"Perbandingan Kinerja Algoritma *Branch and Bound* dan *Greedy* dalam Menyelesaikan *Travelling Salesman Problem* Studi Kasus : Optimasi Rute Pengunjung Kebun Binatang"

Rahma Setiani[1], Daniya Auly Yasmin[2]
Program Studi S1 Teknik Informatika
Fakultas Informatika
Institut Teknologi Telkom Purwokerto, Jl. D.I. Panjaitan No.128 Purwokerto 53147, Indonesia
[1]*21102304@ittelkom-pwt.ac.id*, [2]*21102041@ittelkom-pwt.ac.id*

***Abstrak*—** Algoritma berperan krusial dalam pengembangan perangkat lunak dan pemrograman karena memungkinkan penyelesaian masalah secara efisien. Dalam penelitian ini, kami melakukan percobaan kasus *Travelling Salesman Problem* (TSP) pada dua jenis kebun binatang, yaitu Nirmala dan Bimasakti, menggunakan algoritma *Greedy* dan *Branch and Bound*. Hasil percobaan menunjukkan perbedaan dalam pencarian jalur minimum dan waktu eksekusi. Pada Kebun Binatang Nirmala, kedua algoritma memberikan hasil yang sama dengan jalur minimum sepanjang 80 meter. Meskipun hasilnya sama, algoritma *Branch and Bound* menunjukkan waktu eksekusi yang lebih cepat. Namun, pada Kebun Binatang Bimasakti, terdapat perbedaan hasil. Algoritma *Branch and Bound* menemukan jalur minimum sepanjang 91 meter, sedangkan algoritma *Greedy* menemukan jalur minimum sepanjang 96 meter. Algoritma *Branch and Bound* juga menunjukkan waktu eksekusi yang lebih cepat. Berdasarkan hasil percobaan, algoritma *Branch and Bound* lebih efisien dalam menemukan jalur minimum pada kedua jenis kebun binatang (Nirmala dan Bimasakti). Oleh karena itu, disarankan untuk menggunakan algoritma *Branch and Bound* sebagai pilihan utama dalam menyelesaikan masalah *Travelling Salesman Problem* pada kasus kebun binatang. Perlu dicatat bahwa pemahaman tentang kedua algoritma dan analisis performa dalam berbagai kasus dan kondisi sangat penting dalam pengambilan keputusan. Dengan mempertimbangkan karakteristik masalah dan kompleksitas data, kita dapat memilih algoritma yang sesuai untuk mencapai solusi yang optimal atau cukup baik secara efisien.

***Kata Kunci*— *Travelling Salesman Problem*, algoritma *Branch and Bound*, algoritma *Greedy*, optimasi rute, kebun binatang.**

I. PENDAHULUAN

Kebun binatang merupakan salah satu tempat rekreasi yang populer dan menarik bagi pengunjung dari berbagai kalangan. Di dalam kebun binatang, pengunjung dapat menikmati keindahan dan keunikan berbagai satwa dari seluruh dunia. Namun, semakin berkembangnya kebun binatang dan meningkatnya jumlah pengunjung, muncul berbagai masalah terkait pengelolaan dan optimalisasi pada rute pengunjung di dalamnya. Rute yang tidak teratur atau kurang dioptimalkan dapat menyebabkan antrean panjang, waktu tunggu yang lama, dan mengurangi pengalaman pengunjung secara keseluruhan. Selain itu, pengaturan rute yang tidak tepat juga dapat mengganggu aktivitas satwa dan mengurangi kenyamanan hewan.

Untuk mengatasi masalah ini, pengelola kebun binatang perlu mencari solusi yang tepat. Penerapan teknologi dan strategi optimisasi dapat menjadi solusi untuk mengatur rute pengunjung agar memberikan pengalaman yang maksimal, efisien, dan menyenangkan. Dalam hal ini, *Travelling Salesman Problem* (TSP) menjadi permasalahan yang relevan, di mana tujuan utamanya adalah menyusun rute terpendek untuk mengunjungi setiap kandang hanya sekali dan kembali ke pintu masuk.

Dengan mencari solusi yang mendekati optimal untuk TSP dalam rute kebun binatang, diharapkan akan meningkatkan kepuasan pengunjung, mengurangi waktu tempuh, serta memaksimalkan pengalaman dan kenyamanan selama mengunjungi kebun binatang. Dengan demikian, pengelola kebun binatang dapat memastikan bahwa rute pengunjung teratur dan efisien, sehingga menjadikan pengalaman berwisata lebih menyenangkan dan bermanfaat bagi para pengunjung dan satwa yang ada di dalam kebun binatang.

Untuk mencari solusi TSP pada rute kebun binatang antar kandang, pengelola kebun binatang dapat menggunakan berbagai algoritma optimasi, antara lain adalah algoritma *Greedy* dan *Branch and Bound*. Algoritma *Greedy* bekerja dengan memilih langkah terbaik secara lokal pada setiap tahap, namun hasilnya mungkin tidak selalu optimal. Di sisi lain, algoritma *Branch and Bound* bekerja dengan membangun dan memotong pohon pencarian untuk mencari solusi yang optimal.

Dalam tulisan ini, kami akan membandingkan kedua algoritma tersebut dalam menyelesaikan TSP pada rute kebun binatang antar kandang. Hasil perbandingan akan memberikan wawasan tentang efisiensi dan keakuratan masing-masing algoritma dalam mengoptimalkan rute pengunjung kebun binatang. Diharapkan dengan pemahaman yang lebih mendalam tentang perbandingan ini, pengelola kebun binatang dapat memilih algoritma yang sesuai dengan kebutuhan dan kompleksitas rute pengunjung, sehingga dapat meningkatkan efisiensi dan kenyamanan bagi para pengunjung dan satwa yang ada di dalam kebun binatang.

II. TINJAUAN PUSTAKA

A. *Travelling Salesman Problem*

William Rowan Hamilton dan Thomas Penyngton merupakan orang yang pertama kali menemukan *travelling salesman problem*. Permainan Icosian Hamilton merupakan permainan klasik yang menjadi asal mula ditemukannya permasalahan *Travelling Salesman Problem* (TSP). Dalam permainan ini, pemain diminta untuk menyelesaikan

perjalanan dari 20 titik berbeda dengan melewati jalur-jalur tertentu.

TSP adalah permasalahan yang sering dihadapi oleh seorang salesman yang harus mencari rute terpendek untuk mengunjungi beberapa tempat tanpa mengulang untuk melewati tempat yang telah dilewati sebelumnya. Dengan kata lain, salesman harus memastikan bahwa ia hanya mengunjungi setiap tempat sekali dan akhirnya kembali ke tempat awal.

Seiring dengan perkembangan permainan *Icosian Hamilton*, permasalahan TSP menjadi menarik bagi para peneliti matematika dan ilmu komputer. Mereka mulai mempelajari algoritma dan strategi untuk mencari solusi optimal atau mendekati optimal dalam mengatasi TSP.

Permasalahan TSP memiliki kompleksitas yang tinggi, dan untuk jumlah titik yang besar, mencari solusi eksak dalam waktu yang efisien menjadi sulit. Oleh karena itu, berbagai pendekatan algoritma seperti algoritma *greedy* dan *branch and bound* telah dikembangkan untuk menemukan solusi yang efisien dan mendekati optimal dalam berbagai kasus TSP.

B. Algoritma

Algoritma adalah serangkaian langkah atau instruksi yang digunakan untuk mencari solusi atas sebuah masalah atau menyelesaikan tugas tertentu dalam pembangunan sebuah program komputer. Salah satu jenis algoritma yang sering digunakan dalam memecahkan masalah optimasi adalah algoritma *greedy* dan algoritma *branch and bound.*

C. Algoritma *Branch and Bound* dalam TSP

Algoritma *Branch and Bound (Brach and Bound)* adalah salah satu metode eksak yang digunakan untuk menyelesaikan permasalahan optimasi dan pencarian solusi optimal. Algoritma ini sering digunakan untuk permasalahan kombinatorial, seperti *Travelling Salesman Problem (TSP), Knapsack Problem*, dan banyak permasalahan lain yang kompleks.

Prinsip dasar dari algoritma *Branch and Bound* adalah dengan membagi permasalahan menjadi beberapa submasalah yang lebih kecil (*branching*) dan kemudian mencari solusi secara rekursif pada setiap submasalah tersebut. Selama proses pencarian, algoritma ini juga melakukan pemangkasan (*pruning*) pada cabang yang tidak memungkinkan untuk menghasilkan solusi optimal. Hal ini dilakukan dengan memeriksa batasan atau batas atas *(upper bound*) pada setiap cabang dan membandingkannya dengan solusi terbaik yang telah ditemukan sejauh ini. Jika sebuah cabang memiliki *upper bound* yang lebih buruk dari solusi terbaik yang telah ada, maka cabang tersebut tidak akan dijelajahi lebih lanjut (*pruning*), sehingga menghemat waktu komputasi.

D. Algoritma *Greedy* dalam TSP

Algoritma *Greedy* (atau sering disebut metode serakah) adalah mengambil keputusan yang dianggap paling optimal pada setiap langkah atau tahap, tanpa mempertimbangkan konsekuensi atau dampak keputusan tersebut pada langkah-langkah berikutnya. Algoritma *Greedy* berfokus pada memaksimalkan atau meminimalkan suatu kriteria lokal pada setiap tahap, dengan harapan bahwa keputusan-keputusan lokal tersebut akan mengarah ke solusi yang optimal secara keseluruhan.

Prinsip utama dari algoritma *Greedy* adalah bahwa pada setiap langkah, algoritma hanya mempertimbangkan keuntungan atau nilai yang dapat diperoleh dari pilihan terbaik saat itu, tanpa melihat bagaimana pilihan tersebut akan mempengaruhi solusi di langkah-langkah berikutnya. Pilihan yang diambil pada setiap langkah dianggap sebagai langkah optimal secara lokal, dengan harapan bahwa langkah-langkah lokal tersebut akan menghasilkan solusi yang optimal secara keseluruhan.

E. Bahasa Pemograman Java

Bahasa pemrograman Java adalah bahasa pemrograman yang didesain untuk berjalan pada platform apapun tanpa harus mengalami perubahan kode secara besar-besaran (*write once, run anywhere* - WORA). Bahasa ini pertama kali dikembangkan oleh James Gosling dan timnya di *Sun Microsystems* pada tahun 1995. Java populer karena kemampuannya dalam membuat aplikasi yang dapat berjalan di berbagai perangkat, termasuk komputer pribadi, server, perangkat seluler, dan bahkan perangkat tertanam (*embedded devices*).

Bahasa pemrograman Java dapat digunakan untuk menyelesaikan berbagai macam algoritma dan permasalahan pemrograman. Java memiliki fitur-fitur yang kuat dan luas, serta dukungan untuk berbagai struktur data dan algoritma yang memungkinkan para pengembang untuk merancang dan mengimplementasikan solusi yang efisien.

F. Penelitian Terdahulu

Penelitian mengenai penggunaan aplikasi untuk menyelesaikan permasalahan *travelling salesman problem* telah dilakukan oleh beberapa orang. Penelitian pertama dilakukan oleh Djorgy, dkk dengan judul jurnal "Aplikasi Navigasi Perjalanan Paket Dengan Sistem Pemilihan Rute Tercepat Menggunakan Algoritma *Greedy*" merupakan perancangan aplikasi yang dapat mengatur rute perjalanan paket dari yang terdekat hingga yang terjauh. Pengurutan alamat dari aplikasi ini berdasarkan info dari *QR code* yang di *scan* oleh kurir lalu sistem pemilih rute akan mensortir alamat dari yang terdekat hingga terjauh dengan menggunakan Algoritma *Greedy*.

Penelitian lain dengan menggunakan aplikasi untuk menyelesaikan permasalahan *travelling salesman problem* telah dilakukan oleh Muhammad Zakaria Usman dan Teguh Oktiarso dengan judul "Implementasi Algoritma *Greedy* untuk Menyelesaikan *Travelling Salesman Problem* di Distributor PT. Z" melakukan implementasi untuk menyelesaian masalah yang dihadapi oleh PT. X yaitu permasalahan rute distribusi dalam menyalurkan produk obat dan barang kebutuhan sehari hari menggunakan *travelling salesman problem* dengan pendekatan algoritma *Greedy* untuk mendapatkan rute distribusi terpendek.

Penelitian lain untuk menyelesaikan permasalahan TSP dilakukan oleh Justin Eduardo Simarmata dengan judul "Penerapan Algoritma *Branch And Bound* pada Persoalan Pedagang Keliling *Travelling Salesman Problem*" penelitian ini membahas tentang algoritma *Branch and Bound* dalam menyelesaikan persoalan TSP. Dengan menerapkan algoritma *Branch and Bound* pada contoh soal persoalan pedagang keliling (*Travelling Salesman Problem*) dalam penelitian ini maka diperoleh rute perjalanan terpendek.

Penelitian serupa untuk menyelesaikan permasalahan TSP juga dilakukan oleh Andika Pranata dan Hutrianto dengan judul "Rekayasa Perangkat Lunak Penentuan Jarak Terdekat Dalam Pengiriman Darah di PMI Kota Palembang Dengan Algoritma *Branch dan Bound*" Dalam melayani kebutuhan darah yang ada

dikota Palembang, PMI kota Palembang selalu memberikan pelayanan yang baik, tertama dalam hal ketepatan waktu dalam pengiriman darah yang di butuhkan oleh rumah sakit. *travelling salesman problem* merupakan perasalahan yang selalu di hadapi dalam suatu proses yang berkaitan dengan jarak dan waktu, salah satu metode nya yaitu metode Algoritma *branch and bound* yang menerapkan dengan membuat simpul –simpul dalam menentukan note untuk menempuh jarak tertentu.

III. METODE PENELITIAN

Metode yang digunakan pada penelitian ini yaitu :

A. Rancangan Penelitian

Penelitian ini menggunakan rancangan eksperimen dengan tujuan untuk membandingkan kinerja algoritma *Branch and Bound* dan *Greedy* dalam menyelesaikan *Travelling Salesman Problem* (TSP) dengan studi kasus optimasi rute pengunjung kebun binatang. Rancangan eksperimen ini memungkinkan kami untuk mengamati perbedaan performa dan efisiensi dari kedua algoritma dalam menyelesaikan permasalahan TSP pada dataset yang berbeda.

B. Studi Literatur

Mengidentifikasi topik penelitian yang spesifik, Menentukan sumber-sumber pustaka yang relevan, seperti jurnal ilmiah, buku, artikel, konferensi, dan sumber-sumber terpercaya lainnya yang berhubungan dengan TSP, algoritma *Branch and Bound*, dan algoritma *Greedy*. Melakukan pencarian literatur menggunakan basis data akademik, perpustakaan digital, mesin pencari, dan sumber-sumber online terpercaya. Memilih sumber informasi yang paling relevan dan berkualitas untuk digunakan dalam analisis penelitian. Menganalisis isi dari setiap sumber informasi yang dipilih, termasuk teori dan konsep yang terkait dengan TSP, algoritma *Branch and Bound*, dan algoritma *Greedy*. Membuat sinopsis dari setiap sumber informasi yang dipilih untuk memahami konten dan temuan utama dari masing-masing sumber. Mencatat rujukan dari setiap sumber informasi yang digunakan dalam studi literatur.

C. Data Penelitian

Data penelitian yang digunakan dalam penelitian ini terdiri dari informasi tentang kebun binatang yang bersangkutan, termasuk lokasi dan jarak antara kandang-kandang di dalamnya. Selain itu, kami juga menggunakan dataset simulasi TSP dengan berbagai skenario jumlah kandang untuk melengkapi analisis perbandingan algoritma.

D. Implementasi Algoritma *Branch and Bound*

Pada langkah awal penelitian ini, kami mengimplementasikan algoritma *Branch and Bound* untuk menyelesaikan TSP pada dataset kebun binatang yang sebenarnya dan dataset simulasi. Langkah-langkah algoritma ini dijalankan untuk mencari rute optimal dengan memanfaatkan pendekatan pemotongan dan batasan pada setiap simpul dalam ruang pencarian.

E. Implementasi Algoritma *Greedy*

Selanjutnya, kami mengimplementasikan algoritma *Greedy* untuk menyelesaikan TSP pada dataset yang sama. Algoritma *Greedy* ini memilih jalur terbaik yang tersedia pada setiap langkah berdasarkan kriteria jarak terpendek. Meskipun algoritma ini cenderung mengambil pilihan lokal terbaik, kami ingin melihat efisiensi dan kualitas solusi yang dihasilkan dalam kasus ini.

F. Pengujian dan Analisis Perbandingan

Melakukan pengujian kedua algoritma pada dataset yang sama dengan menggunakan berbagai ukuran masalah (misalnya, jumlah kandang yang berbeda).Membandingkan hasil dari kedua algoritma dalam hal solusi yang dihasilkan, waktu eksekusi, dan kompleksitas waktu dan ruang. Menganalisis keunggulan dan kelemahan masing-masing algoritma dalam menyelesaikan permasalahan TSP pada studi kasus kebun binatang.

Adapun *hardware* yang digunakan untuk pengujian menggunakan laptop dengan kapasitas HDD 1TB dan RAM 4GB yang menggunakan proses intel core i3 dan sistem operasi nya adalah windows 11. Sedangkan *software* yang digunakan adalah applikasi intelej IDE dimana program dituliskan menggunakan bahasa pemograman java.

G. Evaluasi Kinerja

Mengevaluasi kinerja kedua algoritma berdasarkan hasil pengujian dan analisis. Menarik kesimpulan mengenai algoritma mana yang lebih unggul dalam menyelesaikan TSP pada studi kasus kebun binatang.

H. Validasi Hasil

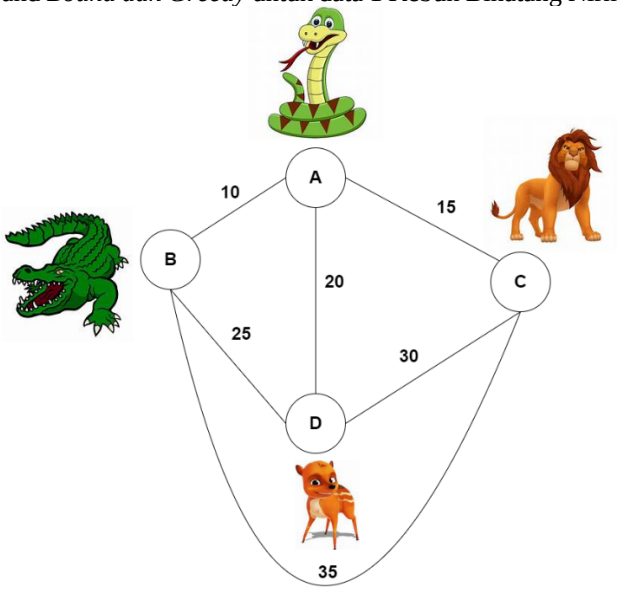
Untuk memastikan validitas hasil, kami melakukan uji coba pada beberapa skenario dataset dengan nilai yang berbeda

IV. HASIL DAN PEMBAHASAN.

A. Hasil

1. Data 1 Kebun Binatang Nirmala

Berikut merupakan hasil implementasi algoritma *Branch and Bound dan Greedy* untuk data 1 Kebun Binatang Nirmala.



Gambar 1 Denah Kebun Binatang Nirmala

Keterangan :
A = Kandang ular (Pintu masuk dan keluar)
B = Kandang buaya
C = Kandang singa
D = Kandang kancil

Tabel 1 Jarak antar kandang Data 1 Nirmala

Kandang	Ular	Buaya	Singa	Kancil
Ular	-	10 m	15 m	20 m
Buaya	10 m	-	35 m	25 m
Singa	15 m	35 m	-	30 m
Kancil	20 m	25 m	30 m	-

Tabel 2 Perbandingan Hasil Program Data 1 Nirmala

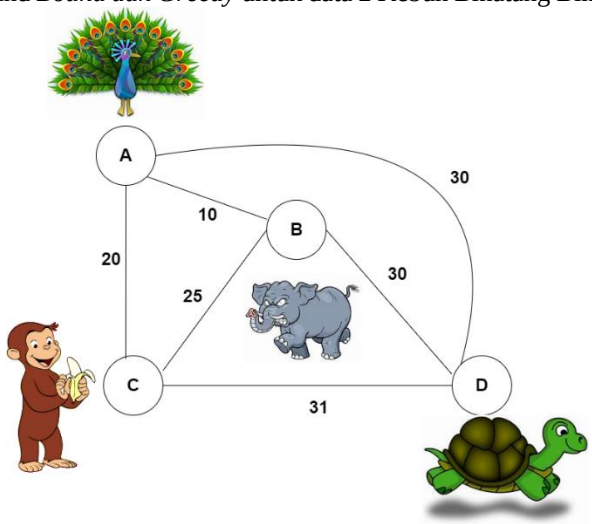
	Branch And Bound	*Greedy*
Jarak Minimum	80 m	80 m
Path Taken	A-B-D-C-A	A-B-D-C-A
Execution Time	0.012 ms	0.259 ms

Untuk Kebun Binatang Nirmala (Data 1), algoritma *Branch and Bound* dan algoritma *Greedy* memberikan hasil yang sama, yaitu jalur minimum sejauh 80 meter, dan keduanya menemukan jalur yang sama, yaitu A - B - D - C – A. Artinya pengunjung kebun Binatang Nirmala dapat memulai rute perjalanannya dari pintu masuk kandang ular selanjutnya ke kandang buaya lalu ke kandang kancil selanjutnya ke kandang singa dan kembali lagi ke kandang ular sebagai pintu keluar.

Meskipun kedua algoritma menghasilkan hasil yang sama, algoritma *Branch and Bound* menunjukkan waktu eksekusi yang lebih cepat dibandingkan dengan algoritma *Greedy*, yaitu hanya 0.012 milidetik dibandingkan dengan 0.259 milidetik.

2. Data 2 Kebun Binatang Bimasakti

Berikut merupakan hasil Implementasi Algoritma *Branch and Bound dan Greedy* untuk data 2 Kebun Binatang Bimasakti.



Gambar 2 Denah Kebun Binatang Bimasakti

Keterangan :
A = Kandang burung merak (Pintu masuk dan keluar)
B = Kandang gajah
C = Kandang monyet
D = Kandang kura-kura

Tabel 3 Jarak antar kandang Data 2 Bimasakti

Kandang	Merak	Gajah	Monyet	Kura-kura
Merak	-	10 m	20 m	30 m
Gajah	10 m	-	25 m	30 m
Moyet	20 m	25 m	-	31 m
Kura-kura	30 m	30 m	31 m	-

Tabel 4 Perbandingan Hasil Program Data 2 Bimasakti

	Branch And Bound	*Greedy*
Jarak Minimum	91 m	96 m
Path Taken	A-B-D-C-A	A-B-C-D-A
Execution Time	0.013 ms	0.242 ms

Pada data Kebun Binatang Bimasakti, terdapat perbedaan hasil antara algoritma *Branch and Bound* dan *Greedy*. Algoritma *Branch and Bound* menemukan jalur minimum mengelilingi kebun binatang bimasakti sejauh 91 meter dan jalur yang diambil adalah A - B - D - C – A. Artinya pengunjung kebun Binatang Bimasakti dapat memulai rute perjalanannya dari pintu masuk kandang merak selanjutnya ke kandang gajah lalu ke kandang kura-kura selanjutnya ke kandang monyet dan kembali lagi ke kandang merak sebagai pintu keluar, dengan waktu eksekusi 0.013 milidetik.

Sementara algoritma *Greedy* menghasilkan jalur minimum untuk berkeliling kebun binatang bimasakti dengan jarak 96 meter dan jalur yang diambil adalah A - B - C - D – A. Artinya pengunjung kebun Binatang Bimasakti dapat memulai rute perjalanannya dari pintu masuk kandang merak selanjutnya ke kandang gajah lalu ke kandang monyet selanjutnya baru ke kandang kura-kura dan kembali lagi ke kandang merak sebagai pintu keluar dengan waktu eksekusi 0.242 milidetik.

Dengan demikian, terdapat perbedaan hasil antara algoritma *Branch and Bound* dan *Greedy* pada data Kebun Binatang Bimasakti. Algoritma *Branch and Bound* menemukan jalur minimum dengan jarak lebih pendek (91 meter), sedangkan algoritma *Greedy* menemukan jalur minimum dengan jarak sedikit lebih panjang (96 meter). Selain itu, algoritma *Branch and Bound* juga memerlukan waktu eksekusi yang lebih singkat dibandingkan algoritma *Greedy* baik di kebun binatang Nirmala (Data 1) maupun di kebun binatang Bimasakti (Data 2).

Maka algoritma *Branch and Bound* cocok digunakan untuk menyelesaikan masalah *Travelling Salesman Problem* lebih tepatnya untuk studi kasus optimasi rute kebun binatang dibandingkan dengan algoritma *Greedy*.

B. Pembahasan

1. Algoritma *Branch and Bound*

a) Perhitungan Manual kebun binarang Nirmala

Langkah 1: Buatlah matriks jarak seperti yang telah diberikan.

$$\begin{Bmatrix} \infty & 10 & 15 & 20 \\ 10 & \infty & 35 & 25 \\ 15 & 35 & \infty & 30 \\ 20 & 25 & 30 & \infty \end{Bmatrix}$$

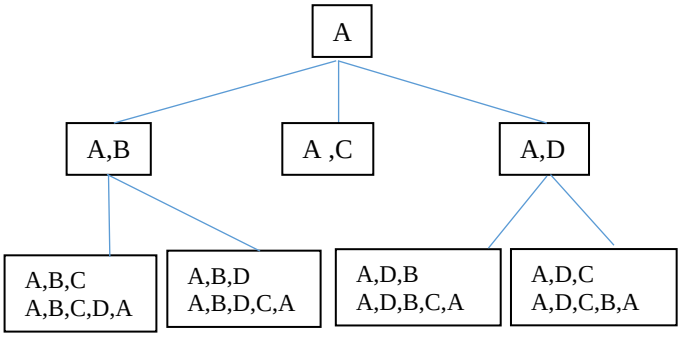
Langkah 2 : Kita terlebih dahulu menghitung batas bawah (*lower bound*) titik A sebagai titik awal.

$$\begin{matrix} A & B & C & D \end{matrix}\begin{pmatrix} \infty & 10 & 15 & 20 \\ 10 & \infty & 35 & 25 \\ 15 & 35 & \infty & 30 \\ 20 & 25 & 30 & \infty \end{pmatrix}$$

$Va = min\{10,15,20\} = 10\ (diambil\ yang\ paling\ kecil)$
$Vb = min\{10,35,25\} = 10$
$Vc = min\{15,35,30\} = 15$
$Vd = min\{20,25,30\} = 20$

$Lb(0) = 10 + 10 + 15 + 20 = 55$
Jadi lb titik awal adalah 55.

Langkah 3 : Buatlah pohon ruang status atau representasi visual dari langkah-langkah yang diambil untuk menyelesaikan masalah optimasi kombinatorial seperti *Travelling Salesman Problem* (TSP). dimulai dari kandang A



Langkah 4 : Pada pohon ruang status, kita memiliki tiga cabang karena dari titik awal A kita dapat memilih kandang B, C, atau D selanjutnya. Kita harus menghitung batas bawah (*lower bound*) untuk setiap cabang.

Cabang 1 A ke B

$$\begin{matrix} A & B & C & D \end{matrix}\begin{pmatrix} \infty & 10 & 15 & 20 \\ \blacksquare & \blacksquare & 35 & 25 \\ 15 & \blacksquare & \infty & 30 \\ 20 & \blacksquare & 30 & \infty \end{pmatrix}$$

$Va = Jarak\ dari\ A\ ke\ B = 10$
$Vb = min\{35,25\} = 25$
$Vc = min\{15,30\} = 15$
$Vd = min\{20,30\} = 20$

$Lb(1) = 10 + 25 + 15 + 20 = 70$
Jadi lb(1) A,B adalah 70.

Cabang 2 A ke C

$$\begin{matrix} A & B & C & D \end{matrix}\begin{pmatrix} \infty & 10 & 15 & 20 \\ 10 & \infty & \blacksquare & 25 \\ \blacksquare & 35 & \blacksquare & 30 \\ 20 & 25 & \blacksquare & \infty \end{pmatrix}$$

$Va = Jarak\ dari\ A\ ke\ C = 15$
$Vb = min\{10,25\} = 10$
$Vc = min\{35,30\} = 30$

$Vd = min\{20,25\} = 20$

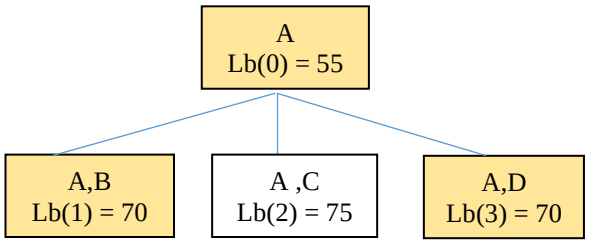
$Lb(2) = 15 + 10 + 30 + 20 = 75$
Jadi lb(2) A,C adalah 75.

Cabang 3 A ke D

$$\begin{matrix} A & B & C & D \end{matrix}\begin{pmatrix} \infty & 10 & 15 & 20 \\ 10 & \infty & 35 & \blacksquare \\ 15 & 35 & \infty & \blacksquare \\ \blacksquare & 25 & 30 & \blacksquare \end{pmatrix}$$

$Va = Jarak\ dari\ A\ ke\ D = 20$
$Vb = min\{10,35\} = 10$
$Vc = min\{15,35\} = 15$
$Vd = min\{25,30\} = 25$

$Lb(3) = 20 + 10 + 15 + 25 = 70$
Jadi lb(2) A,D adalah 70.



Langkah 5 : Pada pohon ruang status, kita memilih cabang dengan nilai batas bawah (*lower bound*) yang paling kecil. Mak a kita memilih cabang A,B dan A,D dengan nilai sama 70. Selanjutnya kita lihat dari titik A,B kita bisa memilih kandang C dengan menyisakan kandang D atau D dulu dengan menyisakan kandang C dan kembali lagi ke titik A. sedangkan dari titik A,D kita bisa memilih kandang B dengan menyisakan kandang C atau ke kandang C dulu dengan menyisakan kandang B dan Kembali lagi ke kandang A. Kita akan menghitung batas bawah (*lower bound*) untuk setiap cabang tersebut.

$$\begin{matrix} A & B & C & D \end{matrix}\begin{pmatrix} \infty & 10 & 15 & 20 \\ 10 & \infty & 35 & 25 \\ 15 & 35 & \infty & 30 \\ 20 & 25 & 30 & \infty \end{pmatrix}$$

Cabang 4 A,B,C (A,B,C,D,A)
$Va = Jarak\ dari\ A\ ke\ B = 10$
$Vb = Jarak\ dari\ B\ ke\ C = 35$
$Vc = Jarak\ dari\ C\ ke\ D = 30$
$Vd = Jarak\ dari\ D\ ke\ A = 20$

$Lb(4) = 10 + 35 + 30 + 20 = 95$
Jadi lb(4) A,B,C,D,A adalah 95.

Cabang 5 A,B,D (A,B,D,C,A)
$Va = Jarak\ dari\ A\ ke\ B = 10$
$Vb = Jarak\ dari\ B\ ke\ D = 25$
$Vc = Jarak\ dari\ C\ ke\ A = 15$
$Vd = Jarak\ dari\ D\ ke\ C = 30$

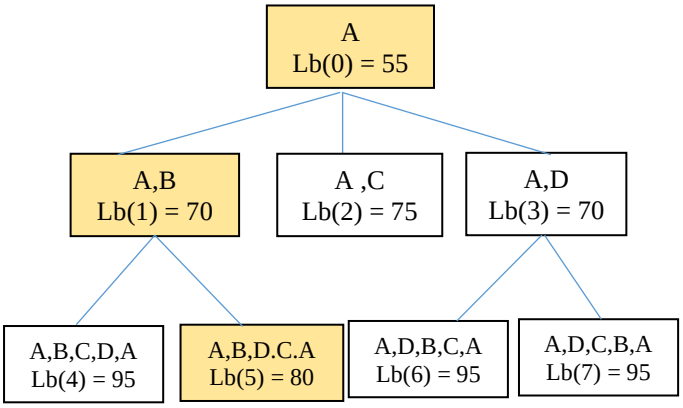
$Lb(5) = 10 + 25 + 15 + 30 = 80$
Jadi lb(5) A,B,D,C,A adalah 80.

Cabang 6 A,D,B (A,D,B,C,A)
$Va = Jarak\ dari\ A\ ke\ D = 20$
$Vb = Jarak\ dari\ B\ ke\ C = 35$
$Vc = Jarak\ dari\ C\ ke\ A = 15$
$Vd = Jarak\ dari\ D\ ke\ B = 25$

$Lb(6) = 20 + 35 + 15 + 25 = 95$
Jadi lb(6) A,D,B,C,A adalah 95.

Cabang 7 A,D,C (A,D,C,B,A)
$Va = Jarak\ dari\ A\ ke\ D = 20$
$Vb = Jarak\ dari\ B\ ke\ A = 10$
$Vc = Jarak\ dari\ C\ ke\ B = 35$
$Vd = Jarak\ dari\ D\ ke\ C = 30$

$Lb(7) = 10 + 10 + 35 + 30 = 95$
Jadi lb(7) A,D,C,B,A adalah 95.



Jadi, karena semua titik sudah dilewati dan kembali lagi ke titik awal maka kita memilih cabang dengan nilai batas bawah *(lower bound*) yang paling kecil.yaitu melalui rute titik kandang A – B – D – C- A dengan total jarak 80 meter.

b) Perhitungan Manual kebun Binatang Bimasakti
Langkah 1: Buatlah matriks jarak seperti yang telah diberikan.

$$\begin{Bmatrix} \infty & 10 & 20 & 30 \\ 10 & \infty & 25 & 30 \\ 20 & 25 & \infty & 31 \\ 30 & 30 & 31 & \infty \end{Bmatrix}$$

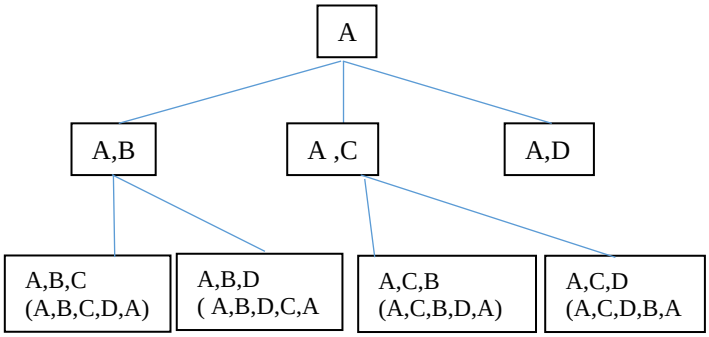
Langkah 2 : Kita terlebih dahulu menghitung batas bawah (*lower bound*) titik A sebagai titik awal.

$$\begin{matrix} A & B & C & D \end{matrix}$$
$$\begin{Bmatrix} \infty & 10 & 20 & 30 \\ 10 & \infty & 25 & 30 \\ 20 & 25 & \infty & 31 \\ 30 & 30 & 31 & \infty \end{Bmatrix}$$

$Va = min\{10,20,30\} = 10$
$Vb = min\{10,25,30\} = 10$
$Vc = min\{20,25,31\} = 20$
$Vd = min\{30,30,31\} = 30$

$Lb(0) = 10 + 10 + 20 + 30 = 70$
Jadi lb titik awal adalah 70.

Langkah 3 : Buatlah pohon ruang status atau representasi visual dari langkah-langkah yang diambil untuk menyelesaikan masalah optimasi kombinatorial seperti *Travelling Salesman Problem* (TSP). dimulai dari kandang A



Langkah 4 : Pada pohon ruang status, kita memiliki tiga cabang karena dari titik awal A kita dapat memilih kandang B, C, atau D selanjutnya. Kita harus menghitung batas bawah (*lower bound*) untuk setiap cabang.

Cabang 1 A ke B

$$\begin{matrix} A & B & C & D \end{matrix}$$
$$\begin{Bmatrix} \infty & 10 & 20 & 30 \\ 10 & & 25 & 30 \\ 20 & & \infty & 31 \\ 30 & & 31 & \infty \end{Bmatrix}$$

$Va = Jarak\ dari\ A\ ke\ B = 10$
$Vb = min\{25,30\} = 25$
$Vc = min\{20,31\} = 20$
$Vd = \min\{30,31\} = 30$

$Lb(1) = 10 + 25 + 20 + 30 = 85$
Jadi lb(1) A,B adalah 85.

Cabang 2 A ke C

$$\begin{matrix} A & B & C & D \end{matrix}$$
$$\begin{Bmatrix} \infty & 10 & 20 & 30 \\ 10 & \infty & & 30 \\ 20 & 25 & & 31 \\ 30 & 30 & & \infty \end{Bmatrix}$$

$Va = Jarak\ dari\ A\ ke\ C = 20$
$Vb = min\{10,30\} = 10$
$Vc = min\{25,31\} = 25$
$Vd = \min\{30,30\} = 30$

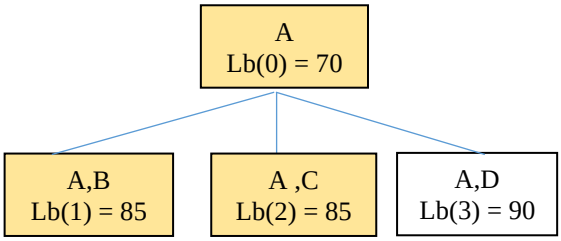
$Lb(2) = 20 + 10 + 25 + 30 = 85$
Jadi lb(2) A,C adalah 85.

Cabang 3 A ke D

$$\begin{matrix} A & B & C & D \end{matrix}$$
$$\begin{Bmatrix} \infty & 10 & 20 & 30 \\ 10 & \infty & 25 & \\ 20 & 25 & \infty & \\ 30 & 30 & 31 & \end{Bmatrix}$$

$Va = Jarak\ dari\ A\ ke\ D = 30$
$Vb = min\{10,25\} = 10$
$Vc = min\{20,25\} = 20$
$Vd = \min\{30,31\} = 30$

$Lb(3) = 30 + 10 + 20 + 30 = 90$
Jadi lb(2) A,D adalah 90.



Langkah 5 : Pada pohon ruang status, kita memilih cabang dengan nilai batas bawah (*lower bound*) yang paling kecil. Mak a kita memilih cabang A,B dan A,C dengan nilai sama 85. Selanjutnya kita lihat dari titik A,B kita bisa memilih kandang C dengan menyisakan kandang D atau D dulu dengan menyisakan kandang C dan kembali lagi ke titik A. Sedangkan dari titik A,C kita bisa memilih kandang B dengan menyisakan kandang D atau ke kandang D dulu dengan menyisakan kandang B dan kembali lagi ke kandang A. Kita akan menghitung batas bawah (*lower bound*) untuk setiap cabang tersebut.

$$\begin{matrix} & A & B & C & D \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{pmatrix} \infty & 10 & 20 & 30 \\ 10 & \infty & 25 & 30 \\ 20 & 25 & \infty & 31 \\ 30 & 30 & 31 & \infty \end{pmatrix} \end{matrix}$$

Cabang 4 A,B,C (A,B,C,D,A)
$Va = Jarak\ dari\ A\ ke\ B = 10$
$Vb = Jarak\ dari\ B\ ke\ C = 25$
$Vc = Jarak\ dari\ C\ ke\ D = 31$
$Vd = Jarak\ dari\ D\ ke\ A = 30$

$Lb(4) = 10 + 25 + 31 + 30 = 96$
Jadi lb(4) A,B,C,D,A adalah 96.

Cabang 5 A,B,D (A,B,D,C,A)
$Va = Jarak\ dari\ A\ ke\ B = 10$
$Vb = Jarak\ dari\ B\ ke\ D = 30$
$Vc = Jarak\ dari\ C\ ke\ A = 20$
$Vd = Jarak\ dari\ D\ ke\ C = 31$

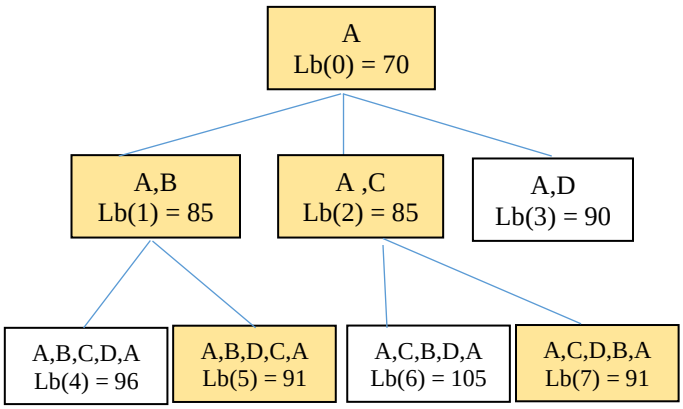
$Lb(5) = 10 + 30 + 20 + 31 = 91$
Jadi lb(5) A,B,D,C,A adalah 91.

Cabang 6 A,C,B (A,C,B,D,A)
$Va = Jarak\ dari\ A\ ke\ C = 20$
$Vb = Jarak\ dari\ B\ ke\ D = 30$
$Vc = Jarak\ dari\ C\ ke\ B = 25$
$Vd = Jarak\ dari\ D\ ke\ A = 30$

$Lb(6) = 20 + 35 + 25 + 30 = 105$
Jadi lb(6) A,C,B,D,A adalah 105.

Cabang 7 A,C,D (A,C,D,B,A)
$Va = Jarak\ dari\ A\ ke\ C = 20$
$Vb = Jarak\ dari\ B\ ke\ A = 10$
$Vc = Jarak\ dari\ C\ ke\ D = 31$
$Vd = Jarak\ dari\ D\ ke\ B = 30$

$Lb(7) = 20 + 10 + 31 + 30 = 91$
Jadi lb(7) A,C,D,B,A adalah 91.



Jadi, karena semua titik sudah dilewati dan kembali lagi ke titik awal maka kita memilih cabang dengan nilai batas bawah (*lower bound*) yang paling kecil. yaitu melalui rute titik kandang A – B – D – C- A dengan total jarak 91 meter atau rute kebalikannya A – C – D – B – A dengan total jarak 91 meter juga.

c) Pseudocode Algoritma *Branch and Bound* kebun Binatang Nirmala (Data 1)

```
import java.util.Arrays;

public class TSP_BranchAndBound_Data2 {
   //Mendefinisikan jumlah kandang
   static int N = 4;
   //Array untuk menyimpan jalur terpendek saat ini
   static int final_path[] = new int[N + 1];
   //Array untuk melacak kandang telah dikunjungi atau belum.
   static boolean visited[] = new boolean[N];
   //untuk menyimpan total biaya jalur terpendek
   static int final_res = Integer.MAX_VALUE;
   //untuk menyimpan waktu mulai dan waktu selesai
   static long startTime, endTime;

   static void copyToFinal(int curr_path[]) {
      for (int i = 0; i < N; i++)
         final_path[i] = curr_path[i] + 1; // Ubah indeks menjadi
1-based (A, B, C, D)
      final_path[N] = curr_path[0] + 1; // Ubah indeks menjadi 1-
based (A, B, C, D)
   }

   //untuk mencari nilai minimum pertama dalam baris i dari
matriks adj.
   static int firstMin(int adj[][], int i) {
      int min = Integer.MAX_VALUE;
      for (int k = 0; k < N; k++)
         if (adj[i][k] < min && i != k)
            min = adj[i][k];
      return min;
   }
```

```
    //untuk mencari nilai minimum kedua dalam baris i dari
matriks adj
    static int secondMin(int adj[][], int i) {
        int first = Integer.MAX_VALUE, second =
Integer.MAX_VALUE;
        for (int j = 0; j < N; j++) {
            if (i == j)
                continue;

            if (adj[i][j] <= first) {
                second = first;
                first = adj[i][j];
            } else if (adj[i][j] <= second && adj[i][j] != first)
                second = adj[i][j];
        }
        return second;
    }

    //pendekatan Branch and Bound untuk mencari jalur terpendek
    static void TSPRec(int adj[][], int curr_bound, int curr_weight,
int level, int curr_path[]) {
        if (level == N) {
            if (adj[curr_path[level - 1]][curr_path[0]] != 0) {
                int curr_res = curr_weight + adj[curr_path[level -
1]][curr_path[0]];
                if (curr_res < final_res) {
                    copyToFinal(curr_path);
                    final_res = curr_res;
                }
            }
            return;
        }

        for (int i = 0; i < N; i++) {
            if (adj[curr_path[level - 1]][i] != 0 && visited[i] == false)
{
                int temp = curr_bound;
                curr_weight += adj[curr_path[level - 1]][i];

                if (level == 1)
                    curr_bound -= ((firstMin(adj, curr_path[level - 1]) +
firstMin(adj, i)) / 2);
                else
                    curr_bound -= ((secondMin(adj, curr_path[level -
1]) + firstMin(adj, i)) / 2);

                if (curr_bound + curr_weight < final_res) {
                    curr_path[level] = i;
                    visited[i] = true;
                    TSPRec(adj, curr_bound, curr_weight, level + 1,
curr_path);
                }

                curr_weight -= adj[curr_path[level - 1]][i];
                curr_bound = temp;
                Arrays.fill(visited, false);
                for (int j = 0; j <= level - 1; j++)
                    visited[curr_path[j]] = true;
            }
        }
    }

    // Fungsi utama untuk memulai algoritma TSP
    static void TSP(int adj[][]) {
        int curr_path[] = new int[N + 1];
        int curr_bound = 0;
        Arrays.fill(curr_path, -1);
        Arrays.fill(visited, false);

        for (int i = 0; i < N; i++)
            curr_bound += (firstMin(adj, i) + secondMin(adj, i));

        curr_bound = (curr_bound == 1) ? curr_bound / 2 + 1 :
curr_bound / 2;

        visited[0] = true;
        curr_path[0] = 0;

        startTime = System.nanoTime(); // Catat waktu mulai

        TSPRec(adj, curr_bound, 0, 1, curr_path);

        endTime = System.nanoTime(); // Catat waktu selesai
    }

    public static void main(String[] args) {
        //Matriks representasi jarak perjalanan antara kandang-
kandang.
        int adj[][] = {
                {0, 10, 15, 20},
                {10, 0, 35, 25},
                {15, 35, 0, 30},
                {20, 25, 30, 0}
        };

        TSP(adj);

        //Mencetak total jarak jalur terpendek.
        System.out.printf("Minimum cost: %d\n", final_res);
        //Mencetak rute untuk jalur terpendek.
        System.out.print("Path Taken: ");
        for (int i = 0; i <= N; i++) {
            System.out.printf("%c ", 'A' + final_path[i] - 1);
        }
        //Menghitung waktu eksekusi algoritma
        long executionTime = endTime - startTime;
        //Mencetak waktu eksekusi dalam milidetik
        System.out.printf("\nExecution Time: %.3f milliseconds",
executionTime / 1000000.0);
    }
}
```

d) Pseudocode Algoritma *Branch and Bound* kebun Binatang Bimasakti (Data 2)

```
import java.util.Arrays;

public class TSP_BranchAndBound_Data2 {
    //Mendefinisikan jumlah kandang
    static int N = 4;
    //Array untuk menyimpan jalur terpendek saat ini
    static int final_path[] = new int[N + 1];
    //Array untuk melacak kandang telah dikunjungi atau belum.
    static boolean visited[] = new boolean[N];
    //untuk menyimpan total biaya jalur terpendek
    static int final_res = Integer.MAX_VALUE;
    //untuk menyimpan waktu mulai dan waktu selesai
    static long startTime, endTime;

    static void copyToFinal(int curr_path[]) {
        for (int i = 0; i < N; i++)
            final_path[i] = curr_path[i] + 1; // Ubah indeks menjadi
1-based (A, B, C, D)
        final_path[N] = curr_path[0] + 1; // Ubah indeks menjadi 1-
based (A, B, C, D)
    }
```

```
    //untuk mencari nilai minimum pertama dalam baris i dari
matriks adj.
    static int firstMin(int adj[][], int i) {
        int min = Integer.MAX_VALUE;
        for (int k = 0; k < N; k++)
            if (adj[i][k] < min && i != k)
                min = adj[i][k];
        return min;
    }

    //untuk mencari nilai minimum kedua dalam baris i dari
matriks adj
    static int secondMin(int adj[][], int i) {
        int first = Integer.MAX_VALUE, second =
Integer.MAX_VALUE;
        for (int j = 0; j < N; j++) {
            if (i == j)
                continue;

            if (adj[i][j] <= first) {
                second = first;
                first = adj[i][j];
            } else if (adj[i][j] <= second && adj[i][j] != first)
                second = adj[i][j];
        }
        return second;
    }

    //pendekatan Branch and Bound untuk mencari jalur terpendek
    static void TSPRec(int adj[][], int curr_bound, int curr_weight,
int level, int curr_path[]) {
        if (level == N) {
            if (adj[curr_path[level - 1]][curr_path[0]] != 0) {
                int curr_res = curr_weight + adj[curr_path[level -
1]][curr_path[0]];
                if (curr_res < final_res) {
                    copyToFinal(curr_path);
                    final_res = curr_res;
                }
            }
            return;
        }

        for (int i = 0; i < N; i++) {
            if (adj[curr_path[level - 1]][i] != 0 && visited[i] == false)
{
                int temp = curr_bound;
                curr_weight += adj[curr_path[level - 1]][i];

                if (level == 1)
                    curr_bound -= ((firstMin(adj, curr_path[level - 1]) +
firstMin(adj, i)) / 2);
                else
                    curr_bound -= ((secondMin(adj, curr_path[level - 1])
+ firstMin(adj, i)) / 2);

                if (curr_bound + curr_weight < final_res) {
                    curr_path[level] = i;
                    visited[i] = true;
                    TSPRec(adj, curr_bound, curr_weight, level + 1,
curr_path);
                }

                curr_weight -= adj[curr_path[level - 1]][i];
                curr_bound = temp;
                Arrays.fill(visited, false);
                for (int j = 0; j <= level - 1; j++)
                    visited[curr_path[j]] = true;
            }
        }
    }

    // Fungsi utama untuk memulai algoritma TSP
    static void TSP(int adj[][]) {
        int curr_path[] = new int[N + 1];
        int curr_bound = 0;
        Arrays.fill(curr_path, -1);
        Arrays.fill(visited, false);

        for (int i = 0; i < N; i++)
            curr_bound += (firstMin(adj, i) + secondMin(adj, i));

        curr_bound = (curr_bound == 1) ? curr_bound / 2 + 1 :
curr_bound / 2;

        visited[0] = true;
        curr_path[0] = 0;

        startTime = System.nanoTime(); // Catat waktu mulai

        TSPRec(adj, curr_bound, 0, 1, curr_path);

        endTime = System.nanoTime(); // Catat waktu selesai
    }

    public static void main(String[] args) {
        //Matriks representasi jarak perjalanan antara kandang-
kandang.
        int adj[][] = {
                { 0, 10, 20, 30 },
                { 10, 0, 25, 30 },
                { 20, 25, 0, 31 },
                { 30, 30, 31, 0 }
        };

        TSP(adj);

        //Mencetak total jarak jalur terpendek.
        System.out.printf("Minimum cost: %d\n", final_res);
        //Mencetak rute untuk jalur terpendek.
        System.out.print("Path Taken: ");
        for (int i = 0; i <= N; i++) {
            System.out.printf("%c ", 'A' + final_path[i] - 1);
        }
        //Menghitung waktu eksekusi algoritma
        long executionTime = endTime - startTime;
        //Mencetak waktu eksekusi dalam milidetik
        System.out.printf("\nExecution Time: %.3f milliseconds",
executionTime / 1000000.0);
    }
}
```

2. Algoritma *Greedy*

Untuk melakukan perhitungan manual menggunakan algoritma *Greedy* pada data Kebun Binatang, kita akan mencari jalur minimum untuk mengunjungi semua kandang (A, B, C, dan D) dan kembali ke kandang awal (A) dengan memilih langkah terbaik berdasarkan jarak terdekat pada setiap langkah.

Langkah pertama adalah memulai dari kandang awal (A) dan mencari kandang terdekat yang belum dikunjungi. Kemudian, kita akan mengunjungi kandang tersebut dan tandai sebagai sudah dikunjungi. Langkah ini diulang sampai

kita telah mengunjungi semua kandang dan kembali ke kandang awal.

a) Perhitungan Manual kebun binarang Nirmala
Matriks jarak kandang pada kebun Binatang Nirmala :
{ -1, 10, 15, 20 },
{ 10, -1, 35, 25 },
{ 15, 35, -1, 30 },
{ 20, 25, 30, -1 }

Langkah 1: Memulai dari kandang awal A
Jarak dari A ke B = 10
Jarak dari A ke C = 15
Jarak dari A ke D = 20
Karena jarak terdekat adalah 10 (ke kandang B), kita akan pergi ke kandang B (A - B).

Langkah 2: Kita sekarang berada di kandang B, tandai sebagai sudah dikunjungi.
Jarak dari B ke A = 10 (kembali ke kandang awal)
Jarak dari B ke C = 35
Jarak dari B ke D = 25
Karena kandang A sudah dikunjungi dan jarak terdekat adalah 25 (ke kandang D), kita akan pergi ke kandang D (A - B - D).

Langkah 3: Kita sekarang berada di kandang D, tandai sebagai sudah dikunjungi.
Jarak dari D ke A = 20 (kembali ke kandang awal)
Jarak dari D ke B = 25
Jarak dari D ke C = 30
Karena kandang A dan B sudah dikunjungi, kita harus pergi ke kandang C karena jarak terdekat adalah 30 (ke kandang C) dan kita harus kembali ke kandang awal (A) setelah mengunjungi kandang C (A - B - D - C).

Langkah 4: Kita sekarang berada di kandang C, tandai sebagai sudah dikunjungi.
Jarak dari C ke A = 15 (kembali ke kandang awal)
Jarak dari C ke B = 35
Jarak dari C ke D = 30
Karena semua kandang sudah dikunjungi dan kembali ke kandang awal (A) memiliki jarak 15, maka jalur minimum yang ditemukan dengan algoritma *Greedy* adalah A - B - D - C - A dengan total jarak 80 meter.

Solusi optimal adalah: A – B – D – C – A dengan total jarak 80 meter, maka algoritma *greedy* memberikan solusi optimal untuk persoalan TSP kebun binatang Nirmala.

b) Perhitungan Manual kebun Binatang Bimasakti
Matriks jarak kandang pada kebun Binatang Bimasakti :
{ -1, 10, 20, 30 },
{ 10, -1, 25, 30 },
{ 20, 25, -1, 31 },
{ 30, 30, 31, -1 }

Langkah 1: Memulai dari kandang awal A
Jarak dari A ke B = 10
Jarak dari A ke C = 20
Jarak dari A ke D = 30
Karena jarak terdekat adalah 10 (ke kandang B), kita akan pergi ke kandang B (A - B).

Langkah 2: Kita sekarang berada di kandang B, tandai sebagai sudah dikunjungi.
Jarak dari B ke A = 10 (kembali ke kandang awal)
Jarak dari B ke C = 25
Jarak dari B ke D = 30
Karena kandang A sudah dikunjungi dan jarak terdekat adalah 25 (ke kandang C), kita akan pergi ke kandang C (A - B - C).

Langkah 3: Kita sekarang berada di kandang C, tandai sebagai sudah dikunjungi.
Jarak dari C ke A = 20 (kembali ke kandang awal)
Jarak dari C ke B = 25
Jarak dari C ke D = 31
Karena kandang A dan B sudah dikunjungi, kita harus pergi ke kandang D karena jarak terdekat adalah 31 (ke kandang D) dan kita harus kembali ke kandang awal (A) setelah mengunjungi kandang D (A - B - C - D).

Langkah 4: Kita sekarang berada di kandang D, tandai sebagai sudah dikunjungi.
Jarak dari D ke A = 30 (kembali ke kandang awal)
Jarak dari D ke B = 30
Jarak dari D ke C = 31
Karena semua kandang sudah dikunjungi dan kembali ke kandang awal (A) memiliki jarak 30, maka jalur minimum yang ditemukan dengan algoritma Greedy adalah A - B - C - D - A dengan total jarak 96 meter.

Solusi optimal adalah: A – B – D – C – A dengan total jarak 91 meter, sedangkan algoritma greedy A - B - C - D - A dengan total jarak 96 meter maka algoritma greedy tidak memberikan solusi optimal untuk persoalan TSP kebun binatang Bimasakti.

c) Pseudocode Algoritma *Greedy* Kebun Binatang Nirmala (Data 1)

```
import java.util.ArrayList;
import java.util.List;

public class TSP_Greedy_Data2 {
   //melacak kota yang telah dikunjungi
   static List<Integer> visitedRouteList = new ArrayList<>();
   //menyimpan jalur terpendek
   static char[] route;

   //menemukan jalur terpendek
   static void findMinRoute(int[][] tsp) {
      int sum = 0;
      int counter = 0;
      int j = 0, i = 0;
      int min = Integer.MAX_VALUE;
      visitedRouteList.add(0);
      route = new char[tsp.length];

      while (i < tsp.length && j < tsp[i].length) {
         if (counter >= tsp[i].length - 1) {
            break;
         }

         if (j != i && !(visitedRouteList.contains(j))) {
            if (tsp[i][j] < min) {
               min = tsp[i][j];
```

```
                route[counter] = (char) ('A' + j);
            }
        }
        j++;

        if (j == tsp[i].length) {
            sum += min;
            min = Integer.MAX_VALUE;
            visitedRouteList.add(route[counter] - 'A');
            j = 0;
            i = route[counter] - 'A';
            counter++;
        }
    }

    i = route[counter - 1] - 'A';

    for (j = 0; j < tsp.length; j++) {
        if ((i != j) && tsp[i][j] < min) {
            min = tsp[i][j];
            route[counter] = (char) ('A' + j);
        }
    }
    sum += min;
    //Mencetak nilai total jarak minimum.
    System.out.print("Minimum Cost is : ");
    System.out.println(sum);
}

public static void main(String[] args) {
    ////Matriks jarak perjalanan tsp diinisialisasi.
    int[][] tsp = {
            { -1, 10, 15, 20 },
            { 10, -1, 35, 25 },
            { 15, 35, -1, 30 },
            { 20, 25, 30, -1 }
    };

    long startTime = System.nanoTime();
    findMinRoute(tsp);
    long endTime = System.nanoTime();

    long executionTime = endTime - startTime;

    //Mencetak label untuk jalur terpendek.
    System.out.print("Path Taken: ");
    System.out.print("A ");
    for (int i = 0; i < route.length; i++) {
        System.out.print(route[i] + " ");
    }
    System.out.println();
    //Mencetak waktu eksekusi dalam milidetik
    System.out.printf("Execution Time: %.3f milliseconds\n",
executionTime / 1000000.0);
    }
}
```

d) Pseudocode Algoritma *Greedy* Kebun Binatang Bimasakti (Data 2)

```
import java.util.ArrayList;
import java.util.List;

public class TSP_Greedy_Data2 {
    //melacak kota yang telah dikunjungi
    static List<Integer> visitedRouteList = new ArrayList<>();
    //menyimpan jalur terpendek
    static char[] route;

    //menemukan jalur terpendek
    static void findMinRoute(int[][] tsp) {
        int sum = 0;
        int counter = 0;
        int j = 0, i = 0;
        int min = Integer.MAX_VALUE;
        visitedRouteList.add(0);
        route = new char[tsp.length];

        while (i < tsp.length && j < tsp[i].length) {
            if (counter >= tsp[i].length - 1) {
                break;
            }

            if (j != i && !(visitedRouteList.contains(j))) {
                if (tsp[i][j] < min) {
                    min = tsp[i][j];
                    route[counter] = (char) ('A' + j);
                }
            }
            j++;

            if (j == tsp[i].length) {
                sum += min;
                min = Integer.MAX_VALUE;
                visitedRouteList.add(route[counter] - 'A');
                j = 0;
                i = route[counter] - 'A';
                counter++;
            }
        }

        i = route[counter - 1] - 'A';

        for (j = 0; j < tsp.length; j++) {
            if ((i != j) && tsp[i][j] < min) {
                min = tsp[i][j];
                route[counter] = (char) ('A' + j);
            }
        }
        sum += min;
        //Mencetak nilai total jarak minimum.
        System.out.print("Minimum Cost is : ");
        System.out.println(sum);
    }

    public static void main(String[] args) {
        ////Matriks jarak perjalanan tsp diinisialisasi.
        int[][] tsp = {
                { -1, 10, 20, 30 },
                { 10, -1, 25, 30 },
                { 20, 25, -1, 31 },
                { 30, 30, 31, -1 }
        };

        long startTime = System.nanoTime();
        findMinRoute(tsp);
        long endTime = System.nanoTime();

        long executionTime = endTime - startTime;

        //Mencetak label untuk jalur terpendek.
        System.out.print("Path Taken: ");
        System.out.print("A ");
        for (int i = 0; i < route.length; i++) {
            System.out.print(route[i] + " ");
        }
```

```
        System.out.println();
        //Mencetak waktu eksekusi dalam milidetik
        System.out.printf("Execution Time: %.3f milliseconds\n",
executionTime / 1000000.0);
    }
}
```

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Selain perbedaan dalam hasil dan waktu eksekusi, terdapat beberapa tambahan informasi penting yang perlu diperhatikan terkait algoritma *Greedy* dan *Branch and Bound* dalam konteks penyelesaian persoalan TSP:

1. Algoritma *Greedy*:
   a) Algoritma *Greedy* merupakan pendekatan yang sederhana dan cepat untuk mencari solusi dalam perhitungan manual untuk persoalan TSP. Ia selalu memilih langkah terbaik dalam setiap langkahnya berdasarkan kriteria lokal, seperti jarak terdekat dari titik saat ini ke titik selanjutnya.
   b) Kelemahan algoritma *Greedy* adalah ia tidak dapat menjamin mencari solusi yang optimal dalam semua kasus TSP. Pendekatan "ambil langkah terbaik saat ini" tidak selalu mengarah pada jalur yang secara keseluruhan menghasilkan jalur minimum.
   c) Meskipun demikian, algoritma Greedy memiliki kelebihan dalam kecepatan menghitung secara manual yang lebih cepat dibandingkan dengan algoritma yang lebih kompleks seperti *Branch and Bound*. Oleh karena itu, algoritma Greedy sering digunakan untuk mencari solusi yang cukup baik secara cepat dengan perhitungan manual, terutama untuk masalah dengan jumlah titik yang lebih besar.
2. Algoritma *Branch and Bound*:
   a) Algoritma *Branch and Bound* merupakan pendekatan *eksak* atau *brute-force* dalam mencari solusi TSP. Ia mencoba semua kemungkinan kombinasi jalur untuk menemukan jalur minimum.
   b) Kelebihan utama dari algoritma *Branch and Bound* adalah kemampuannya untuk menjamin solusi optimal dalam kasus TSP. Karena mencoba semua kemungkinan jalur, ia dapat menemukan jalur minimum tanpa mengorbankan keakuratan hasil.
   c) Namun, kelemahan dari algoritma *Branch and Bound* adalah waktu menghitung secara manual cenderung lebih lama dan kompleksitas yang lebih tinggi, terutama ketika jumlah titik dalam masalah TSP semakin besar.

Kesimpulannya, pemilihan antara algoritma *Greedy* dan *Branch and Bound* harus dipertimbangkan berdasarkan kebutuhan spesifik dan karakteristik masalah TSP yang dihadapi. Jika akurasi hasil yang optimal menjadi prioritas utama dan jumlah titik dalam masalah tidak terlalu besar, algoritma *Branch and Bound* adalah pilihan yang tepat. Namun, jika kecepatan perhitungan secara manual lebih penting dan hasil yang cukup baik sudah mencukupi, algoritma *Greedy* dapat memberikan solusi dengan waktu perhiutngan manual yang lebih cepat meskipun tidak menjamin optimalitas hasil. Selalu perlu melakukan analisis dan uji coba untuk memahami performa kedua algoritma dalam berbagai kasus dan kondisi.

B. Saran

Adapun saran yang penulis berikan untuk pengembangan penelitian ini sebagai berikut :

1. Berdasarkan hasil percobaan, algoritma *Branch and Bound* lebih efisien dalam menemukan jalur minimum pada kedua jenis kebun binatang (Nirmala dan Bimasakti). Oleh karena itu, disarankan untuk menggunakan algoritma *Branch and Bound* sebagai pilihan utama dalam menyelesaikan masalah *Travelling Salesman Problem* pada kasus kebun binatang.
2. Perhatikan bahwa hasil dan waktu eksekusi algoritma dapat dipengaruhi oleh kompleksitas data dan jumlah titik yang harus dikunjungi. Sebaiknya lakukan pengujian lebih lanjut dengan data yang lebih besar dan beragam untuk mendapatkan gambaran yang lebih komprehensif tentang kinerja kedua algoritma.

REFERENCES

[1] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[2] W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[3] H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.

[4] B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[5] E. H. Miller, "A note on reflector arrays (Periodical style—Accepted for publication)," *IEEE Trans. Antennas Propagat.*, to be published.

[6] J. Wang, "Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication)," *IEEE J. Quantum Electron.*, submitted for publication.

[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

PERNYATAAN

Dengan ini saya menyatakan bahwa *paper* yang kami tulis ini adalah tulisan kami sendiri, bukan saduran atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Purwokerto, 20 Juli 2023

Rahma Setiani (21102304)

Daniya Auly Yasmin (21102041)