Pengujian kinerja kompresi pohon Huffman satu pohon dan dua pohon pada objek text

Ridwan Wulida Siam[1] dan M. Didik R. Wahyudi*[2]

1 **Teknik Informatika UIN Sunan Kalijaga**
Jl. Marsda Adisucipto Yogyakarta
14650024@student.uin-suka.ac.id
2 **Teknik Informatika UIN Sunan Kalijaga**
Jl. Marsda Adisucipto Yogyakarta
m.didik@uin-suka.ac.id

Abstrak

Perkembangan teknologi semakin hari semakin cepat dan banyak, mengakibatkan banyaknya data digital yang tersimpan dalam sebuah media penyimpanan. Jika data tersebut secara terus menerus disimpan dalam waktu yang tidak ditentukan maka akan membutuhkan media penyimpanan yang cukup besar untuk menampung data tersebut. Salah satu cara bentuk untuk mengurangi besarnya data yang tersimpan adalah pemampatan data menjadi lebih kecil ukurannya. Huffman merupakan salah satu algoritma kompresi data teks terbaik yang bisa digunakan. Namun demikian terdapat beberapa kelemahan didalam algoritma Huffman ini, diantaranya adalah dalam proses pembentukan pohon. Penelitian ini menawarkan konsep algoritma Huffman dengan menggunakan dua buah pohon untuk memangkas setengah dari pembentukan pohon Huffman. Dengan menggunakan dua buah pohon Huffman dalam pembentukan pohon terutama kode prefik akan menjadi lebih pendek. Namun demikian, dalam implementasi kompresi data dengan algoritma Huffman ini akan memakan memori dan waktu yang lebih besar karena adanya proses tambahan sebelum data tersebut disimpan ataupun ditampilkan.

Kata Kunci algoritma huffman, rasio kompresi, kompresi, dekompresi, pohon huffman

1 Pendahuluan

Perkembangan teknologi informasi dan komunikasi sangat cepat dan pesat. Teknologi informasi dan komunikasi sudah menjadi bagian penting dan mempengaruhi kehidupan di beberapa bidang. Banyak kemudahan yang diberikan, salah satunya adalah dalam penulisan dan penyimpanan data dokumen dalam bentuk digital. Oleh karena itu, keamanan data digital, efektifitas dan efisiensi penyimpanan data merupakan hal yang perlu untuk diperhatikan. Kebutuhan akan keamanan *database* timbul dari kebutuhan untuk melindungi data. Pertama, dari kehilangan dan kerusakan data. Kedua, adanya pihak yang tidak diijinkan hendak mengakses atau mengubah data. Permasalahan lainnya mencakup perlindungan data dari *delay* yang berlebihan dalam mengakses atau menggunakan data, atau mengatasi gangguan *denial of service*.

Kontrol akses terhadap informasi yang sensitif merupakan perhatian terutama oleh manajer, pekerja di bidang informasi, *application developer*, dan *database administrator*. Kontrol akses selektif berdasarkan otorisasi keamanan dari level user dapat menjamin kerahasiaan tanpa batasan yang terlalu luas. Level dari kontrol akses ini menjamin rahasia informasi sensitif yang tidak akan tersedia untuk orang yang tidak diberi ijin (otorisasi)

* Corresponding author.

Jnanaloka
Jurnal Open Access
Yayasan Lentera Dua Indonesia

bahkan terhadap *user* umum yang memiliki akses terhadap informasi yang dibutuhkan, kadang-kadang pada tabel yang sama.

Mengijinkan informasi dapat dilihat atau digunakan oleh orang yang tidak tepat dapat menyulitkan, merusak, atau membahayakan individu, karir, organisasi, agensi, pemerintah, atau negara. Namun untuk data tertentu seringkali bercampur dengan data lainnya, yaitu pada informasi yang kurang sensitif yang secara legal dibutuhkan oleh berbagai *user*. Membatasi akses terhadap semua table atau memisahkan data sensitif ke *database* terpisah dapat menciptakan lingkungan kerja yang tidak nyaman yang membutuhkan biaya besar pada *hardware*, *software*, waktu *user*, dan administrasi [1].

Kompresi data adalah proses mengodekan informasi menggunakan bit atau informasi bearing unit lain yang lebih rendah dari representasi data yang tidak terkodekan dengan suatu *encoding* tertentu. Setelah data berhasil dikompresi, maka untuk membaca data diperlukan proses dekompresi data. Dekompresi data adalah teknik atau proses mengembalikan data yang telah terkompresi kembali seperti semula atau mengodekan kembali bit menjadi informasi seperti semula [2].

Kompresi data adalah sebuah cara untuk memadatkan data sehingga hanya memerlukan ruang simpan yang lebih kecil dengan demikian lebih efisien dalam penyimpanan atau dapat mempersingkat waktu pertukaran data tersebut [3]. Salah satu algoritma untuk kompresi data yang cukup dikenal adalah Huffman. Algoritma ini dibuat oleh seorang mahasiswa *Massachusetts Institute of Technology* bernama David Huffman pada tahun 1952, merupakan salah satu metode paling lama dan paling terkenal dalam kompresi teks [2]. Algoritma Huffman memakai prinsip pengkodean yang mirip dengan kode morse, yaitu setiap karakter dikodekan hanya dengan beberapa bit, dimana melihat frekuensi karakter yang muncul, jika karakter tersebut sering muncul maka akan dikodekan dengan bit yang pendek dan sebaliknya pada karakter yang jarang muncul akan dikodekan pada bit yang lebih panjang. Jadi prinsip awal algoritma Huffman adalah meminimalkan kode yang *redudant* atau muncul berkali-kali dengan mengubah ke dalam kode tertentu sehingga data akan menjadi lebih kecil dan tetap bisa dibaca oleh penerima pesan [4].

Algoritma Huffman mengurangi redudansi pada data yang ada dengan membentuk kode yang dapat diterima oleh penerima pesan dengan cara melakukan kompresi data. Diharapkan data dokumen akan memiliki ukuran yang lebih kecil sehingga dapat menghemat ruang media penyimpanan. Kode Huffman pada dasarnya merupakan kode prefiks (*prefix code*). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner, dimana pada kode prefiks ini tidak ada kode biner yang menjadi awal bagi kode biner yang lain. Kode prefiks biasanya direpresentasikan sebagai pohon biner yang diberikan nilai atau label. Untuk cabang kiri pada pohon *biner* diberi label 0 sedangkan pada cabang kanan pada pohon *biner* diberi label 1 [5].

Namun algoritma kompresi data Huffman dengan membentuk langsung sebuah pohon Huffman akan menimbulkan lebar dan dalam level pohon tersebut. Sebuah solusi diperlukan untuk mengurangi kedalaman dan lebar pohon Huffman tersebut untuk penentuan kode prefik di setiap karakternya. Dengan membentuk dua buah pohon Huffman akan memangkas panjang kode prefik yang dibentuk pada tiap karakter. Sehingga karakter yang jarang muncul dalam pohon Huffman akan memiliki kode prefik yang besar dan mungkin saja lebih dari 8-bit (penyimpanan ASCII/UTF-8) sehingga ketika karakter tersebut muncul dalam sebuah kalimat ataupun kata maka ukuran file tersebut akan lebih besar. Terlebih jika menggunakan metode statik, jika pohon Huffman tersebut tidak dapat mewakili data yang akan dikompresi maka ukuran kompresi bisa jadi lebih besar dan rasio kompresi akan sangat kecil hingga minus.

Hipotesis awal hasil dari pembentukan dua buah pohon Huffman ini adalah hasil kode prefix akan menjadi lebih pendek pada *string* yang jarang muncul. Hal ini disebabkan karena pembentukan pohon Huffman akan lebih sederhana mengingat data karakter yang akan dibentuk lebih sedikit untuk tiap kelompoknya. Lebih cepatnya proses *encoding* dan *decoding* dikarenakan proses pencarian pada pohon Huffman lebih sedikit mengingat lebih pendeknya tingkat pada pohon Huffman yang terbentuk jika dibandingkan dengan algoritma kompresi data Huffman dengan satu pohon saja.

2 Metodologi

Penelitian ini dilakukan dalam 2 proses utama yakni, pertama, merancang dan membuat algoritma Huffman dua pohon dan kedua, pengambilan data dan pengujian performa. Modifikasi algoritma Huffman untuk pembentukan dua buah pohon dilakulan pada penelitian ini. Terdapat dua kelompok yang akan diberi label "0" dan "1" sebagai penanda kelompok tersebut. Pada tahap pembentukan pohon Huffman tetap sama menggunakan aturan Huffman namun pohon yang dihasilkan ada dua pohon yang disimpan dalam bentuk file. Dan pada tahap kompresi dan dekompresi menggunakan pohon yang telah terbentuk tadi dan kemudian dikonversikan dalam bentuk karakter untuk tahap kompresi dan bentuk biner untuk tahap dekompresi agar mempermudah proses selanjutnya.

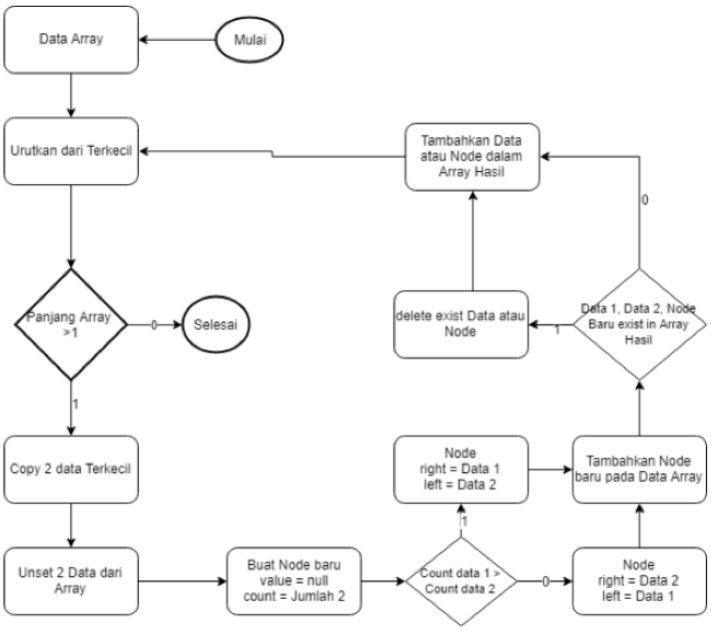
Data yang digunakan didapatkan dari salah satu portal berita daring yaitu detik.com dengan metode pengambilan data menggunakan teknik DOM (*Document Object Model*) [6]. Dalam pengujian performa terdapat tiga indikator yang akan dihitung yaitu rasio hasil kompresi tersebut, waktu eksekusi, dan memori yang digunakan pada proses eksekusi tersebut. Kemudian pada perhitungan waktu dan memori ditetapkan ke dalam dua kondisi yaitu pada saat proses penyimpanan dan pada saat proses menampilkan data. Data ini kemudian dilakukan *preprocessing* terlebih dahulu agar data yang diproses lebih akurat.

2.1 Implementasi algoritma Huffman

Pembentukan dua buah pohon Huffman diawali dengan membuat dua kelompok besar *string* berdasarkan frekuensi kemunculannya. Kemudian dari dua kelompok tersebut dibuat masing-masing sebuah pohon Huffman dengan menerapkan *binary tree*. Sehingga terdapat dua buah pohon Huffman. Kemudian diberikan label pada kedua kelompok tersebut dengan nol untuk kelompok pertama dan satu untuk kelompok kedua. Secara lebih detail, pembentukan pohon Huffman disajikan dalam Gambar 1.

Langkah awal dalam pembentukan pohon Huffman adalah dengan mengurutkan data *array* berdasarkan banyaknya *string* yang muncul dalam data *training*. Kemudian diperiksa apakah banyaknya data dalam *array* tersebut lebih dari satu. Jika kurang dari satu maka hanya tersisa satu data yang mana data tersebut merupakan *root* atau puncak dari pohon Huffman yang akan dibentuk. Jika lebih dari satu maka salin dua data terkecil kemudian hapus dari *array*. Kemudian buat sebuah *node* baru yang nilainya *null* dan *count* dari *node* tersebut yang merupakan penjumlahan *count* dari dua data yang diambil dari *array* .

Dengan mengikuti aturan pada pembuatan pohon, perbandingan dua data yang diambil dari *array* digunakan untuk mengisi bagian *left* dan *right* pada *node* tersebut. Bagian *left* untuk *count* data yang lebih *kecil* dari *count* data lain. Setelah bagian *left* dan *right* pada *node* baru terisi, inputkan *node* tersebut ke dalam *array* awal dan *array* hasil. Ulangi hingga hanya tersisa satu data pada *array* awal. Pembentukan pohon tersebut dilakukan pada kedua kelompok data yang telah dibuat pada proses sebelumnya. Ketika proses telah selesai dan hanya tersisa satu untuk data pada *array* awal, maka pohon Huffman telah terbentuk.



Gambar 1 Flowchart detil pembentukan pohon Huffman.

Tahapan selanjutnya adalah mengisi *prefixCode* dengan aturan bahwa anak sebelah kiri akan diberikan code "0" dan anak sebelah kanan akan diberikan kode "1", kemudian kode pada level berikutnya akan dibuat dari kode induk ditambahkan kode pada jalur mana anak tersebut berada apakah sebelah kiri atau sebelah kanan dari induk.

2.2 Kompresi dan dekompresi

Proses kompresi dimulai dengan mengambil setiap karakter data yang akan dimampatkan dan mencari kode *prefix* dalam pohon Huffman untuk menggantikan karakter. Kode *prefix* tersebut disimpan dalam variabel dan proses tersebut diulangi untuk setiap karakter data yang akan dimampatkan. Tahapan awal tersebut akan membentuk suatu *binary string* yang perlu dikonversikan kembali dalam bentuk *string ASCII* dengan mengambil delapan *bit* kode dari data *binary string*. Proses tersebut dilakukan hingga semua data *binary string* terkonversi secara keseluruhan. Permasalahan akan timbul jika ternyata data *binary string* tersebut tidak habis dibagi oleh delapan, maka akan tersisa beberapa bit kode diakhir proses. Solusi yang diambil dalam penelitian ini adalah dengan tidak mengkonversikan ke dalam bentuk ASCII namun langsung menambahkan saja pada hasil konversi pada bagian akhir, sehingga tidak ada data yang hilang dalam proses pemampatan ini.

Setelah proses tersebut selesai, maka akan terbentuk hasil pemampatan data berupa data *string* karakter kembali, yang siap untuk disimpan ke dalam *database*. Penyimpanan ke dalam *database* menggunakan tipe data *binary large object* untuk menjaga karakter agar tidak berubah saat disimpan. Proses *decoding* dilakukan untuk setiap bit yang ada dan dimulai dari bit pertama pada kode *biner* yang merupakan kode indek pada pohon Huffman.

Kode ini yang akan menentukan pohon Huffman yang akan digunakan, pohon Huffman dengan indek "0" atau indek "1". Jadi jika bit pertama menunjukan "0" maka akan merujuk pada pohon Huffman pada index "0". Bit kedua pada kode biner diambil kembali kemudian dicari kode biner tersebut dalam pohon Huffman. Jika hasilnya *null* atau tidak terdapat kode

biner tersebut dalam pohon Huffman, maka ambil satu bit biner selanjutnya kemudian ulangi lagi pencarian dalam pohon. Begitu seterusnya hingga ditemukan kode bit tersebut dalam pohon Huffman. Huruf pada *node* tersebut kemudian diambil dan simpan pada variabel.

3 Hasil dan pembahasan

Hasil pengujian kompresi dengan algortima Huffman 1 dan 2 pohon adalah tertampil pada Tabel 1. Pengujian pertama dilakukan ketika program menyimpan data ke dalam *database* dengan dan tanpa menggunakan kompresi data Huffman. Skenario pengujian mirip dengan pengujian rasio kompresi yaitu dengan menggunakan beberapa data untuk disimpan kemudian dihitung waktu eksekusi dan memori yang digunakan. Hasil dari pengujian pertama ini tertampil pada Tabel 2

Tabel 1 Hasil pengujian rasio kompresi.

percobaan ke	jumlah data	ukuran asli (KB)	kompresi 1 Pohon		kompresi 2 pohon	
			ukuran (KB)	rasio (%)	ukuran (KB)	rasio (%)
1	1	2,3	0,674	70,69	1,2	47,83
2	1	1,3	0,515	60,38	0,672	48,31
3	1	0,997	1,1	-10,33	0,511	48,75
4	1	2,2	0,949	56,86	1,1	50,00
5	1	1,8	0,740	58,88	0,951	47,17
6	100	224	128	42,81	128	42,81
7	1000	2.576	1.552	39,75	1.552	39,75
8	5000	10.768	5.648	47,54	5.648	47,54
9	10000	22.032	11.792	45,87	10.768	51,12
10	13.749	33.328	16.912	49,25	15.888	52,32

Tabel 2 Hasil pengujian performa penyimpanan data.

percobaan ke	waktu (detik)			memori (bytes)		
	tanpa Huffman	1 pohon	2 pohon	tanpa Huffman	1 pohon	2 pohon
1 (10 data)	0,7635	0,8197	0,8915	1.820.568	1.902.016	1.859.736
2 (50 data)	2,9560	3,2902	3,7203	1.914.136	1.949.632	1.907.224
3 (100 data)	4,7072	5,2292	6,1640	2.035.224	2.023.360	1.973.400
4 (500 data)	35,7843	35,0032	40,7771	2.949.496	2.542.368	2.491.640
5 (1500 data)	97,2917	118,3313	123,9583	5.398.520	3.839.016	3.829.504
Rata-rata	28,3000	32,5347	35,1022	2.823.588	2.451.278	2.412.300

Hasil pengujian penyimpanan data yang disajikan dalam Tabel 2 menunjukkan bahwa rata-rata waktu eksekusi proses menyimpan data dengan menggunakan algoritma Huffman 2 pohon jauh lebih lama 6,8022 detik dibandingkan tidak menggunakan algoritma Huffman namun demikian rata-rata penggunaan memori lebih kecil 411.288 byte. Pada algoritma Huffman 1 pohon mencatatkan waktu lebih cepat dibandingkan dengan menggunakan 2 pohon dengan selisih 2,5675 detik lebih lama tetapi 38.978 byte lebih hemat memori. Tabel 3 merupakan hasil dari pengujian terhadap waktu dan memori yang digunakan untuk proses membaca data.

Dari hasil pengujian performa pembacaan file yang disajikan pada Tabel 3 menunjukkan bahwa waktu yang dibutuhkan oleh algoritma Huffman lebih lama jika dibandingan dengan

Tabel 3 Hasil pengujian performa pembacaan data.

percobaan ke	waktu (detik)			memori (bytes)		
	tanpa Huffman	1 pohon	2 pohon	tanpa Huffman	1 pohon	2 pohon
1 (10 data)	0,1160	0,2551	0,1680	1.881.816	1.909.248	1.921.304
2 (50 data)	0,1653	1,0291	1,0673	2.072.520	2.000.624	2.013.360
3 (100 data)	0,1734	1,9027	2,0135	2.305.640	2.125.072	2.138.064
4 (500 data)	0,2506	8,7417	9,5053	4.027.816	3.048.464	3.060.944
5 (1500 data)	0,9132	27,7126	29,1658	8.613.032	5.448.528	5.463.440
Rata-rata	0,3237	7,9282	8,3840	3.780.165	2.906.387	2.919.422

tanpa pemakaian algoritma. Sedangkan untuk penggunaan memori, pemakaian algortima Huffman lebih kecil jika dibandingkan dengan tanpa pemakaian algortima Huffman.

4 Kesimpulan dan saran

Pembentukan 2 pohon Huffman pada proses awal memiliki pengaruh terhadap hasil dari kompresi data. Data latih yang merepresentasikan data yang akan dikompresi menunjukkan hasil kompresi memiliki ukuran file lebih kecil. Rasio kompresi algoritma Huffman dengan 2 pohon Huffman memiki nilai di atas 39,75% jika dibandingkan dengan algoritma Huffman 1 pohon. Secara keseluruhan algoritma Huffman dengan menggunakan 2 pohon menghasilkan rasio kompresi lebih besar jika dibandingkan dengan algoritma Huffman dengan 1 pohon.

Waktu pemrosesan algoritma Huffman 2 pohon dan 1 pohon tidak memiliki perbedaan yang signifikan, namun memori yang dipergunakan lebih efisien jika dibandingkan dengan tidak menggunakan algoritma Huffman. Dalam proses membaca data dan proses data, waktu proses yang dibutuhkan lebih lama tetapi membutuhkan memori lebih sedikit.

Dengan menggunakan algoritma Huffman ini keamanan data pada lebih terjaga karena hasil kompresi tidak bisa secara langsung dibaca oleh manusia atau komputer tanpa melalui proses dekompresi terlebih dahulu.

Pustaka

1 M. D. R. Wahyudi, "Enkripsi field tabel database dengan pgp," *Jurnal Teknologi IST AKPRIND*, vol. 2, 2009.

2 I. Hanif, "Kompresi teks menggunakan algoritma dan pohon huffman." *Program Studi Teknik Informatika Institut Teknologi Bandung*, vol. 2006-2007, 2007.

3 S. Senthil and L. Robert, "Text compression algorithms: A comparative study," *Journal On Communication Technology*, vol. 2, no. 4, pp. 444–451, 2011.

4 D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

5 Y. Adrisatria, "Penerapan algoritma huffman dalam dunia kriptografi," *Bandung: Program Studi Teknik Informatika, Institut Teknologi Bandung*, 2015.

6 M. D. R. Wahyudi, "Pengambilan isi berita online dengan document object model berbasis php untuk sumber data mining," *Informasi Interaktif*, vol. 1, no. 1, pp. 1–7, 2016.