Pemampatan Data dengan Pengkodean Huffman Dinamis

Mohammad Satrio Utomo[1], Azman Nurgozali[2], Julian Sukmana Putra[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if14134@students.if.itb.ac.id[1], if141422@students.if.itb.ac.id[2],
if14143@students.if.itb.ac.id[3]

Abstrak

Pengkodean Huffman banyak dipakai untuk memampatkan data. Namun, pengkodean Huffman yang biasa mempunyai dua kekurangan utama, yaitu keharusan untuk membaca berkas yang akan dimampatkan dua kali dan penyimpanan pohon Huffman pada berkas hasil kompresi. Perbaikan algoritma Huffman tersebut telah dilakukan, dan menghasilkan algoritma Pengkodean Huffman Dinamis. Dengan algoritma ini, waktu pemampatan data dapat dilakukan lebih cepat daripada waktu pengkodean Huffman biasa. Bahkan, algoritma Pengkodean Huffman Dinamis yang terbaru dapat menghasilkan panjang kode bit yang lebih pendek daripada kode Huffman biasa.

***Kata kunci**: algoritma, pemampatan data, waktu, Pengkodean Huffman Dinamis*

1. Pendahuluan

Pengkodean suatu berkas menjadi ukuran yang lebih kecil (pemampatan) merupakan bagian yang penting dalam bidang penyimpanan dan komunikasi data. Pengkodean Huffman merupakan salah satu algoritma pemampatan yang banyak digunakan. Namun, pengkodean Huffman yang biasa (kadang disebut pengkodean Huffman statis) memiliki beberapa kekurangan. Pertama, kita harus membaca berkas masukan yang akan dimampatkan dua kali, yaitu untuk mengetahui frekuensi setiap karakter yang muncul dan melakukan pengkodean untuk setiap karakter tersebut. Kedua, pohon yang dibuat untuk pengkodean harus turut disertakan pada berkas hasil pemampatan, sehingga hal ini memeperbesar ukuran berkas hasil pemampatan dan menurunkan rasio pemampatan.

Pengkodean Huffman Dinamis (*Dynamic Huffman Coding*) merupakan algoritma pengkodean yang dikembangkan dari pengkodean Huffman statis dengan memperbaiki kekurangan-kekurangan di atas. Perbaikan tersebut dilakukan dengan membuat pohon Huffman yang digunakan untuk pengkodean secara dinamis, tergantung karakter yang dibaca dari berkas masukan. Bobot tiap-tiap simpul dalam pohon diperbaharui pada setiap pembacaan satu karakter. Dapat dikatakan, pohon Huffman yang dibuat beradaptasi dengan karakter yang dibaca, sehingga algortima ini disebut juga Pengkodean Huffman Adaptif (*Adaptive Huffman Coding*). Secara keseluruhan, waktu yang digunakan dalam proses pengkodean dapat lebih singkat dibandingkan waktu yang dibutuhkan oleh algoritma Huffman biasa. Bahkan, algoritma pengkodean Huffman dinamis tertentu dapat menghasilkan kode bit yang lebih pendek daripada hasil yang didapat dari pengkodean Huffman statis [1].

Makalah ini mengeksplorasi kelebihan algoritma Pengkodean Huffman Dinamis dibandingkan dengan algoritma Huffman statis.

2. Pengkodean Huffman Dinamis

Pengkodean Huffman Dinamis, atau disebut juga Pengkodean Huffman Adaptif, pertama kali disusun oleh Faller dan Gallager. Kemudian, D. E. Knuth melakukan peningkatan terhadap algoritma tersebut dan menghasilkan algoritma baru yang disebut Algoritma FGK. Versi terbaru dari Pengkodean Huffman Dinamis dideskripsikan oleh Vitter pada tahun 1987, yang disebut sebagai Algoritma V.

Dua kekurangan utama dari pengkodean Huffman statis diatasi dengan baik oleh algoritma Pengkodean Huffman Dinamis. Hal ini dilakukan dengan melakukan penghitungan frekuensi karakter yang muncul dan pengkodean karakter tersebut dalam satu kali *pass*. Pohon Huffman yang dibangun selama pengkodean tersebut bersifat dinamis, yaitu terus berubah setiap pembacaan karakter dari teks sumber. Pohon Huffman saat ini adalah pohon Huffman yang berhubungan dengan bagian dari teks yang sudah dibaca, sehingga kode dinamis yang digunakan untuk memroses huruf ke t + 1 dalam teks adalah kode Huffman berdasarkan t huruf pertama dalam teks. Dengan demikian, pengkodean dapat lansung dilakukan tanpa perlu mengetahui frekuensi seluruh karakter dalam teks. Efisiensi dari metode ini berdasarkan sebuah karakterisasi dari pohon Huffman, yang dikenal dengan nama *siblings property*.

***Siblings Property*:** Jika T adalah sebuah pohon Huffman dengan n daun, maka simpul-simpul dari T dapat disusun dalam sebuah rangkaian ($x_0, x_1,...,x_{2n-2}$) sehingga:

1. rangkaian bobot, (*bobot*(x_0), *bobot*(x_1), ..., *bobot*(x_{2n-2})) tersusun secara menurun
2. untuk setiap i ($0 \leq i \leq n-2$), simpul yang berturutan x_{2i+1} dan x_{2i+2} adalah *siblings* (saudara), karena memiliki *parent* yang sama.

3. Proses *Encoding* dan *Decoding*

Proses *encoding* dan *decoding* dalam Pengkodean Huffman Dinamis menginisialisasi pohon Huffman dengan sebuah pohon bersimpul tunggal yang berkorespondensi dengan sebuah karakter artifisial, ditunjukkan dengan simbol `ART`. Bobot dari simpul tunggal ini adalah 1.

3. 1 Proses *Encoding*

Setiap pembacaan simbol *a* dari teks sumber, *codeword*-nya dalam pohon dikirimkan. Namun, hal ini hanya dilakukan jika *a* telah muncul sebelumnya. Jika tidak, kode dari ART dikirimkan diikuti oleh *codeword* asli dari *a*. Kemudian, pohon tersebut dimodifikasi sebagai berikut:
Jika *a* belum pernah muncul sebelumnya, sebuah simpul internal dibuat dan kedua simpul anaknya berisi *a* dan ART. Kemudian, pohon tersebut diperbaharui untuk mendapat pohon Huffman dari teks yang sudah dibaca.

Algoritma *encoding* di atas ditunjukkan pada *pseudocode* di bawah ini (kode ini dan kode-kode selanjutnya diadaptasi dari [3]):

```
DynamicHuffmanEncoding(fin, fout)
   DynamicTreeInit()
   while not EOF(fin) and nextSymbol(a) do
      EncodeSymbol(a,fout)
      UpdateDynamicTree()
   DynamicHuffmanEncodeSymbol(END,fout)


EncodeSymbol(a,fout)
   S ← stack kosong
   n ← daun(a)
   if n = NULL then
      n ← daun(ART)
   while n ≠ akar do
      if ganjil(n) then
         push(S,1)
      else
         push(S,0)
      n ← parent(n)
   send(S,fout)
   if (leaf(a) = NULL) then
      print(code(a))
      AddNode(a)


AddNode(a)
   node ← leaf(ART)
bobot(node,1)
left(node) ← leaf(a)
right(node) ← leaf(ART)
    bobot(left(node),0)
    bobot(right(node),1)
```

3. 1 Proses *Decoding*

Pada waktu *decoding*, teks hasil pemampatan di-*parse* dengan menggunakan pohon pengkodean. Simpul saat ini diinisialisasi oleh akar seperti algoritma *encoding*, kemudian pohon tersebut tumbuh secara simetris. Setiap sebuah 0 dibaca dari berkas hasil kompresi, pohon tumbuh mengikuti ke kiri. Jika 1 yang dibaca, pohon tumbuh ke kanan. Ketika simpul terkini adalah daun, simpul tersebut terasosiasi dengan simbol yang ditulis di output file dan pohon diperbaharui sehingga menjadi persis sama seperti seperti pohon pada proses decoding.

```
DynamicHuffmanDecoding(fin,fout)
   DynamicTreeInit()
   a ← DecodeSymbol(fin)
while a ≠ END do
   UpdateDynamicTree(a)
   a ← DecodeSymbol(fin)


DecodeSymbol(fin)
   n ← akar
   while child(n) ≠ NULL do
      read(b,fin)
      n ← child(n) + b
   a ← label(n)
   if a = ART then
       a ← next(fin,9) {simbol yang
                        berkorespondensi dengan
                        9 bit selanjutnya yang
                        dibaca dari berkas fin}
       AddNode(a)
   return (a)
```

3.3 Proses *Updating*

Selama proses *encoding* dan *decoding*, pohon sementara harus diperbaharui untuk mendapatkan frekuensi simbol yang benar. Ketika karakter baru diketahui bobotnya, bobot daun yang berasosiasi dengan karakter tersebut beratambah satu, dan bobot dari simpul-simpul di atasnya dimodifikasi. Kebenaran pohon yang terbentuk itu, dilihat berdasarkan *siblings property* pohon tersebut.

Proses *update* bekerja sebagai berikut:
Pertama, bobot dari daun *n* yang berkorespondesi dengan *a* ditambah 1. Lalu, jika *siblings property*-nya tidak sesuai lagi, maka simpul *n* ditukar dengan simpul terdekat, *m* ($m < n$) sehingga bobot(m) < bobot(n). Oleh karena itu, simpul-simpul itu tersusun menurun sesuai bobotnya.

Algoritma *updating* di atas ditunjukkan pada *pseudocode* di bawah ini:

```
DynamicHuffmanUpdating(a)
   n ← daun(a)
   while n ≠ akar do
```

```
    bobot(n) ← bobot(n) + 1
    m ← n
    while bobot(m − 1) < bobot(n) do
        m ← m − 1
    tukarSimpul(m,n)
    n ← parent(m)
  bobot(akar) ← bobot(akar) + 1
```

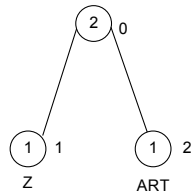
4. Langkah-Langkah Pengkodean

Langkah-langkah pengkodean pada algoritma Pengkodean Huffman Dinamis adalah sebagai berikut:

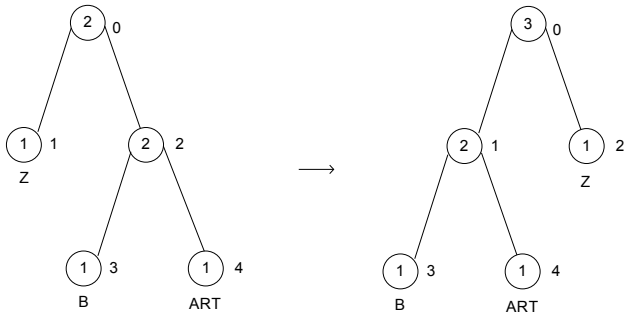
y = ZBZH {teks masukan}

inisialisasi pohon:

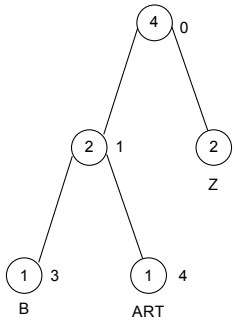
next = Z {simbol selanjutnya Z}



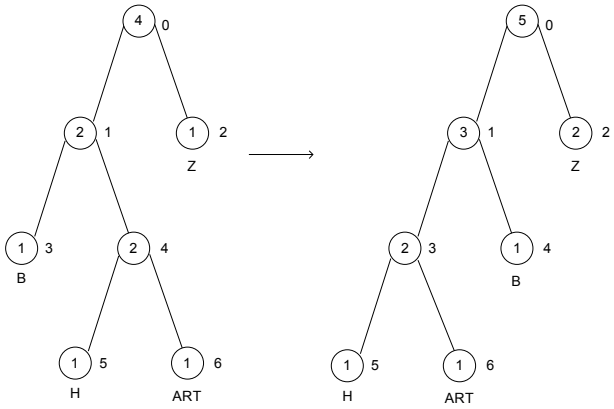
next = B
tukarSimpul(1,2)



next = Z



next = h
tukarSimpul(3,4)



Dari hasil pengkodean di atas, panjang kode untuk tiap karakter tidak lebih sedikit daripada yang dihasilkan oleh algoritma Huffman statis. Hal ini disebabkan teks yang dibaca tidak terlalu panjang. Namun dari proses pengkodeannya, jelas waktu yang dibutuhkan lebih kecil, karena pembacaan berkas hanya dilakukan satu kali.

5. Kesimpulan

Pengkodean Huffman Dinamis dapat mempercepat proses pemampatan data daripada pengkodean Huffman statis. Untuk teks yang pendek, panjang kode yang dihasilkan untuk setiap karakter tidak lebih baik daripada kode Huffman. Untuk itu, diperlukan cara lain untuk menghasilkan kode yang lebih pendek daripada kode Huffman. Contohnya, pada Algoritma V, jumlah bit yang dihasilkan dapat lebih pendek daripada kode Huffman.

6. Daftar Pustaka

1. Jeffrey Scott Vitter, *Design and Analysis of Dynamic Huffman Codes*, Journal of the Association for Computing Machinery, Vol. 34, No. 4, 1987.

2. Debra A. Lelewer dan Daniel S. Hirschberg, *Data Compression*.

3. *Handbook of Algorithms and Theory of Computation*, Chapter 10, Text data compression algorithms.