Modul Panduan - Implementasi Sorted Neighborhood Method menggunakan Pemrograman Java

Yang Tidak Dipublikasikan

Yusuf Lestanto, ST., MSc., MBA

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK DAN ILMU KOMPUTER
UNIVERSITAS BAKRIE
Semester Ganjil 2025/2026

LEMBAR PENGESAHAN

1. Judul PkM	:	Modul Panduan - Implementasi Sorted Neighborhood Method menggunakan Pemrograman Java
2. Ketua Tim Pengabdian		
a. Nama lengkap	:	Yusuf Lestanto, ST., MSc., MBA.
b. NIDN	:	0302057105
c. Pangkat/Golongan	:	Penata Muda Tk. I - III/b
d. Jabatan	:	Dosen Tetap
e. Telp/Alamat Surel	:	087775084255 / yusuf.lestanto@bakrie.ac.id
3. Anggota Tim Pengabdian		
a. Dosen	:	1. .....
	:	2. .....
b. Praktisi	:	1. .....
	:	2. .....
4. Peserta		
a. Mahasiswa	:	1. .....
	:	2. .....
b. Alumni	:	1. .....
	:	2. .....
5. Biaya Kegiatan		
a. Universitas Bakrie	:	Rp. 0,-
b. Sumber lain	:	Rp. 0,-
6. Tahun Pelaksanaan	:	Tahun 2025

Jakarta, 19 Desember 2025

Mengetahui,
Kaprodi

Pelaksana Pengabdian

Iwan Adhicandra, Ph.D. (Sydney), SMIEEE
NIP: 0018025806

Yusuf Lestanto, S.T., MSc., MBA
NIDN: 0302057105

Mengetahui,
Ketua LPkM Universitas Bakrie

Prof. Ardiansyah, S.TP., M.Si., Ph.D.
NIDN: 0318107501

Modul Panduan - Implementasi Sorted Neighborhood Method menggunakan Pemrograman Java

Yusuf Lestanto

1 Pendahuluan

Dalam era digital saat ini, organisasi dan perusahaan mengumpulkan data dari berbagai sumber yang berbeda-beda. Hal ini sering kali mengakibatkan terjadinya duplikasi data, yaitu kondisi di mana satu entitas nyata direpresentasikan oleh lebih dari satu record dalam basis data. Duplikasi data dapat menyebabkan berbagai masalah serius, mulai dari pemborosan ruang penyimpanan, ketidakakuratan analisis, hingga pengambilan keputusan yang keliru.

Proses mendeteksi dan menghilangkan duplikasi data dikenal sebagai *record linkage*, *entity resolution*, atau *deduplication*. Salah satu algoritma yang paling populer dan efisien untuk tujuan ini adalah *Sorted Neighborhood Method* (SNM), yang diperkenalkan oleh Hernández dan Stolfo pada tahun 1995 [1].

SNM bekerja dengan cara mengurutkan data berdasarkan *sorting key* tertentu, kemudian membandingkan record-record yang berdekatan dalam jendela geser (*sliding window*). Pendekatan ini lebih efisien daripada metode perbandingan naive yang membandingkan setiap pasangan record (O(n2)), karena SNM hanya membandingkan record dalam jendela berukuran w, sehingga kompleksitasnya menjadi O(w·n).

Proses ini memungkinkan identifikasi data yang mirip atau duplikat dengan lebih efisien dibandingkan dengan membandingkan setiap pasangan data secara keseluruhan. Keuntungan utama dari metode Sorted Neighborhood adalah kemampuannya untuk mengurangi jumlah perbandingan yang diperlukan, terutama pada dataset yang besar. Namun, efektivitas teknik ini sangat bergantung pada pemilihan kunci pengurutan yang tepat dan penentuan ukuran window yang optimal.

Tujuan dari modul ini adalah memberikan panduan langkah demi langkah untuk mengimplementasikan Sorted Neighborhood menggunakan bahasa pemrograman Java.

2 Konsep Dasar Data Quality & Record Linkage

2.1 Data Quality

Kualitas data mengacu pada kondisi data berdasarkan faktor-faktor seperti akurasi, kelengkapan, konsistensi, ketepatan waktu, dan keandalan. Data berkualitas tinggi adalah data yang memenuhi tujuan penggunaannya dan bebas dari kesalahan. Menurut Wang dan Strong [2], dimensi kualitas data dapat dikategorikan menjadi empat kelompok utama (tabel 1): kualitas intrinsik, yang

berkaitan dengan akurasi dan objektivitas data itu sendiri; kualitas kontekstual, yang membahas relevansi dan kelengkapan dalam kaitannya dengan konteks tugas; kualitas representasional, yang melibatkan konsistensi dan interpretasi format dan makna; dan kualitas aksesibilitas, yang mencakup ketersediaan dan keamanan data. Memastikan kualitas data sangat penting untuk pemrosesan, analisis, dan pengambilan keputusan data yang efektif.

Kualitas data yang tinggi mendukung validitas hasil analisis dan meningkatkan kepercayaan pengguna terhadap informasi yang diperoleh. Sebaliknya, data yang buruk dapat menyebabkan kesalahan interpretasi dan keputusan yang tidak tepat. Oleh karena itu, proses pengelolaan data harus mencakup langkah-langkah untuk memantau dan meningkatkan kualitas data secara berkelanjutan.

Table 1: Kategori, dimensi, dan deskripsi kualitas data

Kategori	**Dimensi**	**Deskripsi**
Intrinsik	Akurasi, Objektivitas	Kualitas data itu sendiri
Kontekstual	Relevansi, Kelengkapan	Kesesuaian dengan konteks tugas
Representasional	Konsistensi, Interpretabilitas	Format dan makna data
Aksesibilitas	Ketersediaan, Keamanan	Kemudahan akses data

2.2 Permasalahan Duplikasi Data

Duplikasi data terjadi ketika satu entitas nyata direpresentasikan oleh dua atau lebih record dalam basis data. Duplikasi data dapat menyebabkan inkonsistensi dan kesalahan dalam pengolahan informasi. Hal ini juga dapat mengakibatkan pemborosan ruang penyimpanan dan menurunkan kinerja basis data. Oleh karena itu, penting untuk menerapkan mekanisme pencegahan dan deteksi duplikasi secara efektif. Penyebab utama duplikasi data antara lain:

- **Kesalahan pengetikan:** "Budi Santoso" vs "Budi Santosa"
- **Singkatan:** "Jl. Merdeka" vs "Jalan Merdeka"
- **Urutan kata berbeda:** "Ahmad Fauzi" vs "Fauzi Ahmad"
- **Data dari sumber berbeda:** Integrasi sistem lama dan baru
- **Nilai hilang (missing values):** Field kosong atau NULL

Contoh: Sebuah organisasi rumah sakit memiliki dua record pasien: *"Siti Rahayu, 15-03-1990, Jl. Mawar No.5"* dan *"Siti Rahaju, 15/03/1990, Jalan Mawar 5"*. Keduanya merujuk pada pasien yang sama, namun berbeda karena kesalahan pengetikan dan format yang tidak konsisten.

2.3 Record Linkage & Entity Resolution

Record Linkage adalah proses mengidentifikasi pasangan record dari satu atau lebih dataset yang merujuk pada entitas yang sama di dunia nyata. Proses ini penting dalam berbagai bidang, seperti manajemen data, integrasi sistem, dan analisis data besar. Record linkage membantu mengatasi masalah duplikasi dan inkonsistensi data yang sering terjadi ketika data berasal dari sumber yang berbeda. Teknik ini menggunakan algoritma dan metode statistik untuk mencocokkan dan menggabungkan data yang relevan secara akurat. Proses ini terdiri dari beberapa tahap:

Tahap 1 — Blocking/Indexing: Mengurangi ruang pencarian dengan mengelompokkan record yang berpotensi duplikat ke dalam blok yang sama. Blok ini memungkinkan algoritma untuk membatasi perbandingan hanya pada record yang berada dalam blok yang sama, sehingga mengurangi jumlah perbandingan yang tidak perlu. Dengan demikian, proses pencarian duplikat menjadi lebih efisien dan cepat.

Tahap 2 — Comparison: Membandingkan pasangan record dalam setiap blok menggunakan fungsi similarity. Setiap pasangan record dalam blok yang dibandingkan akan mendapatkan skor similarity berdasarkan fungsi yang digunakan. Skor ini kemudian dievaluasi untuk menentukan tingkat kemiripan antar record. Hasil evaluasi ini menjadi dasar dalam proses pengelompokan atau identifikasi duplikasi data.

Tahap 3 — Classification: Mengklasifikasikan pasangan record sebagai *match*, *non-match*, atau *possible match*. Klasifikasi ini bertujuan untuk mempermudah proses pencocokan data dengan mengelompokkan pasangan record berdasarkan tingkat kemiripan. Pasangan yang dikategorikan sebagai *match* menunjukkan bahwa kedua record tersebut merepresentasikan entitas yang sama. Sementara itu, *non-match* menandakan bahwa kedua record tersebut berbeda, dan *possible match* menunjukkan adanya ketidakpastian yang memerlukan pemeriksaan lebih lanjut.

Tahap 4 — Clustering/Merging: Menggabungkan record yang teridentifikasi sebagai duplikat menjadi satu record representatif. Proses ini melibatkan identifikasi atribut utama dari setiap record yang dianggap duplikat untuk menentukan nilai yang paling representatif. Selanjutnya, data dari record-record tersebut digabungkan dengan mempertimbangkan konsistensi dan keakuratan informasi. Hasil akhirnya adalah satu record tunggal yang mewakili keseluruhan informasi dari record duplikat tersebut.

SNM termasuk dalam kategori teknik Blocking/Indexing yang efisien karena menggunakan pendekatan pengurutan untuk mengurangi jumlah perbandingan yang diperlukan. Metode ini mengurutkan data terlebih dahulu sehingga hanya perlu membandingkan entitas yang berdekatan dalam urutan tersebut. Dengan demikian, jumlah pasangan yang harus diperiksa berkurang secara

signifikan, meningkatkan efisiensi proses pencocokan.

3 Sorted Neighborhood Method (SNM)

3.1 Sejarah dan Latar Belakang SNM

Sorted Neighborhood Method pertama kali diperkenalkan oleh Mauricio A. Hernández dan Salvatore J. Stolfo dalam paper mereka yang berjudul "The Merge/Purge Problem for Large Databases" yang dipresentasikan pada ACM SIGMOD Conference tahun 1995. Algoritma ini kemudian dikembangkan lebih lanjut dalam paper "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem" (1998).

SNM dirancang untuk mengatasi keterbatasan metode perbandingan naive yang memiliki kompleksitas O(n2), yang tidak praktis untuk dataset besar. Dengan menggunakan teknik pengurutan dan sliding window, SNM berhasil menurunkan kompleksitas menjadi O(w·n) di mana w adalah ukuran window yang jauh lebih kecil dari n.

3.2 Konsep Dasar SNM

Ide utama SNM adalah: *"Record yang merupakan duplikat satu sama lain cenderung memiliki nilai yang mirip, sehingga setelah diurutkan berdasarkan kunci tertentu, record-record tersebut akan berada berdekatan satu sama lain."*

Tiga komponen utama dalam Sorted Neighborhood Method (SNM) adalah *sorting key*, *sliding window*, dan *similarity function*, table 2. *Sorting key* merupakan kunci yang digunakan untuk mengurutkan record sehingga data yang memiliki kemiripan berada berdekatan, contohnya bisa berupa tiga karakter pertama dari nama ditambah tahun lahir. *Sliding window* adalah jendela berukuran w yang digeser sepanjang data yang telah diurutkan, di mana setiap record dalam jendela tersebut dibandingkan satu sama lain untuk mendeteksi kemiripan; misalnya, dengan ukuran jendela $w = 3$, maka tiga record berturut-turut akan dibandingkan. Sedangkan *similarity function* adalah fungsi yang mengukur tingkat kesamaan antara dua record, seperti Jaro-Winkler, Levenshtein, atau Cosine Similarity, yang digunakan untuk menentukan apakah dua record merupakan kandidat duplikat berdasarkan nilai kesamaan yang diperoleh. Ketiga komponen ini bekerja secara sinergis untuk meningkatkan efisiensi dan akurasi dalam proses deteksi duplikasi data menggunakan SNM.

3.3 Tahapan Algoritma SNM

Algoritma SNM terdiri dari tiga tahap utama yang saling berkesinambungan untuk mencapai tujuan analisis yang efektif.

Table 2: Tiga komponen utama Sorted Neighborhood Method

Komponen	Deskripsi	Contoh
Sorting Key	Kunci yang digunakan untuk mengurutkan record	3 karakter pertama nama + tahun lahir
Sliding Window	Jendela berukuran w yang bergeser sepanjang data	w = 3 berarti bandingkan 3 record berturut-turut
Similarity Function	Fungsi untuk mengukur kemiripan dua record	Jaro-Winkler, Levenshtein, Cosine

Tahap 1: Pembuatan Sorting Key

Setiap record dalam sebuah database atau kumpulan data diberi sebuah *sorting key*, yaitu representasi ringkas yang berfungsi sebagai identitas unik untuk setiap record tersebut. Kunci ini dirancang sedemikian rupa agar dapat menggambarkan karakteristik utama dari record, sehingga memungkinkan proses pengurutan data menjadi lebih efisien dan terstruktur. Dengan menggunakan *sorting key*, data yang memiliki kemiripan atau kesamaan atribut akan ditempatkan berdekatan, sehingga memudahkan dalam pencarian, pengelompokan, dan analisis data selanjutnya.

Penggunaan *sorting key* tidak hanya mempercepat proses pengurutan, tetapi juga meningkatkan performa sistem dalam mengelola data dalam jumlah besar. Karena kunci ini merupakan representasi ringkas, maka pengolahan dan perbandingan antar record dapat dilakukan dengan lebih cepat dibandingkan membandingkan seluruh isi record secara langsung. Dengan demikian, *sorting key* menjadi komponen penting dalam desain basis data dan algoritma pengurutan, yang berkontribusi pada optimalisasi akses data dan pengelolaan informasi secara keseluruhan.

Contoh Sorting Key:

Record: {*"nama": "Budi Santoso", "tgl_lahir": "1990-05-15", "kota": "Jakarta"*}

Sorting Key: *"BUD199005"* (3 huruf pertama nama + 6 digit tanggal lahir)

Tahap 2: Pengurutan Data

Seluruh record diatur secara leksikografis berdasarkan *sorting key* yang telah ditetapkan untuk memastikan bahwa urutan data mengikuti pola abjad atau nilai yang telah ditentukan secara sistematis. Dengan pengurutan ini, setiap record ditempatkan secara berurutan sesuai dengan kunci pengurutan yang merepresentasikan atribut atau kombinasi atribut tertentu dari data tersebut. Metode ini memudahkan proses pencarian dan pengelolaan data berkat struktur urutan yang konsisten dan terstandarisasi.

Setelah proses pengurutan selesai, record-record yang memiliki kemiripan atau potensi duplikat akan berada berdekatan dalam urutan tersebut. Penempatan yang berdekatan ini memfasilitasi identifikasi dan penanganan terhadap data duplikat, sehingga memungkinkan proses validasi, pembersihan, atau konsolidasi data dapat dilakukan dengan lebih efisien. Dengan demikian, teknik pengurutan leksikografis tidak hanya mengorganisir data secara sistematis, tetapi juga mendukung

langkah-langkah lanjutan dalam pemeliharaan kualitas data.

Tahap 3: Sliding Window & Perbandingan

Sebuah jendela berukuran **w** digeser dari awal hingga akhir data yang telah diurutkan. Pada setiap posisi jendela, record pertama dalam jendela dibandingkan dengan semua record lainnya dalam jendela tersebut menggunakan fungsi *similarity*.

$$Jumlah\ Perbandingan \quad = \quad (n \ - \ w \ + \ 1) \ \times \ (w \ - \ 1) \quad \approx \quad O(w \cdot n)$$

Jika nilai *similarity* antara dua record melebihi *threshold* yang ditentukan, pasangan tersebut diklasifikasikan sebagai kandidat duplikat.

3.4 Analisis Kompleksitas

Analisis kompleksitas pada Sorted Neighborhood Method (SNM) melibatkan dua tahap utama: pengurutan dan perbandingan menggunakan jendela geser. Tahap pengurutan memiliki kompleksitas waktu sebesar $O(n \log n)$ karena proses mengurutkan seluruh record berdasarkan sorting key secara leksikografis. Tahap perbandingan, di mana setiap record dibandingkan dengan record lain dalam jendela berukuran w, memiliki kompleksitas $O(w \cdot n)$. Oleh karena itu, total kompleksitas waktu SNM adalah $O(n \log n + w \cdot n)$. Jika ukuran jendela w relatif kecil dibandingkan dengan jumlah data n, maka kompleksitas pengurutan $O(n \log n)$ menjadi faktor dominan. Pendekatan ini jauh lebih efisien dibandingkan metode naive yang memiliki kompleksitas $O(n^2)$, sehingga SNM sangat cocok untuk dataset besar dengan kebutuhan performa tinggi.

Table 3: Analisis Kompleksitas Waktu Berbagai Metode

Metode	**Kompleksitas Waktu**	**Keterangan**
Naive (Brute Force)	$O(n^2)$	Membandingkan semua pasangan
SNM (Sorting)	$O(n \log n)$	Tahap pengurutan
SNM (Windowing)	$O(w \cdot n)$	Tahap perbandingan
SNM Total	$\boldsymbol{O(n \log n + w \cdot n)}$	Dominan: $O(n \log n)$ jika w kecil

3.5 Kelebihan dan Kekurangan SNM

Sorted Neighborhood Method (SNM) memiliki beberapa kelebihan dan kekurangan yang perlu dipertimbangkan dalam penerapannya. Kelebihan utama SNM adalah efisiensinya dalam menangani dataset besar karena mengurangi jumlah perbandingan yang diperlukan dibandingkan metode naive, sehingga lebih cepat dan skalabel. Selain itu, SNM mudah diimplementasikan dan dapat dikombinasikan dengan metode lain untuk meningkatkan akurasi. Namun, SNM juga memiliki kekurangan, seperti sensitivitas terhadap pemilihan sorting key yang tepat; jika sorting key tidak

dipilih dengan baik, duplikat yang sebenarnya bisa terlewat karena tidak berdekatan dalam urutan. Selain itu, ukuran jendela (window size) harus ditentukan secara manual dan dapat memengaruhi hasil deteksi, sementara performa metode ini sangat bergantung pada kualitas data yang digunakan.

Table 4: Kelebihan dan Kekurangan Sorted Neighborhood Method

Kelebihan	**Kekurangan**
Efisien untuk dataset besar	Sensitif terhadap pemilihan *sorting key*
Mudah diimplementasikan	Duplikat bisa terlewat jika *key* berbeda jauh
Skalabel (*scalable*)	Ukuran *window* w harus ditentukan manual
Dapat dikombinasikan dengan metode lain	Performa bergantung pada kualitas data

4 String Similarity & Fungsi Perbandingan

Dalam SNM, fungsi *similarity* digunakan untuk mengukur seberapa mirip dua record. Berikut adalah beberapa fungsi yang umum digunakan:

4.1 Jaro-Winkler Distance

Jaro-Winkler adalah metrik kesamaan yang dirancang untuk mengukur seberapa mirip dua string, terutama efektif untuk string pendek seperti nama orang. Metrik ini menggabungkan perhitungan jarak Jaro dengan penyesuaian tambahan yang memberikan bobot lebih besar pada kesamaan yang terjadi di bagian awal string. Hal ini penting karena dalam banyak kasus, terutama pada nama, kesamaan di awal kata dianggap lebih signifikan daripada di bagian lain. Dengan demikian, Jaro-Winkler lebih sensitif terhadap kesamaan awal, yang membantu meningkatkan akurasi dalam pencocokan data seperti pencarian nama, deduplikasi, dan verifikasi identitas.

Selain itu, Jaro-Winkler juga mempertimbangkan jumlah karakter yang cocok dan posisi karakter yang berbeda antara dua string, sehingga menghasilkan nilai kesamaan yang lebih representatif dibandingkan metrik jarak string tradisional. Dengan memberikan bobot tambahan pada prefiks yang sama, metrik ini mampu menangkap kesamaan yang sering kali diabaikan oleh metode lain, menjadikannya pilihan populer dalam aplikasi pemrosesan data yang memerlukan pencocokan nama atau entitas dengan tingkat ketelitian tinggi. Pendekatan ini membuat Jaro-Winkler sangat berguna dalam sistem pengenalan data, penggabungan basis data, dan aplikasi lain yang membutuhkan pencocokan string yang akurat dan efisien.

$$Jaro(s1, s2) \quad = \quad (1/3) \times (m/|s1| + m/|s2| + (m - t)/m)$$

$$Jaro\text{-}Winkler(s1, s2) \quad = \quad Jaro + p \times l \times (1 - Jaro)$$

Di mana: m = jumlah karakter yang cocok, t = jumlah transposisi, l = panjang prefix yang sama (maks 4), p = faktor skala (biasanya 0.1).

Contoh: Jaro-Winkler("MARTHA", "MARHTA") = 0.961 — sangat mirip karena hanya ada transposisi huruf.

4.2 Levenshtein Distance

Jarak Levenshtein mengukur jumlah minimum operasi karakter tunggal yang diperlukan untuk mengubah satu string menjadi string lain. Operasi ini meliputi penyisipan, penghapusan, dan penggantian karakter. Metrik ini banyak digunakan di bidang-bidang seperti pemrosesan bahasa alami, bioinformatika, dan pengejaan, di mana penilaian kesamaan atau perbedaan antara sekuens sangat penting.

Dengan menghitung Jarak Levenshtein, seseorang dapat menentukan seberapa dekat hubungan antara dua string, yang memungkinkan aplikasi seperti pencocokan string perkiraan, koreksi kesalahan, dan analisis sekuens DNA. Semakin rendah jaraknya, semakin mirip string tersebut, sedangkan jarak yang lebih tinggi menunjukkan ketidakmiripan yang lebih besar. Algoritma yang efisien ada untuk menghitung jarak ini, seringkali menggunakan teknik pemrograman dinamis untuk mengoptimalkan proses untuk string yang lebih panjang.

$$Similarity \quad = \quad 1 \; - \; (LevenshteinDistance(s1, s2)/max(|s1|, |s2|))$$

Table 5: Contoh perhitungan Similarity menggunakan Levenshtein Distance

String 1	**String 2**	**Edit Distance**	**Similarity**
kitten	sitting	3	0.571
Budi	Buri	1	0.750
Jakarta	Jakrata	2	0.714

4.3 Cosine Similarity

Cosine Similarity adalah metrik yang mengukur kosinus sudut antara dua vektor dalam ruang multidimensi, memberikan ukuran seberapa mirip kedua vektor tersebut dalam hal arahnya terlepas dari besarnya. Dalam konteks analisis teks, ini melibatkan representasi setiap string sebagai vektor, yang biasanya dibangun dari penghitungan frekuensi karakter, kata, atau unit tekstual lainnya. Dengan mengubah data teks menjadi vektor numerik, kesamaan kosinus memungkinkan perbandingan dokumen atau kalimat dengan menilai seberapa dekat vektor yang sesuai diselaraskan dalam ruang vektor.

Pendekatan ini sangat bermanfaat dalam tugas-tugas pemrosesan bahasa alami seperti pengambilan informasi, pengelompokan dokumen, dan klasifikasi teks, di mana tujuannya adalah un-

tuk mengenali kesamaan semantik antara bagian-bagian teks. Karena kesamaan kosinus lebih menekankan pada orientasi vektor daripada panjangnya, ia secara efektif menormalkan perbedaan panjang teks, sehingga kuat dalam membandingkan dokumen dengan ukuran yang berbeda. Oleh karena itu, ini berfungsi sebagai teknik dasar untuk banyak algoritma yang memerlukan ukuran kuantitatif kesamaan tekstual.

$$Cosine(A, B) \quad = \quad (A \cdot B)/(||A|| \times ||B||)$$

Cosine Similarity cocok digunakan untuk membandingkan teks yang lebih panjang seperti alamat atau deskripsi produk.

4.4 Pemilihan Fungsi Similarity

Pemilihan fungsi similarity dalam Sorted Neighborhood Method sangat bergantung pada tipe data yang akan dibandingkan untuk memastikan hasil pencocokan yang akurat dan efisien. Untuk data berupa nama orang yang umumnya pendek, fungsi Jaro-Winkler Distance direkomendasikan karena kemampuannya memberikan bobot lebih pada kesamaan di awal string, yang sering kali penting dalam pencocokan nama. Untuk data seperti kode, ID, atau nomor, Levenshtein Distance lebih sesuai karena mengukur jumlah operasi edit yang diperlukan untuk menyamakan dua string, sehingga efektif mendeteksi perbedaan kecil. Sedangkan untuk data teks yang lebih panjang seperti alamat atau deskripsi produk, Cosine Similarity lebih tepat digunakan karena mengukur kesamaan berdasarkan orientasi vektor frekuensi karakter atau kata, yang menormalkan perbedaan panjang teks. Untuk atribut seperti tanggal lahir, metode pencocokan yang sederhana seperti exact match atau Levenshtein Distance dapat diterapkan. Pemilihan fungsi similarity yang tepat sangat penting untuk meningkatkan akurasi deteksi duplikat dan mengoptimalkan kinerja algoritma.

Table 6: Rekomendasi Fungsi Similarity Berdasarkan Tipe Data

Tipe Data	**Fungsi yang Direkomendasikan**
Nama orang (pendek)	Jaro-Winkler Distance
Kode, ID, nomor	Levenshtein Distance
Alamat, deskripsi (panjang)	Cosine Similarity / Jaccard
Tanggal lahir	Exact Match / Levenshtein

5 Implementasi SNM di Java

Implementasi Sorted Neighborhood Method (SNM) di Java dimulai dengan pembuatan kelas `Record` yang merepresentasikan setiap baris data beserta atribut dan sorting key-nya. Sorting key ini dihasilkan dari atribut tertentu untuk mengurutkan record secara leksikografis. Selanjutnya, kelas `SortedNeighborhoodMethod` mengelola proses utama SNM, yaitu pengurutan record

berdasarkan sorting key dan penerapan sliding window untuk membandingkan record dalam jendela berukuran w. Pada setiap posisi jendela, fungsi similarity seperti Jaro-Winkler digunakan untuk mengukur kemiripan antar record berdasarkan bobot atribut yang telah ditentukan. Jika nilai similarity melebihi threshold, pasangan record tersebut dianggap kandidat duplikat. Fungsi similarity Jaro-Winkler diimplementasikan secara mandiri untuk menghitung nilai kemiripan antara dua string. Program utama (`Main.java`) menyiapkan dataset contoh, menghasilkan sorting key, menjalankan proses SNM, dan menampilkan hasil deteksi duplikat secara lengkap. Pendekatan ini memberikan kerangka kerja modular dan efisien untuk deteksi duplikasi data menggunakan SNM dalam bahasa pemrograman Java.

Sebelum memulai implementasi, pastikan lingkungan pengembangan telah disiapkan:

- **JDK:** Java Development Kit versi 11 atau lebih baru
- **IDE:** IntelliJ IDEA Community / Eclipse / VS Code dengan Java Extension
- **Build Tool:** Maven atau Gradle (opsional)

Buat project Java baru dengan struktur direktori berikut:

Struktur Proyek Java

```
SNM_Project/
├── src/
│   └── main/
│       └── java/
│           ├── model/
│           │   └── Record.java
│           ├── similarity/
│           │   ├── JaroWinkler.java
│           │   └── LevenshteinDistance.java
│           ├── snm/
│           │   └── SortedNeighborhoodMethod.java
│           └── Main.java
└── data/
    └── sample_data.csv
```

5.1 Struktur Data Record

Struktur Data Record merepresentasikan satu baris data dalam sebuah dataset dengan atribut-atribut yang terkait. Kelas `Record` berisi ID unik untuk setiap record, sebuah *sorting key* yang merupakan representasi ringkas dari atribut tertentu untuk keperluan pengurutan, serta sebuah peta (`Map`) yang menyimpan pasangan kunci dan nilai atribut. Metode dalam kelas ini memungkinkan penetapan dan pengambilan nilai atribut, serta pembuatan *sorting key* berdasarkan beberapa bidang atribut yang dipilih dengan mengambil sebagian karakter dari nilai atribut tersebut. Pendekatan ini memudahkan pengelolaan data dan mendukung proses pengurutan serta pencocokan record secara efisien, seperti yang diimplementasikan dalam algoritma Sorted Neighborhood Method (SNM).

Langkah pertama, kita membuat kelas `Record.java` untuk merepresentasikan satu baris data.:

```java
package model;

import java.util.HashMap;
import java.util.Map;

/**
* Kelas Record merepresentasikan satu baris data dalam dataset.
* Setiap record memiliki ID unik, sorting key, dan map atribut.
*/
public class Record {
    private String id;
    private String sortingKey;
    private Map<String, String> attributes;

    public Record(String id) {
        this.id = id;
        this.attributes = new HashMap<>();
    }

    public void setAttribute(String key, String value) {
        attributes.put(key, value != null ? value.trim() : "");
    }

    public String getAttribute(String key) {
        return attributes.getOrDefault(key, "");
    }

    public void generateSortingKey(String... fields) {
        StringBuilder sb = new StringBuilder();
        for (String field : fields) {
            String val = getAttribute(field).toUpperCase();
            int len = Math.min(val.length(), 3);
            sb.append(val, 0, len);
        }
        this.sortingKey = sb.toString();
    }

    public String getId() { return id; }
    public String getSortingKey() { return sortingKey; }

    public String toString() {
        return "Record{" + "id='" + id + '\'' + ", key='" + sortingKey + '\''
```

```
            + ", " +
43          attributes + "}";
44      }
45  }
```

5.2 Implementasi Sorting Key & Pengurutan

Implementasi Sorting Key dan Pengurutan dalam Sorted Neighborhood Method dilakukan dengan membuat kelas yang mengelola daftar record dan mengurutkannya berdasarkan sorting key yang telah dihasilkan. Sorting key merupakan representasi ringkas dari atribut tertentu yang dipilih untuk pengurutan, seperti kombinasi karakter nama dan tanggal lahir. Proses pengurutan dilakukan secara leksikografis menggunakan metode pengurutan bawaan Java dengan membandingkan nilai sorting key setiap record. Setelah pengurutan, record yang memiliki kemiripan akan berada berdekatan dalam daftar, sehingga memudahkan tahap berikutnya dalam mendeteksi duplikat. Implementasi ini memastikan efisiensi karena hanya perlu mengurutkan daftar record sekali sebelum melakukan perbandingan dalam jendela geser.

Kelas `SortedNeighborhoodMethod.java` — Tahap pengurutan:

```
46  package snm;
47
48  import model.Record;
49  import java.util.*;
50
51  public class SortedNeighborhoodMethod {
52
53      private List<Record> records;
54      private int windowSize;
55      private double threshold;
56
57      public SortedNeighborhoodMethod(List<Record> records,
58      int windowSize,
59      double threshold) {
60      this.records = records;
61          this.windowSize = windowSize;
62          this.threshold = threshold;
63      }
64
65      /**
66      * Tahap 1: Urutkan records berdasarkan sorting key
67      */
68      public void sortRecords() {
69          records.sort(Comparator.comparing(Record::getSortingKey));
```

```
70          System.out.println("[INFO] Records berhasil diurutkan berdasarkan
                sorting key.");
71
72          for (int i = 0; i < records.size(); i++) {
73              System.out.printf("  [%d] Key: %-15s | %s%n",
74              i, records.get(i).getSortingKey(), records.get(i).getId());
75          }
76      }
```

5.3 Implementasi Sliding Window

Dalam Metode Sorted Neighborhood, implementasi Sliding Window dilakukan dengan menggeser jendela berukuran w secara berurutan dari awal hingga akhir daftar data yang telah diurutkan berdasarkan kunci pengurutan. Pada setiap posisi jendela, rekaman pertama (anchor) dibandingkan dengan semua rekaman lain di dalam jendela tersebut menggunakan fungsi similaritas yang telah ditentukan, seperti Jaro-Winkler. Jika nilai similaritas antara pasangan rekaman melebihi ambang batas yang telah ditetapkan, pasangan tersebut dianggap sebagai kandidat duplikat dan dicatat. Pendekatan ini secara signifikan mengurangi jumlah perbandingan yang diperlukan dibandingkan dengan metode brute force, karena hanya membandingkan rekaman yang berdekatan dalam jendela, sehingga meningkatkan efisiensi proses deteksi duplikasi. Implementasi ini juga memungkinkan penggunaan bobot atribut yang berbeda dalam perhitungan similaritas, sehingga hasil pencocokan dapat disesuaikan dengan karakteristik data yang dianalisis.

```
77      /**
78      * Tahap 2: Sliding Window - bandingkan record dalam jendela
79      * @return List pasangan record yang merupakan kandidat duplikat
80      */
81      public List<String[]> applyWindow() {
82          List<String[]> candidates = new ArrayList<>();
83          int n = records.size();
84
85          System.out.println("\n[INFO] Memulai Sliding Window (w=" + windowSize
                + ")...");
86
87          for (int i = 0; i < n - 1; i++) {
88              int end = Math.min(i + windowSize, n);
89              Record anchor = records.get(i);
90
91              for (int j = i + 1; j < end; j++) {
92                  Record candidate = records.get(j);
93                  double sim = computeSimilarity(anchor, candidate);
94
```

```
95              System.out.printf("  Bandingkan [%s] vs [%s] => Similarity: %.4
                    f",
96              anchor.getId(), candidate.getId(), sim);
97
98              if (sim >= threshold) {
99                  candidates.add(new String[]{
100                     anchor.getId(), candidate.getId(),
101                     String.format("%.4f", sim)
102                 });
103                 System.out.print(" *** DUPLIKAT TERDETEKSI ***");
104             }
105             System.out.println();
106         }
107     }
108     return candidates;
109     }
110
111     /**
112     * Hitung similarity gabungan dari semua atribut record
113     */
114     private double computeSimilarity(Record r1, Record r2) {
115     String[] fields = {"nama", "tgl_lahir", "alamat"};
116     double[] weights = {0.5, 0.3, 0.2};
117     double totalSim = 0.0;
118
119     for (int k = 0; k < fields.length; k++) {
120         String v1 = r1.getAttribute(fields[k]);
121         String v2 = r2.getAttribute(fields[k]);
122         totalSim += weights[k] * JaroWinkler.similarity(v1, v2);
123     }
124     return totalSim;
125     }
126 }
```

5.4 Implementasi Fungsi Jaro-Winkler Similarity

Implementasi fungsi Jaro-Winkler Similarity bertujuan untuk menilai tingkat kesamaan antara dua string dengan mempertimbangkan karakter yang cocok, posisi karakter, serta prefiks yang sama di awal string. Proses ini dimulai dengan menormalisasi input melalui konversi ke huruf kecil dan penghapusan spasi berlebih. Selanjutnya, algoritma menghitung jumlah karakter yang cocok dalam jangkauan tertentu dan jumlah transposisi karakter yang terjadi. Nilai Jaro dihitung berdasarkan proporsi karakter yang cocok dan transposisi tersebut. Setelah itu, penyesuaian Jaro-

Winkler dilakukan dengan memberikan bobot tambahan pada panjang prefiks yang sama hingga maksimal empat karakter, menggunakan faktor skala standar 0,1. Pendekatan ini meningkatkan sensitivitas terhadap kesamaan di bagian awal string, yang penting dalam pencocokan nama atau entitas.

```
package similarity;

public class JaroWinkler {

public static double similarity(String s1, String s2) {
   if (s1 == null || s2 == null) return 0.0;
   s1 = s1.toLowerCase().trim();
   s2 = s2.toLowerCase().trim();
   if (s1.equals(s2)) return 1.0;
   if (s1.isEmpty() || s2.isEmpty()) return 0.0;

   int matchDist = Math.max(s1.length(), s2.length()) / 2 - 1;
   boolean[] s1Matched = new boolean[s1.length()];
   boolean[] s2Matched = new boolean[s2.length()];
   int matches = 0, transpositions = 0;

   // Hitung karakter yang cocok
   for (int i = 0; i < s1.length(); i++) {
      int start = Math.max(0, i - matchDist);
      int end = Math.min(i + matchDist + 1, s2.length());
      for (int j = start; j < end; j++) {
         if (!s2Matched[j] && s1.charAt(i) == s2.charAt(j)) {
            s1Matched[i] = s2Matched[j] = true;
            matches++;
            break;
         }
      }
   }
   if (matches == 0) return 0.0;

   // Hitung transposisi
   int k = 0;
   for (int i = 0; i < s1.length(); i++) {
      if (s1Matched[i]) {
         while (!s2Matched[k]) k++;
         if (s1.charAt(i) != s2.charAt(k)) transpositions++;
         k++;
      }
```

```
39      }
40
41      double jaro = ((double) matches / s1.length()
42      + (double) matches / s2.length()
43      + (double) (matches - transpositions / 2) / matches) / 3.0;
44
45      // Hitung prefix length (maks 4)
46      int prefix = 0;
47      for (int i = 0; i < Math.min(4, Math.min(s1.length(), s2.length())); i
            ++) {
48          if (s1.charAt(i) == s2.charAt(i)) prefix++;
49          else break;
50      }
51
52      return jaro + prefix * 0.1 * (1.0 - jaro);
53      }
54  }
```

5.5 Program Lengkap — Main.java

Program dalam `Main.java` menyajikan implementasi lengkap dari SNM untuk mendeteksi duplikasi menggunakan Java. Program ini dimulai dengan membentuk kumpulan data sampel yang terdiri dari beberapa catatan, masing-masing memiliki atribut seperti ID, nama, tanggal lahir, dan alamat. Setiap catatan menghasilkan kunci pengurutan berdasarkan atribut yang dipilih (nama dan tanggal lahir) untuk memfasilitasi pengurutan leksikografis. Program kemudian memulai proses SNM dengan ukuran jendela dan ambang batas kesamaan yang telah ditentukan. Program ini mengurutkan catatan berdasarkan kunci pengurutan dan menggunakan pendekatan jendela geser untuk membandingkan catatan dalam setiap jendela dengan fungsi kesamaan berbobot berdasarkan jarak Jaro-Winkler. Pasangan duplikat kandidat yang melebihi ambang batas dikumpulkan dan ditampilkan dengan skor kesamaannya. Output dengan jelas mencantumkan duplikat yang terdeteksi dan jumlah total yang ditemukan, menyediakan kerangka kerja modular dan praktis untuk penghapusan duplikasi data menggunakan SNM.

```
1   import model.Record;
2   import snm.SortedNeighborhoodMethod;
3   import java.util.*;
4
5   public class Main {
6   public static void main(String[] args) {
7
8       // === DATASET CONTOH ===
9       List<Record> dataset = new ArrayList<>();
```

```
10
11      String[][] data = {
12          {"R001", "Budi Santoso", "1990-05-15", "Jl. Merdeka No.10"},
13          {"R002", "Budi Santosa", "1990-05-15", "Jalan Merdeka 10"},
14          {"R003", "Siti Rahayu", "1985-11-20", "Jl. Mawar No.5"},
15          {"R004", "Siti Rahaju", "1985-11-20", "Jalan Mawar 5"},
16          {"R005", "Ahmad Fauzi", "1995-03-08", "Jl. Sudirman No.3"},
17          {"R005", "Ahmad Fauzy", "1995-03-08", "Jl. Sudirman No.3"},
18          {"R006", "Dewi Lestari", "1992-07-22", "Jl. Gatot Subroto"},
19          {"R007", "Dewi Lestary", "1992-07-22", "Jl. Gatot Subroto"}
20      };
21      for (String[] row : data) {
22          Record r = new Record(row[0]);
23          r.setAttribute("nama", row[1]);
24          r.setAttribute("tgl_lahir", row[2]);
25          r.setAttribute("alamat", row[3]);
26          r.generateSortingKey("nama", "tgl_lahir");
27          dataset.add(r);
28      }
29
30      // === JALANKAN SNM ===
31      int windowSize = 3;
32      double threshold = 0.85;
33
34      SortedNeighborhoodMethod snm =
35      new SortedNeighborhoodMethod(dataset, windowSize, threshold);
36
37      System.out.println("=== SORTED NEIGHBORHOOD METHOD ===");
38      System.out.println("Window Size: " + windowSize + " | Threshold: " +
            threshold);
39      System.out.println("====================================");
40
41      snm.sortRecords();
42      List<String[]> duplicates = snm.applyWindow();
43
44      System.out.println("\n=== HASIL DETEKSI DUPLIKAT ===");
45      if (duplicates.isEmpty()) {
46          System.out.println("Tidak ada duplikat terdeteksi.");
47      } else {
48          System.out.printf("%-8s %-8s %-10s%n", "Record 1", "Record 2", "
                Similarity");
49          System.out.println("-".repeat(30));
```

```
50          for (String[] dup : duplicates) {
51              System.out.printf("%-8s %-8s %-10s%n", dup[0], dup[1], dup[2]);
52          }
53      }
54      System.out.println("\nTotal duplikat ditemukan: " + duplicates.size());
55      }
56  }
```

References

[1] Hernández, M. & Stolfo, S. The merge/purge problem for large databases. *ACM Sigmod Record*. **24**, 127-138 (1995)

[2] Wang, R. & Strong, D. Beyond accuracy: What data quality means to data consumers. *Journal Of Management Information Systems*. **12**, 5-33 (1996)