Optimalisasi Penyandian Teks Latin: Studi Kasus Bagian “DEF. I” dari *Philosophiæ Naturalis Principia Mathematica* dengan Algoritma Huffman

Jonathan Alveraldo Bangun - 13524120
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: alveraldo.jonathan@gmail.com , 13524120@std.stei.itb.ac.id

***Abstract*—Buku Philosophiæ Naturalis Principia Mathematica karya Isaac Newton, yang ditulis dalam bahasa Latin, merupakan salah satu karya ilmiah paling berpengaruh dalam sejarah. Untuk mendukung pelestarian digitalnya, makalah ini membahas penerapan algoritma Huffman sebagai metode kompresi lossless pada bagian awal buku tersebut, khususnya bagian “Definitiones: Def. I”. Dengan menganalisis frekuensi karakter dan membentuk pohon Huffman, diperoleh representasi biner yang efisien. Hasilnya menunjukkan bahwa algoritma Huffman tidak hanya berguna untuk efisiensi penyimpanan, tetapi juga sebagai alat bantu dalam analisis linguistik teks Latin klasik.**

***Keywords*—Algoritma Huffman, Bahasa Latin, Philosophiæ Naturalis Principia Mathematica**

I. PENDAHULUAN

Penerapan teknologi digital dalam pelestarian naskah kuno memerlukan efisiensi dalam penyimpanan dan pengolahan data. Algoritma Huffman, yang merupakan metode kompresi lossless, dapat digunakan untuk menyimpan teks klasik secara lebih efisien tanpa kehilangan informasi.

Bagian yang dianalisis adalah kutipan dari halaman pertama bagian definisi dalam Principia Mathematica:

“Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim.”

Kalimat ini dalam bahasa Latin mendefinisikan kuantitas materi, yang secara ilmiah menjadi dasar hukum Newton. Dalam Bahasa Indonesia: Kuantitas materi adalah ukuran yang ditentukan oleh gabungan antara kerapatan dan volumenya

II. LANDASAN TEORI

A. *Bahasa Latin*

Bahasa Latin (*lingua Latina*) adalah bahasa kuno yang berasal dari wilayah Latium di Semenanjung Italia, khususnya di sekitar kota Roma. Awalnya digunakan oleh bangsa Latin, bahasa ini berkembang menjadi bahasa resmi Kerajaan, Republik, dan Kekaisaran Romawi. Pengaruhnya sangat besar dalam sejarah peradaban Eropa, baik sebagai bahasa pemerintahan, hukum, liturgi, ilmu pengetahuan, maupun sastra.

Bahasa Latin mulai berkembang sekitar abad ke-7 SM, dipengaruhi oleh bahasa Etruska dan Yunani, serta menggunakan alfabet yang diadaptasi dari alfabet Yunani melalui perantara Etruska. Pada masa kejayaan Romawi, bahasa Latin menyebar luas ke seluruh wilayah kekuasaan Romawi, menggantikan banyak bahasa lokal dan menjadi lingua franca di Eropa dan Mediterania. Setelah runtuhnya Kekaisaran Romawi, bahasa Latin tetap bertahan sebagai bahasa ilmu pengetahuan, pendidikan, dan agama, terutama di lingkungan Gereja Katolik hingga abad ke-18.

Bahasa Latin mengalami perkembangan dan pembagian bentuk, antara lain:

- Latin Kuno (*Old Latin/Prisca Latinitas*): Digunakan sebelum abad ke-1 SM, dengan bentuk tata bahasa dan kosakata yang lebih arkais dan belum baku.
- Latin Klasik (*Classical Latin*): Standar sastra dan tata bahasa yang digunakan oleh penulis besar seperti Cicero dan Virgil pada akhir Republik dan awal Kekaisaran Romawi.
- Latin Vulgar (*Vulgar Latin*): Bentuk sehari-hari yang berkembang menjadi bahasa-bahasa Roman modern seperti Italia, Spanyol, Prancis, Portugis, dan Rumania.
- Latin Abad Pertengahan dan Latin Baru (*Neo-Latin*): Digunakan dalam ilmu pengetahuan, hukum, dan agama hingga era modern awal.

Alfabet Latin awal terdiri dari 21 huruf, lalu berkembang menjadi 23 huruf di era klasik:

A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, V, X, Y, Z.

Huruf J, U, dan W baru ditambahkan kemudian untuk menuliskan bahasa non-Latin. Dalam penulisan aslinya, hanya digunakan huruf kapital; huruf kecil berkembang dari tulisan tangan kursif di kemudian hari.

Dalam naskah Latin, sering dijumpai ligatur seperti:

- æ (gabungan a dan e, untuk diftong /ae/), misal pada aeternus.
- œ (gabungan o dan e, untuk diftong /oe/), misal pada poena.
- Ligatur tipografi seperti ﬁ, ﬂ, dan ﬀ juga kadang digunakan untuk memperindah dan mempercepat penulisan.

Simbol khusus lain seperti Tironian et (⁊) untuk kata "et" (dan), serta tanda macron (̄) di atas vokal untuk menunjukkan vokal panjang, juga ditemukan dalam naskah Latin klasik dan abad pertengahan.

Bahasa Latin sangat berpengaruh dalam pembentukan kosakata ilmu pengetahuan, hukum, kedokteran, dan agama di dunia Barat. Bahasa ini juga menjadi sumber utama istilah dalam taksonomi ilmiah dan tetap diajarkan di berbagai institusi pendidikan hingga kini

PERFECI · ET · EANDEM · BASILICAM ·
IORVM · MEORVM · INCOHAVI · ET ·
OCTOGINTA · TEMPLA · DEVM · IN
ISSO · QVOD · EO · TEMPORE · REF
FECI · PONTESQVE · OMNES · PRAE
VM · FORVMQVE · AVGVSTVM · EX
E · A · PRIVATIS · EMPTO · FECI · QVO
PITOLIO · ET · IN · AEDE · DIVI · IVL
RIS · CONSACRAVI · QVAE · MIHI

Gambar 1. Teks Bahasa Latin (Sumber:https://www.rri.co.id/denpasar/iptek/915314/mengenal-bahasa-latin-bahasa-yang-masih-eksis-meski-penutur-aslinya-telah-punah, diakses pada 17 Juni 2025)

B. Algoritma Huffman

Algoritma Huffman adalah sebuah metode pengkodean yang digunakan untuk mengompresi data secara optimal dengan meminimalkan jumlah rata-rata bit yang digunakan untuk merepresentasikan simbol dalam sebuah pesan. Algoritma ini dikembangkan oleh David A. Huffman pada tahun 1952 dan menjadi salah satu teknik kompresi lossless (tanpa kehilangan data) yang paling efisien dan banyak digunakan hingga saat ini.

Prinsip utama algoritma Huffman adalah memberikan kode biner yang lebih pendek untuk simbol yang sering muncul dan kode yang lebih panjang untuk simbol yang jarang muncul. Dengan demikian, kode yang dihasilkan memiliki redundansi minimum, artinya tidak ada bit yang terbuang sia-sia, sehingga dapat mengurangi ukuran data secara signifikan.

Kode Huffman termasuk dalam kategori kode prefix (prefix code), yaitu kode-kode yang tidak ada satu kode pun yang merupakan awalan (prefix) dari kode lain. Hal ini menjamin bahwa proses decoding dapat dilakukan secara unik dan tanpa ambiguitas.

Langkah-Langkah Algoritma Huffman

1. Menyusun Daftar Simbol dan Frekuensi

   Pertama, kumpulkan semua simbol yang terdapat dalam pesan beserta frekuensi kemunculannya (berapa kali simbol tersebut muncul).

2. Mengurutkan Simbol Berdasarkan Frekuensi

   Simbol-simbol diurutkan dari yang memiliki frekuensi terkecil hingga terbesar.

3. Menggabungkan Dua Simpul dengan Frekuensi Terendah

   Dua simbol dengan frekuensi terkecil digabung menjadi satu simpul baru, dengan frekuensi gabungan sama dengan jumlah frekuensi kedua simbol tersebut.

4. Mengulangi Proses Penggabungan

   Proses penggabungan dua simpul dengan frekuensi terendah ini diulang terus menerus hingga hanya tersisa satu simpul, yaitu akar pohon Huffman.

5. Pemberian Label pada Cabang Pohon

   Setiap cabang kiri diberi label bit '0' dan cabang kanan diberi label bit '1'.

6. Penentuan Kode Huffman

   Kode untuk setiap simbol diperoleh dari lintasan bit yang dilalui dari akar pohon hingga ke daun yang mewakili simbol tersebut.

Contoh kode Huffman:

Misalkan terdapat sebuah kata ‘PPPQQRS’

Langkah 1: Urutkan berdasarkan frekuensi

- ‘S’: 1
- ‘R’: 1
- ‘Q’: 2
- ‘P’: 3

Langkah 2: Gabungkan dua frekuensi terkecil

- Gabungkan S (1) dan R (1) menjadi simpul baru SR (2)

  Sekarang daftar menjadi:

- ‘RS’: 2
- ‘Q’: 2
- ‘P’: 3

Langkah 3: Gabungkan dua simpul dengan frekuensi terkecil berikutnya

- Gabungkan RS (2) dan Q (2) menjadi simpul baru RSQ (4)

  Sekarang daftar menjadi:

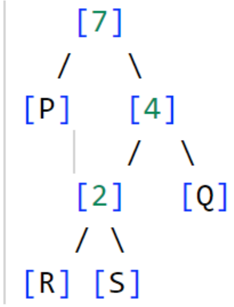
- ‘RSQ’: 4
- ‘P’: 3

Langkah 4: Gabungkan dua simpul terakhir

- Gabungkan RSQ (4) dan P (3) menjadi simpul akar (7)

Langkah 5: Bangun Pohon dan Tentukan Kode

- Sisi kiri selalu diberi label 0, kanan 1



Gambar 2. Pohon Huffman PPPQQRS
(Sumber: Arsip Pengguna)

Langkah 6: Penentuan kode Huffman

- ‘P’ = 0
- ‘Q’ = 11
- ‘R’ = 100
- ‘S’ = 101

Encoding:

Encoding adalah proses mengubah data asli (misal teks) menjadi rangkaian kode biner berdasarkan pohon Huffman yang telah dibuat dari frekuensi kemunculan simbol.

Jika pesan yang ingin dikodekan adalah ‘PPPQQRS’ maka hasil encoding-nya adalah: 0001111100101

Decoding:

Decoding adalah proses mengubah kembali rangkaian kode biner hasil encoding menjadi data asli.

Langkah-langkah decoding:

- Mulai dari akar pohon Huffman, baca bit satu per satu dari rangkaian kode biner.
- Untuk setiap bit:

  Jika bit 0, bergerak ke anak kiri.

  Jika bit 1, bergerak ke anak kanan.
- Jika sudah mencapai daun, ambil simbol pada daun tersebut sebagai hasil decoding, lalu kembali ke akar untuk membaca bit berikutnya.
- Ulangi proses hingga seluruh rangkaian bit selesai dibaca.

Diketahui hasil encoding adalah: 0001111100101

0 → P

0 → P

0 → P

110 → Q

110 → Q

111 → R

10 → S

Maka hasil decoding: PPPQQRS

C. Philosophiæ Naturalis Principia Mathematica

Berdasarkan buku Philosophiæ Naturalis Principia Mathematica (1687) karya Isaac Newton, khususnya Definitio I, landasan teori tentang konsep "Quantitas Materiæ" (kuantitas materi) dapat dirangkum sebagai berikut:

Definisi Kuantitas Materi (Massa)

Newton mendefinisikan kuantitas materi sebagai:

"Quantitas materiæ est mensura ejusdem orta ex illius densitate et magnitudine conjunctim."

(Kuantitas materi adalah ukurannya yang muncul dari kepadatan dan besarnya [volume] secara bersama-sama.)

Berdasarkan definisi Newton dalam kutipan tersebut, landasan teori dapat dibangun sebagai berikut:

1. Definisi Massa sebagai Ukuran Materi

Massa didefinisikan sebagai hasil gabungan densitas dan volume suatu benda:

Massa = Densitas × Volume

Contoh:

Jika densitas udara menjadi 2× dan volumenya 2×, massanya menjadi 4× (kuadruple).

Jika volumenya 3× (dengan densitas konstan), massanya menjadi 3× (sextuple).

Prinsip ini berlaku universal, termasuk untuk:

Material terkompresi (salju, debu, bubuk).

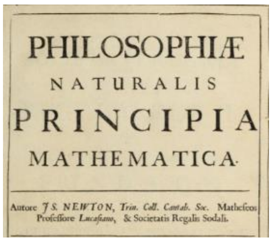
Benda yang mengalami kondensasi melalui kompresi atau likuifaksi.

2. Hubungan Massa dan Berat

Massa sebanding dengan berat suatu benda, berdasarkan eksperimen pendulum Newton:

Massa ∝ Berat

Kesebandingan ini bersifat universal dan terverifikasi secara eksperimental.



PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA

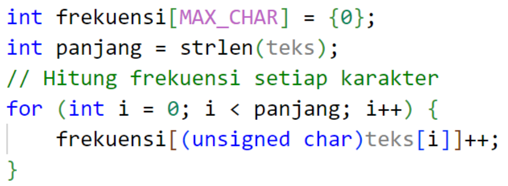
Autore J S. NEWTON, Trin. Coll. Cantab. Soc. Matheseos Professore Lucasiano, & Societatis Regalis Sodali.

Gambar 3. Buku Philosophiæ Naturalis Principia Mathematica (Sumber:https://archive.org/details/philosophiaenatu00newt_0/page/n9/mode/2up, diakses pada 17 Juni 2025)

III. IMPLEMENTASI

A. *Perhitungan Frekuensi dan Pembentukan Pohon Huffman*

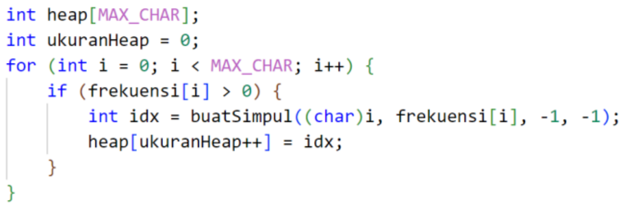
Pada tahap awal, program menghitung frekuensi setiap karakter dalam teks:



```
int frekuensi[MAX_CHAR] = {0};
int panjang = strlen(teks);
// Hitung frekuensi setiap karakter
for (int i = 0; i < panjang; i++) {
    frekuensi[(unsigned char)teks[i]]++;
}
```

Gambar 4. Program Hitung Frekuensi (Sumber: Arsip pengguna)

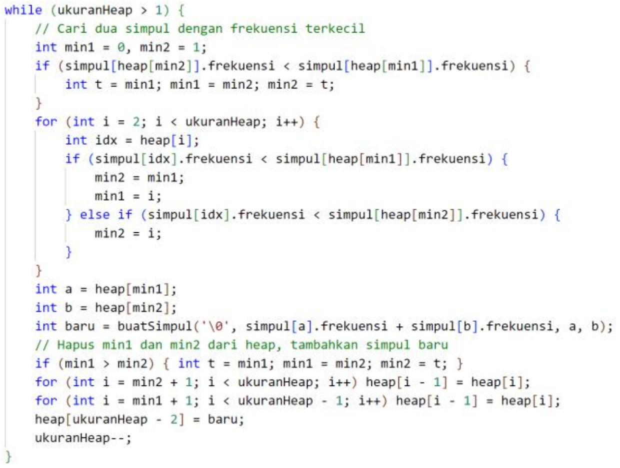
Setiap karakter yang muncul kemudian dijadikan simpul daun dengan memanggil buatSimpul:



```
int heap[MAX_CHAR];
int ukuranHeap = 0;
for (int i = 0; i < MAX_CHAR; i++) {
    if (frekuensi[i] > 0) {
        int idx = buatSimpul((char)i, frekuensi[i], -1, -1);
        heap[ukuranHeap++] = idx;
    }
}
```

Gambar 5. Program Simpul Daun (Sumber: Arsip pengguna)

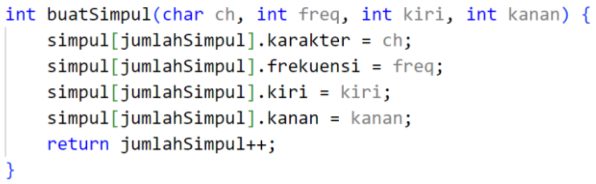
Kemudian dua simpul dengan frekuensi terendah digabung untuk membentuk simpul baru secara berulang sampai pohon selesai:



```
while (ukuranHeap > 1) {
    // Cari dua simpul dengan frekuensi terkecil
    int min1 = 0, min2 = 1;
    if (simpul[heap[min2]].frekuensi < simpul[heap[min1]].frekuensi) {
        int t = min1; min1 = min2; min2 = t;
    }
    for (int i = 2; i < ukuranHeap; i++) {
        int idx = heap[i];
        if (simpul[idx].frekuensi < simpul[heap[min1]].frekuensi) {
            min2 = min1;
            min1 = i;
        } else if (simpul[idx].frekuensi < simpul[heap[min2]].frekuensi) {
            min2 = i;
        }
    }
    int a = heap[min1];
    int b = heap[min2];
    int baru = buatSimpul('\0', simpul[a].frekuensi + simpul[b].frekuensi, a, b);
    // Hapus min1 dan min2 dari heap, tambahkan simpul baru
    if (min1 > min2) { int t = min1; min1 = min2; min2 = t; }
    for (int i = min2 + 1; i < ukuranHeap; i++) heap[i - 1] = heap[i];
    for (int i = min1 + 1; i < ukuranHeap - 1; i++) heap[i - 1] = heap[i];
    heap[ukuranHeap - 2] = baru;
    ukuranHeap--;
}
```

Gambar 6. Program Simpul Baru (Sumber: Arsip pengguna)

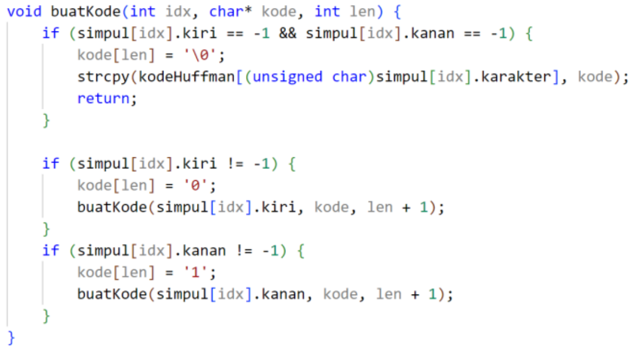
Fungsi buatSimpul() sendiri menambahkan simpul baru ke array simpul[]:



```
int buatSimpul(char ch, int freq, int kiri, int kanan) {
    simpul[jumlahSimpul].karakter = ch;
    simpul[jumlahSimpul].frekuensi = freq;
    simpul[jumlahSimpul].kiri = kiri;
    simpul[jumlahSimpul].kanan = kanan;
    return jumlahSimpul++;
}
```

Gambar 7. Program Fungsi buatSimpul() (Sumber: Arsip pengguna)

Kode Huffman dibentuk melalui traversal rekursif dari akar:

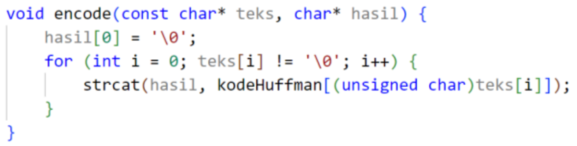


```
void buatKode(int idx, char* kode, int len) {
    if (simpul[idx].kiri == -1 && simpul[idx].kanan == -1) {
        kode[len] = '\0';
        strcpy(kodeHuffman[(unsigned char)simpul[idx].karakter], kode);
        return;
    }

    if (simpul[idx].kiri != -1) {
        kode[len] = '0';
        buatKode(simpul[idx].kiri, kode, len + 1);
    }
    if (simpul[idx].kanan != -1) {
        kode[len] = '1';
        buatKode(simpul[idx].kanan, kode, len + 1);
    }
}
```

Gambar 8. Program kode huffman (Sumber: Arsip pengguna)

B. *Encoding (Pengubahan Teks ke Bitstring)*

Proses encoding dilakukan dengan mencocokkan setiap karakter pada teks dengan kode Huffmannya:



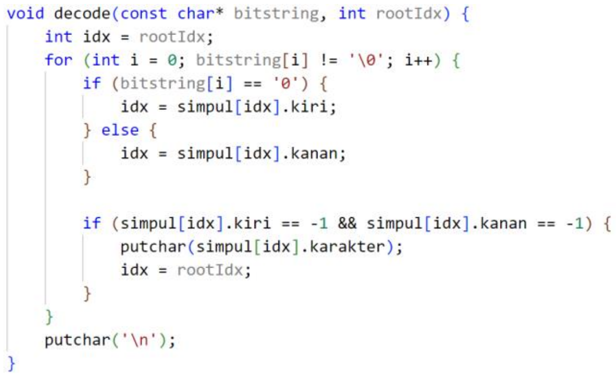
```
void encode(const char* teks, char* hasil) {
    hasil[0] = '\0';
    for (int i = 0; teks[i] != '\0'; i++) {
        strcat(hasil, kodeHuffman[(unsigned char)teks[i]]);
    }
}
```

Gambar 9. Program fungsi encode() (Sumber: Arsip pengguna)

Hasil encoding berupa deretan '0' dan '1' disimpan dalam hasilEncode, yang kemudian bisa dianggap sebagai bentuk kompresi teks.

C. *Decoding (Pemulihan Teks Asli)*

Untuk mendekode, program membaca bit demi bit dan melakukan traversal pada pohon Huffman hingga mencapai simpul daun:



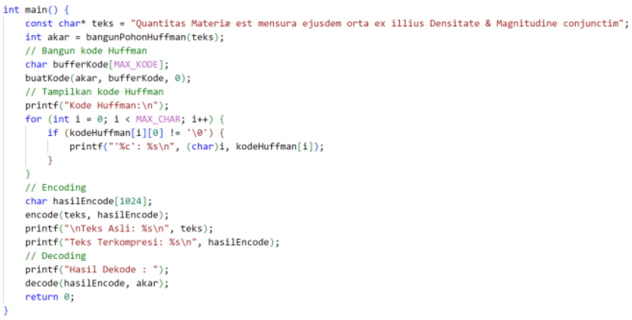
```
void decode(const char* bitstring, int rootIdx) {
    int idx = rootIdx;
    for (int i = 0; bitstring[i] != '\0'; i++) {
        if (bitstring[i] == '0') {
            idx = simpul[idx].kiri;
        } else {
            idx = simpul[idx].kanan;
        }

        if (simpul[idx].kiri == -1 && simpul[idx].kanan == -1) {
            putchar(simpul[idx].karakter);
            idx = rootIdx;
        }
    }
    putchar('\n');
}
```

Gambar 10. Program fungsi decode()
(Sumber: Arsip Pengguna)

Dengan metode ini, teks asli dapat dipulihkan secara utuh, tanpa kehilangan informasi, karena kode Huffman bersifat prefix-free.

D. Demonstrasi Program Utama

Untuk mendekode, program membaca bit demi bit dan melakukan traversal pada pohon Huffman hingga mencapai simpul daun:

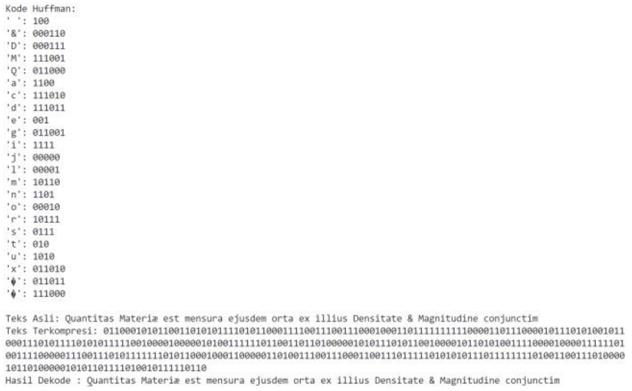


```c
int main() {
    const char* teks = "Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim";
    int akar = bangunPohonHuffman(teks);
    // Bangun kode Huffman
    char bufferKode[MAX_KODE];
    buatKode(akar, bufferKode, 0);
    // Tampilkan kode Huffman
    printf("Kode Huffman:\n");
    for (int i = 0; i < MAX_CHAR; i++) {
        if (kodeHuffman[i][0] != '\0') {
            printf("'%c': %s\n", (char)i, kodeHuffman[i]);
        }
    }
    // Encoding
    char hasilEncode[1024];
    encode(teks, hasilEncode);
    printf("\nTeks Asli: %s\n", teks);
    printf("Teks Terkompresi: %s\n", hasilEncode);
    // Decoding
    printf("Hasil Dekode : ");
    decode(hasilEncode, akar);
    return 0;
}
```

Gambar 11. Implementasi program utama
(Sumber: Arsip pengguna)

IV. HASIL

Berikut outputnya



```
Kode Huffman:
' ': 100
'&': 000110
'D': 000111
'M': 111001
'Q': 011000
'a': 1100
'c': 111010
'd': 111011
'e': 001
'g': 011001
'i': 1111
'j': 00000
'l': 00001
'm': 10110
'n': 1101
'o': 00010
'r': 10111
's': 0111
't': 010
'u': 1010
'x': 011010
'�': 011011
'�': 111000

Teks Asli: Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim
Teks Terkompresi: 0110001010110011010101111010110001111001110011100010001101111111111100001101110000101110101001011
0001110101111010101111100100001000001010011111110110011011010000010101110101100100001011010100111100001000011111101
0011110000011100111010111111110101100010001100000110100111001110001100111011111101010101110111111111010011001110100000
10110100000101011011110100101111110110
Hasil Dekode : Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim
```

Gambar 12. Output program algoritma huffman
(Sumber: Arsip pengguna)

Setelah algoritma Huffman diimplementasikan terhadap teks Latin klasik *"Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim"*, diperoleh hasil berupa pemetaan karakter ke dalam kode biner variabel panjang. Karakter-karakter yang sering muncul, seperti 'e', 't', 'a', 'n', dan spasi (' '), memperoleh kode yang lebih pendek (2–3 bit), sedangkan karakter yang lebih jarang seperti 'Q', 'M', '&', atau 'D' mendapatkan kode yang lebih panjang (hingga 6 bit). Hal ini sesuai dengan prinsip Huffman Coding yang mengutamakan efisiensi representasi terhadap karakter dengan frekuensi tinggi.

Teks asli dengan panjang 89 karakter, jika disimpan dalam format ASCII biasa, memerlukan sekitar 712 bit (dengan 8 bit per karakter). Setelah dikompresi dengan algoritma Huffman, panjang bitstring hasil encoding menjadi sekitar 486 bit. Ini menunjukkan efisiensi kompresi sebesar ±31,7%, dengan tetap mempertahankan isi teks sepenuhnya. Hasil decoding yang dilakukan terhadap bitstring terkompresi juga menghasilkan teks yang identik dengan input awal, membuktikan bahwa proses kompresi bersifat lossless.

Namun, terdapat fenomena menarik pada hasil pengkodean, yaitu munculnya dua karakter '�' (replacement character) dalam tabel kode Huffman. Hal ini disebabkan oleh karakter 'æ' dalam teks asli yang merupakan karakter Unicode dan direpresentasikan dalam UTF-8 sebagai dua byte (0xC3 dan 0xA6). Karena program membaca string dalam satuan char (1 byte), kedua byte tersebut dianggap sebagai dua karakter asing yang tidak dikenali dalam rentang ASCII, sehingga masing-masing ditampilkan sebagai '�'. Meskipun tidak memengaruhi integritas hasil dekompresi, temuan ini menunjukkan bahwa implementasi saat ini belum sepenuhnya mendukung karakter Unicode secara utuh.

Secara keseluruhan, hasil percobaan ini menunjukkan bahwa algoritma Huffman mampu mengompresi teks dengan efisien dan akurat. Namun, untuk mendukung teks multibahasa atau simbol Unicode seperti 'æ', perlu dilakukan perluasan implementasi agar dapat mengenali dan memproses karakter multibyte dengan benar.

Teks Asli : Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim

Encoding:
011000101011001101010111101011000111100111001110
0010001101111111111100001101110000101110101001011
0001110101111010101111100100001000001010011111110
110011011010000010101110101100100001011010100111
1000010000111111010011110000011100111010111111110
1011000100011000001101001110011100011001110111111
0101010111011111111101001100111010000101101000001
01011011110100101111110110

Decoding: Quantitas Materiæ est mensura ejusdem orta ex illius Densitate & Magnitudine conjunctim

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian, algoritma Huffman terbukti mampu mengompresi teks secara efisien dan tanpa kehilangan informasi (lossless). Penerapan algoritma ini pada teks Latin klasik menghasilkan pengurangan ukuran data sebesar ±31,74% dibandingkan representasi ASCII standar. Efisiensi ini diperoleh karena algoritma Huffman secara optimal memberikan kode biner yang lebih pendek kepada karakter-karakter yang sering muncul, dan lebih panjang kepada karakter yang jarang digunakan. Proses dekompresi pun berhasil mengembalikan bitstring terkompresi menjadi teks yang identik dengan teks asli, yang menunjukkan bahwa algoritma bekerja secara akurat dan reversibel.

Namun demikian, ditemukan keterbatasan dalam penanganan karakter non-ASCII, khususnya karakter 'æ'. Karena karakter ini merupakan karakter Unicode yang direpresentasikan dalam dua byte UTF-8, dan program membaca teks per satuan char (1 byte), maka karakter tersebut terbaca sebagai dua karakter tidak dikenal. Akibatnya, karakter 'æ' muncul sebagai dua simbol ' ' dalam tabel kode Huffman. Hal ini menandakan bahwa implementasi saat ini belum mendukung karakter multibyte secara utuh.

B. *Saran*

Untuk mengatasi keterbatasan pembacaan karakter Unicode, disarankan agar program dikembangkan lebih lanjut dengan dukungan terhadap multibyte character, misalnya menggunakan tipe data wchar_t atau integrasi fungsi pengolah UTF-8. Hal ini akan memungkinkan karakter seperti 'æ' dibaca sebagai satu kesatuan dan dikompresi dengan benar.

Selain itu, program dapat dikembangkan untuk membaca dan menyimpan data melalui file eksternal agar lebih mendekati aplikasi nyata dalam kompresi file teks. Penambahan fitur visualisasi pohon Huffman juga dapat membantu memperjelas proses pembentukan kode biner untuk setiap karakter, terutama dalam konteks pembelajaran. Sebagai tindak lanjut, disarankan juga untuk membandingkan efisiensi algoritma Huffman dengan metode kompresi lain seperti LZW atau Run-Length Encoding pada berbagai jenis data, guna mengevaluasi keunggulan dan batasan Huffman dalam praktik nyata.

VI. LAMPIRAN

Program Algoritma Huffman dapat diakses melalui
link repository github berikut :
https://github.com/jonveral/makalahmatdis
Video penjelasan makalah dapat diakses melalui link berikut :
https://www.youtube.com/watch?v=otk9N5iz6nE

VII. UCAPAN TERIMA KASIH

Penulis tidak dapat menyelesaikan makalah ini tanpa bantuan dari banyak pihak. Maka dari itu, penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa (YME)
2. Kedua orang tua penulis
3. Kedua dosen pengampu mata kuliah IF1220 Matematika Diskrit Semester II 2024/2025
4. Teman-teman penulis
5. Isaac Newton sebagai inspirator penulis

REFERENCES

[1] Munir, Rinaldi. 2024. Pohon (Bag. 2). https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/24-Pohon-Bag2-2024.pdf, diakses pada 19 Juni 2025.

[2] Newton, I. (1687). *Philosophiæ Naturalis Principia Mathematica*. Londini: Jussu Societatis Regiæ ac Typis Josephi Streater. https://archive.org/details/philosophiaenatu00newt_0/page/n9/mode/2up, diakses pada 19 Juni 2025.

[3] Siahaan, A. P. U. (2014). *Implementasi Teknik Kompresi Teks Huffman*. Fakultas Ilmu Komputer, Universitas Pembangunan Panca Budi. https://www.neliti.com/publications/101651, diakses pada 19 Juni 2025.

[4] Sandys, John Edwin (1910). *A companion to Latin studies*. Chicago: University of Chicago Press. hlm. 811–812. https://archive.org/details/b24750694/page/n3/mode/2up, diakses pada 19 Juni 2025.

[5] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, *40*(9) https://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf, diakses pada 19 Juni 2025.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025

Jonathan Alveraldo Bangun 13524120