**BAB I**

**DESKRIPSI TUGAS**

Dalam komunikasi data, pesan yang dikirim seringkali ukurannya sangat besar sehingga waktu pengirimannya lama. Begitu juga dengan penyimpanan data, arsip yang berukuran besar membutuhkan ruang penyimpanan yang besar. Kedua masalah ini dapat diatasi dengan mengkodekan pesan atau isi arsip sesingkat mungkin, sehingga waktu pengiriman pesan relatif cepat dan ruang penyimpanan yang dibutuhkan juga sedikit. Cara pengkodean seperti ini disebut kompresi (pemampatan) data dan pemulihan data tersebut kembali seperti aslinya disebut dekompresi (penirmampatan). Salah satu cara pemampatan dan nirmampat data ini adalah dengan menggunakan kode Huffman.

Pada tugas pertama Strategi Algoritma ini, kami diminta membuat aplikasi “*mini WinZip*” yang mengimplementasikan algoritma Huffman untuk memampatkan dan menirmampatkan data. Algoritma Huffman menggunakan prinsip *greedy* dalam pembentukan kode Huffman. Program kami harus dapat memampatkan semua jenis data baik berupa teks, gambar, suara, dan video dan program juga harus mampu mengembalikan data yang sudah dikompres tersebut ke bentuk asalnya (dekompresi). Sebagai contoh, jika *executable file* dimampatkan (misalnya notepad.exe), maka program notepade.exe tersebut harus dapat dijalankan kembali.

Program yang dibuat selain mampu memampatkan dan menirmampatkan data juga harus dapat menunjukkan perubahan hasil kompresi tersebut melalui nisbah pemampatannya. Nisbah (*ratio*) pemampatan dihitung dengan rumus:

*P* = Ukuran *file* hasil pemampatan/Ukuran *file* sebelum dimampatkan × 100%

Yang berarti ukuran *file* menjadi *P* (dalam persentase) dari ukuran semula.

Waktu yang dibutuhkan untuk memampatkan dan menirmampatkan data juga dicatat. Selain itu, program harus dapat memampatkan banyak *file* sekaligus, dan ketika penirmampatan harus dapat mengembalikannya menjadi *file-file* semula dengan nama sama. Entropi pesan dan rata-rata panjang bit setiap simbol di dalam pesan (*file*) juga dihitung.

**Penjelasan tambahan**: Algoritma Huffman standard memampatkan data agak lambat karena *file* harus dibaca dua kali. Pertama, *file* dibaca untuk menghitung frekuensi kemunculan karakter di dalam *file*. Kedua, *file* dibaca kembali untuk mengkodekan setiap karakter di dalam file dengan kode Huffman. Varian dari algoritma Huffman adalah algoritma *adaptive Huffman coding*. Pada algoritma adaptif ini, *file* dibaca hanya satu kali saja. Setiap kali karakter dibaca, pohon Huffman dibangun dan kode Huffman untuk karakter itu ditentukan, sekaligus karakter dikodekan langsung.

Dalam tugas ini, kami juga mengimplementasikan algoritma *adaptive Huffman coding*.

**BAB II**

**DASAR TEORI**

1. Algoritma *Greedy*
2. Definisi Algoritma *Greedy*

Algoritma *greedy* merupakan algoritma yang memberikan solusi persoalan langkah per langkah. Pada setiap langkah solusi, ada lebih dari satu pilihan solusi yang perlu diseleksi. Pilihan solusi yang dipilih biasanya merupakan solusi yang terbaik di setiap langkah. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya. Misalnya, jika kita menggunakan algoritma *greedy* untuk menempatkan komponen pada papan sirkuit, komponen yang sudah diletakkan tidak dapat diubah lagi posisinya.

Algoritma *greedy* menggunakan pendekatan dalam bentuk membuat pilihan yang “tampaknya” memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan harapa bahwa sisanya mengarah ke solusi optimum global.

Dengan kata lain, algoritma *greedy* adalah algoritma yang memecahkan masalah langkah per langkah dan pada setiap langkahnya mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma *greedy* mengasumsikan bahwa optimum lokal merupakan bagian dari optimum global.

1. Skema Umum Algoritma *Greedy*

Elemen-elemen yang menyusun persoalan optimasi dalam konteks algoritma *greedy* adalah :

1. Himpunan kandidat, C

Himpunan ini berisi elemen-elemen pembentuk solusi, misalnya himpunan koin dan himpunan simpul di dalam graf. Satu buah kandidat diambil dari himpunan ini pada setiap langkah.

1. Himpunan solusi, S

Himpunan ini berisi kandidat-kandidat yang terpilih di setiap langkah sebagai solusi persoalan. Himpunan ini merupakan bagian dari himpunan kandidat.

1. Fungsi seleksi

Fungsi ini merupakan fungsi yang pada setiap langkah memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak akan dipertimbangkan lagi pada langkah selanjutnya. Setiap kandidat biasanya dimasukkan nilai numerik, lalu fungsi seleksi memilih kandidat yang mempunyai nilai terbesar atau nilai terkecil bergantung pada persoalannya.

1. Fungsi kelayakan (*feasible*)

Fungsi ini memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kedala yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak akan pernah dipertimbangkan lagi.

1. Fungsi obyektif

Fungsi ini adalah fungsi yang memaksimumkan atau meminimumkan nilai solusi (misalnya panjang lintasan, keuntungan, dan lain-lain).

Skema algoritma *greedy* secara umum, yaitu *S* diinisialisasi dengan kosong, sebuah kandidat dipilih dari *C* dengan fungsi seleksi. Kandidat yang sudah dipilih dihapus dari *C*, kandidat yang sudah dipilih tersebut diperiksa apakah membentuk solusi yang layak bersama himpunan solusi dengan fungsi kelayakan, jika iya kandidat tersebut dimasukkan ke dalam himpunan solusi, jika tidak kandidat tersebut dibuang dan tidak perlu dipertimbangkan lagi, himpunan solusi diperiksa apakah sudah memberikan solusi yang lengkap dengan menggunakan fungsi solusi, jika iya maka selesai, jika tidak langkah diulangi lagi dari pemilihan sebuah kandidat dari *C* dengan fungsi seleksi.

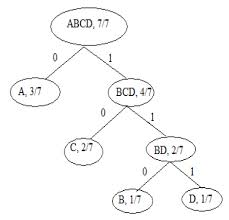
Algoritma *greedy* tidak selalu menghasilkan solusi yang benar-benar optimum karena tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada dan terdapat banyak kemungkinan fungsi seleksi yang berbeda.

1. Algoritma Huffman

Prinsip dasar pada kode Huffman adalah karakter yang paling sering muncul di dalam data dikodekan (*encode*) dengan kode yang lebih pendek, sedangkan karakter yang relatif jarang dikodekan dengan kode yang lebih panjang. Pemampatan (kompresi) data dengan kode Huffman bisa memberi penghematan sebesar 20% hingga 30%.

Cara untuk mendapatkan kode Huffman adalah pertama, dipilih dua simbol dengan frekuensi kemunculan pada data paling kecil. Frekuensi kemunculan kedua simbol tersebut kemudian dijumlahkan dan hasil penjumlahannya diletakkan pada simpul orangtua dari kedua simbol sebelumnya (menggunakan pohon biner). Simpul ini baru ini diperhitungkan dalam mencari simbol selanjutnya yang memiliki peluang paling kecil. Selanjutnya, pilih dua simbol berikutnya, termasuk simbol baru, yang mempunyai peluang terkecil. Hal ini dilakukan terus sehingga terbentuk satu pohon yang mengandung semua simpul berisi semua simbol yang muncul pada data. Pengkodean setiap simbol dilakukan dengan memberi label 0 pada setiap cabang kiri dan label 1 pada setiap cabang kanan.

Proses peng-dekode-an (*decode*) kode Huffman dapat dillakukan dengan memulai dari aka, lalu membaca semua *string* biner. Untuk setiap bit, dilakukan transversal pada cabang yang bersesuaian hingga bertemu daun. Rangkaian bit yang telah dibaca dikodekan dengan karakter di daun. Proses tersebut diulangi hingga semua bit dalam *string* habis.



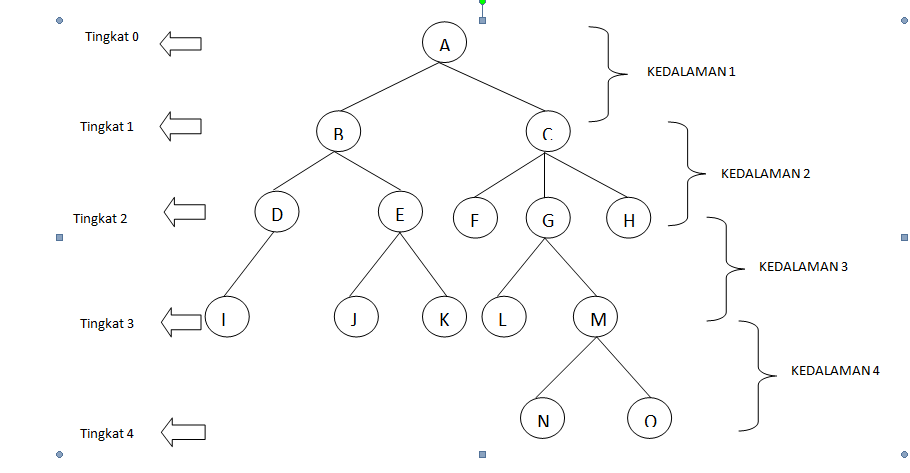
Gambar 2.1 Pohon Huffman

Sumber : http://knxbmt.blogspot.co.id/2010/06/algoritma-huffman.html

Pembentukan Kode Huffman dengan algoritma *greedy* dapat dilakukan dengan membaca terlebih dahulu semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul dimasukkan dengan frekuensi kemunculan karakter tersebut. Lalu, diterapkan strategi *greedy* dengan menggabungkan dua buah pohon yang mempunyai frekuensi terkecil pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon sebelumnya. Langkah-langkah tersebut kemudian diulangi hingga hanya tersisa satu buah pohon Huffman. Untuk pengkodean setiap karakter di dalam data, dibaca kembali karakter-karakter pada pohon, lalu dikodekan dengan karakter yang sesuai (kombinasi 0 dan 1).

1. Pohon Biner

Pohon biner merupakan pohon yang setiap simpulnya memiliki paling banyak dua buah anak. Kedua anak tersebut disebut dengan anak kiri dan anak kanan. Pohon yang akarnya adalah ana kiri disebut upapohon kiri, sedangkan pohon yang akarnya adalah anak kanan disebut upapohon kanan. Karena adanya perbedan anak kiri dan anak kanan, serta upapohon kiri dan upapohon kanan, maka pohon biner adalah pohon terurut. Pohon yang semua simpulnya terletak di bagian kiri saja disebut pohon condong-kiri, sementara yang semua simpulnya terletak di bagian kanan saja disebut pohon condong-kanan. Pohon biner penuh adalah pohon biiner yang setiap simpulnya mempunyai tepat dua buah anak, kecuali simpul pada aras bawah.



Gambar 2.2 Pohon Biner

Sumber : http://ichaajja212.blogspot.co.id/2012/10/struktur-data-dan-algoritma-mencari.html

**BAB III**

**ANALISIS PEMECAHAN MASALAH**

1. Kode Huffman

Program yang dibuat menggunakan algoritma Huffman untuk memampatkan data. Program akan membaca karakter-karakter yang ada pada data dan membuat pohon Huffman berdasarkan frekuensi kemunculan karakter-karakternya. Karakter-karakter tersebut kemudian dikodekan sesuai dengan pohon Huffman yang sudah dibuat. Kemudian, program juga menirmampatkan data dengan cara melakukan *decode* terhadap arsip hasil pemampatansesuai pohon Huffman yang telah dibuat.

Algoritma Huffman biasa termasuk lambat dalam memampatkan data, karenanya kami juga membuat program menggunakan algoritma Huffman adaptif yang bisa memampatkan data lebih cepat, karena program membaca karakter pada data, membuat dan memperbarui pohon, serta membuat arsip hasil pengkodean dalam sekali transversal.

1. Struktur Data
2. Struktur Data Larik (*Array*)

Program menggunakan struktur data larik untuk menyimpan frekuensi kemunculan karakter sebelum dimasukkan ke dalam pohon, untuk menyimpan daftar hasil pengkodean dari masing-masing karakter, Tambahin Lagi Dong Kalau Ada

1. Struktur Data Pohon Biner

Program menggunakan struktur data pohon untuk membuat pohon Huffman yang digunakan untuk pengkodean dan pen-dekode-an. Setiap simpul pohon Huffman menyimpan data bentukan dalam bentuk karakter dan frekuensi kemunculannya.

1. Spesifikasi Program
2. Program bisa memampatkan semua jenis data, baik yang berbentuk gambar, dokumen, video, dan lain-lain.
3. Program bisa menirmampatkan data.
4. Program bisa menunjukkan perubahan hasil kompresi tersebut melalui nisbah pemampatannya.
5. Program bisa menampilkan waktu yang dibutuhkan untuk memampatkan dan menirmampatkan data.
6. Program bisa menampilkan entropi pesan dan rata-rata panjang bit setiap simbol di dalam pesan.
7. Program bisa memampatkan dan menirmampatkan dengan algoritma Huffman biasa dan algoritma Huffman adaptif.
8. Program memiliki antarmuka yang kompatibel.
9. Program bisa menampilkan proses pemampatan dan nirmampat dalam bentuk *progress bar*.

**BAB IV**

**IMPLEMENTASI DAN PENGUJIAN**

1. Implementasi
2. Pengujian

**BAB V**

**KESIMPULAN DAN SARAN**

1. Kesimpulan
2. Saran

**DAFTAR PUSTAKA**

Munir, Rinaldi. “Diktat Kuliah IF2120 Matematika Diskrit”. 2006. Bandung : Institut Teknologi Bandung

Munir, Rinaldi. “Diktat Kuliah IF2211 Strategi Algoritma”. 2009. Bandung : Institut Teknologi Bandung