Lossless Data Compression Rate

Sekarang dalam kehidupan sehari-hari kita saling melakukan interaksi berupa pertukaran informasi dengan memanfaatkan teknologi. Kendala yang sering dihadapi dalam pengiriman maupun penyimpanan data yaitu terlalu besarnya kapasitas data sehingga dalam proses pengiriman data menjadi lambat dan membutuhkan storage yang besar untuk penyimpanannya. Oleh karena itu berkembanglah aplikasi-aplikasi kompresi data untuk mengkompress data agar ukurannya menjadi lebih kecil.

Kompresi data adalah sebuah cara untuk memadatkan data sehingga hanya memerlukan ruangan penyimpangan yang lebih kecil sehingga lebih efisien dalam menyimpannya atau mempersingkat waktu pertukaran data tersebut. Dalam pemampatan data atau kompresi data memiliki 2 jenis hasil kompresi yaitu

1. Kompresi data tanpa kehilangan (lossless data compression)
2. Kompresi data berkehilangan (lossy data compression)

Pada artikel kali ini kita akan membahas lebih lanjut tentang Kompresi data tanpa kehilangan atau Lossless data compression. Lossless data compression adalah teknik kompresi dimana data hasil kompresi dapat didekompres lagi dan menghasilkan hasil yang tepat sama dengan data sebelum dikompres. Contohnya yaitu ZIP, RAR, GZIP, 7-Zip. Teknik ini digunakan jika dibutuhkan data yang setelah di kompresi harus dapat diekstrak/ didekompres lagi tepat sama. Kadang kala ada data-data yang setelah dikompresi dengan Teknik ini ukurannya menjadi lebih besar atau sama.

Run-length Encoding (RLE)
Run-length Encoding (RLE) adalah bentuk dari *lossless data compression* yang sangat sederhana. Algoritma ini menyimpan elemen data yang berurutan dan memiliki nilai yang sama, sebagai sebuah data tunggal yang terdiri dari nilai dan banyaknya. Contohnya, jika string masukkannya adalah “wwwwwaaadexxxxxx” maka algoritma *run-length encoding* akan mengembalikan “w4a3d1e1x6”. Hasil ini dapat diinterpretasikan sebagai empat buah w, tiga buah a, sebuah d, sebuah x dan enam buah x. Berikut adalah contoh perhitungan dari RLE *compression rate* dari sebuah citra.

Kasus Pertama

0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

1	1	1	1	0	0	0
1	1	1	1	0	0	0
0	0	0	0	0	0	0

Dalam bentuk aslinya, citra ini akan membutuhkan 49 bits. Jika kemudian kita mengeksekusi algoritma RLE dengan total 4 bit untuk tiap run-nya (1 bit untuk menyimpan nilainya dan 3 bit untuk menyimpan banyaknya). Maka akan memberi hasil sebagai berikut.
Baris 1 : 07
Baris 2 : 17
Baris 3 : 17
Baris 4 : 17
Baris 5 : 1403
Baris 6 : 1403
Baris 7 : 07
Sehingga setelah dikompresi kita hanya akan membutuhkan 36 bit. Dengan kata lain, pada kasus ini algoritma RLE memiliki *compression rate* sebesar 49 : 36 atau sekitar 1.36 : 1.

Kasus Kedua

0	0	1	1	1	0	0
1	1	1	0	0	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
1	1	1	1	0	0	0
1	1	0	0	0	0	0
0	0	0	0	1	1	1

Untuk kasus kedua, algoritma RLE akan memberikan hasil sebagai berikut.
Baris 1 : 021302
Baris 2 : 130411
Baris 3 : 0314
Baris 4 : 0215
Baris 5 : 1403
Baris 6 : 1205
Baris 7 : 0413

Sehingga setelah dikompresi kita hanya membutuhkan 64 bit. Dengan kata lain, pada kasus ini algoritma RLE justru akan membuat ukuran data yang kita butuhkan untuk menyimpan data yang telah di-*compress* menjadi lebih besar daripada untuk menyimpan data aslinya.
Dari dua contoh kasus di atas, sangat jelas terlihat bahwa algoritma ini tidak cukup baik untuk mengkompresi data secara umum. Algoritma ini akan cocok untuk mengkompresi data yang banyak memiliki nilai yang sama pada elemen data yang berurutan. Namun akan menjadi tidak efektif untuk kasus data yang sebaliknya karena justru akan memberi ukuran data yang sama atau bahkan lebih besar dari ukuran data aslinya.

Half Byte

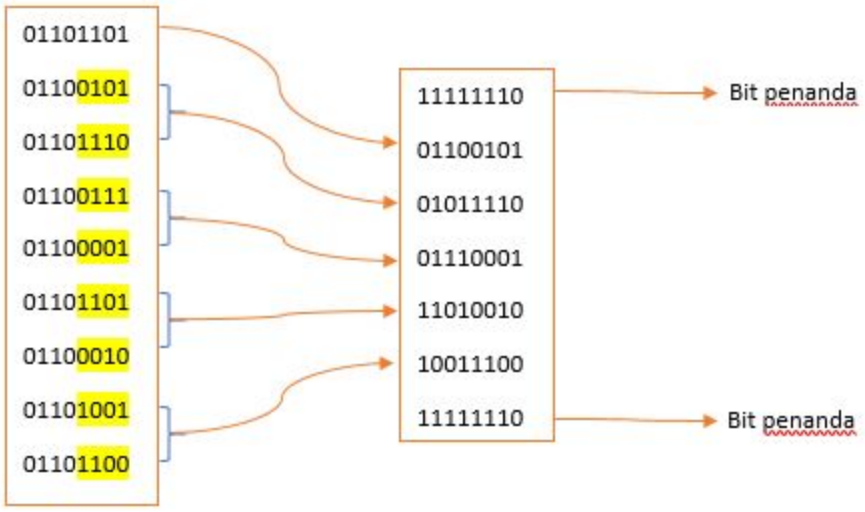
Metode kompresi *Half Byte* merupakan suatu metode kompresi dengan prosesnya adalah memanfatkan empat bit sebelah kiri yang sering sama secara berurutan terutama pada file-file text. Misalnya pada suatu file yang berisi data text bertuliskan “mengambil” yang diterjemahkan dalam heksadesimal dan biner pada tabel berikut

Karakter	Biner	Heksadesimal
m	01101101	6D
e	01100101	65
n	01101110	6E
g	01100111	67
a	01100001	61
m	01101101	6D
b	01100010	62
i	01100001	69
l	01101100	6C

Jika dilihat dari biner tiap-tiap huruf dapat kita amati bahwa bit sebelah kiri memiliki 4 bit yang sama yaitu 0110. Half Byte mamanfaatkan bit 4 bit yang sama sebelah kiri ini untuk melakukan kompresi data.

Saat karakter yang bit kirinya sama secara berderet, maka algoritma ini mengkompres data tersebut diawali dengan bit penanda kemudisan bit pertama dari deretan yang sama diikuti dengan pasangn bit kanan dari deretan tersebut dan ditutup dengan bit penanda. Bit penanda

(marker bit) berupa suatu byte yang dipilih secara acak asalkan digunakan secara konsisten pada seluruh bit penanda.
Dari biner diatas dikompresi menjadi seperti berikut



Deretan data sebelah kiri merupakan deretan data pada file asli, sedangkan deretan data sebelah kanan merupakan deretan data hasil pemampatan dengan algoritma Half-Byte.

Huffman Encoding

Metode ini diberi nama sesuai nama penemunya yaitu D.A. Huffman yang mengembangkan prosedur tersebut pada tahun 1950-an. Ide pengembangan metode ini didasari oleh data penggunaan karakter ASCII, dimana terkadang karakter ASCII dengan ukuran byte yang besar justru adalah karakter yang sering digunakan dan sebaliknya, karakter ASCII dengan ukuran byte yang kecil justru kurang sering digunakan. Sehingga muncul ide untuk menggunakan lebih sedikit bit (satu atau dua bit) dalam menginterpretasikan karakter yang sering digunakan dan membiarkan karakter yang jarang digunakan diinterpretasikan menggunakan lebih banyak bit.

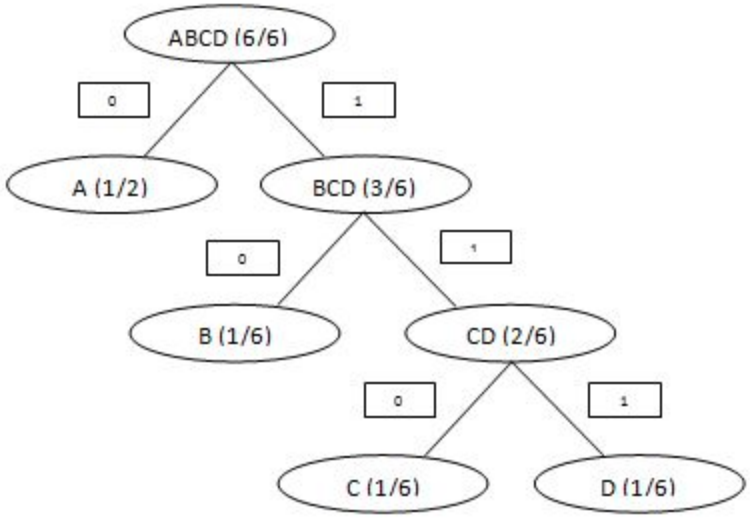
Skema dalam metode Huffman ini diawali dengan menghitung probabilitas kemunculan setiap karakter, kemudian memberi Huffman code untuk tiap karakter dengan aturan karakter yang memiliki probabilitas lebih besar akan memiliki Huffman code dengan ukuran bit yang lebih kecil. Skema penentuan Huffman code ini dapat dilakukan dengan membuat pohon Huffman. Prosedur pembuatan pohon Huffman secara umum adalah dengan mencari dua buah karakter yang memiliki nilai peluang terkecil, kemudian mengkombinasikan kedua karakter tersebut menjadi simpul parent dengan peluang merupakan jumlah peluang dari kedua anaknya. Ulangi langkah ini hingga semua karakter telah menjadi simpul pada pohon Huffman.

Berikut adalah contoh perhitungan compression rate dengan penggunaan Huffman Encoding.

Misal kita memiliki string masukkan berupa ACDABA. Maka kemudian kita akan menghitung probabilitas kemunculan dari tiap karakter dan memberikan Huffman code untuk masing-masing karakter.

Karakter	Probabilitas Kemunculan	Huffman Code
A	$^1/_2$	0
B	$^1/_6$	10
C	$^1/_6$	110
D	$^1/_6$	111

Dengan pohon Huffman yang terbentuk sebagai berikut.



Sehingga data yang telah dikompresi menggunakan Huffman Encoding akan menjadi 01101110100. Data asli yang awalnya membutuhkan 6 byte, setelah dilakukan kompresi menjadi hanya membutuhkan 11 bit saja.

Pembagian Tugas

1. Hani'ah Wafa - 13516053 : RLE dan Huffman Encoding
2. Dinda Yora Islami - 13516067 : Half Byte

Referensi :

http://artofartikel.blogspot.com/2012/05/proses-kompresi-dengan-metode-half-byte.html
http://slailinux.blogspot.com/2012/04/algoritma-kompresi-data.html
www.geeksforgeeks.org/run-length-encoding/amp/
http://academic.mu.edu/phys/matthysd/web226/L0425.htm
http://www.dspguide.com/ch27/3.htm
https://www.prepressure.com/library/compression-algorithm/huffman