TUGAS AKHIR – EF234801

PENERAPAN STRUKTUR K-DIMENSIONAL TREE PADA PENYELESAIAN PERSOALAN DOMINASI PARETO

THOMAS JUAN MAHARDIKA SURYONO

NRP 5025211016

Dosen Pembimbing I

Rully Soelaiman, S.Kom., M.Kom.

NIP. 197002131994021001

Dosen Pembimbing II

Dr. Yudhi Purwananto, S.Kom., M.Kom.

NIP. 197007141997031002

Program Studi S-1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – EF234801

PENERAPAN STRUKTUR K-DIMENSIONAL TREE PADA PENYELESAIAN PERSOALAN DOMINASI PARETO

THOMAS JUAN MAHARDIKA SURYONO

NRP 5025211016

Dosen Pembimbing I

Rully Soelaiman, S.Kom., M.Kom.

NIP. 197002131994021001

Dosen Pembimbing II

Dr. Yudhi Purwananto, S.Kom., M.Kom.

NIP. 197007141997031002

Program Studi S-1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

(Halaman ini sengaja dikosongkan)

FINAL PROJECT – EF234801

K-DIMENSIONAL TREE STRUCTURE APPLICATION ON PARETO DOMINATION PROBLEM SOLUTION

THOMAS JUAN MAHARDIKA SURYONO

NRP 5025211016

Advisor I

Rully Soelaiman, S.Kom., M.Kom.

NIP. 197002131994021001

Advisor II

Dr. Yudhi Purwananto, S.Kom., M.Kom.

NIP. 197007141997031002

Study Program Bachelor of Informatics

Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2025

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

PENERAPAN STRUKTUR K-DIMENSIONAL TREE PADA PENYELESAIAN PERSOALAN DOMINASI PARETO

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer pada
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh: **THOMAS JUAN MAHARDIKA SURYONO**

NRP. 5025211016

Disetujui oleh Tim Penguji Tugas Akhir:

1. Rully Soelaiman, S.Kom., M.Kom.
   NIP. 197002131994021001
   ..... 30/1 25
   Pembimbing

2. Dr. Yudhi Purwananto, S.Kom., M.Kom.
   NIP. 197007141997031002
   .....
   Ko-pembimbing

3. Ir. Suhadi Lili, M.T.I.
   NIP. 196907281993031001
   .....
   Penguji I

4. Fajar Baskoro, S.Kom., M.T.
   NIP. 197404031999031002
   .....
   Penguji

SURABAYA

Januari, 2025

(Halaman ini sengaja dikosongkan)

APPROVAL SHEET

K-DIMENSIONAL TREE STRUCTURE APPLICATION ON PARETO DOMINATION PROBLEM SOLUTION

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a bachelor degree at
Undergraduate Study Program of Informatics
Department of Informatics
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember

Oleh: **THOMAS JUAN MAHARDIKA SURYONO**

NRP. 5025211016

Approved by Final Project Examiner Team:

1. Rully Soelaiman, S.Kom., M.Kom.
   NIP. 197002131994021001
   30/1/25
   .....
   Advisor
2. Dr. Yudhi Purwananto, S.Kom., M.Kom.
   NIP. 197007141997031002
   .....
   Co-Advisor
3. Ir. Suhadi Lili, M.T.I.
   NIP. 196907281993031001
   .....
   Examiner I
4. Fajar Baskoro, S.Kom., M.T.
   NIP. 197404031999031002
   .....
   Examiner II

SURABAYA

January, 2025

(Halaman ini sengaja dikosongkan)

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP	:	Thomas Juan Mahardika Suryono / 5025211016
Departemen	:	Teknik Informatika
Dosen Pembimbing I / NIP	:	Rully Soelaiman, S.Kom., M.Kom. / 197002131994021001
Dosen Pembimbing II / NIP	:	Dr. Yudhi Purwananto, S.Kom., M.Kom. / 197007141997031002

Dengan ini menyatakan bahwa tugas akhir dengan judul "Penerapan Struktur K-dimensional Tree pada Penyelesaian Persoalan Dominasi Pareto" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 30 Januari 2025

Mahasiswa

Thomas Juan Mahardika Suyono

NRP. 5025211016

Mengetahui,

Dosen Pembimbing I	Dosen Pembimbing II
Rully Soelaiman, S.Kom., M.Kom.	Dr. Yudhi Purwananto, S.Kom., M.Kom.
NIP. 197002131994021001	NRP. 197007141997031002

(Halaman ini sengaja dikosongkan)

STATEMENT OF ORIGINALITY

The undersigned below:

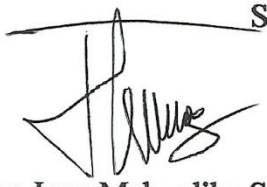
Name of student / NRP	:	Thomas Juan Mahardika Suryono / 5025211016
Department	:	Teknik Informatika
Advisor I / NIP	:	Rully Soelaiman, S.Kom., M.Kom. / 197002131994021001
Advisor II / NIP	:	Dr. Yudhi Purwananto, S.Kom., M.Kom. / 197007141997031002

Hereby declare that undergraduate thesis with the title of "K-dimensional Tree Structure Application on Pareto Domination Problem Solution" is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Sepuluh Nopember Institute of Technology.

Surabaya, 30 January 2025

Student

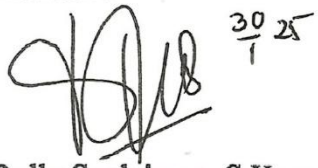


Thomas Juan Mahardika Suyono

NRP. 5025211016

Acknowledge,

Advisor I



Rully Soelaiman, S.Kom., M.Kom.

NIP. 197002131994021001

Advisor II

Dr. Yudhi Purwananto, S.Kom., M.Kom.

NRP. 197007141997031002

(Halaman ini sengaja dikosongkan)

ABSTRAK

PENERAPAN STRUKTUR K-DIMENSIONAL TREE PADA PENYELESAIAN PERSOALAN DOMINASI PARETO

Nama Mahasiswa / NRP	**:**	**Thomas Juan Mahardika Suryono / 5025211016**
Departemen	**:**	**Teknik Informatika FTEIC - ITS**
Dosen Pembimbing I	**:**	**Rully Soelaiman, S.Kom., M.Kom.**
Dosen Pembimbing II	**:**	**Dr. Yudhi Purwananto, S.Kom., M.Kom.**

Abstrak

Eolymp merupakan situs penilaian kode daring yang berasal dari Ukraina. Eolymp memiliki berbagai soal dengan berbagai tingkat kesulitan yang dapat diselesaikan pengguna dengan menggunakan kode. Studi kasus yang digunakan pada tugas akhir ini terdapat pada Eolymp kode soal 100 berjudul "Pareto Domination" yang memiliki tautan https://basecamp.eolymp.com/en/problems/100.

Sebuah titik dengan koordinat (x_0, x_1, …, x_{M-1}) dikatakan didominasi oleh sebuah titik dengan koordinat (y_0, y_1, …, y_{M-1}) jika $x_i \leq y_i$ untuk setiap i ($0 \leq i \leq M$-1), dengan M merupakan dimensi. Konsep dominasi ini sama seperti dominasi Pareto. Diberikan himpunan titik. Titik x merupakan titik tidak didominasi jika titik x tidak didominasi oleh titik lain manapun dalam himpunan. Tujuan dari studi kasus ini adalah menemukan banyak titik tidak didominasi dari himpunan titik yang diberikan.

Persoalan dominasi Pareto termasuk kasus *NP-hard* karena perlu mereduksi kompleksitas kuadratik agar dapat memenuhi batasan waktu eksekusi, yakni 10 detik. Solusi menggunakan penerapan *k-d tree* (*k-dimensional tree*) yang dikombinasikan dengan teknik *bounding box* dan pendekatan dua fase.

Berdasarkan hasil uji coba, solusi penerapan *k-d tree* mampu menyelesaikan persoalan dalam waktu 326 milidetik hingga 329 milidetik dengan penggunaan memori antara 4,15 MB dan 4,16 MB pada 22 September 2024.

Kata kunci: ***Dominasi Pareto, K-dimensional Tree, Pendekatan Dua Fase***.

(Halaman ini sengaja dikosongkan)

ABSTRACT

K-DIMENSIONAL TREE STRUCTURE APPLICATION ON PARETO DOMINATION PROBLEM SOLUTION

Student Name / NRP	**:**	**Thomas Juan Mahardika Suryono / 5025211016**
Department	**:**	**Informatics Engineering FTEIC - ITS**
Advisor I	**:**	**Rully Soelaiman, S.Kom., M.Kom.**
Advisor II	**:**	**Dr. Yudhi Purwananto, S.Kom., M.Kom.**

Abstract

Eolymp is an online code judge site originating from Ukraine. Eolymp offers a variety of problems with different levels of difficulty that users can solve using code. The case study used in this final project is found on the Eolymp under problem code 100, titled “Pareto Domination”, with the link: https://basecamp.eolymp.com/en/problems/100.

A point with coordinates (x_0, x_1, …, x_{m-1}) is said to be dominated by a point with coordinates (y_0, y_1, …, y_{m-1}) if $x_i \leq y_i$ for every i ($0 \leq i \leq M-1$), where M represents the number of dimensions. This concept of domination is the same as Pareto domination. Given a set of points, a point x is considered a non-dominated point if it is not dominated by any other point in the set. The goal of this case study is to determine the number of non-dominated points in the given set of points.

The Pareto domination problem is NP-hard because it requires reducing quadratic complexity to meet the execution time limit of 10 seconds. The solution involves applicating k-d tree (k-dimensional tree) combined with the bounding box technique and two-phase approach.

Based on test results, the k-d tree application was able to solve the problem in 326 milliseconds to 329 milliseconds, with memory usage ranging from 4,15 MB to 4.16 MB, as of September 22, 2024.

Keywords: ***Pareto Domination, K-dimensional Tree, Two Phase Approach***.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Dengan nama Bapa Yang Maha Pengasih dan Maha Penyayang, puji syukur penulis panjatkan ke hadirat-Nya atas segala rahmat, petunjuk, serta karunia-Nya yang telah senantiasa melimpah dalam setiap langkah perjalanan hidup, serta kepada Tuhan Yesus Kristus yang menjadi teladan bagi seluruh umat sehingga penulis berhasil menyelesaikan tugas akhir dan merampungkan laporan akhir dalam bentuk buku ini.

Penulisan buku ini dilakukan dengan tujuan mendalami lebih lanjut topik-topik yang kurang dijelaskan secara mendalam pada kurikulum kampus, namun menarik untuk menjadi perhatian bagi penulis. Penulis berharap dengan menyelesaikan tugas akhir dan buku ini dapat menjadi langkah awal yang bermanfaat dalam perjalanan hidup penulis dalam menambah wawasan dan pengetahuan.

Dengan kesempatan ini, penulis ingin mengucapkan terima kasih kepada semua pihak yang telah berkontribusi besar selama proses penulisan tugas akhir ini dan selama menempuh studi, yaitu:

1. Bapak Rully Soelaiman, S.Kom., M.Kom., selaku pembimbing utama penulis, yang selalu memberikan perhatian, arahan, dukungan, pengajaran, nasihat, dan doa dengan tiada hentinya agar penulis dapat menjadi pribadi yang cerdas, berbudi pekerti, saleh, dan berhasil dalam dunia maupun akhirat.

2. Bapak Dr. Yudhi Purwananto, S.Kom., M.Kom., selaku ko-pembimbing, atas perhatian, didikan, pengajaran, nasihat, dan kesempatannya di bawah bimbingan beliau selama studi terutama saat pengerjaan tugas akhir ini.

3. Keluarga, terutama Ayah, Ibu, dan adik-adik saya yang senantiasa memberikan dukungan, perhatian, dan kasih sayang sebagai penyemangat selama perkuliahan dan pengerjaan tugas akhir.

4. Serta seluruh pihak lainnya yang tidak dapat disebutkan satu per satu, namun telah memberikan banyak dukungan selama penulisan tugas akhir ini.

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SEMU

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I PENDAHULUAN

Pada bab ini, dibahas mengenai latar belakang yang mendasari tugas akhir ini. Latar belakang tersebut berisi deskripsi dan identifikasi permasalahan. Kemudian, dirumuskan permasalahan beserta batasannya, tujuan, dan manfaat tugas akhir.

1.1 Latar Belakang

Eolymp merupakan situs penilaian kode daring yang berasal dari Ukraina. Eolymp memiliki berbagai soal dengan berbagai tingkat kesulitan yang dapat diselesaikan pengguna dengan menggunakan kode. Jika pengguna mengajukan kode solusi, Eolymp memberikan status umpan balik berupa "Accepted" jika kode benar, "Wrong Answer" jika kode salah, atau "Runtime Error" jika kode gagal dijalankan. Hingga tugas akhir ini selesai disusun, Eolymp memiliki tautan https://basecamp.eolymp.com/en. Berdasarkan tingkat kesulitannya, soal pada Eolymp terbagi menjadi lima kategori, yakni *very easy*, *easy*, *medium*, *hard*, *very hard*. Studi kasus yang digunakan pada tugas akhir ini terdapat pada laman Eolymp kode soal 100 berjudul "Pareto Domination" yang memiliki tautan https://basecamp.eolymp.com/en/problems/100. Soal ini termasuk kategori *hard* dengan *acceptance rate* 23%.

Setiap preferensi individual dapat direpresentasikan dengan fungsi utilitas, U_i. Jika individu i lebih menyukai sebuah kebijakan x daripada kebijakan y, $U_i(x) > U_i(y)$. Jika individu i seimbang antara x dan y, $U_i(x) = U_i(y)$. Sebuah kebijakan x mendominasi Pareto kebijakan lain y jika memenuhi dua kondisi. Pertama, tidak ada individu yang lebih menyukai y daripada x, sehingga untuk setiap i, $U_i(x) \geq U_i(y)$. Kedua, setidaknya terdapat satu individu yang lebih menyukai x daripada y, sehingga untuk setidaknya satu i, $U_i(x) > U_i(y)$. Sebuah kebijakan x merupakan *Pareto efficient* jika tidak ada kebijakan lain yang mendominasi Pareto kebijakan x. Definisi ini diaplikasikan pada ekonomi kesejahteraan, yakni bagaimana upaya untuk menyusun kebijakan yang *Pareto efficient*. Selain itu, juga diaplikasikan pada optimisasi multiobjektif dengan cara menemukan *Pareto front*, yaitu himpunan solusi yang *Pareto efficient*[1].

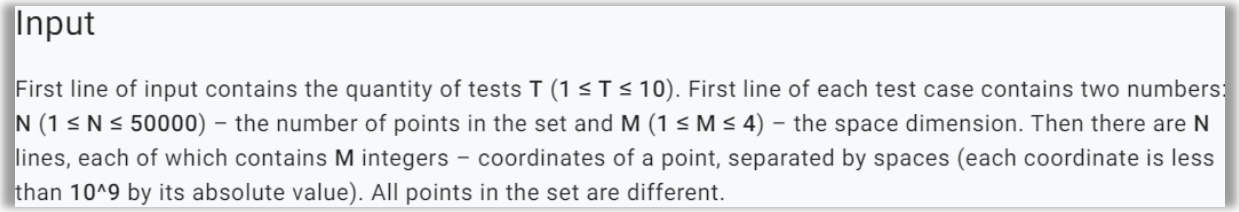
1.1.1 Studi Kasus Eolymp 100 "Pareto Domination"

Berikut dijelaskan konten yang terdapat pada studi kasus Eolymp 100 "Pareto Domination." Berikut deskripsi soal berbahasa Inggris yang disediakan laman pada Gambar 1.1.

A point with coordinates (**x1**, **x2**, ..., **xn**) is called dominated in Pareto's sense by a point with coordinates (**y1**, **y2**, ..., **yn**), if for each **i** (**1 ≤ i ≤ n**) the inequality **xi ≤ yi** holds. A set of some points is given. Your task is to find the number of points in this set, that are not dominated in Pareto's sense by any other point in the given set.

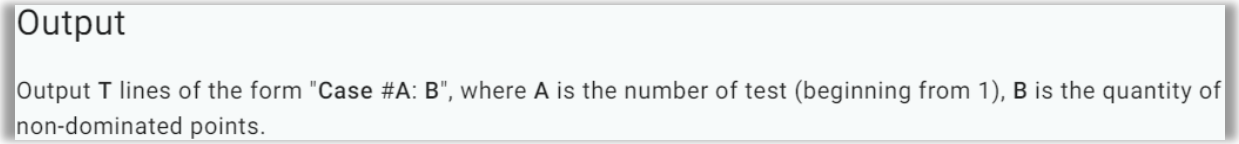
Gambar 1.1 Deskripsi Permasalahan Studi Kasus

Sebuah titik dengan koordinat (x_0, x_1, …, x_{M-1}) dikatakan didominasi oleh sebuah titik dengan koordinat (y_0, y_1, …, y_{M-1}) jika $x_i \leq y_i$ untuk setiap i ($0 \leq i \leq M$-1). Konsep dominasi ini sama seperti dominasi Pareto. Diberikan himpunan titik. Titik x merupakan titik tidak didominasi jika titik x tidak didominasi oleh titik lain manapun dalam himpunan. Tujuan dari studi kasus ini adalah menemukan banyak titik tidak didominasi dari himpunan titik yang diberikan.



Gambar 1.2 Deskripsi Masukan Studi Kasus

Berikut deskripsi masukan sesuai Gambar 1.2. Baris pertama memuat banyak kasus uji T ($1 \leq T \leq 10$). Baris pertama dari tiap kasus uji memuat dua bilangan, N ($1 \leq N \leq 50000$) yang merupakan banyak titik pada himpunan dan M ($1 \leq M \leq 4$) yang merupakan dimensi ruangnya. Selanjutnya, terdapat N baris yang di tiap barisnya memuat M *integer* yang merupakan titik koordinat, dipisahkan spasi. Setiap koordinat nilainya kurang dari 10^9 terhadap nilai absolutnya. Tidak ada titik yang sama di dalam himpunan.



Gambar 1.3 Deskripsi Keluaran Studi Kasus

Berikut deskripsi keluaran sesuai Gambar 1.3. Keluaran berupa T baris dalam format "Case #A: B", A merupakan urutan kasus uji (dimulai dari satu), B merupakan banyak titik tidak didominasi.

Gambar 1.4 Contoh Masukan dan Keluaran Studi Kasus

Berikut deskripsi untuk contoh masukan dan keluaran pada Gambar 2.4. Baris pertama masukan, yakni T, bernilai dua yang menunjukkan terdapat dua kasus uji. Baris selanjutnya, yakni N dan M, bernilai empat dan satu yang menunjukkan pada kasus uji pertama, terdapat empat titik berdimensi satu. Oleh karena itu, empat baris selanjutnya merupakan himpunan titik berdimensi satu. Keluaran dari kasus uji ini terdapat pada "Case #1: 1" yang menunjukkan bahwa banyak titik tidak didominasi adalah satu, yakni titik berkoordinat (4). Baris masukan selanjutnya bernilai empat dan dua yang menunjukkan pada kasus uji kedua, terdapat empat

titik berdimensi dua sehingga pada empat baris selanjutnya memuat himpunan titik berdimensi dua. Keluaran dari kasus uji ini terdapat pada "Case #2: 3" yang menunjukkan bahwa banyak titik tidak didominasi adalah tiga, yakni titik berkoordinat (1, 1), (2, 0), dan (0, 2).

Jika diringkas, batasan yang terdapat pada studi kasus adalah sebagai berikut.

1. Batas nilai T yang merupakan banyak kasus uji adalah 1 sampai 10.
2. Batas nilai N yang merupakan banyak titik adalah 1 sampai 50000.
3. Batas nilai M yang merupakan dimensi adalah 1 sampai 4.
4. Nilai koordinat bertipe data *integer*.
5. Batas nilai koordinat kurang dari 10^9 terhadap nilai mutlaknya.
6. Tidak ada titik yang sama.
7. Batas waktu eksekusi program adalah 10 detik.
8. Batas memori yang digunakan ketika program dieksekusi adalah 64 MB.
9. Keluaran hanya menampilkan banyak titik tidak didominasi, tanpa menampilkan daftar titik tidak didominasi.

1.1.2 Identifikasi Permasalahan

Melalui observasi mandiri, penulis mengemukakan solusi bahwa untuk persoalan dimensi satu, banyak titik tidak didominasi pasti satu yakni titik berkoordinat terbesar sehingga dapat disebut sebagai keluaran konstan. Pada dimensi dua, solusi yang dikemukakan adalah menggunakan algoritma *greedy*, yakni dengan mengurutkan himpunan titik bertipe data *array of pair* menggunakan fungsi *sort* pada bahasa C++, kemudian diiterasi dari kanan ke kiri dengan menyimpan nilai *second* terbesar. Jika *second* yang ditinjau melebihi *second* yang disimpan, titik tersebut merupakan titik tidak didominasi. Penulis belum berhasil mengemukakan solusi untuk persoalan dimensi tiga dan empat sehingga diperlukan studi literatur lebih lanjut.

Studi kasus Eolymp 100 "Pareto Domination" merupakan persoalan yang dapat dikategorikan *NP-hard* karena persoalan tersebut dapat dan perlu direduksi dari kompleksitas $O(N^2)$ agar dapat memenuhi batasan waktu eksekusi.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah bagaimana persoalan dominasi Pareto dapat diselesaikan?

1.3 Batasan Masalah

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut.

1. Implementasi algoritma menggunakan bahasa C++.
2. Implementasi penyelesaian persoalan dominasi Pareto hanya berfokus pada studi kasus Eolymp 100 "Pareto Domination."
3. Durasi waktu pengerjaan tugas akhir selama 6 bulan.
4. Tugas akhir ini tidak membandingkan solusi studi kasus Eolymp 100 "Pareto Domination" dengan soal lain yang serupa

1.4 Tujuan

Tujuan penulisan tugas akhir ini adalah menyelesaikan persoalan dominasi Pareto.

1.5 Manfaat

Tugas akhir ini diharapkan dapat memberikan gambaran proses berpikir serta menjelaskan algoritma yang berhasil menyelesaikan persoalan dominasi Pareto.

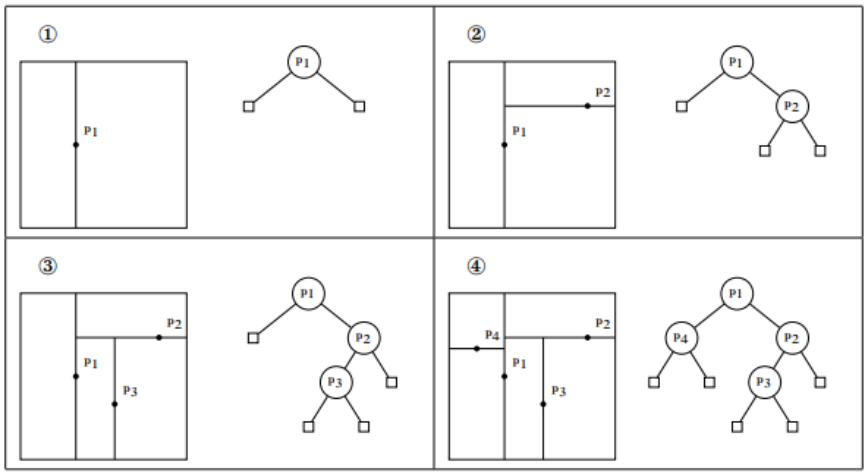
BAB II TINJAUAN PUSTAKA

Pada bab ini, dibahas mengenai dasar teori baik yang digunakan maupun yang tidak jadi digunakan dalam penyusunan tugas akhir. Dasar teori yang terpakai menjadi landasan dalam peracangan dan analisis algoritma.

2.1 Penerapan *K-d Tree*

Metode penerapan *k-d tree* dikemukakan oleh Wei-Mei Chen, Hsien-Kuei Hwang, dan Tsung-Hsi Tsai pada referensi [2] yang berjudul *Maxima-finding algorithms for multidimensional samples: A two-phase approach*. Maxima merupakan himpunan titik tidak didominasi dari himpunan titik sehingga metode ini bertujuan untuk menemukan himpunan tersebut.

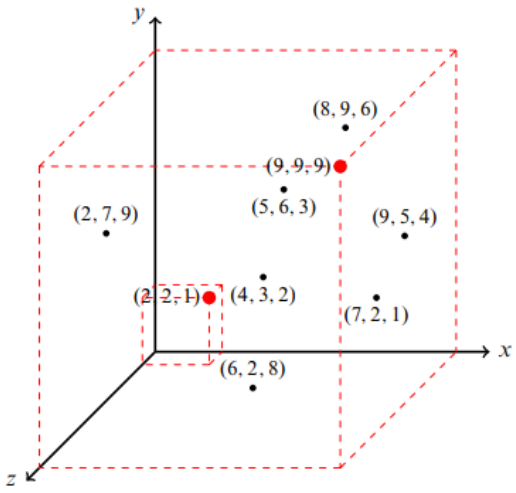
K-d tree merupakan ekstensi dari *binary search tree* untuk data multidimensional. Ketika sebuah titik dimasukkan, *k-d tree* menggunakan setiap dari *M* koordinat secara siklis menurut level *tree* sebagai *discriminator* yang menentukan *subtree* mana yang akan dipilih pada level selanjutnya. Jika sebuah simpul yang menyimpan titik $r = (r_1, \ldots, r_M)$ menggunakan koordinat ke-*i* sebagai *discriminator*, untuk simpul dalam *subtree* dari *r* yang menyimpan titik $w = (w_1, \ldots, w_M)$, maka berelasi $w_i < r_i$ jika *w* berada dalam *subtree* kiri, $w_i \geq r_i$ jika *w* berada dalam *subtree* kanan dari *r*. *Child* dari *r* berganti menggunakan koordinat ke-$(i \bmod M) + 1$ sebagai *discriminator*. Gambar 2.1 merupakan ilustrasi *k-d tree* menggunakan empat titik dengan persegi di sebelah kiri menunjukkan posisi titik pada bidang kartesius dan *tree* di sebelah kanan menunjukkan hasil *k-d tree*.



Gambar 2.1 Ilustrasi *K-d Tree*

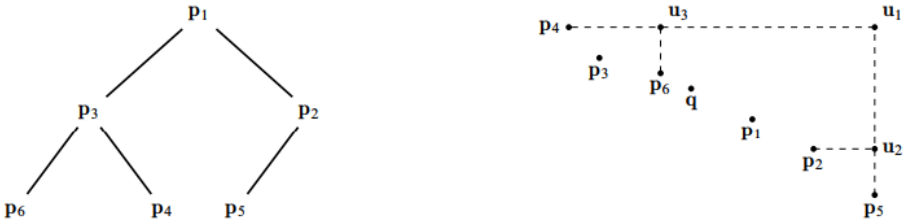
Bounding box merupakan teknik sederhana untuk meningkatkan perfoma berbagai algoritma, terutama yang berkaitan dengan objek geometri berpotongan, yang telah digunakan secara luas dalam berbagai situasi teoretis dan praktis. $u_r = (u_1, \ldots, u_M)$ merupakan *upper bound* dari *subtree* ber-*root r* jika u_i merupakan nilai maksimum koordinat ke-*i* dari semua titik dalam subpohon ber-*root r*. $v_r = (v_1, \ldots, v_M)$ merupakan *lower bound* dari *subtree* ber-*root r* jika v_i merupakan nilai minimum koordinat ke-*i* dari semua titik dalam *subtree* ber-*root r*.

Sebagai contoh, jika sebuah *subtree* memiliki titik {(4, 3, 2), (9, 5, 4), (7, 2, 1), (5, 6, 3), (8, 9, 6), (2, 7, 9), (6, 2, 8)}, maka (9, 9, 9) merupakan *upper bound*-nya dan (2, 2, 1) merupakan *lower bound*-nya. Ilustrasinya terdapat pada Gambar 2.2.



Gambar 2.2 Ilustrasi *Bounding Box*

Jika titik *p* tidak didominasi oleh u_r, maka *p* tidak didominasi oleh sembarang titik dalam *subtree* ber-*root r*. Ini berarti semua perbandingan antara *p* dan semua titik dalam *subtree* ber-*root r* dapat dilewati. Ketika mencari titik dalam *subtree* ber-*root r* yang didominasi oleh *p*, jika v_r tidak didominasi oleh *p*, maka tidak ada titik dalam *subtree* ber-*root r* yang didominasi oleh *p* sehingga perbandingan dapat dilewati.



Gambar 2.3 Ilustrasi Pengecekan Dominasi

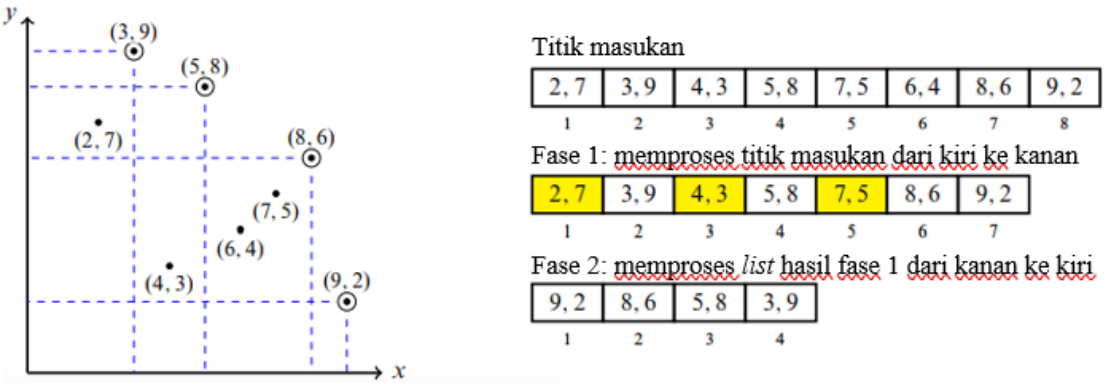
Gambar di atas merupakan ilustrasi dari contoh pengecekan dominasi dengan gambar kiri merupakan *k-d tree* dan gambar kanan merupakan posisi titik dalam bidang kartesius. Terdapat *k-d tree* dengan enam titik p_1, p_2, …, p_6 dan sebuah titik baru *q*. *Upper bound* dari *subtree* ber-*root* pada p_1, p_2, dan p_3 masing-masing adalah u_1, u_2, dan u_3. Untuk mengecek apakah *q* didominasi oleh suatu titik dalam pohon, perbandingan antara *q* dan *subtree* ber-*root* pada p_2 dan p_3 dapat dilewati karena *q* tidak didominasi oleh u_2 dan u_3.

Digunakan pendekatan dua fase yang hanya memerlukan *upper bound* karena di setiap fase hanya mendeteksi apakah titik baru didominasi oleh titik yang tersimpan. Titik tidak hanya disimpan dalam *k-d tree*, tetapi juga dalam *list* untuk menyimpan urutannya.

Titik masukan pertama langsung disimpan ke *k-d tree* dan *list*. Untuk titik masukan selanjutnya hingga titik masukan habis, perlu dilakukan pengecekan apakah titik masukan didominasi oleh salah satu titik yang tersimpan dalam *k-d tree*. Jika tidak, titik masukan dimasukkan pada *k-d tree* dan *list*. Setiap titik masukan dimasukkan, *upper bound* setiap simpul

yang dilalui titik masukan perlu diperbarui dengan nilai maksimal pada setiap koordinat antara *upper bound* awal dan titik masukan. Tahap ini disebut fase pertama.

Pada fase kedua, menggunakan *k-d tree* baru, urutan pengecekan dibalik, yakni dimulai dari akhir *list* dengan titik akhir *list* langsung dimasukkan *k-d tree*, kemudian bergilir dilakukan pengecekan dominasi hingga titik awal *list*. Pada setiap pengecekan, titik yang tidak didominasi dimasukkan *k-d tree*, sedangkan titik yang didominasi dihapus dari *list*. Banyak titik tidak didominasi dapat dihitung dari banyak titik yang terdapat pada *list* setelah seluruh pengecekan selesai.



Gambar 2.4 Ilustrasi Pendekatan Dua Fase

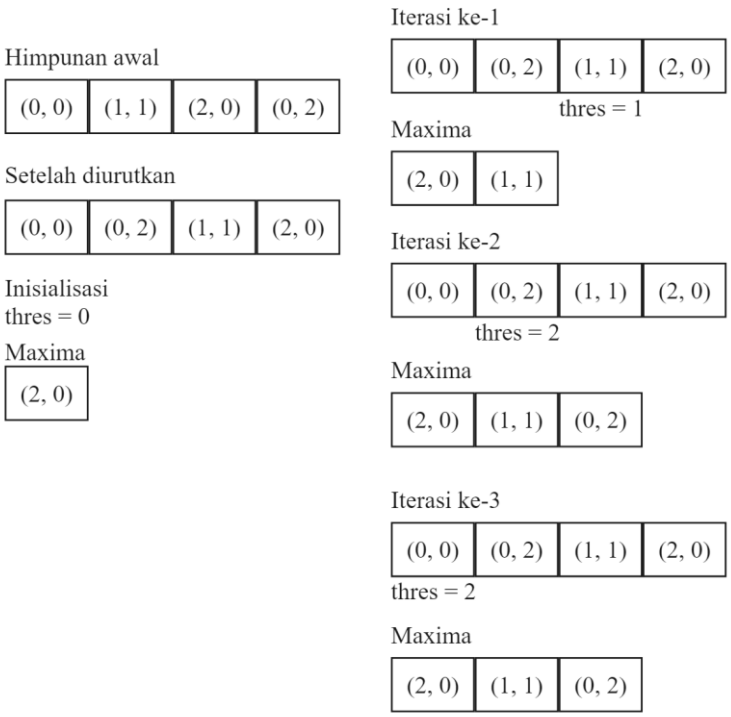
Gambar 2.4 merupakan ilustrasi dari contoh alur pendekatan dua fase. Titik tidak didominasi pada bidang kartesius ditunjukkan dengan titik berlingkaran. Pada hasil akhir fase pertama, masih ada titik yang bukan titik tidak didominasi yang ditunjukkan oleh petak berwarna kuning. Hal ini teratasi pada fase kedua sehingga hasil fase kedua merupakan himpunan titik tidak didominasi.

2.2 Algoritma *Greedy*

Algoritma *greedy* yang khusus untuk menemukan banyak titik tidak didominasi ini dikemukakan oleh H. T. Kung, F. Luccio, dan F. P. Preparata pada referensi [3] yang berjudul *On Finding the Maxima of a Set of Vectors*. Algoritma ini khusus sebagai penyelesaian persoalan dimensi dua.

Semua titik masukan dimasukkan dalam *array of pair* terlebih dahulu. Kemudian, dilakukan proses pengurutan secara *ascending* pada *array of pair* tersebut. Dideklarasikan *maxima* yang menyimpan himpunan titik tidak didominasi dan variabel yang menyimpan nilai maksimal koordinat kedua *thres* dengan nilai awal koordinat kedua titik terbesar pada *array*. Iterasi dimulai dari titik setelah titik terbesar menuju titik terkecil pada *array* dengan pernyataan, jika koordinat kedua > *thres*, maka nilai *thres* diganti dengan koordinat kedua dan titik tersebut dimasukkan *maxima*.

Gambar 2.5 merupakan ilustrasi algoritma *greedy*. Nilai awal *thres* bernilai nol yang berasal dari nilai *second* titik posisi terakhir setelah diurutkan, yakni titik (2, 0). Pada iterasi pertama, $x_1 >$ *thres* sehingga *thres* menjadi sama dengan satu dan titik tersebut masuk *maxima*. Sama halnya pada iterasi kedua, $x_1 >$ *thres* sehingga *thres* menjadi sama dengan dua dan titik tersebut masuk *maxima*. Pada iterasi ketiga, $x_1 \leq$ *thres* sehingga nilai *thres* dan *maxima* tetap. *Maxima* pada iterasi ketiga menjadi hasil akhir himpunan titik tidak didominasi.



Gambar 2.5 Ilustrasi Algoritma *Greedy*

2.3 Penerapan *Quad Tree*

Metode penerapan *quad tree* dikemukakan oleh Minghe Sun dan Ralph E. Steuer pada referensi [4] yang berjudul *Quad-trees and Linear Lists for Identifying Nondominated Criterion Vectors*. Pada *quad tree*, terdapat konsep *successor*. Misal z^u, $z^v \in R^k$, maka z^v merupakan $(\psi_1\psi_2\ldots\psi_k)_2$-*successor* dari z^u yang mana

$$\psi_i = \begin{cases} 1, & z_i^v > z_i^u \\ 0, & z_i^v \leq z_i^u \end{cases} \tag{2.1}$$

atau secara ekuivalen, z^v merupakan ψ-*successor* dari z^u yang mana

$$\psi = (\psi_1\psi_2\ldots\psi_k)_2$$

$\psi = (\psi_1\psi_2\ldots\psi_k)_2$ merupakan basis 2 ekuivalen dengan ψ yang mana

$$\psi = \sum_{i=1}^{k} \psi_i 2^{k-i} \tag{2.2}$$

ψ-*son* merupakan ψ-*successor* yang tersimpan secara langsung di bawahnya (*child*-nya). Karena *quad tree* bersifat *domination free*, tidak ada $(00\ldots0)_2$- atau $(11\ldots1)$-*successor* karena akan terjadi dominasi jika ada sehingga setiap *vector* memiliki maksimal $2^k - 2$ *son*. Didefinisikan berikut.

$$S_0(\psi) = \{i|\psi_i = 0, \psi = (\psi_1\psi_2 \ldots \psi_k)_2\} \tag{2.3}$$

$$S_1(\psi) = \{i|\psi_i = 1, \psi = (\psi_1\psi_2 \ldots \psi_k)_2\} \tag{2.4}$$

Misalkan diproses $z^v \in R^6$ untuk kemungkinan pemasukkan dalam *domination-free quad-tree* ber-*root* pada $z^r \in R^6$ dan z^v merupakan $\psi = 37 = (100101)_2$-*successor* dari z^r. Dengan $S_0(37) = \{2, 3, 5\}$ dan $S_1(37) = \{1, 4, 6\}$, diobservasi bahwa

1) *Vector* yang didominasi oleh z^v hanya terdapat pada *subtree* yang *root*-nya merupakan *son* dari z^r yang memiliki angka 0 setidaknya pada semua posisi $(100101)_2$ yang berangka 0. Dengan kata lain, dalam *subtree* yang memiliki *root* Φ-*son* dari z^r yang mana $\Phi \leq 37$ dan $S_0(37) \subset S_0(\Phi)$
2) *Vector* yang mendominasi z^v hanya terdapat pada *subtree* yang *root*-nya merupakan *son* dari z^r yang memiliki angka 1 setidaknya pada semua posisi $(100101)_2$ yang berangka 1. Dengan kata lain, dalam *subtree* yang memiliki *root* Φ-*son* dari z^r yang mana $\Phi \geq 37$ dan $S_1(37) \subset S_1(\Phi)$

Ketika memproses sebuah *vector* untuk kemungkinan pemasukkan dalam *domination-free quad-tree*, *vector* tersebut bisa jadi dibuang sebagai titik didominasi oleh sebuah *vector* dalam *quad-tree* atau dimasukkan ke *quad-tree*. Akan tetapi, ketika sebuah *vector* dimasukkan, *vector* ini bisa jadi mendominasi *vector* yang ada dalam *quad-tree* dan *vector* tersebut harus dihapus. Namun ketika menghapus sebuah *vector*, struktur *subtree* yang *root*-nya pada *vector* tersebut hancur sehingga semua *successor* dari *vector* tersebut perlu dipertimbangkan agar dimasukkan pada *quad-tree*. Konstruksi tree dimulai dengan *vector* pertama z^r sebagai *root* awal. Lalu, setiap *vector* z^s diproses kemungkinan pemasukkannya menggunakan fungsi process. Berikut penjabaran semua fungsi yang digunakan.

PROCESS(z^r, z^s). Akan dicek apakah (a) z^s menggantikan z^r, (b) z^s didominasi *vector* dalam *quad-tree*, (c) z^s mendominasi *vector* dalam *quad-tree*.

1. Tentukan ψ yang mana z^s adalah ψ-*successor* dari z^r
   (a) Jika $\psi = 0$, buang z^s dan STOP.
   (b) Jika $z_i^r = z_i^s$, untuk setiap $i \in S_0(\psi)$, eksekusi REPLACE(z^r, z^s) dan STOP.
2. Untuk setiap Φ yang mana $\psi < \Phi \leq 2^k - 2$ dan $S_1(\psi) \subset S_1(\Phi)$, jika terdapat Φ-*son* dari z^r, nyatakan sebagai z^t dan eksekusi TEST2(z^t, z^s).
3. Untuk setiap Φ yang mana $1 \leq \Phi < \psi$ dan $S_0(\psi) \subset S_0(\Phi)$, jika terdapat Φ-*son* dari z^r, nyatakan sebagai z^t dan eksekusi TEST1(z^t, z^s).
4. Jika terdapat Φ-*son* dari z^r, nyatakan sebagai z^t dan eksekusi PROCESS(z^t, z^s). Jika tidak, masukkan z^s sebagai ψ-*son* dari z^r dan STOP.

TEST1(z^c, z^s). Fungsi ini mendeteksi semua *vector* yang terdapat pada *subtree* ber-*root* z^c yang didominasi oleh z^s.

1. Tentukan ψ yang mana z^s adalah ψ-*successor* dari z^c.
2. Jika $z_i^c \neq z_i^s$, untuk setidaknya satu $i \in S_0(\psi)$, lanjut langkah 4. Jika tidak,
3. Eksekusi DELETE(z^c). Jika *subtree* ber-*root* z^c dengan z^c terhapus tidak kosong, jadikan z^t (*son* dari z^c dengan index terkecil) menjadi *root* baru dari *subtree* dan eksekusi TEST1(z^t, z^s).
4. Untuk setiap Φ yang mana $1 \leq \Phi \leq \psi$ dan $S_0(\psi) \subset S_0(\Phi)$, jika terdapat Φ-*son* dari z^c, nyatakan sebagai z^t dan eksekusi TEST1(z^t, z^s).
5. Jika DELETE(z^c) dieksekusi, buang z^c.

TEST2(z^c, z^s). Fungsi ini mengecek apakah ada *vector* pada *subtree* ber-*root* z^c yang mendominasi z^s.

1. Tentukan ψ yang mana z^s adalah ψ-*successor* dari z^c.
2. Jika $\psi = 0$, buang z^s dan STOP. Jika tidak,
3. Untuk setiap Φ yang mana $\psi \leq \Phi \leq 2^k - 2$ dan $S_1(\psi) \subset S_1(\Phi)$, jika terdapat Φ-*son* dari z^c, nyatakan sebagai z^t dan eksekusi TEST2(z^t, z^s).

DELETE(z^c). Fungsi ini menghapus z^c dari *subtree* ber-*root* z^c. Jika *subtree* ber-*root* z^c dengan z^c terhapus tidak kosong, *son* berindeks terkecil menjadi *root* baru bagi *subtree*.

1. Anggap $\Phi = 1$. Lepaskan *subtree* ber-*root* z^c.
2. Jika terdapat Φ-*son* dari z^c, nyatakan sebagai z^t, pindah z^t ke posisi z^c, dan lanjut langkah 4.
3. Anggap $\Phi = \Phi + 1$. Jika $\Phi > 2^k - 2$, RETURN. Jika tidak, lanjut langkah 2.
4. Untuk setiap φ yang mana $\Phi + 1 \leq \varphi \leq 2^k - 2$, jika terdapat φ-*son* dari z^c, nyatakan sebagai z^s dan eksekusi REINSERT(z^t, z^s).

REPLACE(z^c, z^s). Pada fungsi ini, karena z^c didominasi oleh z^s, z^c menggantikan z^s. Karena struktur dari *subtree* ber-*root* z^c dihancurkan, *successor* dari z^c (jika ada) harus dipertimbangkan kembali untuk kemungkinan pemasukkan pada *subtree* baru ber-*root* z^s (karena beberapanya bisa jadi didominasi *root* yang baru).

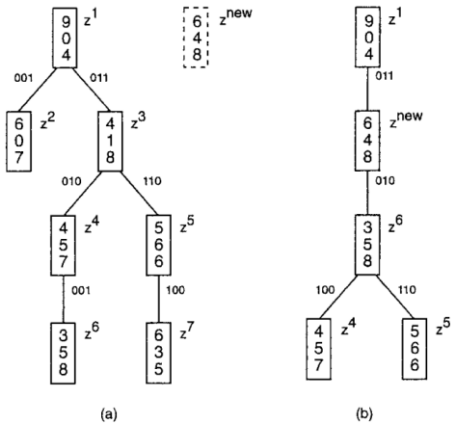
1. Lepaskan *subtree* ber-*root* z^c. Biarkan z^s mengambil alih posisi yang sebelumnya ditempati z^c.
2. Untuk setiap Φ yang mana $1 \leq \Phi \leq 2^k - 2$, jika terdapat Φ-*son* dari z^c dalam *subtree* yang dilepaskan, nyatakan sebagai z^t dan eksekusi RECONSIDER(z^s, z^t).
3. Buang z^c.

REINSERT(z^c, z^s). Tujuan fungsi ini hanyalah menemukan posisi yang benar dalam *subtree* ber-*root* z^c untuk memasukkan z^s dan *successor*-nya. Ketika memasukkan kembali *vector*, *vector* yang berada di level terendah dari *subtree* ber-*root* z^s yang diproses pertama.

1. Untuk setiap Φ yang mana $1 \leq \Phi \leq 2^k - 2$, jika terdapat Φ-*son* dari z^s, nyatakan sebagai z^t dan eksekusi REINSERT(z^c, z^t).
2. Tentukan ψ yang mana z^s adalah ψ-*successor* dari z^c.
3. Jika terdapat ψ-*son* dari z^c, nyatakan sebagai z^t dan eksekusi REINSERT(z^t, z^s). Jika tidak,
4. Pindahkan z^s pada posisi ψ-*son* dari z^c.

RECONSIDER(z^c, z^s). Perbedaan mendasar fungsi ini dengan REINSERT adalah z^s yang dicoba untuk dimasukkan kembali bisa jadi didominasi z^c.

1. Untuk setiap Φ yang mana $1 \leq \Phi \leq 2^k - 2$, jika terdapat Φ-*son* dari z^s, nyatakan sebagai z^t dan eksekusi RECONSIDER(z^c, z^t).
2. Tentukan ψ yang mana z^s adalah ψ-*successor* dari z^c.
   (a) Jika $\psi = 0$, buang z^s dan RETURN.
   (b) Jika terdapat ψ-*son* dari z^c, nyatakan sebagai z^t dan eksekusi REINSERT(z^t, z^s). Jika tidak, pindahkan z^s pada posisi ψ-*son* dari z^c.



Gambar 2.6 Ilustrasi Pemasukan Titik pada *Quad Tree*

Pada Gambar 2.6, bagian (a) merupakan *tree* awal, sedangkan bagian (b) merupakan hasil *tree* setelah z^{new} dimasukkan. Berikut prosesnya. Dimulai dengan memanggil PROCESS untuk mempertimbang z^{new} dapat dimasukkan *tree*. Setelah ditemukan bahwa z^{new} $(011)_2$-*successor* dari z^1, TEST1 dipanggil untuk melihat apakah z^{new} mendominasi z^2. Karena memang benar, DELETE dipanggil untuk menghapus z^2. PROCESS dipanggil kembali untuk kemungkinan pemasukan ke *subtree* ber-*root* z^3. Karena z^{new} mendominasi z^3, REPLACE dipanggil untuk melepas *subtree* ber-*root* z^3 dan menggantikan z^3 dengan z^{new} pada *subtree*. Semua *successor* z^3 pada *subtree* yang terlepas perlu diproses kembali untuk kemungkinan pemasukan pada *subtree* baru ber-*root* z^{new}.

RECONSIDER dipanggil untuk memproses kembali z^4, namun karena z^4 mempunyai *son*, RECONSIDER dipanggil kembali dan dihasilkan bahwa z^6 merupakan $(010)_2$-*successor* z^{new} dan ditempatkan di bawah z^{new}. Kembali ke atas, z^4 juga $(010)_2$-*successor* z^{new} sehingga REINSERT dipanggil untuk memasukkan z^4 pada *subtree* ber-*root* z^6 yang hasilnya menjadi $(100)_2$-son z^6. RECONSIDER meninjau z^5, namun karena mempunyai son, RECONSIDER dipanggil kembali untuk mengevaluasi z^7. z^7 dihapus karena didominasi z^{new}. Kembali ke atas, z^5 juga $(010)_2$-*successor* z^{new} sehingga REINSERT dipanggil untuk memasukkan z^5 pada *subtree* ber-*root* z^6 yang hasilnya menjadi $(110)_2$-son z^6.

2.4 Pembangkit Titik Acak Berdistribusi Seragam Dalam *M-ball*

Pada uji coba kinerja lokal, digunakan pembangkit titik acak berdistribusi seragam dalam *M-ball*. Menurut referensi [5], untuk membangkitkan titik acak berdistribusi seragam dalam unit *M-ball* dapat menggunakan persamaan (2.5).

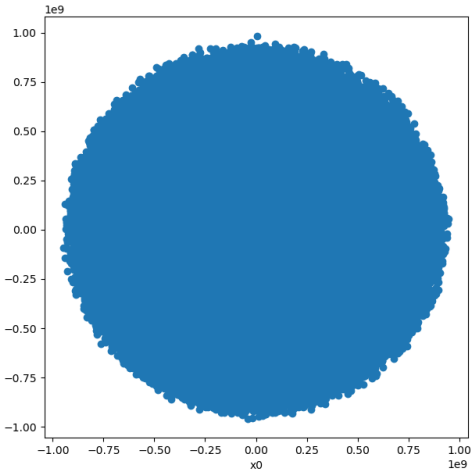
$$(x_0, x_1, \dots x_{M-1}) = \frac{U^{\frac{1}{M}}}{\sqrt{{X_0}^2 + {X_1}^2 + \cdots + {X_{M-1}}^2}} (X_0, X_1, \dots, X_{M-1}) \qquad (2.5)$$

U merupakan bilangan riil acak berdistribusi seragam dengan rentang [0, 1). $X_0, X_1, \dots, X_{M-1}$ merupakan bilangan riil acak berdistribusi normal secara independen dengan rata-rata 0 dan varians 1.

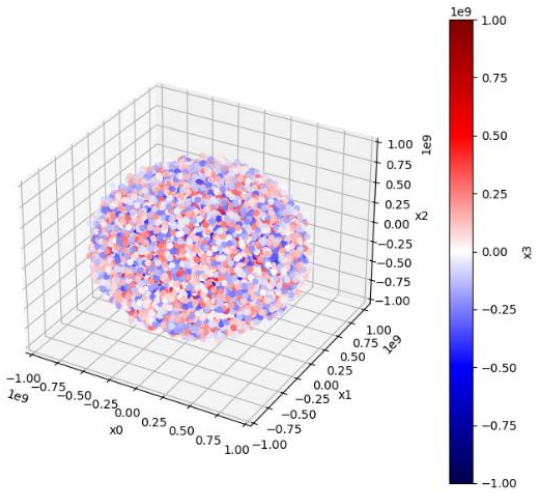
Dari persamaan (2.1), dapat dikembangkan pembangkit titik acak berdistribusi seragam dalam *M-ball* berjari-jari *R*. Pengembangan tersebut terdapat pada persamaan (2.6).

$$(x_0, x_1, \dots x_{M-1}) = \frac{RU^{\frac{1}{M}}}{\sqrt{{X_0}^2 + {X_1}^2 + \cdots + {X_{M-1}}^2}} (X_0, X_1, \dots, X_{M-1}) \quad (2.6)$$

Salah satu batasan pada tugas akhir ini adalah nilai koordinat yang kurang dari 10^9 dari nilai mutlaknya sehingga nilai *R* yang digunakan pada pembangkit untuk uji coba kinerja lokal adalah 10^9. Dapat dipastikan nilai koordinat tidak akan pernah mencapai 10^9 dari nilai mutlaknya karena rentang nilai U tidak termasuk nilai 1. Terdapat batasan lain, yakni koordinat bernilai integer, sedangkan pembangkit titik acak pada persamaan (2.2) menghasilkan titik yang berupa bilangan riil. Oleh karena itu, perlu pembulatan di tiap koordinat agar titik tersebut mengandung bilangan bulat di tiap koordinatnya. Berikut visualisasi persebaran hasil pembangkitan titik acak berdistribusi seragam dalam *2-ball* pada Gambar 2.7 dan *4-ball* pada Gambar 2.8 hingga **Error! Reference source not found.** dengan *N* = 50000.



Gambar 2.7 Persebaran Hasil Pembangkitan Titik Acak dalam *2-ball*



Gambar 2.8 Persebaran Hasil Pembangkitan Titik Acak dalam *4-ball* dengan Kamera Menyamping

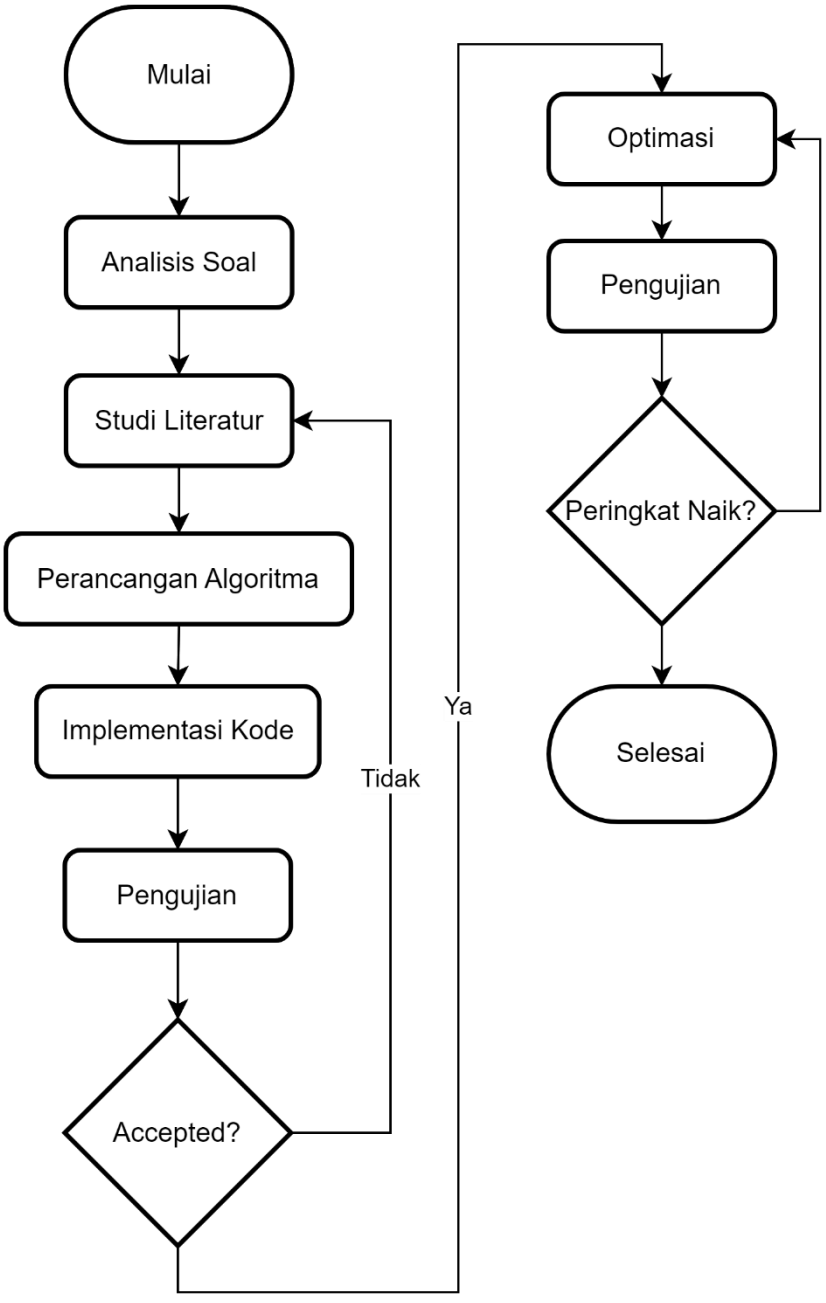
Karena keterbatasan visualisasi, nilai x_3 pada dimensi empat dilambangkan dengan peta warna. Semakin mendekati 10^9, semakin merah, sedangkan semakin mendekati -10^9, semakin biru.

BAB III METODOLOGI

Pada bab ini, dibahas mengenai metode dan tahapan dalam penyusunan tugas akhir. Disertai juga penjelasan peralatan pendukung.

3.1 Tahapan Penelitian

Pelaksanaan tugas akhir dilaksanakan dengan tahapan seperti *flowchart* pada Gambar 3.1.



Gambar 3.1 Tahapan Penelitian Tugas Akhir

Tahapan pertama dimulai dengan menganalisis soal, yakni dengan membaca betul-betul deskripsi soal sehingga tidak salah memahami, mengetahui batasan-batasan yang ada, dan

mencoba menyelesaikan permasalahan secara mandiri terlebih dahulu agar dapat mengetahui dan merasakan tingkat kesulitan soal. Ketika sudah buntu, penulis melakukan studi literatur secara *daring* lewat *google* dan *semantic scholar* menggunakan berbagai kata kunci, seperti *Pareto domination, multidimensional non-dominated points, maxima-finding, set of maxima,* dan sebagainya. Studi literatur merupakan salah satu tahapan yang cukup menyita waktu karena sulitnya mencari kata kunci yang relevan, referensi yang relevan, dan referensi yang setidaknya cukup dapat dipahami penulis.

Setelah didapat referensi yang relevan, penulis melakukan perancangan algoritma yang menyesuaikan dengan studi kasus karena referensi tidak secara langsung menggunakan studi kasus Eolymp 100 “Pareto domination” sebagai fokus permasalahan. Penyesuaian tersebut berupa penggantian dari algoritma pada referensi yang berguna untuk menyimpan himpunan titik tidak didominasi menjadi hanya digunakan untuk mengetahui banyak titik tidak didominasi sehingga penyesuaian tersebut dapat mengoptimasi waktu eksekusi. Rancangan algoritma diimplementasi menjadi kode menggunakan bahasa C++ karena bahasa tersebut memiliki waktu eksekusi tercepat dari antara bahasa lain dan digunakan dalam dunia pemrograman kompetitif.

Pengujian dilakukan dengan mengirimkan soal pada situs Eolymp. Jika masih belum *accepted*, penulis melakukan studi literatur kembali berharap menemukan referensi yang relevan kembali. Jika *accepted*, penulis mencari kemungkinan optimasi dari kode yang sudah dihasilkan untuk menaikkan peringkat. Namun, halaman peringkat sudah tidak tersedia sejak September 2024. Optimasi yang ditemukan penulis adalah mengganti struktur data dengan yang lebih ringan dan menghapus *pointer*.

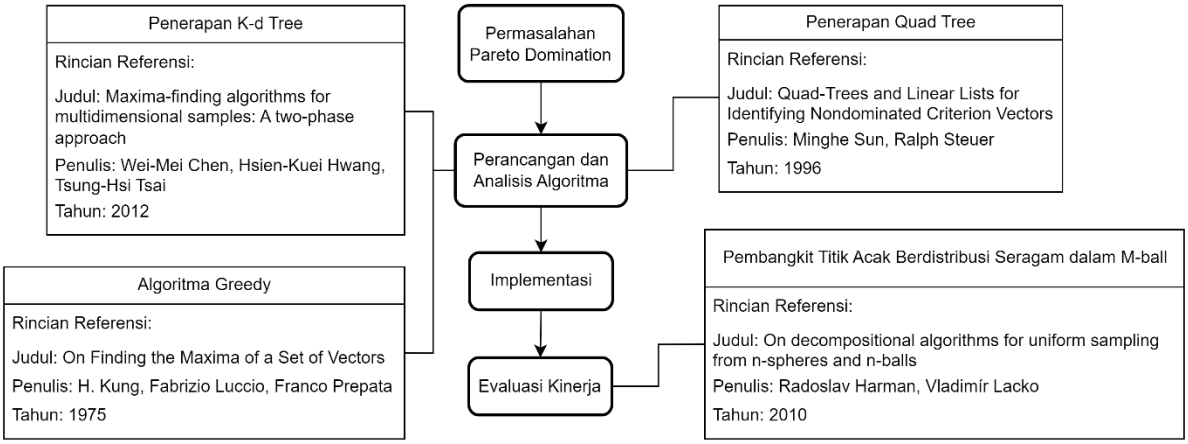
3.2 Peralatan Pendukung

Dalam pengerjaan tugas akhir ini, penulis tidak memerlukan peralatan perangkat keras yang spesifik, namun penulis menggunakan peralatan perangkat lunak pendukung, yaitu Visual Studio Code sebagai teks editor untuk menuliskan kode program.

BAB IV HASIL DAN PEMBAHASAN

Pada bab ini, dijelaskan mengenai hasil dan pembahasan penyelesaian permasalahan Eolymp 100 “Pareto Domination” pada tugas akhir ini.

4.1 Ringkasan Kerangka Berpikir



Gambar 4.1 Bagan Kerangka Berpikir

Pada peracangan algoritma, referensi yang jadi digunakan sebagai hasil akhir implementasi adalah referensi penerapan *k-d tree* [2] dan algoritma *greedy* [3], sedangkan penerapan *quad tree* tidak jadi dipakai karena ketika dikirimkan pada situs Eolmyp, statusnya masih *wrong answer*. Berikut rincian iterasi percobaan pengiriman kode pada situs Eolmyp.

Tabel 4.1 Iterasi Percobaan Pengumpulan

Iterasi ke-	Metode			Keterangan Metode	Accepted	Keterangan Hasil
	Dimensi Satu	Dimensi Dua	Dimensi Tiga dan Empat			
1	Keluaran konstan	Penerapan *k-d tree*	Penerapan *k-d tree*		×	
2	Keluaran konstan	Penerapan *quad tree*	Penerapan *quad tree*		×	
3	Keluaran konstan	Penerapan *k-d tree*	Penerapan *k-d tree*	Perbaikan pada penggunaan *reverse iterator vector*	✓	
4	Keluaran konstan	Penerapan *k-d tree*	Penerapan *k-d tree*	Optimasi mengganti *vector* dengan *unordered map*	×	
5	Keluaran konstan	Penerapan *k-d tree*	Penerapan *k-d tree*	Optimasi mengganti *vector* dengan *array*	✓	Optimasi berhasil
6	Keluaran konstan	Algoritma *greedy*	Penerapan *k-d tree*		✓	Waktu eksekusi tidak terduga lebih cepat
7	Keluaran konstan	Algoritma *greedy*	Penerapan *k-d tree*	Optimasi delete *Node* untuk menghapus *tree*	✓	Optimasi berhasil

Pada evaluasi kinerja lokal, digunakan pembangkit titik acak berdistribusi seragam dalam *M-ball*. Hasil pembangkit digunakan sebagai titik masukan uji coba kinerja lokal untuk membuktikan kompleksitas algoritma.

4.2 Desain dan Analisis Algoritma

Penjelasan desain dan analisis algoritma akan dibagi berdasarkan dimensinya.

4.2.1 Dimensi Satu

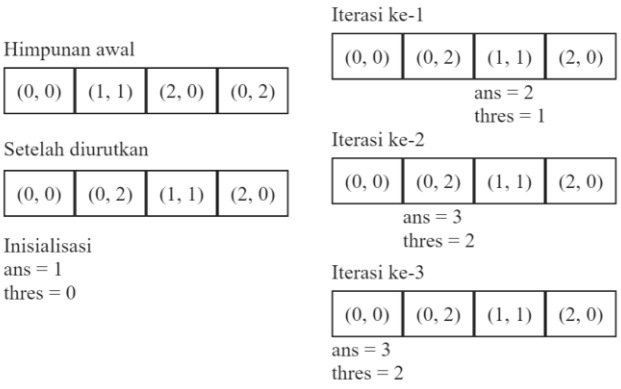
Pada dimensi satu, banyak titik tidak didominasi pasti bernilai satu, yakni titik yang memiliki koordinat terbesar. Hal ini terjadi karena titik dengan koordinat terbesar mampu mendominasi semua titik lain yang koordinatnya bernilai lebih kecil sehingga menyisakan titik dengan koordinat terbesar sebagai satu-satunya titik tidak didominasi. Karena keluarannya tetap, metode ini dapat disebut keluaran konstan yang berkompleksitas $O(1)$. Sebagai contoh, seperti pada contoh masukan studi kasus, pada himpunan titik $\{(1), (2), (3), (4)\}$, yang merupakan titik tidak didominasi adalah titik (4) sebagai koordinat terbesar sehingga banyak titik tidak didominasi adalah satu.

4.2.2 Dimensi Dua

Titik x (x_0, x_1) merupakan titik dalam himpunan titik. Himpunan titik tersebut diurutkan secara menaik dengan x_0 sebagai prioritas utama, kemudian x_1 sebagai prioritas selanjutnya. Sebagai contoh, terdapat dua titik berbeda a (a_0, a_1) dan b (b_0, b_1). Jika $a_0 \neq b_0$, terlepas berapapun nilai a_1 dan b_1, yang dibandingkan hanya a_0 dan b_0. Jika $a_0 < b_0$, urutan titiknya menjadi a kemudian b. Jika $a_0 > b_0$, urutan titiknya menjadi b kemudian a. Namun, jika $a_0 = b_0$, yang dibandingkan adalah a_1 dan b_1. Jika $a_1 < b_1$, urutan titiknya menjadi a kemudian b. Jika $a_1 > b_1$, urutan titiknya menjadi b kemudian a.

Setelah diurutkan, titik yang berada pada posisi terakhir pasti titik tidak didominasi. Dinisialisasi variabel *ans*, untuk menyimpan banyak titik tidak didominasi, sama dengan satu dan variabel *thres* sama dengan nilai x_1 titik posisi terakhir. Selanjutnya, dilakukan iterasi pada himpunan, mundur satu titik, dimulai dari titik terakhir sebelum titik posisi terakhir. Pada tiap iterasi, dibandingkan x_1 dengan *thres*. Jika $x_1 > thres$, titik x merupakan titik tidak didominasi sehingga *ans* ditambah satu dan *thres* sama dengan x_1.

Cara penyelesaian ini dapat dilakukan karena setelah proses pengurutan, nilai x_0 dalam himpunan sudah pasti menaik, sedangkan nilai x_1 belum tentu menaik. Nilai x_1 masih menjadi keraguan ketika iterasi apakah titik x dapat didominasi. *Thres* diperbarui di setiap iterasi dengan nilai x_1 jika $x_1 > thres$ karena titik x tersebut yang berpeluang tertinggi mendominasi titik pada iterasi selanjutnya. Titik y (y_0, y_1) merupakan titik yang y_1-nya menjadi nilai *thres*. Jika $x_1 \leq thres$, dapat dipastikan bahwa $x_0 \leq y_0$, hasil dari proses pengurutan, dan $x_1 \leq y_1$ sehingga titik x didominasi titik y.



Gambar 4.2 Contoh Persoalan Dimensi Dua

Gambar 4.2 merupakan ilustrasi contoh persoalan dimensi dua. Nilai awal *thres* bernilai nol yang berasal dari nilai x_1 titik posisi terakhir setelah diurutkan, yakni titik (2, 0). Pada iterasi pertama, $x_1 > thres$ sehingga *ans* bertambah satu menjadi dua dan *thres* menjadi sama dengan satu. Titik tersebut merupakan titik tidak didominasi. Sama halnya pada iterasi kedua, $x_1 > thres$ sehingga *ans* bertambah satu menjadi tiga dan *thres* menjadi sama dengan dua. Pada iterasi ketiga, $x_1 \leq thres$ sehingga nilai *ans* dan *thres* tetap dan titik tersebut bukan titik tidak didominasi. Jadi, banyak titik tidak didominasi pada himpunan adalah *ans*, yakni tiga. Berikut kode semu untuk memperjelas penyelesaian pada Kode Semu 4.1.

Kode Semu 4.1 Fungsi greedy

```
Input:  p = Himpunan titik dimensi dua
        n = Banyak titik dalam p
Output: ans = Banyak titik tidak didominasi
1:   function greedy(p, n)
2:     p ← sort(p)
3:     ans ← 1
4:     n ← n - 1
5:     thres ← p[n].second
6:     while n > 0 do
7:       n ← n - 1
8:       if p[n].second > thres then
9:         thres ← p[n].second
10:        ans ← ans + 1
11:    return ans
```

Penyelesaian persoalan pada dimensi dua secara konsep merupakan penerapan algoritma *greedy* sehingga untuk selanjutnya, Kode Semu 4.1 juga dapat disebut algoritma *greedy*. Proses pengurutan pada baris 1 berkompleksitas O(N log N), sedangkan perulangan *while* pada baris 5 masih berkompleksitas O(N) sehingga secara keseluruhan, algoritma *greedy* berkompleksitas O(N log N).

4.2.3 Dimensi Tiga dan Empat

Penyelesaian persoalan dimensi tiga dan empat menggunakan struktur data *k-d tree* yang dikombinasikan dengan teknik *bounding box* dan pendekatan dua fase. *K* pada *k-d tree* menunjukkan dimensinya. Jika *k-d tree* menyimpan data titik dimensi empat, *tree* tersebut dapat disebut *4-d tree*. *K-d tree* merupakan pengembangan dari *binary search tree* untuk data

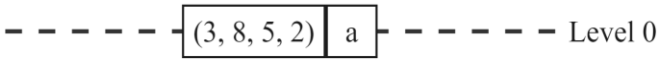
multidimensional. Oleh karena itu, salah satu kesamaan *k-d tree* dengan *binary search tree* adalah *node* yang dapat memiliki dua *child*, yakni *child* kiri dan kanan. Perbedaannya terdapat pada data yang disimpan. *Binary search tree* menyimpan bilangan, sedangkan *k-d tree* menyimpan titik multidimensional. Perbedaan juga terdapat pada proses pemasukan. *Binary search tree* menggunakan *key*, sedangkan *k-d tree* menggunakan *discriminator* untuk penentuan letak *subtree*.

Titik x ($x_0, x_1, \ldots, x_{M-1}$) merupakan titik yang disimpan di *node*. Koordinat ke-i pada titik x dilambangkan x_i. Dimasukkan titik y ($y_0, y_1, \ldots, y_{M-1}$) pada *tree*. Penentuan *subtree* untuk titik y didasarkan pada *discriminator* yang berupa koordinat ke-i. Jika $y_i < x_i$, titik y dimasukkan ke *subtree* kiri. Jika $y_i \geq x_i$, titik y dimasukkan ke *subtree* kanan. Jika koordinat ke-i digunakan sebagai *discriminator* pada level kini, pada level selanjutnya digunakan koordinat ke-$(i+1) \bmod M$ sebagai *discriminator*. Dimulai dari digunakannya koordinat ke-0 sebagai *discriminator* pada level 0.

a	b	c	d	e	f	g
(3, 8, 5, 2)	(2, 9, 7, 6)	(3, 7, 8, 1)	(4, 6, 1, 3)	(6, 5, 2, 9)	(5, 2, 5, 7)	(7, 4, 3, 8)

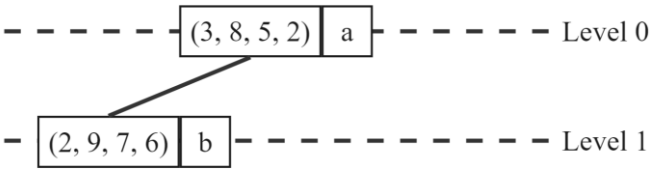
Gambar 4.3 Himpunan Titik Penyusun *4-d Tree*

Sebagai contoh, himpunan titik dimensi empat pada Gambar 4.3 ingin disusun menjadi *4-d tree*. Berikut ilustrasi pemasukan titik demi titik dari himpunan pada *tree*.



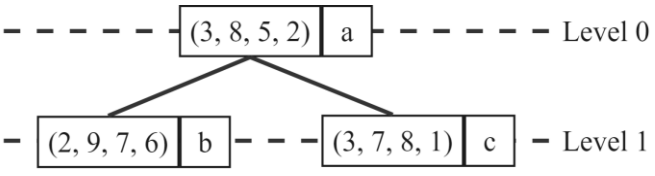
Gambar 4.4 Pemasukan Titik *a* pada *4-d Tree*

Berikut proses pemasukan titik *a* pada Gambar 4.4. Titik *a* sebagai titik paling awal dalam himpunan digunakan sebagai *root* dari *4-d tree*.



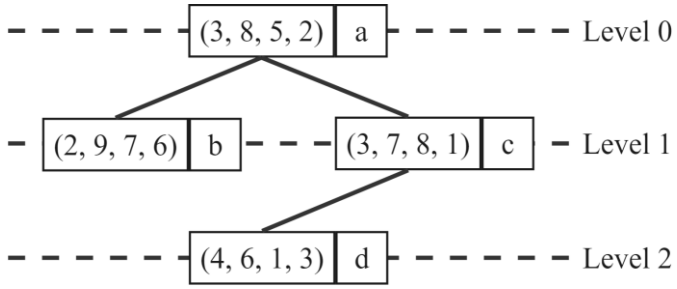
Gambar 4.5 Pemasukan Titik *b* pada *4-d Tree*

Berikut proses pemasukan titik *b* pada Gambar 4.5. Pada node *a*, karena terletak pada level 0, digunakan koordinat ke-0 sebagai *discriminator*. $b_0(2) < a_0(3)$ sehingga titik *b* masuk *subtree* kiri dari *node a*. Jika *node* belum memiliki *child* dari arah *subtree* yang telah ditentukan, titik yang dimasukkan dimasukkan sebagai *child* dari *node* tersebut. *Node a* belum memiliki *child* kiri sehingga titik *b* dimasukkan sebagai *child* kiri dari *node a*.



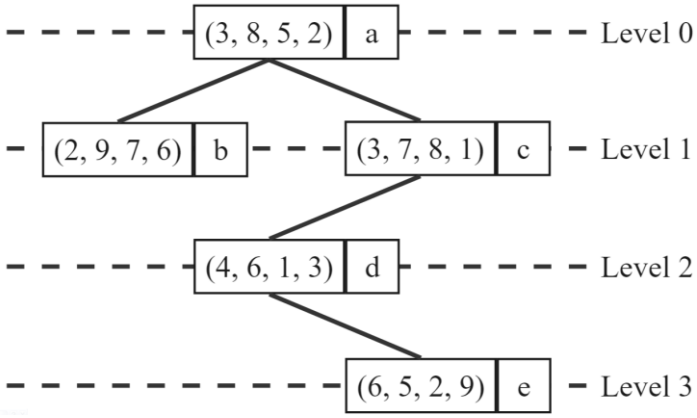
Gambar 4.6 Pemasukan Titik *c* pada *4-d Tree*

Berikut proses pemasukan titik *c* pada Gambar 4.6. Pada node *a*, c_0 (3) ≥ a_0 (3) sehingga titik *c* masuk *subtree* kanan dari *node a*. *Node a* belum memiliki *child* kanan sehingga titik *c* dimasukkan sebagai *child* kanan dari *node a*.



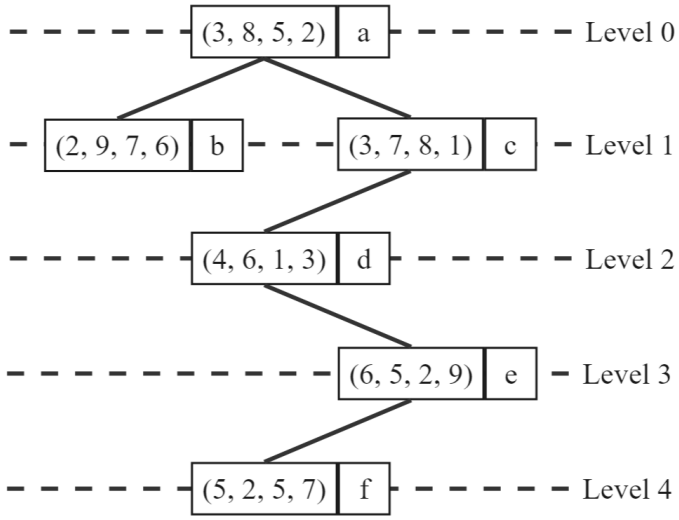
Gambar 4.7 Pemasukan Titik *d* pada *4-d Tree*

Berikut proses pemasukan titik *d* pada Gambar 4.7. Pada node *a*, d_0 (4) ≥ a_0 (3) sehingga titik *d* masuk *subtree* kanan dari *node a*. Jika *node* sudah memiliki *child* dari arah *subtree* yang telah ditentukan, proses pemasukan dilanjut pada *child* tersebut. *Node a* sudah memiliki *child* kanan *node c* sehingga proses pemasukan dilanjut pada *node c*. Karena turun satu level, *discriminator* yang digunakan berubah menjadi koordinat ke-(0 + 1) mod 4, yakni koordinat ke-1. Pada node *c*, d_1 (6) < c_1 (7) sehingga titik *d* masuk *subtree* kiri dari *node c* dan menjadi *child* kiri dari *node c*.



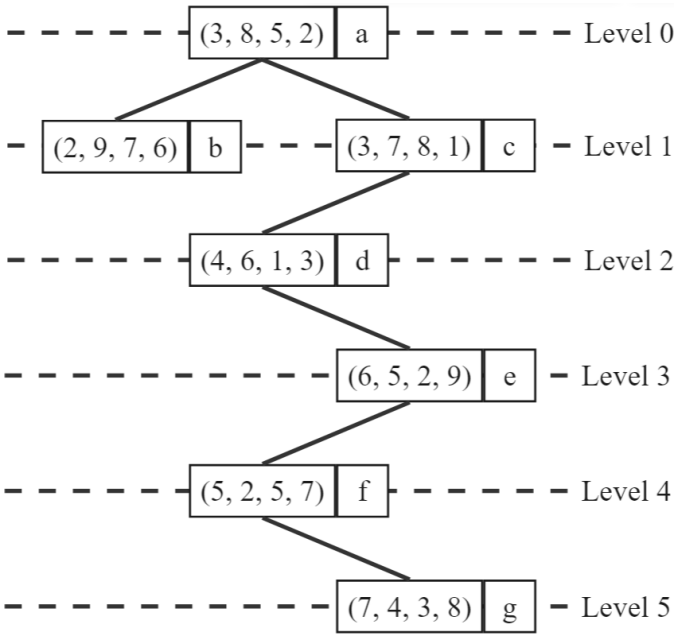
Gambar 4.8 Pemasukan Titik *e* pada *4-d Tree*

Berikut proses pemasukan titik *e* pada Gambar 4.8. Pada node *a*, e_0 (6) ≥ a_0 (3) sehingga titik *e* masuk *subtree* kanan dari *node a*. Pada node *c*, e_1 (5) < c_1 (7) sehingga titik *e* masuk *subtree* kiri dari *node c*. *Node d* berada pada level 2 sehingga *discriminator* yang digunakan berubah menjadi koordinat ke-(1 + 1) mod 4, yakni koordinat ke-2. e_2 (2) ≥ d_2 (1) sehingga titik *e* masuk *subtree* kanan dari *node d* dan menjadi *child* kanan dari *node d*.



Gambar 4.9 Pemasukan Titik *f* pada *4-d Tree*

Berikut proses pemasukan titik *f* pada Gambar 4.9. Pada node *a*, f_0 (5) ≥ a_0 (3) sehingga titik *f* masuk *subtree* kanan dari *node a*. Pada node *c*, f_1 (2) < c_1 (7) sehingga titik *f* masuk *subtree* kiri dari *node c*. Pada node *d*, f_2 (5) ≥ d_2 (1) sehingga titik *f* masuk *subtree* kanan dari *node d. Node e* berada pada level 3 sehingga *discriminator* yang digunakan berubah menjadi koordinat ke-(2 + 1) mod 4, yakni koordinat ke-3. f_3 (7) < e_3 (9) sehingga titik *f* masuk *subtree* kiri dari *node e* dan menjadi *child* kiri dari *node e*.

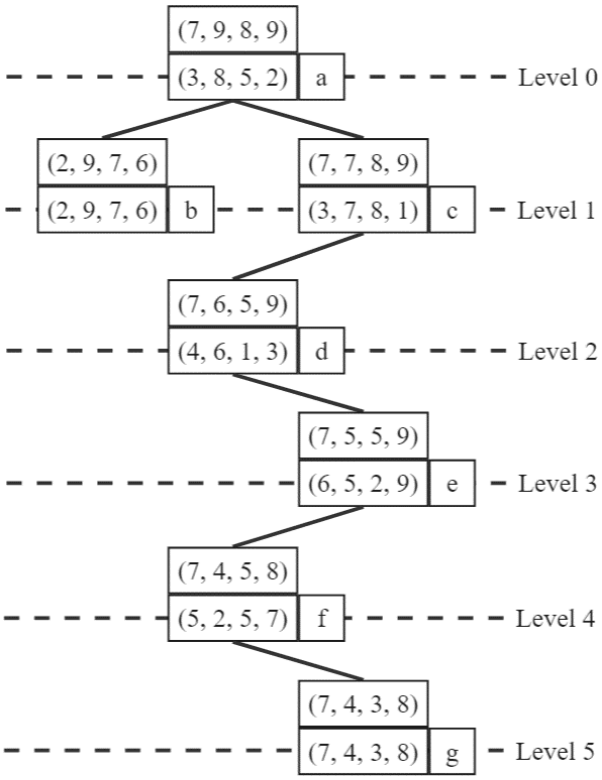


Gambar 4.10 Pemasukan Titik *g* pada *4-d Tree*

Berikut proses pemasukan titik *g* pada Gambar 4.10. Pada node *a*, g_0 (7) ≥ a_0 (3) sehingga titik *g* masuk *subtree* kanan dari *node a*. Pada node *c*, g_1 (4) < c_1 (7) sehingga titik *g* masuk *subtree* kiri dari *node c*. Pada node *d*, g_2 (3) ≥ d_2 (1) sehingga titik *g* masuk *subtree* kanan dari *node d.* Pada *node e*, g_3 (8) < e_3 (9) sehingga titik *g* masuk *subtree* kiri dari *node e*. *Node f* berada pada level 4 sehingga *discriminator* yang digunakan berubah menjadi koordinat ke-(3 +

1) mod 4, yakni koordinat ke-0. $g_0(7) \geq f_0(5)$ sehingga titik *g* masuk *subtree* kanan dari *node f* dan menjadi *child* kanan dari *node f*.

Bounding box merupakan teknik yang seolah-olah memberikan batas berbentuk kotak menggunakan *upper bound* dan *lower bound* dari kumpulan titik. Titik x ($x_0, x_1, \ldots, x_{M-1}$) merupakan titik yang berada pada *subtree* dengan *root r*. *Upper bound* dari *subtree* tersebut adalah u ($u_0, u_1, \ldots, u_{M-1}$), sedangkan *lower bound* dari *subtree* tersebut adalah l ($l_0, l_1, \ldots, l_{M-1}$). Nilai u_i merupakan nilai maksimal dari semua x_i dalam *subtree*, sedangkan nilai l_i merupakan nilai minimum dari semua x_i dalam *subtree*. Namun, penyelesaian ini hanya menggunakan *upper bound*.



Gambar 4.11 *4-d Tree* yang Ditambahkan *Upper Bound*

Gambar 4.11 merupakan *4-d tree* pada Gambar 2.9 yang telah diberi *upper bound* pada masing-masing node di bagian atasnya. Sebagai contoh, *node d* sebagai *root* dari *subtree* beranggotakan *node d*, *e*, *f*, dan *g* memiliki *upper bound* (7, 6, 5, 9) yang berasal dari (g_0, d_1, f_2, e_3) sebagai nilai maksimum dari semua x_i pada tiap *i*. *Node a* sebagai *root* dari *subtree* yang beranggotakan semua *node* dalam *tree* memiliki *upper bound* (7, 9, 8, 9) yang berasal dari (g_0, b_1, c_2, e_3).

Kode Semu 4.2 Struct Node dan Variabel Global M

```
1:   global M
2:   struct Node
3:       p, u
4:       left, right
5:       Node(p)
6:           for i ← 0 to M-1 do
7:               this.p[i] ← p[i]
```

```
8:                u[i] ← p[i]
9:            left ← NULL
10:           right ← NULL
11:       ~Node()
12:           delete left
13:           delete right
```

Kode Semu 4.2 berisi variabel global *M* yang merupakan dimensi pada baris 1 dan *struct Node* pada baris 2-13 yang terus digunakan pada kode semu berikutnya hingga akhir subbab 4.2.3. Seperti yang ditujukkan pada baris 3-4, *Node* memiliki atribut *p* yaitu titik yang disimpan, *u* yaitu *upper bound*, *left* yaitu *child* kiri, dan *right* yaitu *child* kanan. *Node* memiliki *constructor* seperti yang ditunjukkan pada baris 5-10. *Constructor* tersebut berisi perintah untuk memberikan nilai *p* dari parameter pada *p* dan *u* serta memberikan nilai NULL pada *left* dan *right*. *Node* memiliki *desctuctor* pada baris 11-13 yang berfungsi untuk menghapus *left* dan *right* ketika *Node* dihapus.

Kode Semu 4.3 Fungsi insert

```
Input:  r = node root dari subtree
        p = titik yang ingin dimasukkan
        d = discriminator
Output:  r = node root dari subtree
1:   function insert(r, p, d)
2:     if r = NULL then
3:       return new Node(p)
4:     for i ← 0 to M-1 do
5:       r.u[i] ← max(r.u[i], p[i])
6:     d ← d mod M
7:     if p[d] < r.p[d] then
8:       r.left ← insert(r.left, p, d + 1)
9:     else
10:      r.right ← insert(r.right, p, d + 1)
11:    return r
```

Kode Semu 4.3 merupakan kode yang menunjukkan proses bagaimana titik dimasukkan dalam *k-d tree*. Pada baris 2-3, jika *r* masih bernilai NULL, yang artinya *Node* masih belum dibentuk pada *r*, dibuat *Node* baru menggunakan *p*, lalu dikembalikan. Baris 4-5 digunakan untuk memperbarui nilai *u* pada *r* karena *p* sudah menjadi anggota dari *subtree* dengan *root r*. Operasi *modulo* pada baris 6 merupakan bagian dari pembaruan nilai *discriminator* yang sebelumnya sudah di tambah satu ketika pemanggilan fungsi *insert* pada baris 8 dan 10. Pada baris 7-10, *discriminator* berperan sebagai penentu arah *subtree*. Jika $p_d < r.p_d$, *p* masuk *subtree* kiri dari *r*. Jika tidak, *p* masuk *subtree* kanan dari *r*. Diakhiri dengan mengembalikan *r* pada baris 11.

Kode Semu 4.4 Fungsi dominated_point

```
Input:  p1 = titik pertama
        p2 = titik kedua
Output:  Nilai boolean apakah titik pertama didominasi titik kedua
1:  function dominated_point(p1, p2)
2:    for i ← 0 to M-1 do
```

```
3:      if p1[i] > p2[i] then
4:        return false
5:    return true
```

Kode Semu 4.4 merupakan kode fungsi yang berguna untuk melakukan pengecekan apakah titik *p1* didominasi titik *p2*. Caranya dengan mengecek pada setiap *i* apakah ada kondisi $p1_i > p2_i$. Jika ada kondisi tersebut, *p1* tidak dapat didominasi *p2* karena agar dapat didominasi, kondisi yang harus dipenuhi adalah $p1_i \leq p2_i$ untuk semua *i* sehingga dikembalikan *boolean false*. Sebaliknya, jika kondisi $p1_i > p2_i$ tidak pernah terpenuhi pada semua *i* dikembalikan *boolean true* karena $p1_i \leq p2_i$ untuk semua *i* yang menandakan titik *p1* didominasi titik *p2*.

Kode Semu 4.5 Fungsi dominated_node

```
Input:   p =  titik yang ingin dicek apakah didominasi salah satu
              titik dalam tree
         r = node root dari subtree
Output:  Nilai boolean apakah p didominasi salah satu titik dalam
         tree
1:   function dominated_node(p, r)
2:     if dominated_point(p, r.p) then
3:       return true
4:     if r.left <> NULL and dominated_point(p, r.left.u) then
5:       if dominated_node(p, r.left) then
6:         return true
7:     if r.right <> NULL and dominated_point(p, r.right.u) then
8:       if dominated_node(p, r.right) then
9:         return true
10:    return false
```

Kode Semu 4.5 merupakan kode fungsi yang berguna untuk melakukan pengecekan apakah titik *p* didominasi salah satu titik dalam *subtree* dengan *root r*. Pertama, pada baris 2-3, dicek apakah titik *p* didominasi titik yang disimpan dalam *r* yakni *r.p*. Jika benar, dikembalikan nilai *true*. Jika tidak, pada baris 4-9, pengecekan dilanjutkan pada *subtree* kiri dan kanan, jika memang ada, secara *depth first search*. Namun, pengecekan tersebut terbantu dengan adanya *upper bound*. Jika *p* tidak didominasi *upper bound* dari *subtree*, tidak ada titik di dalam *subtree* yang mampu mendominasi *p*. Titik x $(x_0, x_1, \ldots, x_{M-1})$ merupakan titik dalam *subtree*. *Upper bound u* menyimpan nilai maksimum x_i pada tiap *i* sehingga semua x_i nilainya kurang dari sama dengan u_i pada tiap *i*. Oleh karena itu, jika *u* tidak bisa mendominasi *p*, apalagi titik *x* sehingga pengecekan pada *subtree* dapat dilewati. Jika salah satu *x* berhasil mendominasi *p*, nilai *true* pada baris 3 akan dikembalikan secara rekursif lewat baris 5 atau 8. Jika tidak ada *x* yang dapat mendominasi *p*, dikembalikan nilai *false*.

Kode Semu 4.6 Fungsi two_phase

```
Input:  p = himpunan titik
        n = banyak titik dalam p
Output:  ans = banyak titik tidak didominasi
1:   function two_phase(p, n)
2:     ans ← 1
3:     r ← new Node(p[0])
4:     q[0] ← p[0]
```

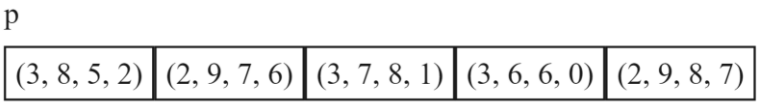
```
5:    for i ← 1 to n-1 do
6:      if dominated_node(p[i], r) = false then
7:        insert(r, p[i], 0)
8:        q[ans] ← p[i]
9:        ans ← ans + 1
10:   delete r
11:   i ← ans - 1
12:   r ← new Node(q[i])
13:   while i > 0 do
14:     i ← i - 1
15:     if dominated_node(q[i], r) = true then
16:       ans ← ans - 1
17:     else
18:       insert(r, q[i], 0)
19:   delete r
20:   return ans
```

Kode Semu 4.6 merupakan kode fungsi untuk menentukan banyak titik tidak didominasi pada himpunan titik *p*. Himpunan titik *p* beranggotakan titik p_0, titik p_1, …, titik p_{n-1}. *p* bukan himpunan kosong sehingga paling sedikit *p* memiliki satu titik tidak didominasi sesuai baris 2. p_0 digunakan sebagai *root* dari *k-d tree* sesuai baris 3. *q* merupakan himpunan yang digunakan untuk menyimpan titik yang diduga merupakan titik tidak didominasi. Iterasi pada baris 5-9 digunakan untuk menemukan titik yang diduga titik tidak didominasi. Di dalam iterasi, jika semua titik dalam *k-d tree* tidak ada yang mampu mendominasi titik p_i, p_i dimasukkan ke dalam *k-d tree* dan *q* dan nilai *ans* bertambah satu. Iterasi ini disebut fase pertama. Dalam iterasi tersebut, dominansi p_i hanya dibandingkan dengan p_0 hingga p_{i-1}. Masih terdapat kemungkinan p_i didominasi salah satu dari p_{i+1} hingga p_{n-1} ketika p_i dimasukkan *q*. Oleh karena itu, kekurangan ini diatasi dengan iterasi pada fase kedua. Sebelumnya, sesuai baris 10, *r* dihapus untuk menghemat penggunaan memori karena pada fase kedua, digunakan *k-d tree* yang baru. Pada baris 11-12, *i* diperbarui dengan *ans-1* dan *r* diperbarui menggunakan titik terakhir dari *q*, yakni q_{ans-1}. Baris 13-18 merupakan iterasi sebagai fase kedua. Awalnya, *i* dikurangi 1 untuk penyesuaian index. Kemudian, jika q_i didominasi salah satu titik dalam *k-d tree*, dugaan sebelumnya bahwa q_i tidak didominasi ternyata salah sehingga *ans* dikurangi satu. Jika tidak, q_i dimasukkan ke dalam *k-d tree*. Pada baris 19-20, *r* dihapus dan dikembalikan nilai *ans*. Oleh karena digunakan dua fase, pendekatan tersebut disebut pendekatan dua fase.

Operasi pemasukan dan pencarian pada *k-d tree* berkompleksitas O(log N). Operasi pencarian memang berbeda dengan operasi pengecekan dominasi, yakni fungsi pada Kode Semu 4.5. Operasi pencarian hanya perlu secara spesifik mencari titik dengan koordinat yang sesuai sehingga arah pencarian selalu ke dalam berdasar *discriminator*, tidak pernah melebar. Namun pada operasi pengecekan dominasi, arah pengecekan bisa jadi melebar dari *subtree* kiri berlanjut ke *subtree* kanan. Tanpa mempertimbangkan keberadaan *upper bound*, kompleksitas operasi pengecekan dominasi bisa menjadi O(N) karena masih perlu menjelajah semua *node* dalam *tree*. Namun, hal tersebut teratasi dengan adanya *upper bound* yang menjadi acuan jika *subtree* dapat dilewati dalam operasi pengecekan sehingga kompleksitas pengecekan dominasi masih dapat dianggap O(log N). Operasi pengecekan dilakukan pada setiap titik dalam himpunan sehingga secara keseluruhan, metode penyelesaian menggunakan *k-d tree* yang dikombinasikan dengan teknik *bounding box* dan pendekatan dua fase berkompleksitas O(N

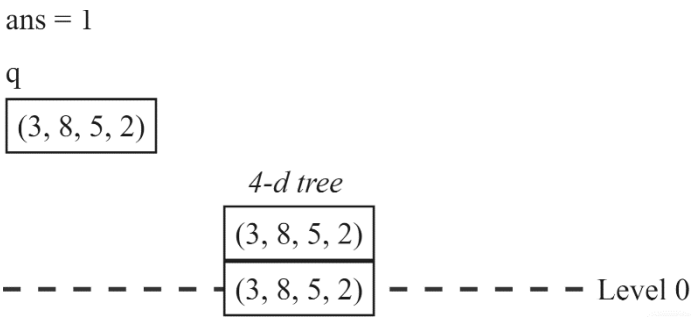
log N). Jika dikaitkan dengan *M*, penerapan *k-d tree* berkompleksitas O(MN log N) yang berasal dari perulangan pada fungsi *insert* pada Kode Semu 4.3 baris 4 dan fungsi dominated_point pada Kode Semu 4.4 baris 2.

Penerapan *k-d tree* yang diterapkan sebagai penyelesaian persoalan pada dimensi tiga dan empat sebenarnya bisa diterapkan pada dimensi satu dan dua. Namun, pada dimensi satu, banyak titik tidak didominasi sudah pasti satu sehingga operasi lebih lanjut akan menambah biaya lagi. Pada dimensi dua, algoritma *greedy* dan penerapan *k-d tree* memiliki kelebihan dan kekurangan masing-masing yang akan ditelusuri lebih lanjut lewat uji coba yang dijelaskan pada subbab 4.6.5.



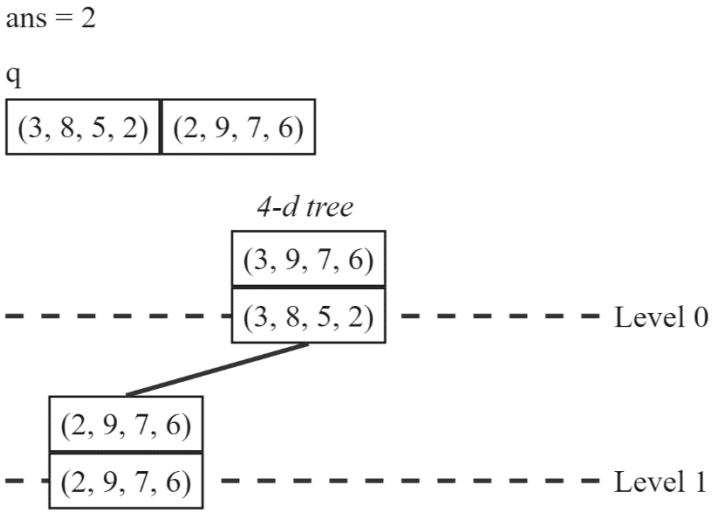
Gambar 4.12 Himpunan titik *p* untuk Simulasi Pendekatan Dua Fase

Sebagai contoh, Gambar 4.12 merupakan himpunan titik *p* yang ingin ditemukan banyak titik didominasinya. Berikut penjelasan penyelesaiannya.



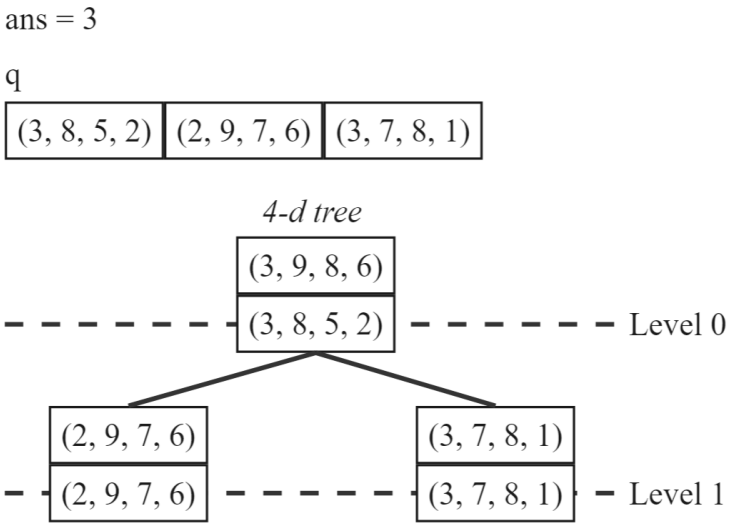
Gambar 4.13 Persiapan Fase Pertama pada Simulasi Pendekatan Dua Fase

Gambar 4.13 merupakan tahap persiapan sebelum memulai fase pertama, yakni deklarasi *ans* = 1, q = {p_0}, dan *r* = *Node*(p_0).



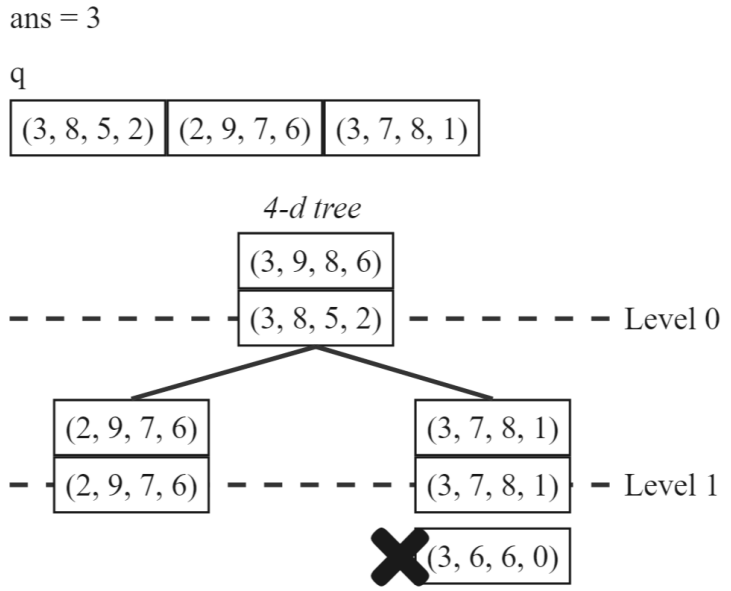
Gambar 4.14 Fase Pertama *i* = 1 pada Simulasi Pendekatan Dua Fase

Gambar 4.14 merupakan hasil dari fase pertama ketika *i* = 1. p_i tidak didominasi oleh *r.p* sehingga dapat dimasukkan pada *q* dan *4-d tree* dan *ans* bertambah satu menjadi dua.



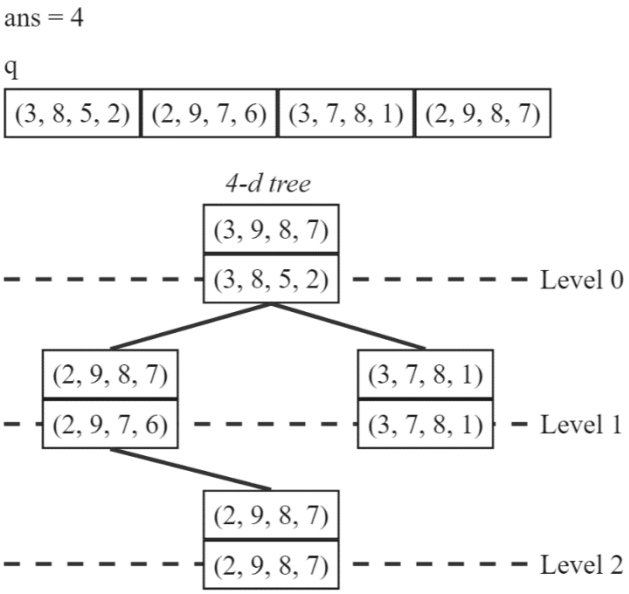
Gambar 4.15 Fase Pertama *i* = 2 pada Simulasi Pendekatan Dua Fase

Gambar 4.15 merupakan hasil dari fase pertama ketika *i* = 2. p_2 tidak didominasi oleh *r.p*. Kemudian, p_2 juga tidak didominasi oleh *r.left.u* sehingga pengecekan pada *subtree* kiri dapat dilewati dan p_2 dimasukkan pada *q* dan *4-d tree* dan *ans* bertambah satu menjadi tiga.



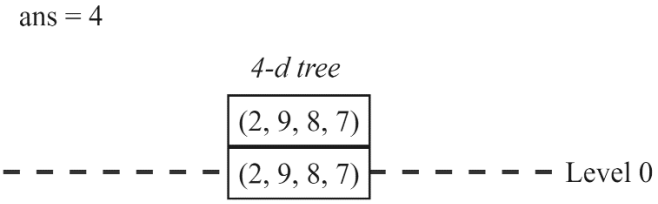
Gambar 4.16 Fase Pertama *i* = 3 pada Simulasi Pendekatan Dua Fase

Gambar 4.16 merupakan hasil dari fase pertama ketika *i* = 3. p_3 tidak didominasi oleh *r.p*. Kemudian, p_3 juga tidak didominasi oleh *r.left.u*. Namun, p_3 masih didominasi oleh *r.right.u* sehingga perlu dicek apakah ada titik dalam *subtree* kanan yang mampu mendominasi p_3. Ternyata memang ada, yakni *r.left.p* mampu mendominasi p_3 sehingga p_3 bukan titik tidak didominasi. Nilai *ans*, isi *q*, dan isi *4-d tree* tetap.



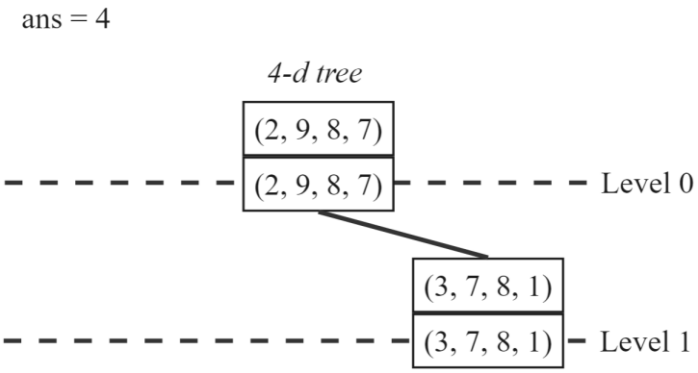
Gambar 4.17 Fase Pertama *i* = 4 pada Simulasi Pendekatan Dua Fase

Gambar 4.17 merupakan hasil dari fase pertama ketika *i* = 4. p_4 tidak didominasi oleh *r.p*. Kemudian, p_4 juga tidak didominasi baik oleh *r.left.u* maupun *r.right.u* sehingga pengecekan baik *subtree* kiri maupun kanan dapat dilewati dan p_4 dimasukkan pada *q* dan *4-d tree* dan *ans* bertambah satu menjadi empat. Fase pertama telah selesai. Selanjutnya, dilakukan persiapan untuk fase kedua.



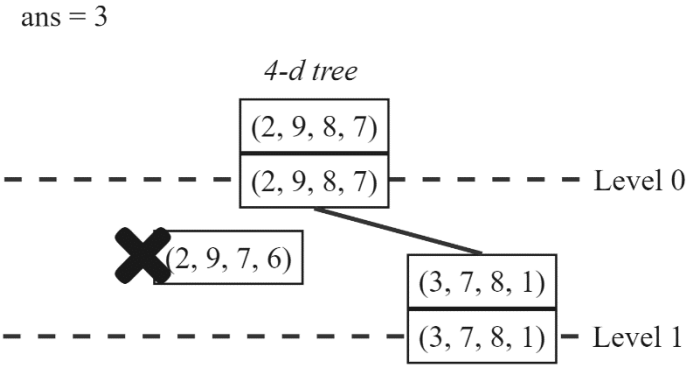
Gambar 4.18 Persiapan Fase Kedua pada Simulasi Pendekatan Dua Fase

Persiapan fase kedua terdiri dari penghapusan *r* yang secara rekursif menghapus keseluruhan *4-d tree*, pembaruan nilai *i* dengan *ans* – 1, dan membuat *r* baru menggunakan *Node*(q_3) sehingga hasilnya menjadi seperti pada Gambar 4.18.



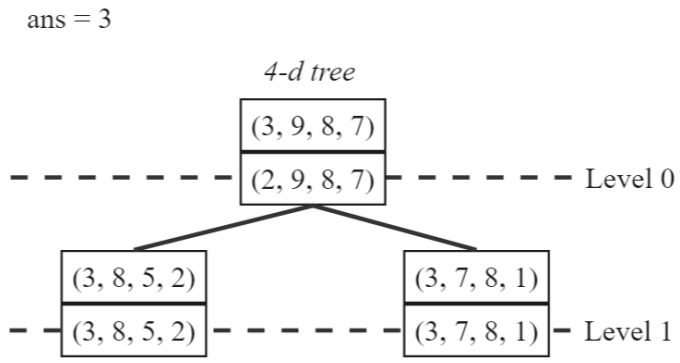
Gambar 4.19 Fase Kedua *i* = 3 pada Simulasi Pendekatan Dua Fase

Gambar 4.19 merupakan hasil dari fase kedua ketika *i* = 3. q_2 tidak didominasi oleh *r.p* sehingga dapat dimasukkan pada *4-d tree* dengan *ans* tetap.



Gambar 4.20 Fase Kedua *i* = 2 pada Simulasi Pendekatan Dua Fase

Gambar 4.20 merupakan hasil dari fase kedua ketika *i* = 2. q_1 didominasi oleh *r.p* sehingga q_1 bukan titik tidak didominasi dan *ans* berkurang satu menjadi tiga.



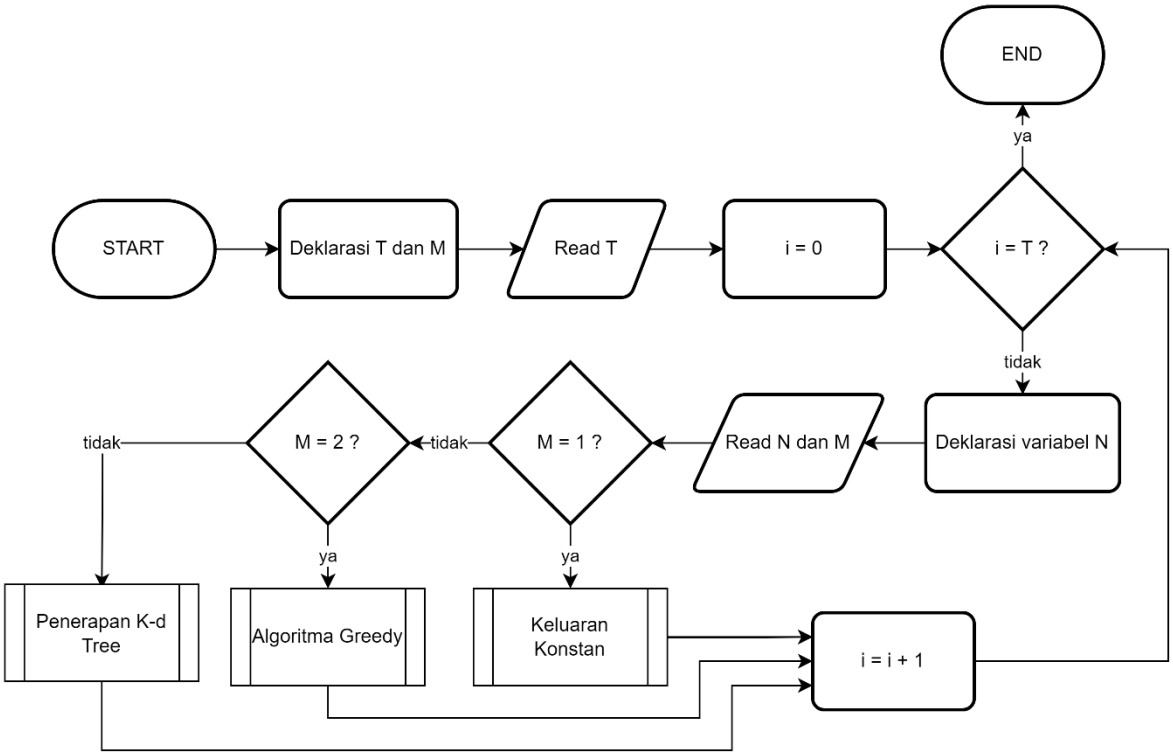
Gambar 4.21 Fase Kedua *i* = 1 pada Simulasi Pendekatan Dua Fase

Gambar 4.21 merupakan hasil dari fase kedua ketika *i* = 1. q_0 tidak dapat didominasi oleh *r.p*. Kemudian, q_0 juga tidak dapat didominasi oleh *r.right.u* sehingga pengecekan pada *subtree* kanan dapat dilewati dan q_0 dimasukkan pada *4-d tree* dengan *ans* tetap.

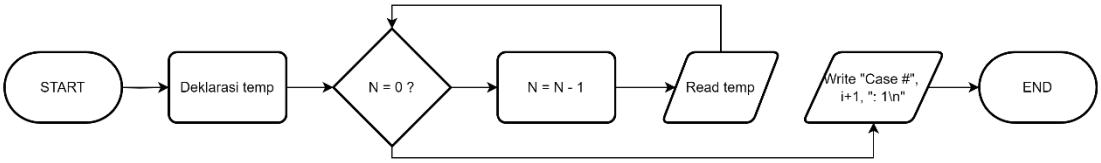
Fase kedua telah berakhir. *r* dihapus, kemudian dikembalikan nilai *ans*, yakni 3 yang menandakan banyak titik tidak didominasi dari himpunan titik *p* adalah tiga.

4.3 Alur Kerja Solusi

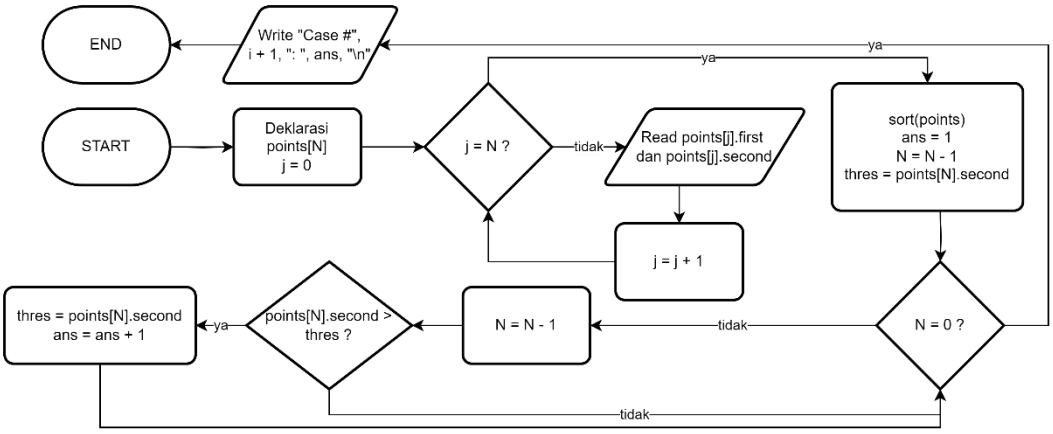
Alur kerja solusi studi kasus Eolymp 100 “Pareto Domination” menggunakan keluaran konstan, algoritma greedy, dan penerapan k-d tree. Pada Gambar 4.22, T merupakan banyak kasus uji, M merupakan dimensi, dan N merupakan banyak titik. Diagram alir pada Gambar 4.22 membagi kasus menjadi tiga bagian, yakni dimensi satu yang menggunakan keluaran konstan pada Gambar 4.23, dimensi dua yang menggunakan algoritma greedy pada Gambar 4.24, dan dimensi tiga dan empat yang menggunakan penerapan k-d tree pada Gambar 4.25. Masing-masing bagian menghasilkan keluaran banyak titik tidak didominasi pada akhir diagram. Pada Gambar 4.23, temp merupakan variabel penyimpan masukan sementara yang tidak dipakai karena keluarannya konstan. Pada Gambar 4.24, points merupakan array of pair yang menyimpan masukan himpunan titik dimensi dua. Pada Gambar 4.25, points merupakan array integer dua dimensi berukuran N x 4 untuk menyimpan titik yang diduga merupakan titik tidak didominasi, sedangkan point merupakan array integer satu dimensi berukuran 4 untuk menerima masukan titik dimensi tiga atau empat satu per satu.



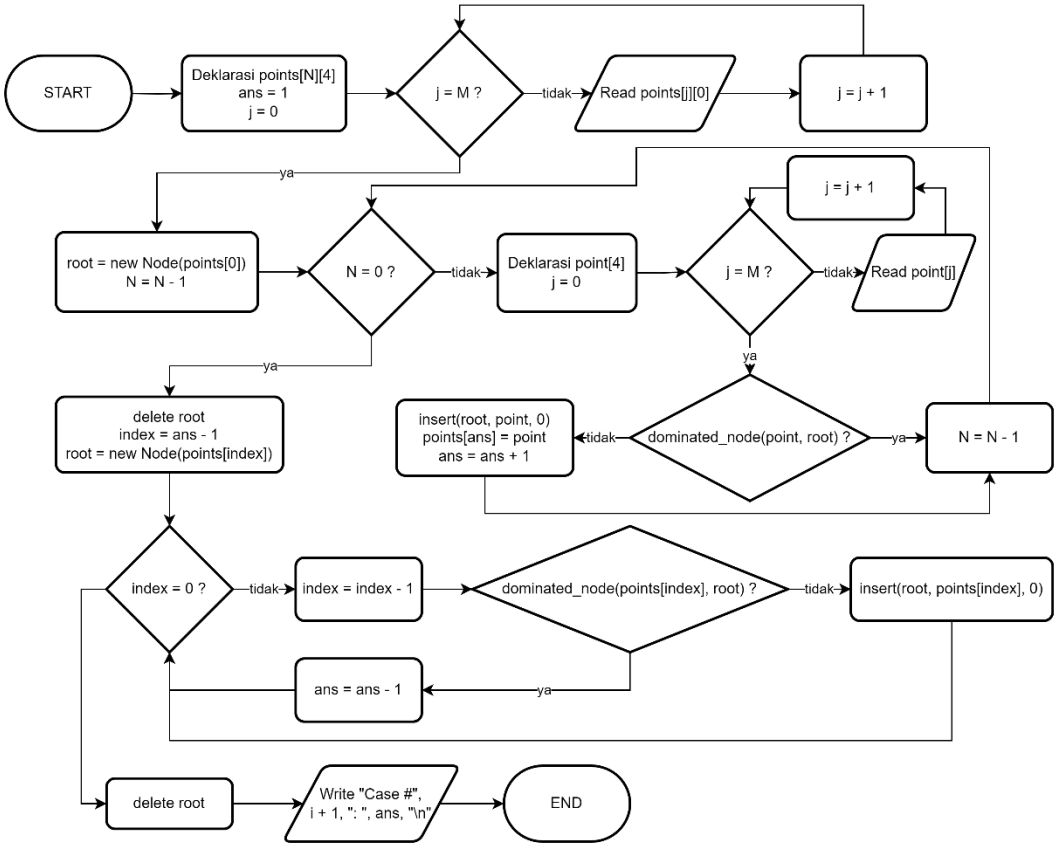
Gambar 4.22 Diagram Alir Desain Umum Solusi



Gambar 4.23 Diagram Alir Keluaran Konstan



Gambar 4.24 Diagram Alir Algoritma *Greedy*



Gambar 4.25 Diagram Alir Penerapan *K-d Tree*

4.4 Desain Solusi

Pada subbab ini akan dijelaskan mengenai gambaran secara umum metode yang dirancang. Sistem kerja dari algoritma yang telah dirancang dimulai dengan menerima masukan berupa bilangan bulat positif *T* yang menyatakan jumlah kasus uji untuk satu kali program dijalankan. Setiap kasus uji terdiri dari dua bilangan bulat N dan M, yakni banyak titik dan dimensi, lalu diikuti dengan baris berjumlah N yang tiap barisnya berisi bilangan bulat berjumlah M yang terpisah spasi. Keluaran akhir dari program ini adalah banyak titik tidak didominasi. Pada penyelesaian permasalahan ini, desain kode program solusi permasalahan memiliki 4 fungsi, yaitu fungsi *main*, *insert*, *dominated_point*, *dominated_node*. Selain fungsi, juga terdapat fungsi makro *REP*, variabel global *M*, dan *struct Node*.

4.4.1 Desain Fungsi Makro REP dan Variabel Global M

Fungsi makro *REP* digunakan untuk mempersingkat penulisan perulangan *for*, sedangkan variabel global *M* digunakan untuk menyimpan nilai dimensi. Desain fungsi dan penjelasan variabelnya terdapat pada Kode Semu 4.7 dan Tabel 4.2.

Kode Semu 4.7 Fungsi Makro REP dan Variabel Global M

```
1:  define REP(i, n) for(int i=0; i<n; i++)
2:  global M
```

Tabel 4.2 Penjelasan Fungsi Makro REP dan Variabel Global M

No	Nama	Tipe	Fungsi
1	REP	fungsi makro	Menyingkat penulisan perulangan for
2	M	integer	Menyimpan dimensi

4.4.2 Desain Struct Node

Struct Node merupakan struktur data yang menjadi basis dalam pembuatan node. Desain fungsi dan penjelasan variabelnya terdapat pada Kode Semu 4.8 dan Tabel 4.3.

Kode Semu 4.8 Struct Node

```
1:   struct Node
2:       point, upper_bound
3:       left, right
4:       Node(point)
5:           for i ← 0 to M-1 do
6:               this.point[i] ← point[i]
7:               upper_bound[i] ← point[i]
8:           left ← NULL
9:           right ← NULL
10:      ~Node()
11:          delete left
12:          delete right
```

Tabel 4.3 Penjelasan Struct Node

No	Nama	Tipe	Fungsi
1	Node	struct	Basis dalam pembuatan node
2	point	integer array	Menyimpan titik yang terdapat pada *node* ini
3	upper_bound	integer array	Menyimpan *upper bound* dari *subtree* dengan *root node* ini
4	left	pointer Node	Merujuk pada *node child* kiri
5	right	pointer Node	Merujuk pada *node child* kanan
6	Node()	constructor	Mengisi atribut ketika Node dibuat
7	~Node()	destructor	Menghapus *child* secara rekursif ketika Node dihapus

4.4.3 Desain Fungsi main

Fungsi *main* merupakan fungsi yang bertanggung jawab sebagai induk program untuk menerima masukan dan menampilkan keluaran yang sesuai. Fungsi *main* membagi metode penyelesaian menjadi tiga menurut dimensi. Pada fungsi *main*, digunakan fungsi *insert* dan *dominated_node*. Desain fungsi dan penjelasan variabelnya terdapat pada Kode Semu 4.9 dan Tabel 4.4.

Kode Semu 4.9 Fungsi main

```
1:   function main()
2:       read(T)
3:       for i ← 0 to T-1 do
4:       read(N, M)
5:       if M = 1 then
6:           while N > 0 do
```

```
7:                N ← N - 1
8:                read(temp)
9:            write("Case #", i + 1, ": 1\n")
10:       else if M = 2 then
11:           for j ← 0 to N-1 do
12:               read(points[j].first, points[j].second)
13:           sort(points)
14:           ans ← 1
15:           N ← N - 1
16:           thres ← points[N].second
17:           while N > 0 do
18:               if points[N].second > thres do
19:                   thres ← points[N].second
20:                   ans ← ans + 1
21:           write("Case #", i + 1, ": ", ans, "\n")
22:       else
23:           ans ← 1
24:           for j ← 0 to M-1 do
25:               read(points[0][j])
26:           root ← new Node(points[0])
27:           N ← N - 1
28:           while N > 0 do
29:               for j ← 0 to M-1 do
30:                   read(point[j])
31:               if dominated_node(point, root) = false then
32:                   insert(root, point, 0)
33:                   points[ans] ← point
34:                   ans ← ans + 1
35:               N ← N – 1
36:           delete root
37:           index ← ans – 1
38:           root ← new Node(points[index])
39:           while index > 0 do
40:               index ← index – 1
41:               if dominated_node(points[index], root) = true then
42:                   ans ← ans – 1
43:               else
44:                   insert(root, points[index], 0)
45:           delete root
46:           write("Case #", i + 1, ": ", ans, "\n")
47:   return 0
```

Tabel 4.4 Penjelasan Fungsi main

No	Nama	Tipe	Fungsi
1	T	integer	Menyimpan banyak kasus uji
2	N	integer	Menyimpan banyak titik
3	temp	integer	Menyimpan sementara masukan yang tidak diperlukan
4	points	pair array	Menyimpan masukan titik dimensi dua
5	ans	integer	Menyimpan banyak titik tidak didominasi

6	thres	integer	Menyimpan riwayat points.second terbesar
7	points	array 2d	Menyimpan titik dimensi tiga atau empat yang diduga merupakan titik tidak didominasi
8	root	pointer Node	Merujuk pada Node yang merupakan *root* dari *k-d tree*
9	point	integer array	Menyimpan masukan titik dimensi tiga atau empat
10	index	integer	Sebagai perantara mengakses points untuk fase kedua

4.4.4 Desain Fungsi insert

Fungsi *insert* merupakan fungsi yang digunakan pada kasus uji berdimensi tiga atau empat untuk memasukkan titik pada *k-d tree*. Fungsi *insert* secara rekursif menerima balikan untuk ditempatkan sebagai *child*. Desain fungsi dan penjelasan variabelnya terdapat pada Kode Semu 4.10 dan Tabel 4.5.

Kode Semu 4.10 Fungsi insert

```
1:  function insert(root, point, discriminator)
2:    if root = NULL then
3:      return new Node(point)
4:    for i ← 0 to M-1 do
5:      root.upper_bound[i] ← max(root.upper_bound[i], point[i])
6:    discriminator ← discriminator mod M
7:    if point[discriminator] < root.point[discriminator] then
8:      root.left ← insert(root.left, point, discriminator + 1)
9:    else
10:     root.right ← insert(root.right, point, discriminator + 1)
11:   return root
```

Tabel 4.5 Penjelasan Fungsi insert

No	Nama	Tipe	Fungsi
1	root	pointer Node	Merujuk pada Node yang merupakan *root* dari *subtree*
2	point	integer array	Titik yang ingin dimasukkan
3	discriminator	integer	Koordinat ke berapa penentu arah *child subtree*

4.4.5 Desain Fungsi dominated_point

Fungsi *dominated_point* digunakan pada kasus uji berdimensi tiga atau empat untuk menentukan apakah titik pertama didominasi titik kedua. Desain fungsi dan penjelasan variabelnya terdapat pada Kode Semu 4.11 dan Tabel 4.6.

Kode Semu 4.11 Fungsi dominated_point

```
1: function dominated_point(point1, point2)
2:    for i ← 0 to M-1 do
3:       if point1[i] > point2[i] then
4:          return false
5:    return true
```

Tabel 4.6 Penjelasan Fungsi dominated_point

No	Nama	Tipe	Fungsi
1	point1	integer array	Titik yang dicek apakah didominasi point2
2	point2	integer array	Titik yang dicek apakah mendominasi point1

4.4.6 Desain Fungsi dominated_node

Fungsi *dominated_node* digunakan pada kasus uji berdimensi tiga atau empat untuk menentukan apakah titik dari parameter didominasi oleh salah satu titik dalam *subtree*. Desain fungsi dan penjelasan variabelnya terdapat pada Kode Semu 4.12 dan Tabel 4.7.

Kode Semu 4.12 Fungsi dominated_node

```
1:   function dominated_node(point, root)
2:      if dominated_point(point, root.point) then
3:         return true
4:      if root.left <> NULL and dominated_point(point,
        root.left.upper_bound) then
5:         if dominated_node(point, root.left) then
6:            return true
7:      if root.right <> NULL and dominated_point(point,
        root.right.upper_bound) then
8:         if dominated_node(point, root.right) then
9:            return true
10:     return false
```

Tabel 4.7 Penjelasan Fungsi dominated_node

No	Nama	Tipe	Fungsi
1	point	integer array	Titik yang dicek apakah didominasi salah satu titik dalam *subtree*
2	root	pointer Node	Merujuk pada Node yang merupakan *root* dari *subtree*

4.5 Implementasi Solusi

Implementasi dari kode melibatkan beberapa *library* standar yang memberikan dukungan fundamental untuk pengembangan program dalam bahasa C++ dengan rinciannya terdapat pada Kode Sumber 4.1 dan Tabel 4.8.

```
1: #include<cstdio>
2: #include<algorithm>
```

Kode Sumber 4.1 *Library* yang digunakan

Tabel 4.8 Penjelasan Utilitas Library

No	Nama *Library*	Fungsi
1	cstdio	Memberikan akses pada fungsi yang berkaitan dengan operasi masukan dan keluaran berbasis teks (file I/O). Pada kode implementasi, fungsi yang digunakan adalah *scanf* dan *printf*.
2	algorithm	Memberikan akses pada struktur data dan fungsi yang berkaitan dengan operasi pada tipe data berentang elemen seperti *array* dan *vector*. Pada kode implementasi, struktur data yang digunakan adalah *pair*, sedangkan fungsi yang digunakan adalah *max* untuk mentukan nilai maksimum dari dua nilai, *sort* untuk mengurutkan *pair*, dan *copy* untuk menyalin *array*.

4.5.1 Implementasi Fungsi Makro REP dan Variabel Global M

Implementasi fungsi makro *REP* dan variabel global *M* menggunakan bahasa C++ sesuai pada Kode Semu 4.7 terdapat pada Kode Sumber 4.2.

```
#pragma GCC optimize("Ofast")
using namespace std;
#define REP(i, n) \
    for(int i=0; i<n; i++)
int M;
```

Kode Sumber 4.2 Implementasi Fungsi Makro REP dan Variabel Global M

4.5.2 Implementasi Struct Node

Implementasi *struct Node* menggunakan bahasa C++ sesuai pada Kode Semu 4.8 terdapat pada Kode Sumber 4.3.

```
struct Node {
  int point[4], upper_bound[4];
  Node *left, *right;
  Node(const int (&point)[4]){
    REP(i, M)
      this->point[i] = upper_bound[i] = point[i];
    left = right = NULL;
  }
  ~Node(){
    delete left;
    delete right;
  }
};
```

Kode Sumber 4.3 Implementasi struct Node

4.5.3 Implementasi Fungsi main

Implementasi fungsi *main* menggunakan bahasa C++ sesuai pada Kode Semu 4.9 terdapat pada Kode Sumber 4.4.

```
int main(){
  int T;
  scanf("%d", &T);
  REP(i, T){
    int N;
    scanf("%d%d", &N, &M);
    if(M == 1){
      int temp;
      while(N--)
        scanf("%d", &temp);
      printf("Case #%d: 1\n", i+1);
    }else if(M == 2){
      pair<int, int> points[N];
      REP(j, N)
        scanf("%d%d", &points[j].first, &points[j].second);
      sort(points, points + N);
```

```
17:         int ans = 1, thres = points[--N].second;
18:         while(N--)
19:           if(points[N].second > thres){
20:             thres = points[N].second;
21:             ans++;
22:           }
23:         printf("Case #%d: %d\n", i+1, ans);
24:       }else{
25:         int points[N][4], ans = 1;
26:         REP(j, M)
27:           scanf("%d", &points[0][j]);
28:         Node *root = new Node(points[0]);
29:         while(--N){
30:           int point[4];
31:           REP(j, M)
32:             scanf("%d", &point[j]);
33:           if(!dominated(point, root)){
34:             insert(root, point, 0);
35:             copy(point, point+M, points[ans++]);
36:           }
37:         }
38:         delete root;
39:         int index = ans;
40:         root = new Node(points[--index]);
41:         while(index--)
42:           if(dominated(points[index], root))
43:             ans--;
44:           else
45:             insert(root, points[index], 0);
46:         delete root;
47:         printf("Case #%d: %d\n", i+1, ans);
48:       }
49:     }
50:     return 0;
51:   }
```

Kode Sumber 4.4 Implementasi Fungsi main

4.5.4 Implementasi Fungsi insert

Implementasi fungsi *insert* menggunakan bahasa C++ sesuai pada Kode Semu 4.10 terdapat pada Kode Sumber 4.5.

```
1:   Node* insert(Node *root, const int (&point)[4], int
     discriminator){
2:     if(!root)
3:       return new Node(point);
4:     REP(i, M)
5:       root->upper_bound[i] = max(root->upper_bound[i],
         point[i]);
6:     discriminator %= M;
```

```
7:     if(point[discriminator] < root->point[discriminator])
8:       root->left = insert(root->left, point, discriminator + 1);
9:     else
10:      root->right = insert(root->right, point, discriminator +
         1);
11:    return root;
12:  }
```

Kode Sumber 4.5 Implementasi Fungsi insert

4.5.5 Implementasi Fungsi dominated_point

Implementasi fungsi *dominated_point* menggunakan bahasa C++ sesuai pada Kode Semu 4.11 terdapat pada Kode Sumber 4.6.

```
1:  bool dominated(const int (&point1)[4], const int (&point2)[4]){
2:     REP(i, M)
3:        if(point1[i] > point2[i])
4:           return false;
5:     return true;
6:  }
```

Kode Sumber 4.6 Implementasi Fungsi dominated_point

4.5.6 Implementasi Fungsi dominated_node

Implementasi fungsi *dominated_node* menggunakan bahasa C++ sesuai pada Kode Semu 4.12 terdapat pada Kode Sumber 4.7.

```
1:   bool dominated(const int (&point)[4], Node *root){
2:     if(dominated(point, root->point))
3:       return true;
4:     if(root->left && dominated(point, root->left->upper_bound))
5:       if(dominated(point, root->left))
6:         return true;
7:     if(root->right && dominated(point, root->right->upper_bound))
8:       if(dominated(point, root->right))
9:         return true;
10:    return false;
11:  }
```

Kode Sumber 4.7 Implementasi Fungsi dominated_node

4.6 Evaluasi Solusi

Pada subbab ini, dijelaskan hasil evaluasi yang dilakukan untuk membuktikan kebenaran dan mengukur kinerja program.

4.6.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan untuk melakukan uji coba kinerja lokal merupakan laptop pribadi penulis dengan spesifikasi perangkat keras dan perangkat lunak yang ditunjukkan pada Tabel 4.9.

Tabel 4.9 Spesifikasi Lingkungan Uji Coba Kinerja Lokal

No	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Mobile Processor* AMD Ryzen 7 7840HS (3.8 GHz / up to 5.1 GHz / 8 *Cores* / 16 *Threads*) • *Random Access Memory* 16384MB
2	Perangkat Lunak	• Sistem Operasi Windows 11 *Home Single Language* 64-bit • Visual Studio Code • *Compiler* GCC 13.1.0 (Rev6, Built by MSYS2 project)

Lingkungan uji coba untuk melakukan uji coba kebenaran dan kinerja luar menggunakan situs penilaian daring Eolymp dengan spesifikasi perangkat lunak yang ditunjukkan pada Tabel 4.10.

Tabel 4.10 Spesifikasi Lingkungan Uji Coba Kebenaran dan Kinerja Luar

No	Jenis Perangkat	Spesifikasi
1	Perangkat Lunak	• Bahasa pemrograman C++ 20 (GNU 10.2)

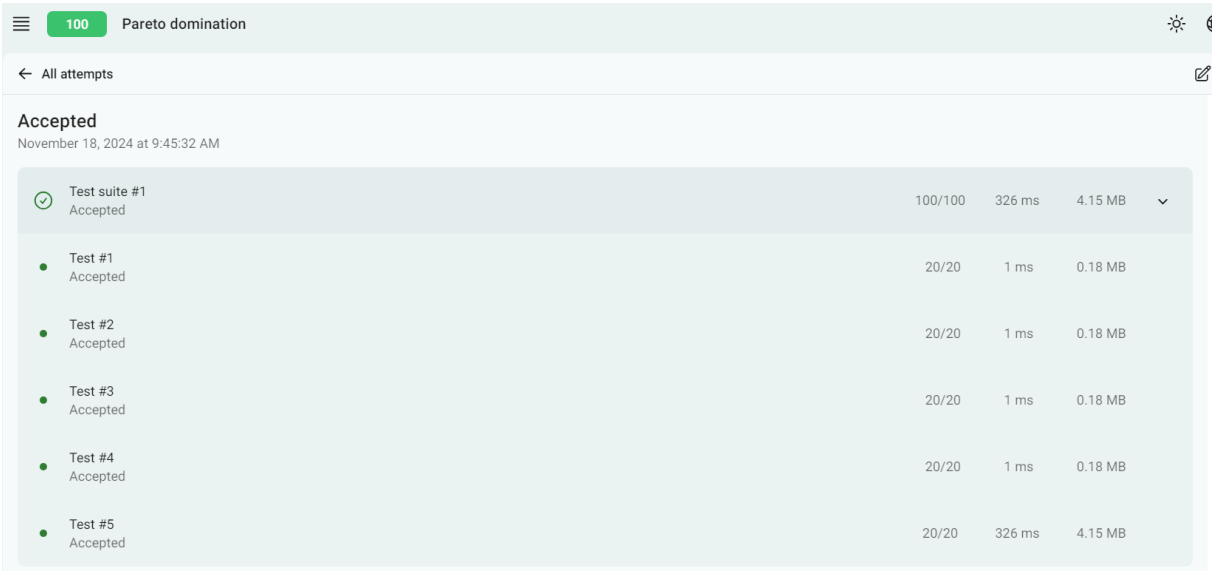
4.6.2 Skenario Uji Coba

Pengujian dilakukan dalam dua skenario, yakni pengujian lokal dan luar dengan rincian sebagai berikut.

1. Pengujian Luar
   Pengujian luar merupakan pengujian yang dilakukan menggunakan situs penilaian daring, yaitu Eolymp, untuk menguji kebenaran dan kinerja program. Terdapat dua kode yang perlu diujikan sebagai perbandingan kinerja algoritma. Kode pertama adalah kode yang menggunakan keluaran konstan pada dimensi satu, algoritma *greedy* pada dimensi dua, dan penerapan *k-d tree* pada dimensi tiga dan empat. Kode kedua adalah kode yang menggunakan keluaran konstan pada dimensi satu dan penerapan *k-d tree* pada dimensi dua, tiga, dan empat. Uji coba dilakukan dengan mengirimkan kode program algoritma ke situs penilaian daring Eolymp sebanyak 10 kali.
2. Pengujian Lokal
   Pengujian lokal merujuk pada proses pengujian yang menggunakan mesin yang sama dengan yang digunakan dalam tahap pengembangan dengan tujuan untuk mengevaluasi performa program. Pengujian melibatkan eksekusi kode program dan dilakukan pada sejumlah kasus uji yang disesuaikan dengan batasan yang telah ditetapkan dalam permasalahan. Dalam pengujian ini, kasus uji yang digunakan adalah kasus dengan N sama dengan 50, 500, 5000, dan 50000. Untuk menguji kompleksitas algoritma *greedy*, digunakan kasus uji $M = 2$. Untuk menguji kompleksitas penerapan k-d tree, digunakan kasus uji semua M, yakni dari 1 hingga 4. Uji berikutnya adalah perbandingan algoritma *greedy* dengan penerapan *k-d tree* pada M = 2 dengan data masukan yang sudah diurutkan. Masing-masing kasus dilakukan 10 kali uji coba.

4.6.3 Uji Coba Kebenaran

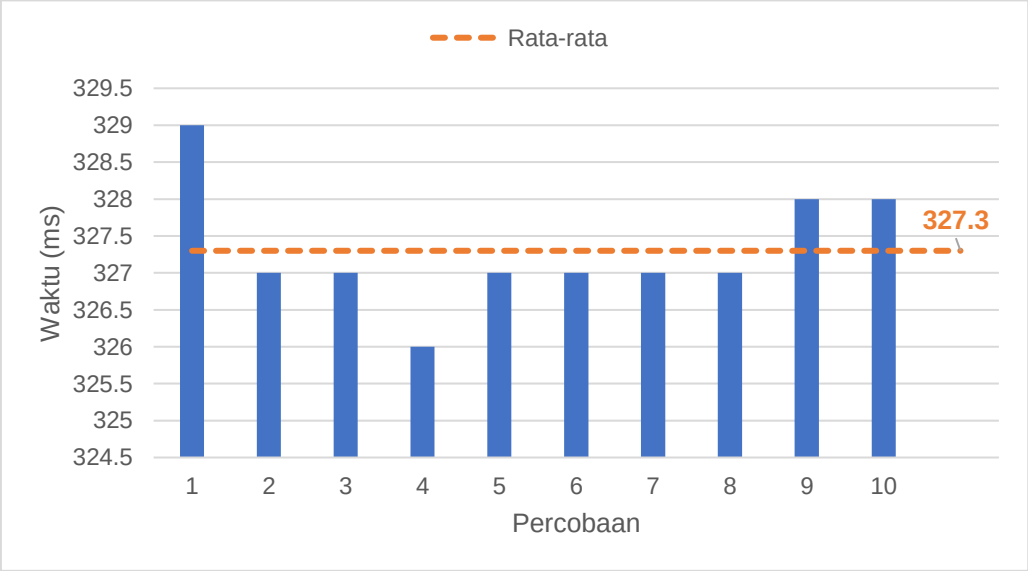
Uji coba kebenaran dari penyelesaian permasalahan Eolymp 100 “Pareto Domination” dilakukan dengan mengirim kode sumber terkait ke dalam situs penilaian daring Eolymp. Bukti hasil pengujian metode pada 18 November 2024 terdapat pada Gambar 4.26.



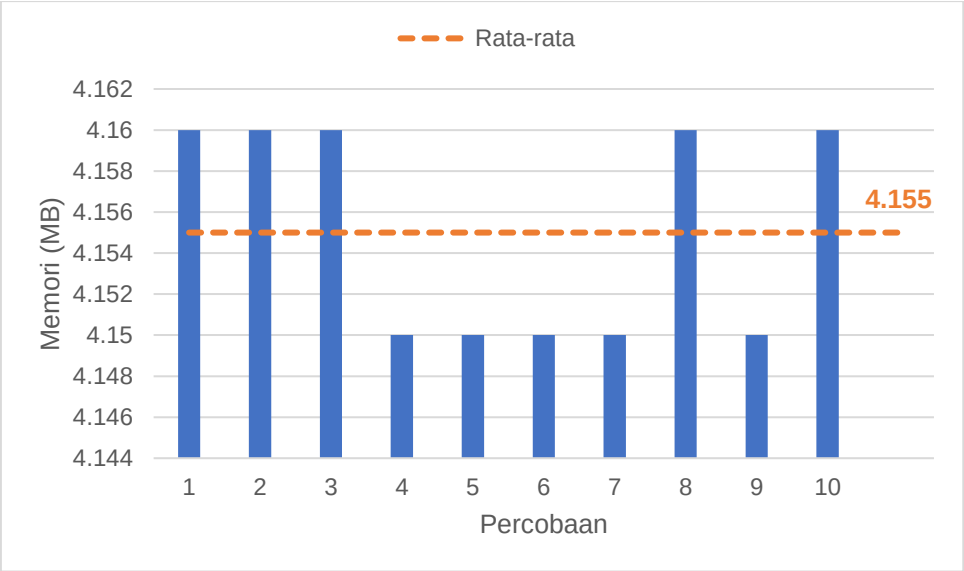
Gambar 4.26 Hasil Uji Coba Kebenaran Penerapan *K-d Tree*

4.6.4 Uji Coba Kinerja Luar

Subbab ini akan menjelaskan hasil uji coba kinerja luar hasil program penyelesaian permasalahan Eolymp 100 “Pareto Domination” dengan menggunakan keluaran konstan pada dimensi satu, algoritma *greedy* pada dimensi dua, dan penerapan *k-d tree* pada dimensi tiga dan empat serta keluaran konstan pada dimensi satu dan penerapan *k-d tree* pada dimensi dua, tiga, dan empat. Pengujian dilakukan dengan mengirim kode sumber terkait ke dalam situs penilaian daring Eolymp sebanyak 10 kali. Detail mengenai hasil uji kinerja dapat dilihat pada Lampiran B.

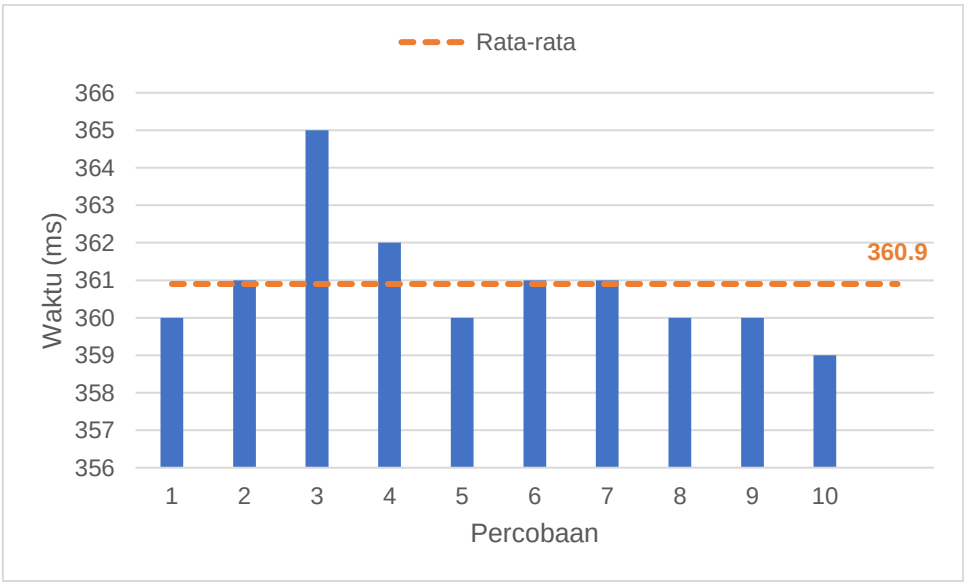


Gambar 4.27 Grafik Waktu Kinerja Luar Menggunakan Keluaran Konstan, Algoritma *Greedy*, dan Penerapan *K-d Tree*

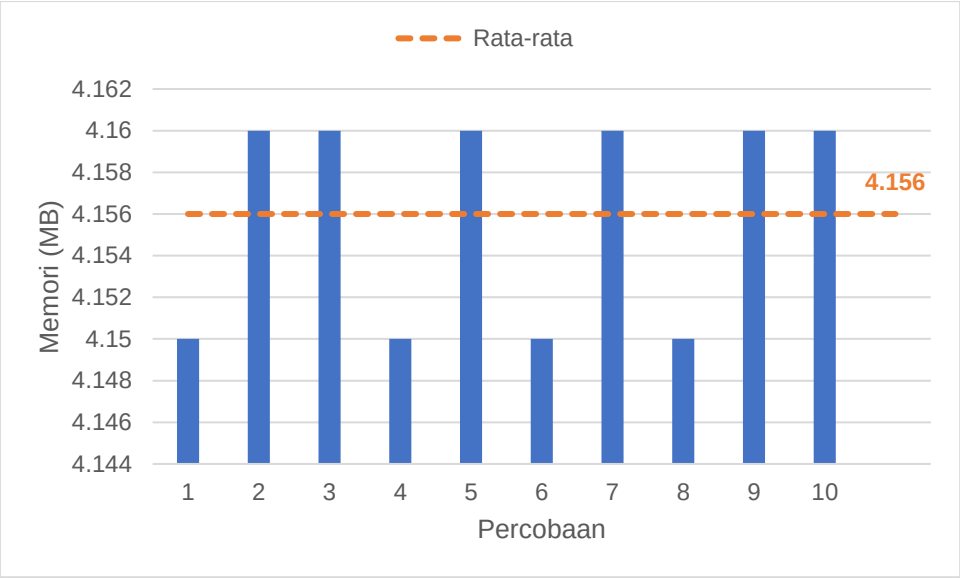


Gambar 4.28 Grafik Memori Kinerja Luar Menggunakan Keluaran Konstan, Algoritma *Greedy*, dan Penerapan *K-d Tree*

Pada penggunaan metode penyelesaian menggunakan keluaran konstan, algoritma *greedy*, dan penerapan *k-d tree*, pengujian 10 kali percobaan *submit* pada laman Eolymp pada 22 September 2024 menghasilkan rata-rata waktu kinerja sebesar 327,3 milidetik, dengan waktu minimal sebesar 326 milidetik dan waktu maksimal sebesar 329 milidetik sesuai pada Gambar 4.27. Rata-rata penggunaan memori sebesar 4,155 MB, dengan penggunaan memori minimum sebesar 4,15 MB dan penggunaan memori maksimum sebesar 4,16 MB sesuai Gambar 4.28.



Gambar 4.29 Grafik Waktu Kinerja Luar Menggunakan Keluaran Konstan dan Penerapan *K-d Tree*



Gambar 4.30 Grafik Memori Kinerja Luar Menggunakan Keluaran Konstan dan Penerapan *K-d Tree*

Pada penggunaan metode penyelesaian menggunakan keluaran konstan dan penerapan *k-d tree*, pengujian 10 kali percobaan *submit* pada laman Eolymp pada 18 November 2024 menghasilkan rata-rata waktu kinerja sebesar 360,9 milidetik, dengan waktu minimal sebesar 359 milidetik dan waktu maksimal sebesar 365 milidetik sesuai pada Gambar 4.29. Rata-rata penggunaan memori sebesar 4,156 MB, dengan penggunaan memori minimum sebesar 4,15 MB dan penggunaan memori maksimum sebesar 4,16 MB sesuai Gambar 4.30.

Pada uji coba kinerja luar, hasil waktu eksekusi kode yang menggunakan keluaran konstan, algoritma *greedy*, dan penerapan *k-d tree* lebih singkat dibandingkan dengan kode yang menggunakan keluaran konstan dan penerapan *k-d tree*. Namun, hal ini belum menandakan bahwa penggunaan algoritma *greedy* lebih baik daripada penerapan *k-d tree* sebagai penyelesaian persoalan dimensi dua karena tidak diketahui bagaimana format titik masukan untuk dimensi dua pada kasus uji. Oleh karena itu, untuk memastikan, perlu dilakukan uji coba kinerja lokal dengan membandingkan kedua metode.

4.6.5 Uji Coba Kinerja Lokal

Pada subbab ini, ditampilkan hasil uji coba kinerja lokal hasil program penyelesaian permasalahan Eolymp 100 “Pareto Domination” dengan algoritma *greedy* dan penerapan *k-d tree*. Pengujian lokal dilakukan terhadap kelompok masukan yang rinciannya terlampir pada Lampiran A dengan langkah sebagai berikut.

1. Rekam waktu tepat sebelum komputasi penyelesaian masalah.
2. Melakukan komputasi penyelesaian masalah untuk masukan kasus uji.
3. Rekam waktu tepat setelah komputasi penyelesaian masalah.
4. Hitung selisih waktu setelah dan sebelum komputasi.
5. Ulangi sebanyak 10 kali untuk masing-masing kombinasi kasus.
6. Hasil pengulangan tersebut dirata-rata sebagai bahan evaluasi.

Pengujian menggunakan *library chrono* sebagai media pengambilan waktu dengan ketelitian waktu paling tingginya menggunakan bilangan bulat bersatuan nanodetik. Masukan

titik menggunakan pembangkit titik acak berdistribusi seragam dalam *M-ball* sesuai dasar teori pada subbab 2.4.

Kasus uji $M = 2$ menggunakan algoritma *greedy* digunakan untuk mengukur kinerja dan membuktikan kompleksitas algoritma *greedy*. Hasil uji coba tersebut terdapat pada Tabel 4.11.

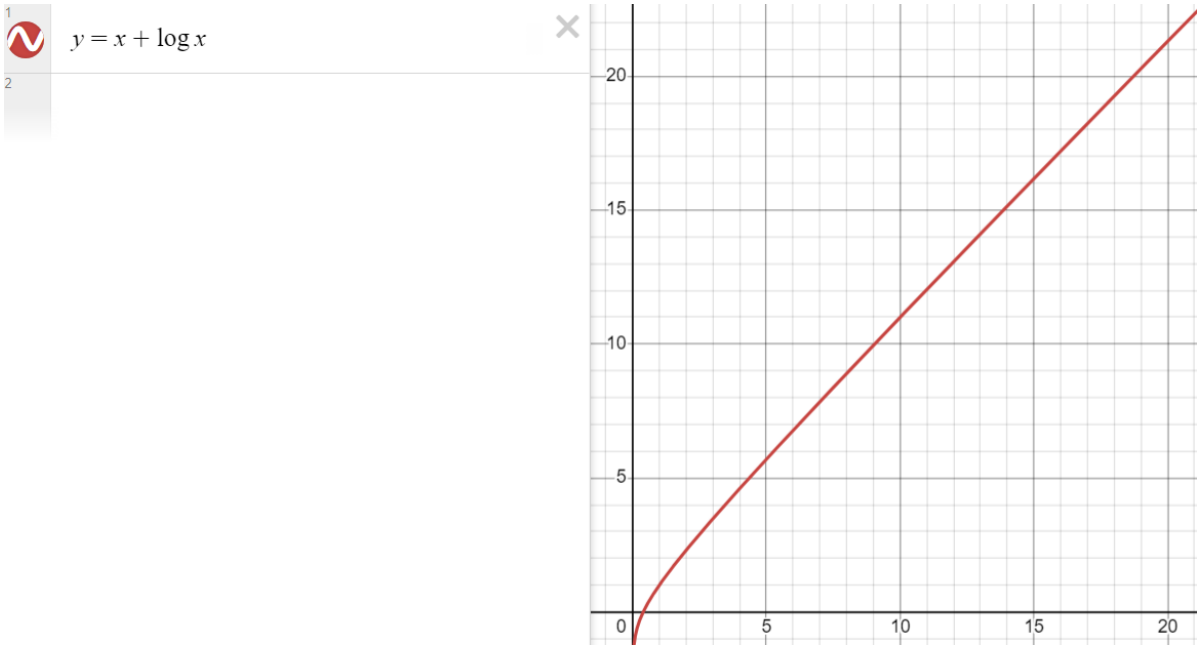
Tabel 4.11 Hasil Uji Coba Kinerja Lokal $M = 2$ Menggunakan Algoritma *Greedy*

N	Rata-rata waktu (nanodetik)
50	2.930
500	43.580
5000	660.490
50000	5.413.440

Untuk memudahkan visualisasi antara nilai N dengan rata-rata waktu ($\bar{t}$), kedua variabel akan divisulisasikan dalam skala logaritma. Sesuai kompleksitas algoritma *greedy*, dapat dibentuk persamaan (4.3).

$$N' = \log(N) \quad (4.1)$$
$$\bar{t}' = \log(\bar{t}) \quad (4.2)$$
$$\bar{t} = N\log(N) \quad (4.3)$$
$$\log(\bar{t}) = \log(N\log(N)) \quad (4.4)$$
$$\log(\bar{t}) = \log(N) + \log(\log(N)) \quad (4.5)$$
$$\bar{t}' = N' + \log(N') \quad (4.6)$$

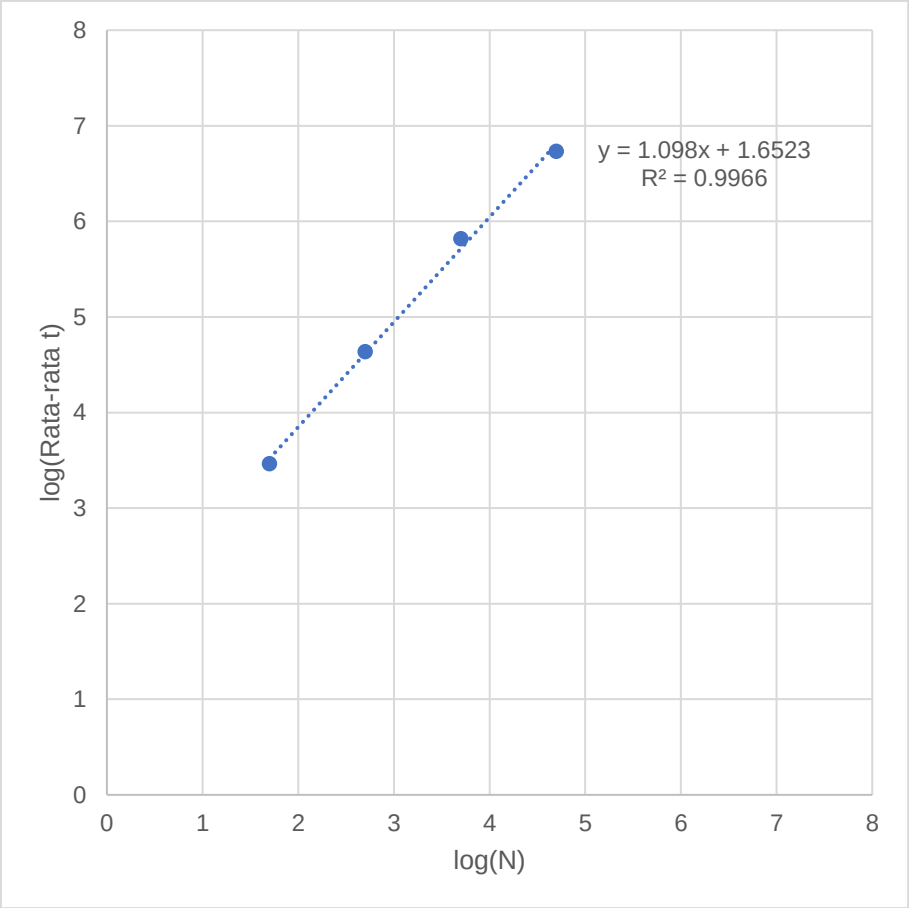
Persamaan (4.6) merupakan persamaan yang digunakan untuk visualisasi hasil uji coba dengan sumbu x menggunakan nilai N' dan sumbu y menggunakan nilai $\bar{t}'$. Persamaan (4.6) merupakan persamaan yang menghasilkan garis linear sesuai Gambar 4.31 yang menunjukkan hasil grafik pada situs desmos dengan persamaan serupa. Oleh karena itu, diharapkan visualisasi hasil juga membentuk persamaan linear.



Gambar 4.31 Grafik Persamaan $y = x + log\ x$

Tabel 4.12 Hasil Uji Coba Kinerja Lokal M = 2 Menggunakan Algoritma *Greedy* Skala Logaritma

N	N'	$\bar{t}'$
50	1,698970004	3,46686762
500	2,698970004	4,639287226
5000	3,698970004	5,819866247
50000	4,698970004	6,733473328



Gambar 4.32 Hasil Uji Coba Kinerja Lokal M = 2 Menggunakan Algoritma *Greedy* Skala Logaritma

Tabel 4.12 merupakan rincian hasil berskala logaritma, sedangkan Gambar 4.32 merupakan visualisasi hasil berskala logaritma menggunakan algoritma *greedy*. Hasilnya membentuk persamaan linear sehingga algoritma *greedy* terbukti berkompleksitas $O(N \log N)$.

Kasus uji untuk semua nilai M menggunakan penerapan *k-d tree* digunakan untuk mengukur kinerja, membuktikan kompleksitas, serta membandingkannya dengan algoritma *greedy*. Hasil uji coba tersebut terdapat pada Tabel 4.13 dan Tabel 4.14.

Tabel 4.13 Hasil Uji Coba Kinerja Lokal Menggunakan Penerapan *K-d Tree* Bagian Pertama

M	N	Rata-rata waktu (nanodetik)
1	50	2.050
	500	10.240

Tabel 4.14 Hasil Uji Coba Kinerja Lokal Menggunakan Penerapan *K-d Tree* Bagian Kedua

M	N	Rata-rata waktu (nanodetik)
1	5000	83.860
	50000	827.140
2	50	3.470
	500	18.870
	5000	133.710
	50000	1.235.900
3	50	4.950
	500	35.750
	5000	379.090
	50000	2.924.580
4	50	10.040
	500	99.980
	5000	899.030
	50000	6.803.970

Untuk memudahkan visualisasi antara nilai *N* dengan rata-rata waktu ($\bar{t}$), kedua variabel akan divisualisasikan dalam skala logaritma. Sesuai kompleksitas penerapan *k-d tree*, dapat dibentuk persamaan (4.10).

$$M' = \log(M) \tag{4.7}$$
$$N' = \log(N) \tag{4.8}$$
$$\bar{t}' = \log(\bar{t}) \tag{4.9}$$
$$\bar{t} = MN\log(N) \tag{4.10}$$
$$\log(\bar{t}) = \log(MN\log(N)) \tag{4.11}$$
$$\log(\bar{t}) = \log(M) + \log(N) + \log(\log(N)) \tag{4.12}$$
$$\bar{t}' = N' + \log(N') + M' \tag{4.13}$$

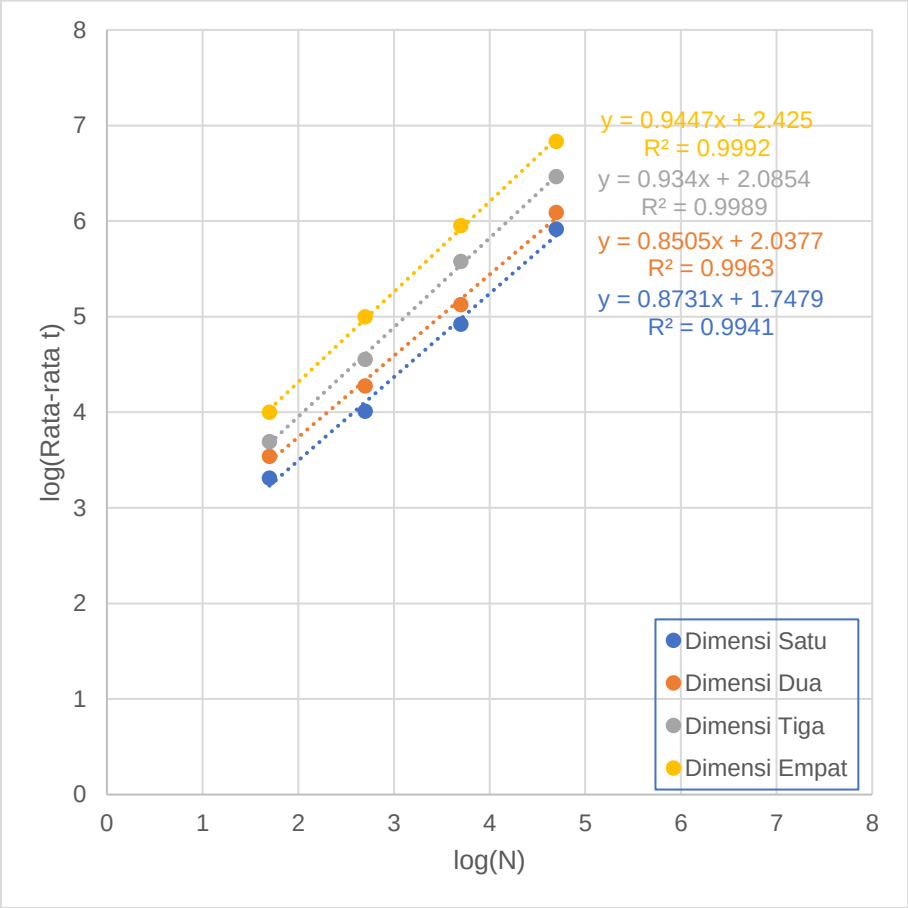
Persamaan (4.13) merupakan persamaan yang digunakan untuk visualisasi hasil uji coba dengan sumbu *x* menggunakan nilai N' dan sumbu *y* menggunakan nilai $\bar{t}'$. Untuk membedakan nilai *M*, digunakan variasi warna. Persamaan (4.13) jika digambarkan dengan mengabaikan M' akan membentuk garis linear sesuai pembuktian pada Gambar 4.31. Jika dibandingkan antar M', nilai yang berpengaruh ketika regresi linear adalah konstantanya. Oleh karena itu, diharapkan visualisasi hasil membentuk persamaan linear dengan bertambahnya konstanta seiring bertambahnya dimensi.

Tabel 4.15 Hasil Uji Coba Kinerja Lokal Menggunakan Penerapan *K-d Tree* Skala Logaritma Bagian Pertama

M	N	N'	$\bar{t}'$
1	50	1,698970004	3,311753861
	500	2,698970004	4,010299957
	5000	3,698970004	4,923554858
	50000	4,698970004	5,917579024
2	50	1,698970004	3,540329475
	500	2,698970004	4,2757719
	5000	3,698970004	5,126163889

Tabel 4.16 Hasil Uji Coba Kinerja Lokal Menggunakan Penerapan *K-d Tree* Skala Logaritma Bagian Kedua

M	N	N'	$\bar{t}'$
2	50000	4,698970004	6,091983332
3	50	1,698970004	3,694605199
	500	2,698970004	4,553276046
	5000	3,698970004	5,578742328
	50000	4,698970004	6,466063506
4	50	1,698970004	4,001733713
	500	2,698970004	4,999913132
	5000	3,698970004	5,953774184
	50000	4,698970004	6,83276239



Gambar 4.33 Hasil Uji Coba Kinerja Lokal Menggunakan Penerapan *K-d Tree* Skala Logaritma

Tabel 4.15 dan Tabel 4.16 merupakan rincian hasil berskala logaritma, sedangkan Gambar 4.33 merupakan visualisasi hasil berskala logaritma menggunakan penerapan *k-d tree*. Hasilnya membentuk persamaan linear di setiap dimensi serta konstanta bertambah seiring bertambahnya dimensi sehingga penerapan *k-d tree* terbukti berkompleksitas $O(MN \log N)$.

Tabel 4.17 Perbandingan Kinerja Algoritma *Greedy* dengan Penerapan *K-d Tree* pada $M = 2$

N	$\bar{t}$ *greedy*	$\bar{t}$ *k-d tree*
50	2.930	3.470
500	43.580	18.870
5000	660.490	133.710
50000	5.413.440	1.235.900

Tabel 4.17 menunjukkan perbandingan rata-rata waktu eksekusi uji coba kinerja lokal antara algoritma *greedy* dan penerapan *k-d tree* pada persoalan dimensi dua. Hasilnya, waktu eksekusi penerapan *k-d tree* secara garis besar lebih baik daripada algoritma *greedy*. Hal ini berbanding terbalik dengan hasil pada uji coba kinerja luar.

Penulis berasumsi bahwa kasus uji untuk dimensi dua pada situs Eolymp menggunakan data masukan yang hampir terurut sehingga operasi pengurutan pada algoritma *greedy* berjalan lebih ringan. Oleh karena itu, perlu dilakukan uji coba kembali menggunakan titik masukan yang sudah diurutkan. Hasil uji coba terdapat pada Tabel 4.18.

Tabel 4.18 Perbandingan Kinerja Algoritma *Greedy* dengan Penerapan *K-d Tree* pada $M = 2$ dengan Titik Masukan yang Sudah Diurutkan

N	$\bar{t}$ *greedy*	$\bar{t}$ *k-d tree*
50	2.110	8.550
500	21.820	90.520
5000	279.310	1.721.240
50000	3.060.520	29.413.060

Hasil pada Tabel 4.18 menunjukkan bahwa waktu eksekusi algoritma *greedy* jauh lebih singkat daripada penerapan *k-d tree* sehingga asumsi bahwa kasus uji untuk dimensi dua pada situs Eolymp menggunakan data masukan yang hampir terurut terbukti benar. Hal ini terjadi karena jika menggunakan titik masukan yang hampir terurut, *2-d tree* yang terbentuk tidak seimbang, condong menuju *subtree* kanan sehingga kompleksitas waktu yang seharusnya bisa dipersingkat menjadi logaritmik kembali menjadi linear.

BAB V KESIMPULAN

5.1 Kesimpulan

Dari hasil uji coba pada permasalahan Eolymp 100 “Pareto Domination”, dapat diambil beberapa kesimpulan terkait penyelesaian persoalan dominasi Pareto untuk menentukan banyak titik tidak didominasi, yakni sebagai berikut.

1. Struktur *k-d tree* diterapkan sebagai penyelesaian persoalan dominasi Pareto, secara spesifik pada studi kasus Eolymp 100 “Pareto Domination”, dengan dikombinasikan teknik *bounding box* dan pendekatan dua fase. Namun, penerapan tersebut pada studi kasus hanya digunakan pada persoalan dimensi tiga dan empat. Pada dimensi satu, banyak titik tidak didominasi pasti sama dengan satu sehingga digunakan keluaran konstan. Pada dimensi dua, digunakan algoritma *greedy* karena titik masukan hampir terurut. Penyelesaian persoalan dimensi satu menggunakan keluaran konstan menghasilkan kompleksitas waktu $O(1)$. Penyelesaian persoalan dimensi dua menggunakan algoritma *greedy* menghasilkan kompleksitas waktu $O(N \log N)$. Penyelesaian persoalan dimensi tiga dan empat menggunakan penerapan *k-d tree* menghasilkan kompleksitas waktu $O(MN \log N)$, dengan *M* merupakan dimensi dan *N* merupakan banyak titik.
2. Dengan mematuhi batasan masalah yang tercantum pada permasalahan Eolymp 100 “Pareto Domination”, penerapan *k-d tree* berhasil menyelesaikan permasalahan dengan waktu eksekusi minimum 326 milidetik, waktu eksekusi maksimum 329 milidetik, penggunaan memori minimum 4,15 MB, dan penggunaan memori maksimum 4,16 MB pada 22 September 2024. Namun, hasil yang teroptimal tersebut didapat dengan menerapkan *k-d tree* hanya pada persoalan dimensi tiga dan empat, sedangkan pada dimensi satu, digunakan keluaran konstan dan pada dimensi dua, digunakan algoritma *greedy*.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] The University of Chicago, “Pareto Concepts”, Diakses: 22 Januari 2025. [Daring]. Tersedia pada: https://home.uchicago.edu/bdm/pepp/pareto.pdf

[2] W.-M. Chen, H.-K. Hwang, dan T.-H. Tsai, “Maxima-finding algorithms for multidimensional samples: A two-phase approach,” *Computational Geometry*, vol. 45, no. 1, hlm. 33–53, 2012, doi: https://doi.org/10.1016/j.comgeo.2011.08.001.

[3] H. Kung, F. Luccio, dan F. Preparata, “On Finding the Maxima of a Set of Vectors,” *Journal of the ACM (JACM)*, vol. 22, hlm. 469–476, Sep 1975, doi: 10.1145/321906.321910.

[4] M. Sun dan R. Steuer, “Quad-Trees and Linear Lists for Identifying Nondominated Criterion Vectors,” *INFORMS J Comput*, vol. 8, hlm. 367–375, Nov 1996, doi: 10.1287/ijoc.8.4.367.

[5] R. Harman dan V. Lacko, “On decompositional algorithms for uniform sampling from n-spheres and n-balls,” *J Multivar Anal*, vol. 101, no. 10, hlm. 2297–2304, 2010, doi: https://doi.org/10.1016/j.jmva.2010.06.002.

(Halaman ini sengaja dikosongkan)

LAMPIRAN A: DATA UJI LOKAL

Berikut merupakan data yang digunakan untuk melakukan uji lokal pada Tabel A.1 dan Tabel A.2. Terdapat enam kolom pada data uji. Kolom pertama merupakan nomor data uji. Kolom kedua merupakan masukan *T*, yakni banyak kasus uji. Kolom ketiga merupakan masukan *N*, yakni banyak titik. Kolom keempat merupakan masukan *M*, yakni dimensi. Kolom kelima merupakan masukan himpunan titik. Kolom keeenam merupakan hasil keluaran untuk setiap kasus uji.

Tabel A.1 Data Uji Bagian Pertama

No	T	N	M	Himpunan Titik	Hasil Keluaran	
1	10	50	1	50 titik acak berdimensi satu berdistribusi seragam	Case #1: 1 Case #2: 1 Case #3: 1 Case #4: 1 Case #5: 1	Case #6: 1 Case #7: 1 Case #8: 1 Case #9: 1 Case #10: 1
2	10	500	1	500 titik acak berdimensi satu berdistribusi seragam	Case #1: 1 Case #2: 1 Case #3: 1 Case #4: 1 Case #5: 1	Case #6: 1 Case #7: 1 Case #8: 1 Case #9: 1 Case #10: 1
3	10	5000	1	5000 titik acak berdimensi satu berdistribusi seragam	Case #1: 1 Case #2: 1 Case #3: 1 Case #4: 1 Case #5: 1	Case #6: 1 Case #7: 1 Case #8: 1 Case #9: 1 Case #10: 1
4	10	50000	1	50000 titik acak berdimensi satu berdistribusi seragam	Case #1: 1 Case #2: 1 Case #3: 1 Case #4: 1 Case #5: 1	Case #6: 1 Case #7: 1 Case #8: 1 Case #9: 1 Case #10: 1
5	10	50	2	50 titik acak berdimensi dua berdistribusi seragam dalam *2-ball*	Case #1: 7 Case #2: 7 Case #3: 7 Case #4: 7 Case #5: 7	Case #6: 7 Case #7: 7 Case #8: 7 Case #9: 7 Case #10: 7
6	10	500	2	500 titik acak berdimensi dua berdistribusi seragam dalam *2-ball*	Case #1: 14 Case #2: 14 Case #3: 14 Case #4: 14 Case #5: 14	Case #6: 14 Case #7: 14 Case #8: 14 Case #9: 14 Case #10: 14
7	10	5000	2	5000 titik acak berdimensi dua berdistribusi seragam dalam *2-ball*	Case #1: 30 Case #2: 30 Case #3: 30 Case #4: 30 Case #5: 30	Case #6: 30 Case #7: 30 Case #8: 30 Case #9: 30 Case #10: 30

Tabel A.2 Data Uji Bagian Kedua

No	T	N	M	Himpunan Titik	Hasil Keluaran	
8	10	50000	2	50000 titik acak berdimensi dua berdistribusi seragam dalam *2-ball*	Case #1: 58 Case #2: 58 Case #3: 58 Case #4: 58 Case #5: 58	Case #6: 58 Case #7: 58 Case #8: 58 Case #9: 58 Case #10: 58
9	10	50	3	50 titik acak berdimensi dua berdistribusi seragam dalam *3-ball*	Case #1: 15 Case #2: 15 Case #3: 15 Case #4: 15 Case #5: 15	Case #6: 15 Case #7: 15 Case #8: 15 Case #9: 15 Case #10: 15
10	10	500	3	500 titik acak berdimensi dua berdistribusi seragam dalam *3-ball*	Case #1: 51 Case #2: 51 Case #3: 51 Case #4: 51 Case #5: 51	Case #6: 51 Case #7: 51 Case #8: 51 Case #9: 51 Case #10: 51
11	10	5000	3	5000 titik acak berdimensi dua berdistribusi seragam dalam *3-ball*	Case #1: 140 Case #2: 140 Case #3: 140 Case #4: 140 Case #5: 140	Case #6: 140 Case #7: 140 Case #8: 140 Case #9: 140 Case #10: 140
12	10	50000	3	50000 titik acak berdimensi dua berdistribusi seragam dalam *3-ball*	Case #1: 395 Case #2: 395 Case #3: 395 Case #4: 395 Case #5: 395	Case #6: 395 Case #7: 395 Case #8: 395 Case #9: 395 Case #10: 395
13	10	50	4	50 titik acak berdimensi dua berdistribusi seragam dalam *4-ball*	Case #1: 23 Case #2: 23 Case #3: 23 Case #4: 23 Case #5: 23	Case #6: 23 Case #7: 23 Case #8: 23 Case #9: 23 Case #10: 23
14	10	500	4	500 titik acak berdimensi dua berdistribusi seragam dalam *4-ball*	Case #1: 91 Case #2: 91 Case #3: 91 Case #4: 91 Case #5: 91	Case #6: 91 Case #7: 91 Case #8: 91 Case #9: 91 Case #10: 91
15	10	5000	4	5000 titik acak berdimensi dua berdistribusi seragam dalam *4-ball*	Case #1: 375 Case #2: 375 Case #3: 375 Case #4: 375 Case #5: 375	Case #6: 375 Case #7: 375 Case #8: 375 Case #9: 375 Case #10: 375
16	10	50000	4	50000 titik acak berdimensi dua berdistribusi seragam dalam *4-ball*	Case #1: 1319 Case #2: 1319 Case #3: 1319 Case #4: 1319 Case #5: 1319	Case #6: 1319 Case #7: 1319 Case #8: 1319 Case #9: 1319 Case #10: 1319

Tabel A.3 Data Uji Bagian Ketiga

No	T	N	M	Himpunan Titik	Hasil Keluaran	
17	10	50	2	50 titik acak berdimensi dua berdistribusi seragam dalam *2-ball* yang telah diurutkan	Case #1: 7 Case #2: 7 Case #3: 7 Case #4: 7 Case #5: 7	Case #6: 7 Case #7: 7 Case #8: 7 Case #9: 7 Case #10: 7
18	10	500	2	500 titik acak berdimensi dua berdistribusi seragam dalam *2-ball* yang telah diurutkan	Case #1: 14 Case #2: 14 Case #3: 14 Case #4: 14 Case #5: 14	Case #6: 14 Case #7: 14 Case #8: 14 Case #9: 14 Case #10: 14
19	10	5000	2	5000 titik acak berdimensi dua berdistribusi seragam dalam *2-ball* yang telah diurutkan	Case #1: 30 Case #2: 30 Case #3: 30 Case #4: 30 Case #5: 30	Case #6: 30 Case #7: 30 Case #8: 30 Case #9: 30 Case #10: 30
20	10	50000	2	50000 titik acak berdimensi dua berdistribusi seragam dalam *2-ball* yang telah diurutkan	Case #1: 58 Case #2: 58 Case #3: 58 Case #4: 58 Case #5: 58	Case #6: 58 Case #7: 58 Case #8: 58 Case #9: 58 Case #10: 58

(Halaman ini sengaja dikosongkan)

LAMPIRAN B: HASIL UJI COBA PADA SITUS EOLYMP

Berikut merupakan lampiran hasil *submit* kode program dengan keluaran konstan, algoritma *greedy*, dan penerapan *k-d tree* sebanyak 10 kali seperti yang tampak pada Gambar B.1.

Status	Language	Time	Memory
Accepted 3 months ago	C++ 20 (gnu 10.2)	328 ms	4.16 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	328 ms	4.15 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	327 ms	4.16 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	327 ms	4.15 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	327 ms	4.15 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	327 ms	4.15 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	326 ms	4.15 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	327 ms	4.16 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	327 ms	4.16 MB
Accepted 3 months ago	C++ 20 (gnu 10.2)	329 ms	4.16 MB

Gambar B.1 Hasil Uji Coba Kode Program Keluaran Konstan, *Algoritma Greedy*, dam Penerapan *K-d Tree*

Berikut merupakan lampiran hasil *submit* kode program dengan keluaran konstan dan penerapan *k-d tree* sebanyak 10 kali seperti yang tampak pada Gambar B.2.

Status	Language	Time	Memory
Accepted 1 day ago	C++ 20 (gnu 10.2)	359 ms	4.16 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	360 ms	4.16 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	360 ms	4.15 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	361 ms	4.16 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	361 ms	4.15 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	360 ms	4.16 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	362 ms	4.15 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	365 ms	4.16 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	361 ms	4.16 MB
Accepted 1 day ago	C++ 20 (gnu 10.2)	360 ms	4.15 MB

Gambar B.2 Hasil Uji Coba Kode Program Keluaran Konstan dan Penerapan *K-d Tree*

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Penulis dilahirkan di Surabaya, pada tanggal 7 Maret 2003. Penulis merupakan anak pertama dari empat bersaudara. Penulis telah menempuh pendidikan formal di TK Dharma Wanita Persatuan Surabaya (2007-2009), SDN Baratajaya Surabaya (2009-2015), SMPN 1 Surabaya (2015-2018), dan SMAN 5 Surabaya (2018-2021). Penulis mengikuti SNMPTN dan diterima di Departemen Teknik Informatika FTEIC-ITS pada tahun 2021 dan terdaftar dengan NRP 5025211016. Penulis dapat dihubungi melalui surel (email) di thomasjuanmahardika@gmail.com.