APLIKASI GREEDY PADA ALGORITMA HUFFMAN UNTUK KOMPRESI TEKS

Nessya Callista
13505119

Program Studi Teknik Informatika
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
Jl.Ganeca No.10
e-mail: if15119@students.if.itb.ac.id

ABSTRAK

Saat ini penyimpanan data yang memakan ruang memori yang cukup besar merupakan suatu masalah yang umum. Kompresi data merupakan salah satu solusi yang dapat membantu menyelesaikan masalah tersebut. Banyak algoritma yang digunakan dalam kompresi teks, contohnya Huffman, algoritma LZW (Lempel-Ziv-Welch), algoritma DMC (Dynamic Matrix Control) dan algortima lainnya. Pada algoritma LZW kita menggunakan *dictionary* dimana setiap bagian dari teks digantikan dengan indeks yang diperoleh dari *dictionary* tadi. Algoritma DMC didasarkan pada model awal berupa model FSA (*finite-state-automata*) dengan transisi 0/1 atau 1/1. Algoritma yang paling terkenal dalam kompresi teks adalah algoritma Huffman, yang dapat memberi penghematan sekitar 20% sampai 30%. Algoritma Huffman menerapkan strategi greedy. Setiap algoritma tersebut memiliki kelebihan dan kekurangan masing-masing. Algoritma yang diterapkan pada kompresi disesuaikan dengan jenis file yang akan dikompres. Kecepatan algoritma Huffman hampir merata untuk semua kategori file.

Kata kunci: algoritma Huffman, algoritma greedy.

1. PENDAHULUAN

Kompresi adalah pengubahan data yang berupa kumpulan karakter menjadi bentuk kode dengan tujuan untuk menghemat kebutuhan tempat penyimpanan dan waktu transmisi data. Ada beberapa faktor yang dijadikan sebagai bahan pertimbangan dalam memilih algoritma yang akan digunakan dalam kompresi data yaitu : sumber daya yang dibutuhkan (memory, kecepatan PC), kecepatan kompresi, ukuran hasil kompresi, besarnya redudansi dan kompleksitas algoritma.Pada algoritma Huffman setiap karakter pada teks akan dikodekan dalam bentuk *string* biner. Huffman menggunakan metode statik yang selalu menggunakan peta kode yang sama. Berdasarkan teknik pengkodean simbol yang digunakan, metode kompresi dapat dibagi dalam tiga kategori :

1. Metode *symbolwise*
   Dalam metode ini peluang kemunculan dari tiap simbol dalam file input dihitung, lalu mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Metode ini diterapkan pada algoritma Huffman

2. Metode *dictionary*
   Pada metode ini karakter dalam file input digantikan dengan indeks lokasi dari karakter tersebut dalam sebuah *dictionary*. Metode ini diterapkan pada algoritma LZW.

3. Metode *predictive*
   Metode ini menggunakan model FSA (*finite state automa*) untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya. Metode ini diterapkan pada algoritma DMC.

2. ALGORITMA HUFFMAN

Prinsip kode Huffman adalah mengganti karakter yang paling sering muncul di dalam data dengan kode yang lebih pendek, sedangkan karakter yang lebih jarang muncul dikodekan dengan kode yang lebih panjang. Algoritma memiliki kompleksitas sebesar $O(n \log n)$ untuk himpunan dengan n karakter. Huffman menerapkan metode statik yaitu menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase, sebagai berikut:

1. fase pertama untuk menghitung kemungkinan kemunculan tiap karakter dan menentukan peta kodenya

2. fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditransmisikan.

Misalnya, sebuah berkas yang berisi 100.000 karakter dengan frekuensi kemunculan karakter dalam data:

Karakter	Frekuensi
A	45%
B	13%
C	12%
D	16%
E	9%
F	5%

Digunakan tiga bit untuk mengkodekan setiap karakter :

Karakter	Frekuensi	Kode
A	45%	000
B	13%	001
C	12%	010
D	16%	011
E	9%	100
F	5%	111

Maka untuk memyimpan data di dalam berkas yang berisi 100.000 karakter, kita membutuhkan 300.000 karakter bit. Dan jelas pengkodean diatas tidak mangkus. Tetapi dengan kode Huffman berikut :

Karakter	Frekuensi	Kode
A	45%	0
B	13%	101
C	12%	100
D	16%	111
E	9%	1101
F	5%	1100

Untuk menyimpan data di dalam berkas yang berisi 100.000 karakter, kita membutuhkan :

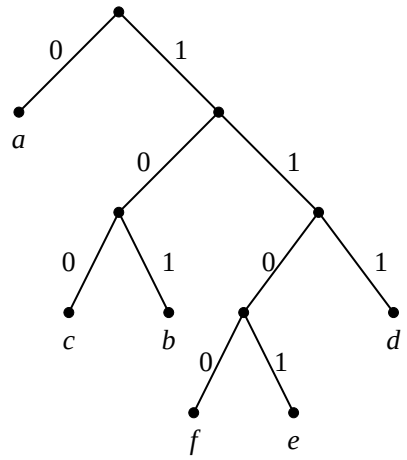
((0.45 x 1) + (0.13 x 3) + (0.12 x 3) + (0.16 x 3) + (0.09 x 4) + (0.05 x 4)) x 100.000 = 224.000 bit.

Dengan demikian kita telah melakukan kompresi data sebesar :

$$\frac{(300.000 - 224.000)}{300.000} \times 100\% = 25.3\%$$

Kode prefiks memberikan pemampatan data yang optimal diantara kode-kode lain.

Kode Huffman merupakan kode prefiks yang direpresentasikan pada pohon biner berlabel, yang dalam hal ini setiap sisi diberi label 0 (cabang kiri) atau 1 (cabang kanan). Kode prefiks adalah himpunan yang berisi sekumpulan kode biner dimana tidak ada kode biner yang menjadi awal bagi kode bine yang lain.



Secara ringkas algoritma Huffman adalah sebagai berikut:

1. Pass pertama
   Baca (*scan*) file input dari awal hingga akhir untuk menghitung frekuensi kemunculan tiap karakter dalam file. *n* adalah jumlah semua karakter dalam file input. *T* adalah semua karakter dan nilai peluang kemunculannya dalam file input. Tiap karakter menjadi *node* daun pada pohon Huffman.

2. Pass kedua
   Ulangi sebanyak (*n*-1) kali :

   a. Item *m1* dan *m2* adalah dua *subset* dalam *T* dengan nilai peluang yang terkecil.

   b. Gantikan *m1* dan *m2* dengan sebuah item {*m1*,*m2*} dalam *T*, dimana nilai peluang dari item yang baru ini adalah penjumlahan dari peluang *m1* dan *m2*.

   c. Buat *node* baru {*m1*,*m2*} sebagai *father node* dari *node m1* dan *m2* dalam pohon Huffman.

3. *T* sekarang tinggal berisi satu item, dari item ini sekaligus menjadi node akar pohon Huffman. Panjang kode untuk suatu simbol adalah jumlah berapa kali simbol tersebut bergabung dengan item lain dalam *T*.

3. ALGORITMA GREEDY PADA HUFFMAN

3.1 Strategi Greedy

Algoritma greedy merupakan salah satu metode pemecahan persoalan optimasi yang paling populer. Prinsip greedy adalah *"take what you can get now"*. Algoritma greedy telah diterapkan dalam berbagai persoalan optimasi. Misalnya pada TSP (*travelling salesperson Problem*), *Job scheduling with deadlines*, Lintasan terpendek (*Shortest Path*), dan persoalan lain.

Algoritma ini membentuk solusi langkah per langkah. Pada setiap langkah dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Pada umumnya algoritma greedy memiliki beberapa elemen-elemen sebagai berikut :

1. Himpunan kandidat
   Himpunan ini dilambangkan dengan C yang berisi elemen-elemen pembentuk solusi.

2. Himpunan solusi
   Himpunan ini dilambangkan dengan S yang berisi kandidat-kandidat yang terpilih sebagai solusi persoalan, dan merupakan bagian dari himpunan kandidat.

3. Fungsi seleksi
   Dinyatakan dengan predikat SELEKSI yang merupakan fungsi yang pada setiap langkah memilih kandidat yang paling memungkinkan mencapai solusi optimal.

4. Fungsi kelayakan
   Dinyatakan dengan predikat LAYAK yang 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 kendala (*constraints*) yang ada.

5. Funsi obyektif
   Merupakan fungsi yang memaksimumkan atau meminimumkan nilai solusi.

Secara umum algoritma greedy memiliki skema sebagai berikut :

1. Inisialisasi S dengan kosong
2. Pilih sebuah kandidat dengan fungsi SELEKSI dari C
3. Kurangi C dengan kandidat yang dipiliih tersebut bersama-sama dengan himpunan solusi membentuk solusi yang layak atau tidak. Jika tidak maka buang kandidat tadi. Jika ya, masukkan kandidat tadi sebagai himpunan solusi
4. Periksa apakah himpunan solusi sudah memberikan solusi yang lengkap. Jika ya, maka berhenti, jika tidak, kembali ke langkah 2.

Pendekatan yang dilakukan algoritma greedy adalah membuat pilihan yang tampaknya memberikan perolehan terbaik yaitu dengan membuat pilihan optimum lokal pada setiap langkah. Pada setiap langkah di dalam algoritma greedy kita baru memperoleh nilai optimum lokal. Bila algoritma berakhir kita berharap optimum lokal dari tiap langkah tadi menjadi optimum global.

3.2 Penerapan Greedy pada Pembentukan Huffman

Algoritma greedy membentuk kode prefiks yang optimal pada kode Huffman. Langkah-langkah pembentukan pohon Huffman adalah sebagai berikut :

1. Baca semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Dan setiap simpul ini di-*assign* dengan frekuensi kemunculan karakter tersebut.

2. Terapkan 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 penyusunnya.

3. Ulangi langkah dua sampai hanya tersisa satu buah pohon Huffman. Agar pemilihan dua pohon yang akan digabungkan berlangsung dengan cepat, maka semua pohon yang ada selalu terurut menaik berdasarka frekuensi.

4. Baca kembali karakter-karakter di dalam data, kodekan setiap karakter dengan kode Huffman yang bersesuaian.

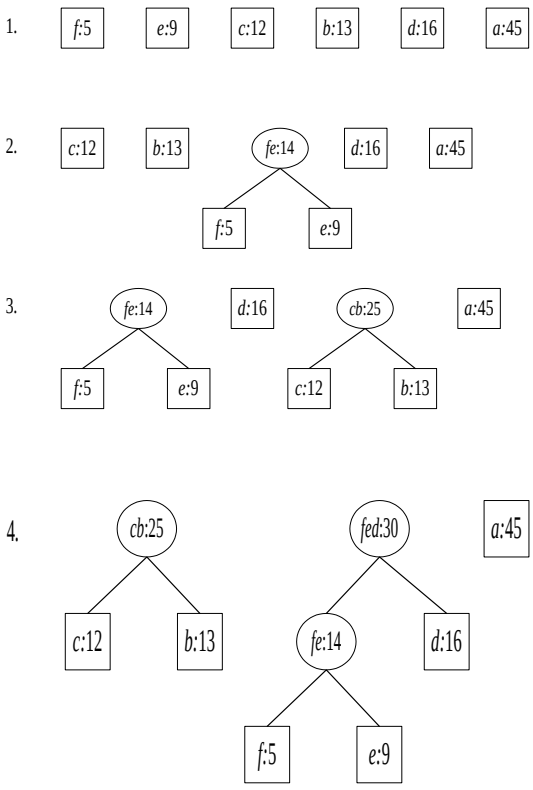
Algoritma greedy digunakan untuk meminimumkan jumlah *cost* yang dibutuhkan untuk menggabungkan dua buah pohon pada akar yang telah disebut diatas.

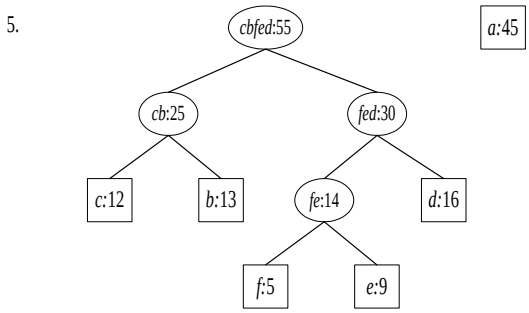
Penggabungan dua buah pohon dilakukan setiap langkah dan algoritma Huffman selalu memilih dua buah pohon yang memiliki frekuensi terkecil untuk meminimumkan total *cost*. Inilah alasan mengapa strategi greedy diterapkan dalam strategi penggabungan dua buah pohon.

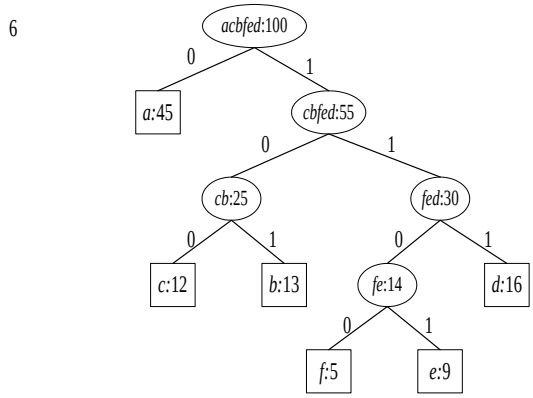
Misalnya, data dengan panjang 100 karakter dan disusun oleh huruf-huruf *a,b,c,d,e* dengan frekuensi kemunculan setiap huruf sebagai berikut :

Karakter	Frekuensi
a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

Langkah – langkah pembentukan pohon Huffman :







Pada langkah (2) diambil akar yang memiliki jumlah kemunculan terkecil. Dalam hal ini karakter yang memiliki nilai frekuensi terkecil adalah *f* dan *e*. Karakter *e* dan *f* kemudian dibentuk menjadi akar dari pohon yang baru.

Pada langkah (3) diambil lagi 2 akar dari pohon yang memiliki jumlah kemunculan terkecil. Dalam hal ini adalah akar *c* dan *b*. Akar *c* dan *b* tadi digabung sehingga membentuk akar dari pohon yang baru dengan jumlah frekuensi sama dengan hasil penjumlahan frekuensi akar *c* dan *b*. Selanjutnya 2 akar terkecil pada langkah (3) digabungkan kembali untuk membentuk pohon dengan akar yang baru dengan frekuensi yang merupakan penjumlahan dari frekuensi akar pembentuknya. Akar terkecil yang diambil pada langkah (3) adalah *fe* dan *d*. Pada langkah (5) akar *fed* dan *cb* digabungkan untuk membentuk pohon yang baru.

Langkah-langkah tadi terus dilakukan hingga semua akar yang ada telah digabung sehingga membentuk satu pohon yang baru dengan akar yang memiliki frekuensi sama dengan jumlah frekuensi seluruh akar pertama kali. Dalam hal ini seluruh akar pada langkah (1).

IV. KESIMPULAN

1. Algoritma greedy pada umumnya digunakan untuk penyelesaian persoalan optimasi, TSP, *Job scheduling with deadlines*, *Shortest Path* (Lintasan terpendek) dan persoalan lain.

2. Algoritma greedy membantu algoritma huffman dalam menentukan akar-akar yang akan digabungkan untuk membentuk akar dari pohon yang baru.

3. Algoritma greedy meminimumkan jumlah *cost* yang dibutuhkan untuk menggabungkan dua buah pohon pada akar dengan frekuensi yang merupakan hasil penjumlahan frekuensi-frekuensi dari pohon penyusunnya.

REFERENSI

[1] Linawati dan Henry P.Panggabean, "Perbandingan Kinerja Algoritma Huffman, LZW dan DMC pada berbagai tipe file", Integral, Volume 9, Nomor 1, 2004.

[2] Philip Chan, "Spam Filtering Using Statistical Data Compression Models", 2006.

[3] Rinaldi Munir, "Bahan Kuliah ke-4 Algoritma Greedy (lanjutan)", 2004.