Informatics

ARMARI FAMILY FOUR

Artikel

Implementasi Sequitur Dalam Kompresi Pola Teks Berulang

Elsya Sabrina Asmita Simorangkir

Teknologi Informasi, Universitas Senior Medan; Medan - 20131; Indonesia; elsyasabrinaas@gmail.com

Abstrak: kemajuan teknologi telah berperan penting dalam mengubah cara manusia bertukar data dan informasi. Dari penggunaan media cetak, kini beralih ke media online, yang menuntut pengguna untuk memiliki akses cepat terhadap informasi. Namun, pergeseran ini juga menimbulkan tantangan baru terkait keterbatasan ruang penyimpanan. Pertumbuhan data teks yang sangat cepat menuntut adanya metode penyimpanan yang efisien. Salah satu pendekatan yang dapat diterapkan untuk mengatasi masalah ini adalah dengan menggunakan teknik kompresi data. Dengan kompresi data, informasi dapat disimpan lebih efisien, memungkinkan pengguna untuk menghemat ruang penyimpanan. Algoritma Sequitur membangun tata bahasa dengan mengganti frase berulang dalam urutan yang diberikan dengan aturan baru pada data sekuensial, khusunya data teks. Penyimpanan data dan kompresi saling berkaitan karena kompresi membantu memaksimalkan kapasitas penyimpanan serta meningkatkan efisiensi. Penelitian ini bertujuan untuk menghasilkan strategi dan mengkaji kinerja Algoritma Sequitur dalam kompresi pola teks berulang.

Kata Kunci: Kompresi, Algoritma Sequitur, Pola Teks Berulang, Teknologi, Sharing Document

1. Pendahuluan

Suatu proses mengurangi ukuran data dengan cara menghilangkan redundansi (pengulangan) atau merepresentasikan data dalam bentuk yang lebih efisien disebut sebagai proses kompresi[1], [2]. kompresi data dilakukan untuk mengubah aliran data masukan menjadi aliran data lain dengan tujuan untuk membuat ukuran menjadi lebih kecil dari ukuran aslinya serta untuk mengurangi pemakaian ruang memori dan mempercepat proses transmisi data[3], [4].

File teks merupakan susunan dari baris data yang berupa karakter huruf, angka, simbol yang merupakan representasi kode ASCII. file teks memiliki beberapa ekstensi, seperti untuk notepad memiliki ekstensi file *.txt, pada Microsoft Word tergantung dari versi MS. Office maka terdapat ekstensi *.doc, *.docx, dan *.rtf. karakter yang tersimpan pada file teks berformat angka 1 dan 0 yang merupakan hasil pembacaan dari kode ASCII. Ukuran file teks dipengaruhi dengan banyaknya karakter yang tersimpan pada file tersebut, semakin besar ukuran sebuah file maka semakin banyak menggunakan ruang memori dan proses transmisi pun menjadi lebih lambat.

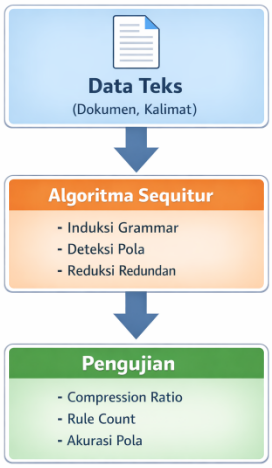
Penerapan algoritma kompresi data seringkali digunakan untuk proses transmisi data (data transmission) dan penyimpanan data (storage). Algoritma kompresi data menawarkan solusi dengan mengurangi ukuran data tanpa mengorbankan informasi esensial sehingga pertukaran data yang lebih efesien[5],[6]. Keuntungan data yang terkompresi antara lain dapat mengurangi bottleneck pada transmisi data, mempersulit pembacaan data oleh pihak yang tidak berkepentingan, dan memudahkan distribusi data[7],[8]. Untuk mengurangi ukuran dari pola teks berulang, ada beberapa algoritma yang digunakan untuk proses kompresi data, seperti algoritma sequitur. algoritma sequitur bekerja untuk mendeteksi pola atau substruktur yang muncul berulang dalam suatu data[9],[10].

Penelitian ini bertujuan untuk menghasilkan strategi dan mengkaji kinerja Algoritma Sequitur dalam kompresi pola teks berulang. Menyelidiki dan membandingkan performa dari kedua algoritma kompresi data dalam hal efisiensi penyimpanan dan kecepatan transfer data. Melalui analisis ini, diharapkan dapat ditemukan algoritma kompresi yang optimal untuk situasi tertentu, memberikan landasan yang kuat untuk pengembangan sistem penyimpanan dan transfer data yang lebih efisien.

Riwayat Artikel:
Diajukan : 29-12-2026
Direvisi : 05-01-2026
Diterima : 10-01-2026
Diterbitkan : 23-01-2026

2. Metodologi

Berikut metologi penelitian yang dilakukan pada penelitian ini Adalah sebagai berikut:



Gambar 1. Metodologi Penelitian

Dalam analisis performa algoritma kompresi data dalam penyimpanan dan transfer data melibatkan pemahaman konsep dasar terkait kompresi data, jenis-jenis algoritma kompresi, dan faktor-faktor yang memengaruhi performa.

2.1 Kompresi Data

Kompresi data adalah proses mengurangi ukuran data dengan cara menghilangkan redundansi atau informasi yang tidak perlu. Dalam teknik kompresi data, redundansi dari data menjadi masalah utama. Redundansi yaitu kejadian berulangnya data atau kumpulan data yang sama dalam sebuah database. Kompresi data ditujukan untuk mereduksi penyimpanan data yang redundan. Terdapat dua jenis kompresi data:

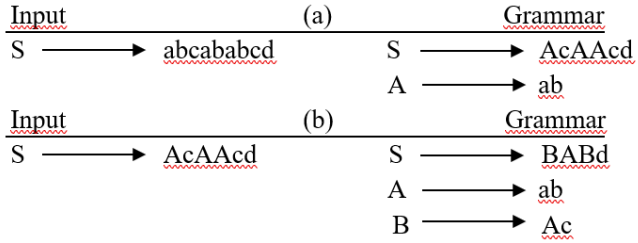
1. Kompresi tanpa kehilangan *(lossless)*: Teknik ini mampu memadatkan data dan mengembalikannya sama persis seperti semula. Tidak ada informasi yang hilang atau harus dikurangi dalam proses untuk mengurangi ukuran besar data
2. Kompresi berkehilangan *(lossy)*: Dengan teknik ini, kehilangan data yang kecil masih dapat diterima. Dengan menghilangkan data yang tidak penting, dapat menghemat ruang penyimpanan.

2.2 Algoritma Sequitur

Sequitur (atau algoritma Nevill-Manning) adalah algoritma rekursif yang dikembangkan oleh Craig Nevill-Manning dan Ian H. Witten pada tahun 1997 yang menyimpulkan struktur hierarkis (tata bahasa bebas konteks) dari urutan simbol diskrit. Algoritma Sequitur membangun tata bahasa dengan mengganti frase berulang dalam urutan yang diberikan dengan aturan baru dan karenanya menghasilkan representasi singkat dari urutan. Dan semua dalam bentuk *lowercase*. Terdapat 2 aturan dalam algoritma Sequitur yaitu:

1. Diagram Keunikan (*Diagram Uniqueness*)
   Diagram Uniqueness mempunyai arti bahwa tidak ada pasangan dari simbol atau diagram muncul lebih dari sekali dalam sebuah tata bahasa. Jika hal ini terjadi maka akan melanggar aturan batasan pertama (diagram uniqueness) sehingga akan membentuk aturan baru (simbol non-terminal) yang akan menggantikan simbol atau diagram yang muncul lebih dari sekali.
2. Aturan Kegunaan (*Rule Utility*)
   Rule Utility mempunyai arti bahwa setiap aturan produksi digunakan lebih dari sekali. Dan jika ada aturan yang hanya digunakan sekali maka akan terjadi pelanggaran pada batasan kedua (rule utility) sehingga aturan yang hanya dipakai sekali akan dihapus atau dihilangkan.

Di bawah ini merupakan algoritma Sequitur:
Masukkan karakter pertama S untuk membuat hasil dari S ⟶ abcababcd



Cara Kerja:
1. Masukkan karakter pertama s untuk membuat hasil dari $S \rightarrow abcababcd$;
2. Mencocokan pada non-terminal yang ada;
3. Membuat non-terminal baru;
4. Memindahkan non-terminal;
5. Memasukkan karakter baru sampai tidak ada karakter yang tersisa.

2.3 Faktor-faktor yang Mempengaruhi Performa

Algoritma kompresi dapat berkinerja berbeda tergantung pada jenis data yang diterapkan. Algoritma kompresi teks umumnya berfokus untuk mengurangi redundansi dalam teks. Setiap jenis data memerlukan pendekatan kompresi yang berbeda sesuai dengan karakteristiknya, dan pemilihan algoritma kompresi yang tepat dapat sangat memengaruhi kinerja kompresi. Kompresi dan dekompresi adalah proses untuk mengurangi ukuran file dan mempercepat pengiriman data. Kecepatan eksekusi algoritma dalam proses kompresi dan dekompresi sangat penting terutama dalam aplikasi yang membutuhkan respons cepat. Pada saat yang sama, kecepatan proses kompresi dan dekompresi sebanding dengan ukuran file, artinya semakin besar file yang diproses, maka akan semakin lama prosesnya. Rasio kompresi mengindikasikan sejauh mana ukuran data dapat dikurangi oleh algoritma, di mana rasio yang tinggi menandakan efektivitas tinggi.

3. Hasil dan Pembahasan

3.1 Kompresi

Misalkan diberikan:

string : "KUMAKAN MAKANAN MAMAKKU"

string lowercase : kumakan makanan mamaku

Tabel 1. Total Bit Sebelum Dikompresi Menggunakan Algoritma Sequitur

Karakter	Frekuensi	ASCII Desimal	ASCII Binary	Bit	Bit×Frekuensi
k	5	40	01101011	8	40
u	2	16	01110101	8	16
m	4	32	01101101	8	32
a	7	56	01100001	8	56
n	3	24	01101110	8	24
sp	2	16	00100000	8	16
		Jumlah Bit × Frekuensi			184 bit

Dari kata "kumakan makanan mamakku" pasangan karakter pertama adalah "ku", kemudian dilakukan pemeriksaan apakah jumlah pasangan "ku" muncul lebih dari satu kali. Dalam string "kumakan makanan mamakku" pasangan "ku" muncul sebanyak dua kali, maka "ku" diubah menjadi "A" di mana "A" merupakan simbol nonterminal.

New string : Amakan makanan mamakA

Kemudian dilakukan lagi pemeriksaan apakah masih terdapat pasangan karakter yang muncul lebih dari satu kali. Pasangan yang ditemukan muncul lebih dari satu kali adalah “ma”, sehingga “ma” diubah menjadi simbol nonterminal “B”.

New String : ABkan Bkanan BbkA

Keseluruhan proses dapat dilihat pada tabel 2 berikut ini:

Tabel 2. Proses Algoritma Sequitur

Proses	String	Diagram	Nonterminal	Rule	New String
1	Kumakan makanan mamakku	Ku	A	A=ku	Amakan makanan mamakA
2	Amakan makanan mamakA	ma	B	B=ma	ABkan Bkanan BBkA
3	ABkan Bkanan BBkA	Bk	C	C=Bk	ACan Canan BCA
4	ACan Canan BCA	Ca	D	D=Ca	ADn Dnan BCA
5	ADn Dnan BCA	Dn	E	E=Dn	AE Ean BCA

Total String Bit yang diperoleh dari proses kompresi dapat dilihat pada tabel 2 berikut:

Tabel 3. Total Bit Setelah Dikompresi Menggunakan Algoritma Sequitur

Karakter	Frekuensi	ASCII Desimal	ASCII Binary	Bit	Bit×Frekuensi
A	2	65	01000001	8	16
E	2	69	01000101	8	16
sp	2	32	00100000	8	16
a	1	97	01100001	8	8
n	1	110	01101110	8	8
B	1	70	01000110	8	8
C	1	107	01101011	8	8
		Jumlah Bit × Frekuensi			80 bit

Dari hasil kompresi tersebut dapat diukur kinerja algoritma *Sequitur* adalah sebagai berikut :

a. Ratio Compression (*RC*)

$$R_C = \frac{\text{ukuran bit data sebelum dikompresi}}{\text{ukuran bit data setelah dikompresi}} = \frac{184}{80} = 2{,}3$$

b. Compression of Ratio (CR)

$$C_R = \frac{\text{ukuran bit data sebelum dikompresi}}{\text{ukuran bit data setelah dikompresi}} = \frac{80}{184} \times 100\% = 43{,}47\%$$

c. Redudancy (*RD*)

$$R_D = 100\% - C_R = 100\% - 43{,}47\% = 56{,}53\%$$

3.2 Dekompresi

Pada Algoritma *Sequitur*, *file* yang baru dibuat berisi informasi tentang *rule* (*diagram* dan nilai *nonterminal*) yang digunakan saat proses kompresi. *Rule* yang diperoleh dari *String* AE Ean BCA adalah A = ku, B = ma, C = Bk, D = Ca, E = Dn, maka String hasil kompresi dibaca dari indeks ke 0 sampai indeks terakhir satu persatu karakter indeks ke 0 adalah A, A terdapat dalam *rule* sehingga berubah menjadi “ku” maka diperoleh string “Amakan makanan kakakA”. Setelah itu, pemeriksaan dilakukan kembali pada *string* apakah masih memiliki simbol *nonterminal* di dalamnya. Jika masih terdapat simbol *nonteriminal*, maka dilakukan pembacaan indeks seterusnya sehingga string hasil dekompresi menjadi “kumakan makanan kakakku”.

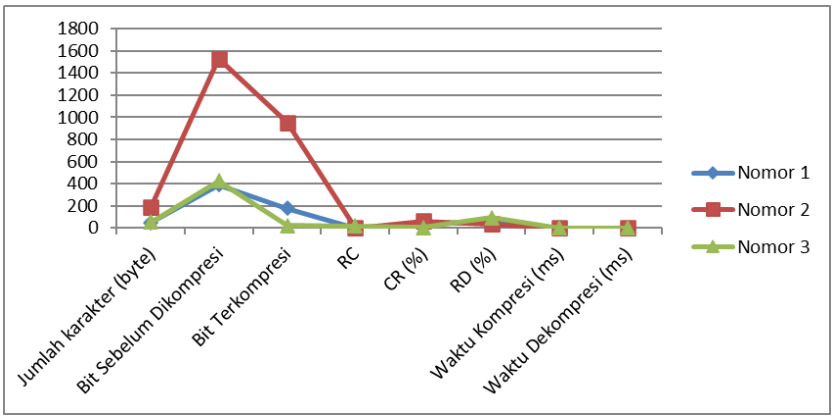
3.3 Pengujian

Berikut merupakan hasil pengujian pada algoritma Sequitur:

Tabel 4. Hasil Pengujian Algoritma Sequitur

No	Jumlah Karakter (byte)	Bit Sebelum Dikompresi	Bit Terkompresi	RC	CR (%)	RD (%)	Waktu Kompresi (ms)	Waktu Dekompresi (ms)
1	49	392	176	2,22	44,898	55,102	0,10	00,12
2	191	1528	952	1,605	62,3037	37,696	0,18	00,14
3	53	424	24	17,66	5,66	94,34	0,12	00,13

Hasil perbandingan dapat dilihat pada gambar 2 berikut:



Gambar 2. Grafik Hasil Pengujian Algoritma Sequitur

Dari hasil uji coba pada aplikasi kompresi teks diperoleh bahwa kedua algoritma tersebut dapat mengkompresi *file* (*.txt) dimana ukuran *file* hasil menjadi berkurang setelah dilakukan kompresi. Dan untuk pengembalian teks, pengujian dekompresi dapat berjalan baik karena isi dan ukuran *file* dapat kembali seperti semula.

4. Kesimpulan

Kesimpulan yang dapat diambil setelah melakukan implementasi sequitur dalam kompresi pola teks berulang adalah sebagai berikut, Aplikasi yang dirancang dalam penelitian telah mampu melakukan proses kompresi file teks dengan algoritma sequitur. Aplikasi yang dirancang dalam penelitian ini telah mampu melakukan proses dekompresi file teks hasil kompresi menjadi file teks semula sebelum file teks dikompresi. Berdasarkan hasil pengujian yang dilakukan disimpulkan bahwa algoritma sequitur lebih unggul jika menggunakan string yang mengandung perulangan frase. Semakin panjang frase yang terdapat dalam sebuah file, maka kinerja algoritma sequitur akan semakin baik. Pengembalian pemampatan data teks dengan algoritma sequitur tidak menghasilkan hasil akhir yang sempurna karena hasil data teks setelah pengembalian, seluruhnya ditampilkan dalam huruf kecil.

Daftar Pustaka

1. D. Salomon, Data Compression: The Complete Reference, 4th ed. London, UK: Springer, 2007.
2. K. Sayood, Introduction to Data Compression, 5th ed. Burlington, MA, USA: Morgan Kaufmann, 2022.
3. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A survey on enabling technologies, protocols, and applications,” IEEE Commun. Surv. \& Tutorials, vol. 23, no. 4, pp. 2347–2380, 2021, doi: 10.1109/COMST.2021.3072823.
4. S. K. Pandey and R. K. Singh, “Data compression techniques for efficient data transmission: A survey,” Int. J. Inf. Technol., vol. 13, no. 3, pp. 1127–1138, 2021, doi: 10.1007/s41870-021-00668-4.
5. R. Grimes, “Character encoding, ASCII, and Unicode in modern text processing,” ACM Comput. Surv., vol. 53, no. 4, pp. 1–36, 2021, doi: 10.1145/3431234.
6. J. Chen, Y. Wang, and X. Liu, “A survey of lossless text compression algorithms,” Inf. Sci. (Ny)., vol. 550, pp. 1–23, 2021, doi: 10.1016/j.ins.2020.11.031.
7. M. S. Obaidat, A. Anpalagan, and I. Woungang, “Efficient data transmission and storage in modern networks,” IEEE Syst. J., vol. 15, no. 2, pp. 2600–2611, 2021, doi: 10.1109/JSYST.2020.3009876.
8. M. Gallé, “Grammar-based compression and its applications,” Theor. Comput. Sci., vol. 807, pp. 3–19, 2020, doi: 10.1016/j.tcs.2019.10.012.
9. P. Deutsch, “Compression in distributed and cloud-based systems,” IEEE Cloud Comput., vol. 7, no. 3, pp. 72–80, 2020, doi: 10.1109/MCC.2020.2986734.

10. A. N. Akram and S. A. Bakar, “Performance evaluation of Sequitur-based grammar compression for repetitive text data,” J. King Saud Univ. -- Comput. Inf. Sci., vol. 34, no. 8, pp. 5124–5134, 2022, doi: 10.1016/j.jksuci.2021.05.006.
11. J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," IEEE Trans. Commun., vol. COM -36, pp. 389-400, Apr. 1988.
12. A. J. Viterbi, "Convolutional codes and their performance in communication systems," IEEE Trans. Commun. Technol., vol. COM -19, pp. 751-772, Oct. 1971
13. Sari., K and M. Riasetiawan, The Development of IoT Compression Technique To The Cloud., 2019, Indonesian Journal of Computing and Cybernetics Systems (IJCCS), vol 13, no.4. doi: 10.22146/ijccs.47270.
14. Sari., K & M. Riasetiawan, The Implementation of Timestamp, Bitmap and RAKE Algorithm on Data Compression and Data Transmission from IoT to Cloud, 2018, IEEE 4th International Conference on Science and Technology (ICST), Yogyakarta, 2018, pp. 1-6. doi: 10.1109/ICSTC.2018.8528698.
15. Vecchio, M., Giaffreda, R. & Marcelloni, F., 2014, Adaptive lossless entropy compressors for tiny iot devices, IEEE Transactions on Wireless Communications, 13, 2, 1088–1100.
16. Zalukhu, Y., Sunandar, H., & Hondro, R. K. (2018). Implementasi Metode Marr-Hilderth Operator Untuk Mendeteksi Tepi Citra Ikonos. *KOMIK (Konferensi Nas. Teknol. Inf. dan Komputer)*, *2*(1), 343-3