ayam dipilih) dengan W = 25.5 + 48.0 = 73.5 dan F = 35.0 + 39.0 = 74.0

  $\hat{c}(3) = F + (K - W)\ p_3 / w_3 = 74 + (80 - 73.5)\ (0.5064102564) = 77.29166667$

- Simpul 4 (makanan bubur ayam tidak dipilih) dengan W = 25.5 dan F = 35.0

  $\hat{c}(4) = F + (K - W)\ p_3 / w_3 = 35.0 + (80 - 25.5) = 62.59935897$

  Karena simpul 3 menghasilkan *cost* paling besar, untuk mendapatkan keuntungan yang optimum dan bobot nya belum melebihi kapasitas, maka simpul 3 selanjutnya akan di ekspansi.

6. Membangkitkan *left child node* dan *right child node* dari simpul 4

- Simpul 5 (makanan sup ayam dipilih) dengan W = 73.5 + 31.2 = 104.7 dan F = 74.0 + 15.8 = 89.8. Bobot W sudah melebihi kapasitas K. Simpul 5 langsung dimatikan

- Simpul 6 (makanan sup ayam tidak dipilih) dengan W = 73.5 dan F = 74.0

  $\hat{c}(6) = F + (K - W)\ p_4 / w_4 = 74.0 + (80 - 73.5)\ (0) = 74.0$

Karena simpul 5 sudah melebihi kapasitas K, maka simpul 6 yang dipilih dan menjadi simpul solusi (*goal node)*. Semua simpul hidup yang *cost*-nya lebih kecil dari 74 dibunuh, yaitu simpul 2 dan simpul 4.

Jadi, solusi optimal dari persoalan tersebut adalah X = (1,1,0), F = 74.0, yang berarti dengan memilih rendang dan bubur ayam, pengguna bisa mendapat protein dan tingkat kepuasan yang lebih tinggi dengan kadar karbohidrat dan gula yang lebih rendah.

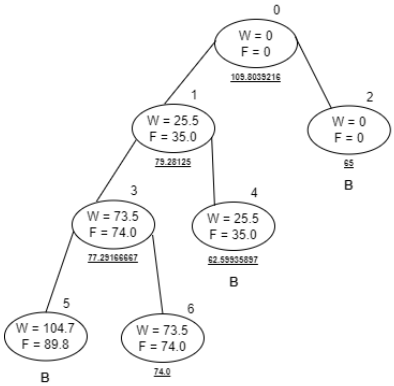

Fig. 9. Pohon Solusi yang Terbentuk dari Penyelesaian Persoalan

Sumber : Dokumentasi Penulis

B. *Implementasi Kode*

Berikut ini merupakan contoh implementasi algoritma *branch and bound* untuk pemilihan makanan dalam bentuk program menggunakan bahasa Python.

```
import heapq

class FoodItem:
    def __init__(self, name, weight, value):
        self.name = name
        self.weight = weight
        self.value = value

def knapsack_01_branch_and_bound(items, max_weight):
    items.sort(key=lambda x: x.value / x.weight,
reverse=True)

    def bound(i, current_weight, current_value):
        if current_weight >= max_weight:
            return 0
        bound_value = current_value
        total_weight = current_weight
        while i < len(items) and total_weight +
items[i].weight <= max_weight:
            total_weight += items[i].weight
            bound_value += items[i].value
            i += 1
        if i < len(items):
            bound_value += (max_weight - total_weight) *
(items[i].value / items[i].weight)
        return bound_value

    n = len(items)
    best_value = 0
    best_items = []

    pq = []
    heapq.heappush(pq, (-bound(0, 0, 0), 0, 0, 0, []))

    while pq:
        _, i, current_weight, current_value,
selected_items = heapq.heappop(pq)

        if current_value > best_value and current_weight
<= max_weight:
            best_value = current_value
            best_items = selected_items

        if i < n:
            if current_weight + items[i].weight <=
max_weight:
                heapq.heappush(pq, (-bound(i + 1,
current_weight + items[i].weight, current_value +
items[i].value),
                i + 1, current_weight + items[i].weight,
current_value + items[i].value, selected_items +
[items[i]]))
            heapq.heappush(pq, (-bound(i + 1,
current_weight, current_value), i + 1, current_weight,
current_value, selected_items))

    return best_value, best_items
```

Program berjalan dengan langkah sebagai berikut :

1. Program dimulai dengan menginisialisasi objek dengan cara mendefinisikan kelas FoodItem yang memiliki atribut name, weight, dan value.

2. Terdapat fungsi utama yang Bernama knapsack_01_branch_and_bound yang menerima dua

parameter, yaitu items (objeks FoodItem) dan max_weight (kapasitas maksimum knapsack).

3. Program akan mengurutkan item berdasarkan rasio nilai per berat (value / weight) secara menurun. Hal ini dilakukan karena item dengan rasio nilai per berat yang lebih tinggi biasanya lebih menguntungkan untuk dipilih.

4. Terdapat fungsi bound dalam fungsi utama knapsack_01_branch_and_bound. Fungsi bound digunakan untuk menghitung batas atas (upper bound) dari nilai total yang dapat dicapai dari kondisi saat ini. Ini membantu dalam memutuskan apakah cabang tertentu layak untuk diekspansi lebih lanjut. Jika berat saat ini (current_weight) melebihi kapasitas maksimum (max_weight), batas atas akan diatur menjadi 0. Batas atas dihitung dengan menambahkan nilai item secara penuh hingga kapasitas knapsack tercapai dan menambahkan fraksi nilai dari item berikutnya jika knapsack tidak penuh.

5. Program selanjutnya akan menginisialisasi *priority queue*. Sebuah priority queue (antrian prioritas) diinisialisasi menggunakan heapq dengan elemen pertama berupa tuple yang berisi nilai negatif dari batas atas (karena heapq adalah min-heap), indeks saat ini, berat saat ini, nilai saat ini, dan daftar item yang dipilih.

6. Kemudian, program akan melakukan proses *branch and bound*. Program melakukan iterasi melalui priority queue hingga kosong. Elemen teratas dari antrian akan diambil, yang mewakili *node* saat ini dalam struktur pohon. Jika nilai saat ini lebih besar dari nilai terbaik (best_value) dan berat saat ini tidak melebihi kapasitas, nilai terbaik diperbarui dan daftar item terbaik (best_items) disimpan. Jika masih ada item yang tersisa untuk dipertimbangkan, program membuat dua cabang :

- Cabang (*Branch*) Memilih Item: Menambahkan item saat ini ke knapsack dan memperbarui berat serta nilai saat ini. Batas atas dihitung untuk *child node* ini, dan *node* ini ditambahkan ke priority queue jika layak.
- Cabang Tidak Memilih Item: Melewatkan item saat ini dan hanya memperbarui indeks. Batas atas dihitung untuk *child node* ini, dan node ini ditambahkan ke priority queue.

7. Setelah *priority queue* kosong, nilai terbaik (best_value) dan daftar item terbaik (best_items) dikembalikan sebagai hasil akhir.

C. Hasil Pengujian

Diberikan contoh kombinasi pemilihan makanan yang melibatkan tiga jenis makanan: rendang, bubur ayam, dan sup ayam. Pengguna ingin memilih antara rendang sebanyak 300 gram, bubur ayam sebanyak 300 gram, dan sup ayam sebanyak 711 gram. Data mengenai kadar karbohidrat, gula, protein, dan tingkat kepuasan untuk ketiga makanan tersebut sudah tersedia dalam database. Kapasitas maksimum karbohidrat dan gula yang dapat pengguna konsumsi adalah 80 gram. Berikut adalah hasil pengujian yang didapatkan dari hasil program.

```
Berapa banyak pilihan makanan yang ingin Anda makan hari ini? 3
Masukkan nama makanan: Rendang
Masukkan jumlah yang ingin Anda makan : 300
Masukkan nama makanan: Bubur Ayam
Masukkan jumlah yang ingin Anda makan : 300
Masukkan nama makanan: Sup Ayam
Masukkan jumlah yang ingin Anda makan : 711
Masukkan kapasitas gula + karbohidrat harian Anda: 80
Total Nilai Keuntungan (Protein + Tingkat Kepuasan) Optimal: 74.0
Makanan yang terpilih:
Rendang
Bubur Ayam
```

Fig. 10. Hasil Pengujian

Sumber : Dokumentasi Penulis

Berdasarkan hasil di atas, didapatkan bahwa makanan yang terpilih untuk diambil adalah rendang dengan bobot 25.5 gram dan bubur ayam dengan bobot 48.0. Hasil ini sesuai dengan hasil yang didapatkan dari perhitungan manual dengan keuntungan optimal yang didapatkan sebesar 74 dan bobot sebesar 73.5 yang tidak melebihi kapasitas $K = 80$.

D. Analisis Kompleksitas Program

Kompleksitas algoritma *branch and bound* pada penyelesaian 0/1 *Knapsack Problem* dapat dianalisis berdasarkan dua aspek utama, yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu mengacu pada jumlah operasi yang dilakukan algoritma dalam menyelesaikan masalah berdasarkan ukuran input, sedangkan kompleksitas ruang mengacu pada jumlah memori yang diperlukan oleh algoritma selama eksekusi.

Kompleksitas waktu algoritma *branch and bound* dipengaruhi oleh jumlah *node* yang harus dieksplorasi dalam pohon ruang status. Pada kasus terburuk, algoritma ini harus memeriksa semua kemungkinan subset dari n item yang ada, sehingga kompleksitas waktu terburuknya adalah $O(2^n)$. Namun, dalam praktiknya, algoritma *branch and bound* seringkali lebih efisien karena menggunakan teknik pemangkasan (*pruning*) untuk mengeliminasi cabang-cabang yang tidak perlu dieksplorasi, sehingga mengurangi jumlah *node* yang harus dieksplorasi.

Pada implementasi algoritma yang telah dibuat dalam program, bound dihitung untuk setiap *node* yang dieksplorasi, dan heap digunakan untuk menyimpan *node-node* tersebut berdasarkan nilai bound-nya. Operasi pada heap seperti penambahan dan penghapusan elemen memiliki kompleksitas O(log n). Oleh karena itu, kompleksitas waktu rata-rata dari algoritma *branch and bound* ini bisa lebih baik dari $O(2^n)$ dalam banyak kasus, terutama jika heuristik yang digunakan untuk menghitung bound efektif dalam mengurangi jumlah cabang yang dieksplorasi.

Kompleksitas ruang algoritma *branch and bound* sangat ditentukan oleh struktur data yang digunakan untuk menyimpan *node* yang akan dieksplorasi. Dalam hal ini, heap digunakan untuk mengelola *node-node* tersebut. Pada skenario terburuk, heap bisa berisi hingga $O(2^n)$ *node*, yang menunjukkan bahwa kompleksitas ruang terburuknya juga sebesar $O(2^n)$. Namun, sama halnya dengan kompleksitas waktu, penggunaan teknik pemangkasan (*pruning)* seringkali

mengurangi jumlah *node* yang disimpan dalam heap pada saat tertentu, sehingga dalam praktik penggunaannya, kompleksitas ruang cenderung lebih rendah dari $O(2^n)$.

Dalam banyak contoh aplikasi penerapan algoritma ini, *branch and bound* dapat menyelesaikan 0/1 *Knapsack Problem* dengan efisiensi yang jauh lebih baik daripada metode *brute force*. Hal ini terjadi karena teknik pemangkasan (*pruning)* yang efektif yang mengurangi eksplorasi ruang solusi. Meskipun kompleksitas terburuknya secara teoritis adalah eksponensial, dalam banyak kasus nyata, algoritma ini mampu menghasilkan solusi dalam waktu yang lebih singkat dan dengan penggunaan memori yang lebih sedikit dibandingkan dengan pendekatan *brute force*.

IV. Kesimpulan dan Saran

Dalam makalah yang telah dibuat, penulis telah menunjukkan bahwa algoritma *Branch and Bound* dapat digunakan secara efektif untuk memilih kombinasi makanan yang optimal dalam konteks pencegahan diabetes. Dengan menggunakan pendekatan *Knapsack* 0/1, algoritma ini mampu menentukan pilihan makanan yang memenuhi kebutuhan nutrisi harian tanpa melebihi batasan gula dan karbohidrat yang ditetapkan. Hal ini penting dalam menjaga kadar gula darah tetap stabil dan mencegah komplikasi yang disebabkan oleh diabetes.

Untuk pengembangan lebih lanjut, penulis menyarankan agar program ini dilengkapi dengan visualisasi antarmuka untuk meningkatkan interaksi pengguna dan membuat pengguna lebih semangat dalam memilih makanan yang sehat. Selain itu, menambahkan lebih banyak variasi makanan ke dalam database dapat membantu dalam menguji efektivitas algoritma secara lebih luas dan memberikan pilihan yang lebih beragam bagi pengguna.

Link Github

https://github.com/Maharanish/Sugar-bNb-for-Diabetic

Link Video Youtube

https://youtu.be/pG6akjBTkT0

Ucapan Terima Kasih

Segenap rasa syukur penulis panjatkan kepada Allah SWT, sumber segala kebijaksanaan, atas rahmat dan petunjuk-Nya yang melimpah, memandu penulis dalam menyelesaikan makalah ini. Terima kasih yang tak terhingga penulis sampaikan kepada Ir. Rila Mandala, M.Eng., Ph.D., dan Monterico Adrian, S.T., M.T., selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas K03, yang dengan penuh dedikasi memberikan bimbingan, ilmu, dan inspirasi yang menjadi landasan utama penulisan. Penghargaan setinggi-tingginya juga diberikan kepada orang tua dan teman-teman penulis yang senantiasa memberikan dukungan moral, motivasi, dan semangat. Terima kasih juga penulis ucapkan kepada salah satu teman penulis, Arrasqya, yang membantu mengoreksi tata bahasa dalam makalah ini. Terakhir, terima kasih kepada semua pihak yang berkontribusi dengan adanya referensi dan bahan pendukung, memperkaya pemahaman penulis terhadap materi. Semua bantuan, dukungan, dan kontribusi ini menjadi pondasi penting dalam kelancaran penyusunan makalah ini. Semoga hasil karya ini dapat memberikan manfaat dan kontribusi positif bagi pembaca..

Referensi

[1] Sherwood, Lauralee. 2014. Human Physiology : From Cells to System Ninth Edition. Boston, USA : Cengage Learning. Halaman 697.

[2] International Diabetes Federation. (2021). IDF Atlas 10th edition. https://www.diabetesatlas.org/data/en/country/94/id.html (diakses pada 10 Juni 2024).

[3] Rini, Hastuti Tri. 2008. Faktor-Faktor Risiko Ulkus Diabetika Pada Penderita Diabetes Mellitus (Studi Kasus Di Rsud Dr. Moewardi Surakarta). Program Studi Magister Epidemiologi Program Pasca Sarjana Universitas Diponegoro Semarang. PhD Thesis. http://eprints.undip.ac.id/18866/1/Rini_Tri_Hastuti.pdf (diakses pada 10 Juni 2024).

[4] World Health Organization. 2023. Diabetes. https://www.who.int/news-room/fact-sheets/detail/diabetes#:~:text=Key%20facts%201%20The%20number%20of%20people%20with,caused%20an%20estimated%202%20million%20deaths.%20More%20items (diakses pada 10 Juni 2024).

[5] Southwick, Courtney. 2023. 7 Proteins to Add to Your Grocery List to Help You Lower Blood Sugar, According to a Dietitian. https://www.eatingwell.com/article/8072838/proteins-to-help-lower-blood-sugar-according-to-a-dietitian/ (diakses pada 10 Juni 2024)

[6] Pengenalan Masalah Knapsack, Jenisnya dan Cara Mengatasinya. 2023. GeeksforGeeks. https://www.geeksforgeeks.org/problems/0-1-knapsack-problem0945/1 (diakses pada 10 Juni 2024).

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

[8] Munir, Rinaldi. 2020. Pohon (Bag. 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf (diakses pada 11 Juni 2024).

[9] Munir, Rinaldi. 2021. Algoritma Runut-Balik (Backtracking) : Bagian 1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf (diakses pada 11 Juni 2024).

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

[11] Munir, Rinaldi. 2022. Algoritma Branch and Bound (Bagian 4). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Branchand-Bound-2022-Bagian4.pdf (diakses pada 11 Juni 2024).

[12] Munir, Rinaldi. 2020. Pohon (Bag. 2). https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf (diakses pada 11 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, 10 Juni 2024

Shabrina Maharani

13522134