Implementasi Divide and Conquer pada Model KNN Menggunakan KD-Tree untuk Pencarian Tetangga Terdekat

Muhammad Adli Arindra - 18222089
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: adliarindra27@gmail.com , 18222089@std.stei.itb.ac.id

***Abstrak*—**Algoritma K-Nearest Neighbor (KNN) merupakan salah satu metode klasifikasi yang sederhana namun efektif dalam berbagai aplikasi pengenalan pola dan pembelajaran mesin. Namun, pendekatan pencarian tetangga terdekat secara brute-force pada algoritma ini memiliki kompleksitas komputasi yang tinggi, terutama saat diterapkan pada dataset berukuran besar. Untuk mengatasi permasalahan tersebut, penelitian ini mengimplementasikan struktur data K-Dimensional Tree (KD-Tree) dengan pendekatan Divide and Conquer guna mengoptimalkan proses pencarian tetangga terdekat. KD-Tree membagi ruang data secara rekursif berdasarkan dimensi fitur, sehingga memungkinkan proses pencarian yang lebih efisien. Eksperimen dilakukan menggunakan dataset acak berdimensi dua sebanyak 100.000 titik dengan nilai K=100K=100. Hasil implementasi menunjukkan bahwa KD-Tree mampu mengurangi jumlah iterasi dan waktu pencarian secara signifikan dibandingkan metode brute-force, tanpa mengurangi akurasi hasil. Dengan demikian, KD-Tree terbukti efektif dalam meningkatkan efisiensi algoritma KNN pada data berdimensi rendah dan berskala besar.

Kata Kunci*—K-Nearest Neighbor; KD-Tree; Divide and Conquer; pencarian tetangga terdekat; klasifikasi.*

I. PENDAHULUAN

A. Latar Belakang

Dalam dunia komputasi modern, pencarian data yang efisien merupakan salah satu tantangan utama, terutama ketika berhadapan dengan data berdimensi tinggi. Algoritma K-Nearest Neighbor (KNN) adalah salah satu metode klasifikasi sederhana namun efektif yang banyak digunakan dalam berbagai bidang, seperti pengenalan pola, sistem rekomendasi, dan pengolahan citra. Namun, kelemahan utama dari KNN terletak pada performa pencarian tetangga terdekatnya yang cukup lambat, terutama ketika ukuran data sangat besar.

Untuk mengatasi permasalahan ini, diperlukan struktur data dan pendekatan algoritmik yang dapat mengurangi waktu pencarian. Salah satu solusi yang populer adalah penggunaan K-Dimensional Tree (KD-Tree), yaitu struktur data pohon yang memungkinkan pencarian spasial dilakukan secara efisien. KD-Tree bekerja berdasarkan prinsip Divide and Conquer, yaitu dengan membagi ruang data ke dalam beberapa bagian secara rekursif sehingga proses pencarian dapat dilakukan lebih cepat dengan mengeliminasi sebagian besar data yang tidak relevan.

Penggabungan konsep Divide and Conquer dalam pembentukan dan penggunaan KD-Tree pada algoritma KNN membuka peluang untuk peningkatan efisiensi, baik dari segi waktu eksekusi maupun penggunaan memori. Oleh karena itu, makalah ini akan membahas bagaimana implementasi KD-Tree sebagai optimalisasi algoritma KNN dapat dilakukan, serta mengevaluasi kinerjanya dibandingkan dengan pendekatan pencarian KNN secara langsung atau brute-force.

B. Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, maka rumusan masalah dalam makalah ini adalah sebagai berikut:

1. Bagaimana cara mengimplementasikan struktur KD-Tree menggunakan pendekatan Divide and Conquer pada algoritma K-Nearest Neighbor (KNN)?
2. Seberapa besar peningkatan efisiensi pencarian tetangga terdekat menggunakan KD-Tree dibandingkan metode brute-force?
3. Apa saja kelebihan dan kekurangan penggunaan KD-Tree dalam konteks pencarian KNN?

C. Tujuan

Adapun tujuan dari penulisan makalah ini adalah:

1. Menjelaskan proses implementasi algoritma KNN yang dioptimalkan menggunakan KD-Tree dengan pendekatan Divide and Conquer.
2. Menganalisis performa pencarian tetangga terdekat antara metode brute-force dan KD-Tree dalam hal efisiensi waktu dan kompleksitas algoritma.
3. Mengidentifikasi keunggulan dan keterbatasan dari struktur KD-Tree dalam penerapan pada model KNN.

II. DASAR TEORI

A. Algoritma K-Nearest Neighbor (KNN)

Algoritma K-Nearest Neighbor (KNN) merupakan salah satu metode dalam machine learning yang termasuk dalam kategori lazy learning dan instance-based learning [2]. KNN bekerja dengan cara menyimpan seluruh data pelatihan, dan

ketika menerima data baru, algoritma akan mencari sejumlah kk tetangga terdekat dari data tersebut untuk menentukan kelas atau nilai prediksinya. Kedekatan antar data umumnya dihitung menggunakan metrik jarak, seperti Euclidean distance, Manhattan distance, atau Minkowski distance.

Kelebihan utama dari KNN adalah kesederhanaan implementasinya serta kemampuannya untuk bekerja dengan baik pada berbagai jenis data, termasuk data yang tidak memiliki asumsi distribusi tertentu. Namun, kelemahan utamanya terletak pada kebutuhan komputasi yang tinggi saat proses prediksi, terutama jika ukuran dataset sangat besar. Setiap pencarian membutuhkan perhitungan jarak ke seluruh data pelatihan, yang menyebabkan waktu pencarian menjadi lambat dan tidak efisien.

Dalam konteks klasifikasi, KNN menentukan label data baru berdasarkan mayoritas dari kk tetangga terdekat. Sedangkan dalam regresi, KNN menggunakan rata-rata dari nilai-nilai tetangga tersebut. Karena KNN tidak melakukan proses pelatihan yang eksplisit, waktu pelatihan sangat singkat atau bahkan tidak ada, tetapi waktu prediksi menjadi mahal secara komputasi. Oleh karena itu, dibutuhkan optimalisasi pencarian tetangga seperti penggunaan struktur data yang lebih efisien, salah satunya adalah KD-Tree.

Dengan menggabungkan KNN dengan struktur data seperti KD-Tree, proses pencarian tetangga terdekat dapat dipercepat secara signifikan, karena tidak perlu memeriksa semua titik dalam dataset. Hal ini memungkinkan KNN untuk tetap digunakan pada skenario real-time atau ketika berhadapan dengan dataset yang sangat besar.

B. Konsep Divide and Conquer

Divide and Conquer merupakan salah satu paradigma algoritma yang sangat umum digunakan dalam penyelesaian masalah komputasi [3]. Prinsip dasar dari pendekatan ini adalah membagi masalah besar menjadi beberapa submasalah yang lebih kecil (divide), menyelesaikan masing-masing submasalah secara rekursif (conquer), lalu menggabungkan hasil-hasilnya untuk mendapatkan solusi akhir (combine). Paradigma ini telah terbukti efektif dalam banyak algoritma klasik, seperti Merge Sort, Quick Sort, dan Binary Search.

Kelebihan utama dari Divide and Conquer terletak pada kemampuannya untuk mengurangi kompleksitas waktu secara signifikan, terutama ketika submasalah dapat diselesaikan lebih cepat dibandingkan menyelesaikan keseluruhan masalah secara langsung. Selain itu, pendekatan ini secara alami cocok diimplementasikan secara rekursif, yang sering kali menghasilkan kode yang lebih sederhana dan modular.

Dalam konteks struktur data spasial seperti KD-Tree, Divide and Conquer digunakan dalam proses pembentukan pohon. Titik-titik data dibagi secara rekursif ke dalam dua bagian berdasarkan satu dimensi tertentu, sehingga setiap simpul pada pohon KD-Tree merepresentasikan keputusan pembagian ruang data. Pembagian ini terus dilakukan sampai mencapai kondisi dasar, seperti jumlah data yang sangat sedikit atau kedalaman tertentu dari pohon.

Dengan menerapkan Divide and Conquer dalam pembentukan dan traversal KD-Tree, pencarian tetangga terdekat dapat dilakukan dengan lebih efisien. Banyak ruang dalam dataset dapat dieliminasi dari pencarian karena tidak mungkin mengandung tetangga yang lebih dekat daripada yang sudah ditemukan, sehingga mengurangi jumlah perhitungan jarak secara drastis dibandingkan metode brute-force.

C. K-Dimensional Tree (KD-Tree)

K-Dimensional Tree (KD-Tree) adalah struktur data pohon biner yang dirancang untuk mengatur dan mencari data dalam ruang berdimensi banyak (multi-dimensional space). KD-Tree pertama kali diperkenalkan oleh Jon Bentley pada tahun 1975 [1] dan menjadi salah satu metode paling populer dalam pencarian spasial, seperti pencarian tetangga terdekat (nearest neighbor search), pencarian dalam jangkauan tertentu (range search), serta aplikasi lainnya dalam pengolahan citra, pengenalan pola, dan machine learning.

KD-Tree bekerja dengan membagi ruang data secara rekursif menggunakan prinsip Divide and Conquer [1]. Setiap simpul pada pohon mewakili pembagian ruang berdasarkan satu dimensi. Misalnya, pada dimensi dua (2D), pembagian dilakukan secara bergantian antara sumbu-x dan sumbu-y. Proses ini berlanjut secara rekursif hingga seluruh titik dimasukkan ke dalam pohon. Hasil akhirnya adalah struktur pohon yang membagi ruang menjadi region-region yang memungkinkan pencarian spasial dilakukan lebih cepat.

Dalam konteks algoritma KNN, KD-Tree digunakan untuk mempercepat proses pencarian tetangga terdekat. Alih-alih menghitung jarak ke seluruh titik dalam dataset (seperti pada metode brute-force), KD-Tree memungkinkan pencarian hanya pada bagian ruang yang relevan. Hal ini dilakukan dengan melakukan traversal ke cabang pohon yang potensial dan melakukan proses backtracking untuk memastikan tidak ada kandidat tetangga terdekat yang terlewatkan.

Keuntungan dari KD-Tree sangat terasa ketika jumlah data besar dan dimensi relatif rendah (umumnya < 20 dimensi). Namun, performa KD-Tree akan menurun pada data berdimensi sangat tinggi karena fenomena curse of dimensionality, yang menyebabkan pembagian ruang menjadi tidak lagi efektif. Meski demikian, KD-Tree tetap menjadi solusi praktis yang efisien dalam banyak kasus nyata, terutama ketika digunakan dalam implementasi KNN untuk klasifikasi dan regresi.

D. Kompleksitas Algoritma Pencarian Tetangga Terdekat

Pencarian tetangga terdekat (nearest neighbor search) merupakan inti dari algoritma KNN dan menjadi faktor utama dalam menentukan efisiensi algoritma secara keseluruhan. Dalam pendekatan brute-force, pencarian dilakukan dengan menghitung jarak antara titik kueri dan seluruh titik dalam dataset. Jika terdapat nn data dan setiap data memiliki dd dimensi, maka kompleksitas waktu pencarian brute-force adalah $O(n \cdot d)O(n \cdot d)$. Pendekatan ini sangat tidak efisien untuk dataset besar karena seluruh titik harus diperiksa satu per satu.

Dengan menggunakan struktur data seperti KD-Tree, kompleksitas waktu pencarian dapat dikurangi secara signifikan. Dalam kasus terbaik, KD-Tree memungkinkan pencarian dilakukan dengan kompleksitas rata-rata $O(logn)O(logn)$, meskipun dalam kasus terburuk—misalnya

jika data tidak seimbang atau dimensi terlalu tinggi—kompleksitasnya dapat mendekati O(*n*)O(*n*). Hal ini tetap memberikan keuntungan signifikan dibandingkan pendekatan brute-force pada banyak kasus praktis, khususnya untuk data berdimensi rendah sampai sedang.

KD-Tree memanfaatkan fakta bahwa sebagian besar ruang data tidak perlu diperiksa selama proses pencarian. Dengan memotong ruang data berdasarkan dimensi tertentu dan membatasi pencarian hanya pada region yang mungkin mengandung solusi, banyak titik dapat dieliminasi tanpa perhitungan jarak eksplisit. Proses ini mengandalkan traversal selektif dan strategi backtracking ketika perlu memeriksa sisi pohon lainnya yang masih berpotensi mengandung tetangga lebih dekat.

Namun, efektivitas KD-Tree sangat dipengaruhi oleh struktur data dan distribusi titik-titik di dalamnya. Jika pembagian pohon tidak seimbang atau data memiliki distribusi acak yang buruk, performa pencarian dapat menurun. Oleh karena itu, dalam implementasi praktis, pemilihan dimensi pembagi dan strategi balancing menjadi aspek penting untuk menjaga efisiensi KD-Tree dalam pencarian KNN.

III. Metodologi Implementasi

A. Representasi Data dalam KD-Tree

Data dalam KD-Tree direpresentasikan sebagai titik-titik dalam ruang berdimensi banyak, di mana setiap titik menggambarkan satu entitas data yang memiliki nilai-nilai numerik pada beberapa fitur. Jumlah dimensi (dd) dari ruang tersebut tergantung pada jumlah fitur yang digunakan dalam proses klasifikasi K-Nearest Neighbors (KNN). Misalnya, jika sebuah dataset memiliki dua fitur seperti panjang dan lebar, maka setiap titik dalam ruang akan direpresentasikan sebagai koordinat dua dimensi (x,y)(x,y). Untuk dataset berdimensi lebih tinggi, seperti pengenalan wajah atau klasifikasi citra dengan banyak atribut, representasi titik akan mencakup lebih banyak komponen koordinat, seperti (x1,x2,..,xd)(x1,x2,..,xd).

Struktur KD-Tree membagi titik-titik ini ke dalam node-node pohon secara hierarkis dengan pendekatan rekursif. Proses pembagian dilakukan berdasarkan satu dimensi tertentu pada setiap tingkat kedalaman pohon. Dimensi yang digunakan sebagai dasar pemisahan (disebut dimensi pembagi atau splitting dimension) akan berganti-ganti secara siklik untuk setiap level dalam pohon, dimulai dari dimensi ke-0, kemudian ke dimensi ke-1, hingga dimensi ke-d−1d−1, lalu kembali ke dimensi ke-0, dan seterusnya. Pendekatan ini memastikan bahwa seluruh dimensi fitur digunakan secara adil dalam proses pembentukan pohon, dan distribusi data tetap seimbang di berbagai arah ruang.

Pemilihan titik median dari dimensi pembagi pada setiap level menjadi strategi utama dalam membentuk pembagian yang optimal. Titik median digunakan sebagai node utama pada level tersebut, sementara titik-titik di bawah median ditempatkan di subtree kiri dan titik-titik di atas median di subtree kanan. Strategi ini secara tidak langsung membagi ruang menjadi dua bagian di sepanjang garis batas yang tegak lurus terhadap dimensi pembagi, menghasilkan partisi spasial yang efisien untuk pencarian.

Dengan cara ini, KD-Tree secara bertahap membangun struktur yang memecah ruang data menjadi sel-sel yang semakin kecil, memungkinkan pengurangan jumlah titik yang harus diperiksa ketika melakukan pencarian tetangga terdekat. Struktur ini sangat efektif dalam menyaring area ruang yang tidak relevan terhadap titik kueri, sehingga pencarian dapat dilakukan secara selektif dan efisien, tanpa harus mengevaluasi semua titik dalam dataset seperti pada metode brute-force.

B. Proses Pembentukan KD-Tree

KD-Tree dibangun secara rekursif menggunakan pendekatan Divide and Conquer. Proses dimulai dengan memilih dimensi pembagi (biasanya berdasarkan kedalaman node % jumlah dimensi), lalu titik-titik diurutkan berdasarkan dimensi tersebut. Median dari titik-titik kemudian dipilih sebagai node akar atau sub-akar, sehingga membagi dataset menjadi dua bagian seimbang: bagian kiri berisi titik-titik yang lebih kecil dari median dan bagian kanan berisi yang lebih besar. Proses ini diulang secara rekursif untuk masing-masing sub-bagian hingga tidak ada titik yang tersisa.

Langkah-langkah pembentukan KD-Tree:

1. Jika jumlah titik kosong, kembalikan null.
2. Tentukan dimensi pembagi: depth % k, dengan kk = jumlah dimensi.
3. Urutkan titik berdasarkan dimensi pembagi.
4. Pilih median sebagai node saat ini.
5. Rekursif bangun subtree kiri dan kanan dengan titik-titik di bawah dan di atas median.

C. Algoritma Pencarian Tetangga Terdekat

Pencarian tetangga terdekat dilakukan dengan menelusuri KD-Tree berdasarkan lokasi titik kueri. Proses pencarian diawali dengan traversal menuju node daun berdasarkan perbandingan koordinat titik kueri dengan node saat ini, sesuai dimensi pembaginya. Setelah mencapai daun, dilakukan backtracking untuk memeriksa kemungkinan adanya node lain yang lebih dekat, dengan membandingkan jarak titik kueri terhadap node yang sudah ditemukan dan batas pemisah dari cabang lain.

Langkah umum pencarian:

1. Telusuri pohon secara rekursif hingga mencapai daun sesuai dimensi pembagi.
2. Simpan titik dengan jarak terdekat sementara.
3. Lakukan backtracking dan periksa apakah cabang lain perlu ditelusuri.
4. Perbarui tetangga terdekat jika ditemukan yang lebih dekat.
5. Ulangi hingga seluruh cabang potensial selesai diperiksa.

Untuk pencarian k-tetangga terdekat, digunakan struktur seperti priority queue atau max-heap untuk menyimpan kandidat terbaik sejauh ini.

D. Analisis Kompleksitas Waktu dan Ruang

1. **Waktu pembentukan KD-Tree**:
   Kompleksitas rata-rata adalah O(*nlogn*)O(*n* *log n*)O(*nlogn*), karena setiap pemanggilan rekursif membagi dataset menjadi dua dan melakukan pengurutan untuk menemukan median.

2. **Waktu pencarian**:
   Rata-rata pencarian tetangga terdekat dalam KD-Tree memiliki kompleksitas O(*logn*)O(*log n*)O(*logn*) pada data berdimensi rendah, namun dalam kasus terburuk (data sangat tidak seimbang atau dimensi tinggi), kompleksitasnya dapat memburuk menjadi O(*n*)O(*n*)O(*n*).

3. **Kompleksitas ruang**:
   Karena setiap node menyimpan satu titik dan dua anak (kiri dan kanan), kompleksitas ruang adalah O(*n*)O(*n*)O(*n*).

Efisiensi KD-Tree sangat bergantung pada distribusi dan dimensi data. Untuk data berdimensi tinggi, alternatif seperti Ball Tree atau metode berbasis Approximate Nearest Neighbor kadang digunakan sebagai pengganti.

IV. HASIL DAN PEMBAHASAN

A. Hasil Implementasi

Implementasi KD-Tree dan algoritma K-Nearest Neighbors (KNN) berhasil direalisasikan secara penuh menggunakan bahasa pemrograman C++ [4], yang dikenal efisien dalam pengelolaan memori dan kecepatan eksekusi untuk proses komputasi intensif. KD-Tree dibangun sebagai struktur data pohon biner yang mampu mengorganisasi titik-titik dalam ruang berdimensi banyak, dengan memanfaatkan pendekatan rekursif berbasis prinsip Divide and Conquer. Dalam proses pembentukannya, setiap titik data dibagi berdasarkan dimensi tertentu yang bergantian pada setiap level pohon, dan nilai median digunakan sebagai node pembagi. Dengan demikian, ruang data terbagi secara seimbang, memungkinkan traversal yang lebih efisien selama proses pencarian.

Setelah struktur KD-Tree terbentuk, proses pencarian tetangga terdekat (KNN) dilakukan dengan menelusuri node-node dalam pohon berdasarkan posisi titik kueri. Algoritma ini memanfaatkan teknik traversal ke bawah menuju node daun dan dilanjutkan dengan proses backtracking untuk mempertimbangkan kemungkinan adanya node lain yang lebih dekat dari hasil sementara. Selama pencarian berlangsung, sebuah struktur data max-heap digunakan untuk mempertahankan K kandidat tetangga terdekat terbaik secara dinamis, dengan memperbarui elemen heap saat ditemukan titik yang lebih dekat. Proses ini secara signifikan mengurangi jumlah titik yang perlu diperiksa dibandingkan metode konvensional.

Untuk mengevaluasi performa dari KD-Tree, dilakukan serangkaian eksperimen dengan dataset acak yang terdiri dari 100.000 titik berdimensi dua (2D), yang merepresentasikan skenario nyata dengan jumlah data besar namun berdimensi rendah—kondisi ideal bagi KD-Tree. Nilai K=100K=100 digunakan sebagai parameter pencarian tetangga terdekat. Sebagai pembanding, diimplementasikan pula metode brute-force secara terpisah, yaitu metode pencarian langsung terhadap seluruh data dengan menghitung jarak Euclidean dari titik kueri ke setiap titik dalam dataset. Metode ini dipilih sebagai tolok ukur karena sifatnya yang sederhana, akurat, dan tanpa optimasi struktural.

Kedua pendekatan ini kemudian dibandingkan dari aspek performa, yaitu jumlah iterasi yang diperlukan untuk pencarian serta waktu eksekusi yang dibutuhkan hingga ditemukan KK tetangga terdekat. Dengan pendekatan ini, efektivitas dan efisiensi KD-Tree dapat diamati secara kuantitatif, sekaligus memberikan pemahaman lebih mendalam mengenai keunggulan struktur data spasial dalam konteks klasifikasi KNN berskala besar.

B. Perbandingan KD-Tree dengan Brute Force

Dalam implementasi algoritma K-Nearest Neighbor (KNN), dua pendekatan umum untuk pencarian tetangga terdekat adalah menggunakan metode brute-force dan menggunakan struktur data seperti KD-Tree. Keduanya memiliki kelebihan dan kekurangan tergantung pada karakteristik dataset dan kebutuhan aplikasi.

Metode brute-force melakukan pencarian dengan cara menghitung jarak antara titik kueri dengan seluruh titik dalam dataset. Hal ini menjamin hasil yang akurat namun memerlukan waktu komputasi yang besar karena kompleksitasnya sebesar O(n·d)O(n·d), dengan nn adalah jumlah data dan dd adalah dimensi. Brute-force juga tidak membutuhkan struktur data tambahan dan lebih mudah diimplementasikan.

Sebaliknya, KD-Tree merupakan struktur data yang dirancang untuk mengurangi kompleksitas pencarian dengan membagi ruang data secara rekursif menggunakan pendekatan Divide and Conquer. Dalam kasus ideal, pencarian dengan KD-Tree memiliki kompleksitas rata-rata O(log⁡n)O(logn), walaupun dalam kasus terburuk bisa mendekati O(n)O(n), khususnya pada data berdimensi tinggi atau distribusi yang tidak merata.

Perbandingan antara kedua metode dalam konteks eksperimen menunjukkan bahwa KD-Tree secara signifikan mengurangi jumlah iterasi pencarian dan waktu eksekusi, terutama pada dataset besar dengan dimensi rendah (seperti 2D). Dengan mempertahankan daftar kandidat tetangga terdekat menggunakan struktur heap, KD-Tree mampu menghasilkan hasil yang identik dengan brute-force, tetapi dengan efisiensi yang jauh lebih tinggi.

Namun demikian, KD-Tree memiliki kelemahan dalam hal implementasi yang lebih kompleks dan sensitivitas terhadap dimensi. Untuk data berdimensi tinggi (biasanya lebih dari 20 dimensi), performa KD-Tree mulai menurun akibat fenomena curse of dimensionality, di mana pembagian ruang menjadi tidak efektif, dan jumlah node yang harus dikunjungi menjadi hampir sama dengan brute-force.

Secara keseluruhan, KD-Tree lebih unggul untuk kasus dataset besar dengan dimensi rendah hingga sedang, sementara brute-force tetap relevan untuk dataset kecil atau ketika kesederhanaan implementasi lebih diutamakan.

C. *Eksperimen dan Output Program*

Untuk mengevaluasi performa dari algoritma K-Nearest Neighbor (KNN) menggunakan KD-Tree dan membandingkannya dengan metode brute-force, dilakukan sebuah eksperimen menggunakan data titik acak dua dimensi sebanyak 100.000 buah dengan nilai K=100. Implementasi dilakukan menggunakan bahasa pemrograman C++ dan dijalankan pada perangkat dengan spesifikasi prosesor Intel Core i5, RAM 8 GB, dan sistem operasi Windows 10. Pengukuran performa difokuskan pada jumlah iterasi pencarian dan waktu eksekusi dalam satuan mikrodetik (µs), dengan hasil pencarian diurutkan dan dibandingkan secara langsung untuk memastikan akurasi hasil yang identik antara kedua metode.

Pada implementasi KD-Tree, proses pencarian hanya mengunjungi sebagian kecil dari keseluruhan node dalam struktur pohon. Hal ini memungkinkan jumlah iterasi pencarian yang jauh lebih sedikit dibandingkan brute-force. Sebagai contoh, dari hasil eksekusi program, KD-Tree hanya memerlukan sekitar 228 iterasi dan waktu pencarian hampir instan untuk menemukan 100 tetangga terdekat dari titik kueri. Sebaliknya, pada metode brute-force, seluruh 100.000 titik tetap diperiksa satu per satu sehingga jumlah iterasi mencapai 100.000 dengan waktu eksekusi sekitar 10087 mikrodetik.

Gambar 4.1 – Output Program KD-Tree

```
Membangun KD-Tree...
Waktu pembangunan KD-Tree: 2144 ms
Melakukan pencarian 100 nearest neighbors...
Iterasi dilakukan: 228
Waktu pencarian: 0 ms
```

Gambar 4.2 – Output Program Brute Force

```
Melakukan pencarian 100 nearest neighbors (brute-force)...
Iterasi dilakukan: 100000
Waktu pencarian: 10087 ms
```

Tabel 4.1 – Perbandingan KD-Tree dan Brute Force

Metode	Jumlah Iterasi	Waktu Eksekusi (ms)
KD-Tree	228	~ 0
Brute Force	100000	10087

Berdasarkan hasil tersebut, dapat disimpulkan bahwa KD-Tree memberikan peningkatan performa yang signifikan, terutama dalam konteks dataset berukuran besar dan berdimensi rendah. Meskipun hasil yang diperoleh oleh kedua metode sama persis (urutan dan nilai jarak tetangga), KD-Tree membutuhkan waktu pencarian yang jauh lebih singkat. Hal ini menjadikan KD-Tree sebagai solusi efisien untuk implementasi KNN dalam skenario real-time atau aplikasi berskala besar. Namun demikian, kompleksitas struktur KD-Tree membuatnya kurang ideal untuk dataset kecil atau data berdimensi tinggi, di mana metode brute-force mungkin justru lebih praktis.

V. KESIMPULAN DAN SARAN

A. *Kesimpulan*

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa penggunaan struktur data KD-Tree pada algoritma K-Nearest Neighbor (KNN) memberikan peningkatan performa yang signifikan dibandingkan metode brute-force. KD-Tree, yang dibangun menggunakan pendekatan Divide and Conquer, mampu meminimalkan jumlah iterasi pencarian dengan membatasi ruang pencarian secara efisien melalui pembagian dimensi.

Pengujian terhadap dataset acak dua dimensi sebanyak 100.000 titik dengan nilai K=100 menunjukkan bahwa KD-Tree hanya memerlukan sebagian kecil dari iterasi yang dibutuhkan metode brute-force, dengan waktu pencarian yang jauh lebih cepat namun tetap menghasilkan hasil yang identik. Hal ini membuktikan bahwa KD-Tree sangat efektif untuk menangani pencarian tetangga terdekat pada dataset berukuran besar dan berdimensi rendah.

Dengan demikian, dapat disimpulkan bahwa penerapan strategi Divide and Conquer dalam pembangunan struktur KD-Tree secara langsung berkontribusi pada efisiensi komputasi algoritma KNN tanpa mengorbankan akurasi hasil.

B. *Saran*

Meskipun KD-Tree memberikan performa yang lebih baik, penggunaannya perlu disesuaikan dengan karakteristik data. Untuk data berdimensi tinggi, performa KD-Tree cenderung menurun karena efek curse of dimensionality. Oleh karena itu, untuk dimensi yang sangat besar, disarankan untuk mempertimbangkan pendekatan lain seperti Ball Tree atau algoritma Approximate Nearest Neighbor (ANN) yang lebih stabil dalam ruang berdimensi tinggi.

Selain itu, pengembangan implementasi ke arah paralelisasi atau integrasi dengan pustaka machine learning modern dapat menjadi langkah lanjutan untuk meningkatkan efisiensi dan skalabilitas sistem. Penelitian lebih lanjut juga dapat dilakukan untuk mengkaji kinerja KD-Tree dalam konteks dataset nyata yang memiliki distribusi non-uniform atau memiliki outlier yang ekstrem.

REFERENCES

[1] Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9), 509–517. https://doi.org/10.1145/361002.361007

[2] Cover, T. M., & Hart, P. E. (1967). *Nearest Neighbor Pattern Classification*. IEEE Transactions on Information Theory, 13(1), 21–27. https://doi.org/10.1109/TIT.1967.1053964

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. (Bab Divide and Conquer & Search Trees)

[4] M. A. Arindra. (2025). implementasi-kdtree. GitHub repository. [Online]. Available: https://github.com/adli-arindra/implementasi-kdtree (accessed: Jun. 21, 2025)

PERNYATAAN

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

Bandung, 21 Juni 2025

Muhammad Adli Arindra
18222089