Kompleksitas Algoritma dari Algoritma Pembentukan pohon Huffman Code Sederhana

Muhammad Fiqri Muthohar – NIM : 13506084[1)]

1) Jurusan Teknik Informatika ITB, Bandung, email: fiqri@arc.itb.ac.id

Abstrak *– makalah ini membahas salah satu implementasi dari pohon yaitu Huffman code. Di mana Huffman code merupakan salah satu bentuk implementasi dari pohon yang digunakan untuk mengkode ulang kode karakter dari suatu data atau pesan dengan kode yang lebih sedikit.*

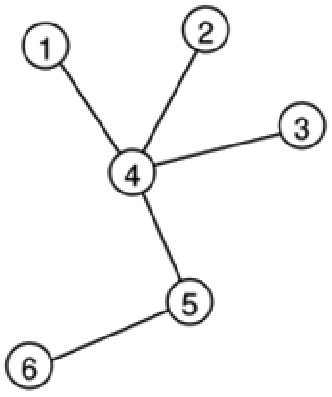
Kompleksitas algoritma total dari sebuah algoritma pembentukan pohon Huffman code secara keseluruhan bergantung pada proses pembentukan pohon Huffman code dan pemilihan algoritma sorting yang digunakan.

Selanjutnya penulis akan menjelaskan lebih lengkap tentang masalah kompleksitas algoritma total dari program pembentukan pohon Huffman code dalam makalah berikut ini.

Kata kunc*i :Huffman code, pohon, kompleksitas algoritma, algoritma pengurutan data*

1. Pendahuluan

Pohon (*tree*) adalah graf yang khusus. Definisi dari pohon adalah graf tak-berarah terhubung yang tidak memiliki sirkuit. Dari definisi tersebut terdapat dua sifat penting dari sebuah pohon, yaitu: terhubung dan tidak memiliki sirkuit.



Gambar 1. Contoh pohon

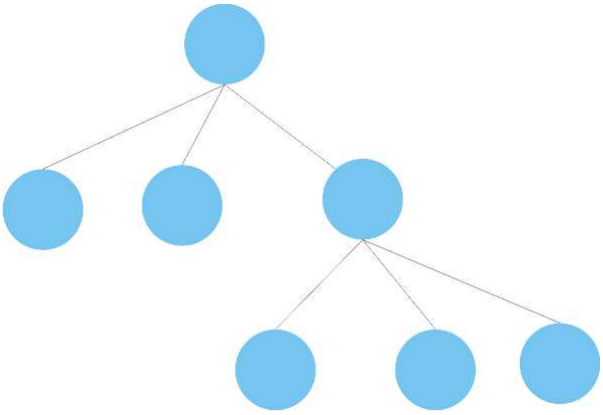
Sifat-sifat pohon dinyatakan dalam teorema berikut.

Misalkan G = (V, E) adalah graf tak-berarah sederhana dan jumlah simpulnya *n*. Maka, semua pernyataan di bawah ini adalah ekivalen :

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki *m = n-1* buah sisi.
4. G tidak memiliki sirkuit dan memiliki *m = n-1* buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang bila dihapus menyebabkan graf terpecah menjadi dua komponen).

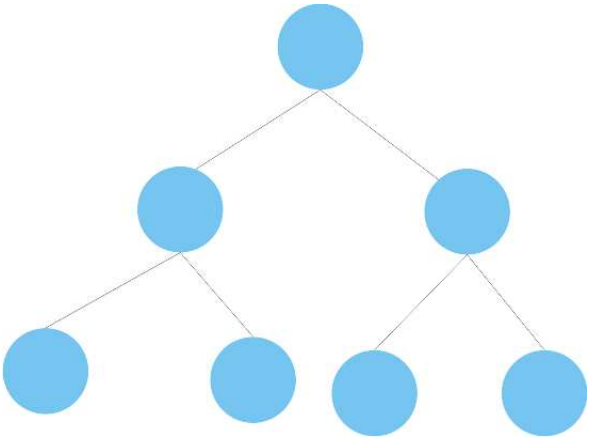
Pada kebanyakan aplikasi pohon, simpul tertentu diperlakukan sebagai akar (*root*). Sekali sebuah simpul ditetapkan sebagai sebuat akar, maka simpul-simpul lainnya dapat dicapai dari akar dengan member arah pada sisi-sisi pohon yang mengikutinya.

Pohon berarah yang setiap simpul cabangnya memmpunyai *n* buah anak disebut pohon *n-ary*. Pohon jenis ini banyak digunakan di berbagai bidang ilmu maupun dalam kehidupan sehari-hari. Dalam terapannya, pohon *n-ary* digunakan sebagai model yang mempresentasikan sebuah struktur.



Gambar 2. Contoh pohon n-airy

Pohon biner merupakan kasus khusus pohon n-ary jika n = 2. Pohon biner adalah pohon yang setiap simpul cabangnya memiliki paling banyak dua buah anak. Yaitu, anak kiri (*left child*) dan anak kanan (*right child*). Pohon yang akarnya adalah anak kiri disebut upapohon kiri (*left subtree*) dan pohon yang akarnya adalah anak kanan disebut upapohon kanan (*right subtree*). Karena adanya perbedaan anak/upapohon kiri dan anak/upapohon kanan, maka pohon biner adalah pohon terurut.



Gambar 3. Contoh pohon biner

Huffman code adalah salah satu implementasi dari pohon biner yang dapat digunakan untuk mengkodekan ulang data yang ada sehingga ukurannya dapat lebih kecil. Pengkodean dengan cara seperti ini disebut juga sebagai perampatan (*compression*) data sederhana. Perampatan data dapat dilakukan dengan mengkodekan setiap karakter di dalam pesan atau di dalam arsip dikodekan dengan kode yang lebih pendek. Dalam hal ini kode yang banyak digunakan adalah kode ASCII. Dengan kode ASCII karakter dikodekan dalam 8 bit biner. Tabel 1 berikut adalah contoh kode ASCII untuk beberapa karakter.

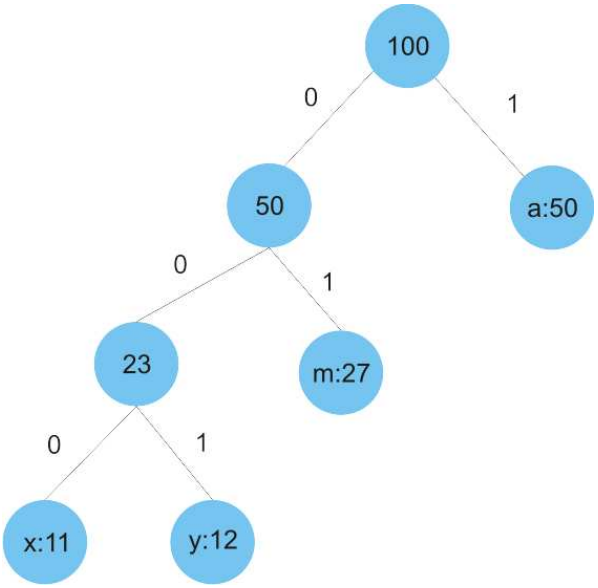
Simbol	KodeASCII
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001

Tabel 1. Contoh kode ASCII

2. Huffman code

Huffman code*[2]* merupakan kasus klasik yang sering digunakan pada pendidikan ilmu komputer. Huffman code dipilih sebagai bahan ajar karena memiliki aspek-aspek penting antara lain struktur data (pohon, pengurutan) dan algoritma (*greedy*). Selain sebagai bahan ajar, Huffman code juga digunakan dalam pengembangan perangkat lunak. Huffman code dikenal sebagai metode kompresi sederhana, yang dapat dikembangkan untuk menghasilkan metode kompresi yang lebih baik atau sesuai kebutuhan.

Huffman encoding [3] adalah sebuah teknik kompresi dokumen yang menggunakan jumlah kemunculan relatif simbol-simbol karakter pada dokumen teks untuk menghasilkan representasi biner dengan panjang tertentu untuk tiap karakter. Representasi biner ini menjadi kode *Huffman* untuk sebuah karakter. Proses *encoding* memiliki karakteristik bahwa tidak ada kode untuk sebuah karakter yang diawali oleh kode karakter lain. Pada umumnya, kode *Huffman* direpresentasikan dalam bentuk pohon biner *Huffman* (“0” merepsentasikan cabang kiri, dan “1” merepresentasikan cabang kanan) seperti ditunjukkan dalam Gambar 4.



Gambar 4. Contoh pohon biner Huffman

Pada gambar 4, kode Huffman untuk karakter "y" adalah 001 dan kode Huffman untuk karakter "a" adalah 1. Bilangan yang ada pada setiap simpul pohon menunjukkan jumlah frekuensi karakter-karakter yang ada di subpohon. Pada gambar 4, simpul yang terdapat bilangan 23 memiliki subpohon yang terdiri dari simpul "x" dengan frekuensi kemunculan sebanyak 11 kali dam simpul "y" yang memiliki frekuensi kemunculan sebanyak 12. Kode Huffman untuk setiap karakter dapat dilihat pada tabel 2.

karakter	Huffman code
x	000
y	001
m	01
a	1

Tabel 2. Huffman code untuk karakter pada gambar 4

Jika kita lihat berdasarkan pengkodean ASCII, representasi 100 huruf menggunakan 100x8 = 800 bit (100 byte). Untuk meminimumkan jumlah bit yang dibutuhkan, panjang kode untuk setiap kode diperpendek, terutama untuk karakter yang kekerapan (*frequency*) kemunculannya besar. Pemikiran ini adalah yang mendasari munculnya kode Huffman.

Dengan menggunakan kode Huffman dari tabel di atas maka jika ada 100 karakter dengan frekuensi yang sama dengan tabel tersebut maka bit yang diperlukan adalah sebanyak 23 x 3 + 27 x 2 + 50 x 1 = 69 + 54 + 50 = 173 bit. Simbol-simbol yang sering muncul direpresentasikan dengan kode yang lebih pendek daripada kode untuk simbol yang lain. Kode untuk setiap simbol tidak boleh merupakan awalan dari kode yang lain sebab akan menimbulkan keraguan (*ambigou*) dalam proses pemulihannya (*decoding* – yaitu pengubahan kembali kode biner ke kode asalnya).

Dapat dilihat pula bahwa kode Huffman tidak bersifat unik, yang berarti kode untuk setiap karakter berbeda-beda pada setiap pesan bergantung pada kekerapan kemunculan karakter tersebut di dalam pesan. Selain itu, keputusan apakah suatu simpul pada pohon Huffman diletakkan di kiri atau di kanan juga menentukan kode yang dihasilkan (tetapi panjang kode tidak terpengaruhi).

3. Kompleksitas Algoritma

Huffman code dibangkitkan menggunakan algoritma greedy terhadap frekuensi kemunculan karakter. Pada langkah awal, kumpulan koleksi yang terurut berdasarkan frekuensi diisi dengan simpul-simpul, satu simpul untuk satu karakter pada dokumen. Kemudian, dua simpul dengan frekuensi minimum dihapuskan dari koleksi, keduanya digabungkan menjadi simpul parent dan frekuensi dari simpul parent tersebut diisi dengan jumlah dari frekuensi kedua simpul anak yang dihapus sebelumnya. Selanjutnya, simpul hasil penggabungan ini dimasukkan kembali kedalam koleksi terurut sebelumnya. Proses ini dilakukan berulang-ulang sampai seluruh koleksi telah direduksi menjadi satu simpul tunggal akar.

Dari definisi di atas dapat di buat sebuah algoritma sederhana sebagai berikut.

```
Procedure Huffman (L : list of tree)

{ membangun pohon Huffman code dari list of tree
yang berisikan karakter-karakter yang ada pada data}

Deklarasi

A, B, T : tree

Algoritma

Sort(L) {mengurutkan elemen list of tree L}
```

While member (L) ≠ 1 do

Deletefirst(L, A) { mendelete elemen pertama dari list of tree L,dan menyimpan nilai dari yang di delete di variable A}

DeleteFirst(L, B) { mendelete elemen pertama dari list of tree L,dan menyimpan nilai dari yang di delete di variable A }

MakeTree(T, A, B) {membentuk sebuah tree T dari variable A dan B}

SearchPosAndInsertAfterPos(T, L) {mencari posisi yang tepat dari tree T pada list of tree L dan meletakkannya di list of tree T}

Endwhile

Contoh algoritma pembentukan pohon Huffman code secara keseluruhan

Dari contoh algoritma sederhana di atas terlihat bahwa hanya ada dua bagian utama dari program Huffman code sederhana ini yaitu pengurutan data dan pembentukan pohon Huffman code itu sendiri.

Proses pembentukan pohon Huffman code jika dilihat dari contoh algoritma sederhana di atas adalah T(f(n))=n-1=O(n).

Proses sorting data dapat mempengaruhi kompleksitas algoritma total, apabila kita mengambil pengurutan data dengan kompleksitas algoritma $O(n^2)$ atau mengambil pengurutan data dengan kompleksitas algoritma O(log n) tentu akan menghasilkan kompleksitas algoritma yang berbeda.

Secara keseluruhan maka kompleksitas algoritma dari pembentukan pohon Huffman code sederhana dipengaruhi oleh kompleksitas algoritma sorting yang digunakan dan juga proses pembentukan pohon Huffman code. Dalam Huffman code menggunakan kode ASCII jumlah jenis karakter yang mungkin ada tidak lebih dari 256. Jika dilihat maka secara waktu masalah pembentukan pohon Huffman code ini tidak akan banyak memerlukan waktu yang lama karena adanya pembatasan maksimum jumlah data yang dapat diproses.

Dengan mengandaikan semua proses diatas sebagai sebuah fungsi. Misal fungsi pengurutan data kita andaikan sebagai fungsi f(n) dan fungsi pembentukan pohon Huffman code sebagai g(n). Maka dapat dirumuskan secara total keseluruhan program akan memiliki perhitungan kompleksitas algoritma sebagai berikut.

$$O(f(n)+g(n))=O(max(\ O(f(n)),\ O(g(n))\ ))\ \ (1)$$

Misalkan kita memilih algoritma pengurutan data bubble sort yang memiliki kompleksitas algoritma $O(n^2)$ maka kompleksitas algoritma secara total adalah dengan cara persamaan (1) didapatkan bahwa kompleksitas algoritma program secara keseluruhan adalah $O(n^2)$ berdasarkan persamaan (1) tersebut. Dapat dilihat bahwa kompleksitas algoritma pada proses pembentukan pohon Huffman code adalah O(n) sehingga nilai tersebut lebih kecil daripada nilai kompleksitas algoritma pengurutan data bubble sort $O(n^2)$.

Demikian pula jika kita memilih algoritma pengurutan data yang memiliki kompleksitas algoritma misalkan O(n log n). Maka berdasarkan persamaan (1) kompleksitas algoritma program secara total menjadi O(n log n) dikarenakan nilai O(n log n) lebih besar daripada O(n).

Misalkan terdapat algoritma pengurutan data yang memiliki kompleksitas algoritma O(log n) atau O(n) maka dalam hal ini berdasarkan persamaan (1) kompleksitas algoritma total baru akan bernilai O(n) dimana proses pembentukan pohon Huffman code memiliki kompleksitas waktu yang lebih besar atau sama dengan kompleksitas algoritma pengurutan data.

Nama algoritma pengurutan data	Kompleksitas algoritma pengurutan data	Kompleksitas algoritma Huffman code total
Bubble sort	$O(n^2)$	$O(n^2)$
X	O(n log n)	O(n log n)
Y	O(n)	O(n)
Z	O(log n)	O(n)

Tabel 3. Perbandingan penggunaan algoritma pengurutan data dengan kompleksitas algoritma yang berbeda

4. Kesimpulan

Makalah ini membahas kompleksitas algoritma dari algoritma pembentukan pohon Huffman code sederhana secara keseluruhan. Terlihat bahwa

pemilihan algoritma pengurutan data yang digunakan akan mempengaruhi kompleksitas algoritma program, secara keseluruhan.

Dengan pemilihan algoritma pengurutan data yang memiliki kompleksitas algoritma yang baik akan menyebabkan kompleksitas algoritma program secara keseluruhan menjadi lebih baik.

Namun nilai kompleksitas algoritma terbaik dari program pembuatan pohon Huffman code sederhana secara keseluruhan ini adalah O(n) dan nilai terburuk yang mungkin didapatkan adalah $O(n^2)$, semua nilai ini didasarkan pada pemilihan algoritma pengurutan data yang akan dipakai dalam program.

Sehingga dapat disimpulkan secara keseluruhan, kompleksitas algoritma dari program pembentukan pohon Huffman code sederhana sangat dipengaruhi oleh pengambilan keputusan pada jenis algoritma pengurutan data yang akan diimplementasikan pada program tersebut.

Daftar Pustaka

[1] Munir, Rinaldi. (2006). Matematika Diskrit, Edisi Ketiga. Penerbit Informatika. Bandung.

[2] D. A. Huffman. (1952). *A Method for the Construction of Minimum were moved from each of the subclasses to the HuffmanNode Redundancy Codes*. Proc. IRE, 40(9).

[3]Wikipedia,http://en.wikipedia.org/wiki/Tree_%28graph_theory%29, waktu akses : 2 Januari 2008, pukul 11.00