A. Penjelasan Algoritma Huffman Trees and Codes

Misalkan kita harus mengkodekan teks yang terdiri dari simbol-simbol dari suatu alfabet dengan n simbol dengan menugaskan untuk masing-masing simbol teks suatu urutan bit yang disebut sebagai codeword. Sebagai contoh, kita dapat menggunakan encoding dengan panjang tetap yang menugaskan pada setiap simbol sebuah string bit dengan panjang yang sama m ($m \geq \log_2 n$). Ini persis seperti yang dilakukan oleh kode ASCII standar. Salah satu cara untuk mendapatkan skema pengkodean yang menghasilkan string bit yang lebih pendek secara rata-rata didasarkan pada ide lama untuk menugaskan codeword yang lebih pendek untuk simbol yang lebih sering muncul dan codeword yang lebih panjang untuk simbol yang kurang sering muncul. Ide ini digunakan, khususnya, dalam kode telegraf yang ditemukan pada pertengahan abad ke-19 oleh Samuel Morse. Pada kode tersebut, huruf yang sering muncul seperti e (.) dan a (.−) diberikan urutan titik dan garis yang pendek sedangkan huruf yang jarang muncul seperti q (− − .−) dan z (− − ..) memiliki urutan yang lebih panjang.

Variable-length encoding, yang memberikan codeword dengan panjang yang berbeda untuk simbol-simbol yang berbeda, memperkenalkan masalah yang tidak dimiliki oleh fixed-length encoding. Yaitu, bagaimana kita dapat mengetahui berapa banyak bit dari teks yang di-encode mewakili simbol pertama (atau, secara umum, simbol ke-i)? Untuk menghindari komplikasi ini, kita dapat membatasi diri pada kode **prefix-free** (atau hanya kode **prefix**). Dalam kode prefix, tidak ada codeword yang merupakan awalan dari codeword dari simbol lain. Oleh karena itu, dengan encoding seperti ini, kita dapat dengan mudah memindai string bit sampai kita mendapatkan grup pertama bit yang merupakan codeword untuk suatu simbol, mengganti bit tersebut dengan simbol ini, dan mengulangi operasi ini sampai akhir string bit tercapai.

Jika kita ingin membuat kode prefix biner untuk beberapa alfabet, hal yang wajar adalah menghubungkan simbol-simbol alfabet dengan daun dari pohon biner di mana semua tepi kiri diberi label 0 dan semua tepi kanan diberi label 1. Codeword untuk suatu simbol kemudian dapat diperoleh dengan mencatat label pada jalur sederhana dari akar ke daun simbol. Karena tidak ada jalur sederhana ke daun yang terus ke daun lain, tidak ada codeword yang dapat menjadi awalan dari codeword lain; oleh karena itu, setiap pohon semacam ini menghasilkan kode prefix.

Di antara banyak pohon yang dapat dibangun dengan cara ini untuk suatu alfabet dengan frekuensi kejadian simbol yang diketahui, bagaimana kita dapat membuat pohon yang akan menetapkan bit string yang lebih pendek ke simbol-simbol yang lebih sering muncul dan bit string yang lebih panjang ke simbol-simbol yang kurang sering muncul? Ini dapat dilakukan dengan algoritma serakah berikut, yang diciptakan oleh David Huffman ketika dia masih mahasiswa pascasarjana di MIT [Huf52].

Algoritma Huffman

Langkah 1 Inisialisasi n pohon satu simpul dan label mereka dengan simbol-simbol alfabet yang diberikan. Catat frekuensi setiap simbol di akar pohonnya untuk menunjukkan berat pohon. (Secara umum, berat pohon akan sama dengan jumlah frekuensi di daun pohon.).

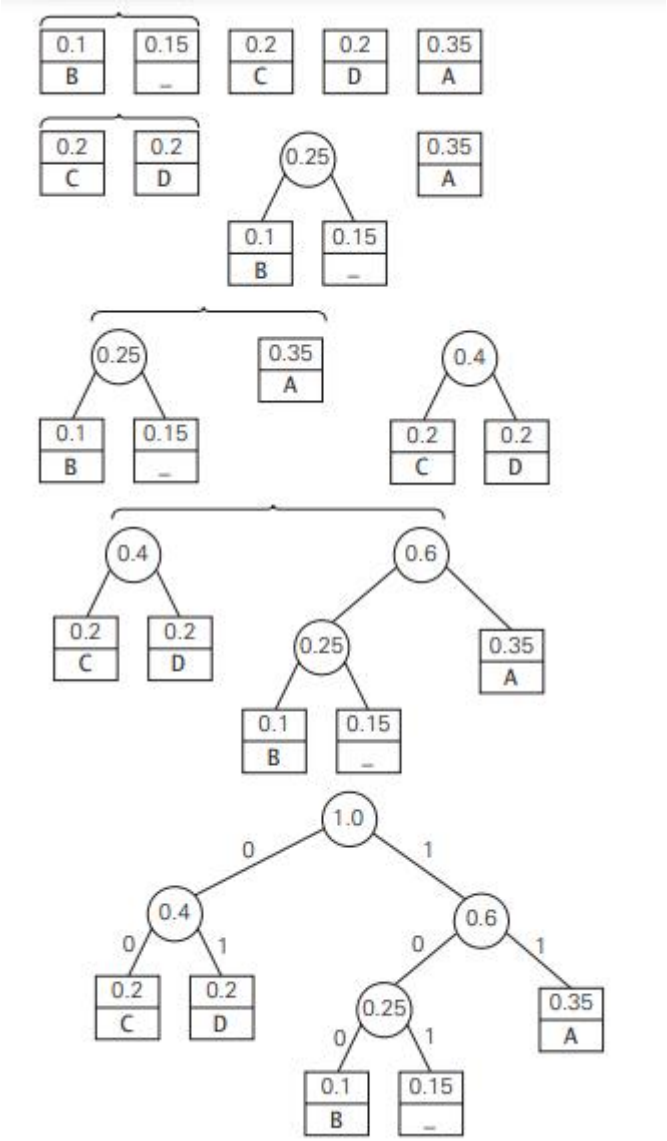
Langkah 2 Ulangi operasi berikut sampai satu pohon diperoleh. Temukan dua pohon dengan berat terkecil (ikatan dapat dipecah secara sembarangan, tetapi lihat Masalah 2 di latihan bagian ini). Buat mereka subtree kiri dan kanan dari pohon baru dan catat jumlah berat mereka di akar pohon baru sebagai beratnya.

Sebuah trees yang dibangun dengan algoritma di atas disebut pohon Huffman. Ini menentukan - seperti yang dijelaskan di atas - kode Huffman.

CONTOH Pertimbangkan alfabet lima simbol {A, B, C, D, _} dengan frekuensi kejadian berikut dalam teks yang terbuat dari simbol-simbol ini:

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

Gambar 9.12 Konstruksi pohon Huffman untuk input



Gambar 9.12 Contoh dari konstruksi Huffman Coding Tree.

Hasil codewords nya adalah :

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

B. Pseudocode Huffman Trees and Codes

Huffman coding adalah algoritma kompresi data lossless yang menetapkan kode panjang variabel untuk karakter input berdasarkan frekuensinya[1]. Berikut adalah contoh pseudocode untuk pohon dan kode Huffman dalam bahasa Python:

1. Buat tabel frekuensi karakter dalam teks input.
2. Buat antrian prioritas node, di mana setiap node berisi karakter dan frekuensinya.
3. Selama ada lebih dari satu node dalam antrian, lakukan hal berikut:
   - Hapus dua node dengan frekuensi terendah dari antrian.
   - Buat node internal baru dengan dua node ini sebagai anak dan dengan frekuensi yang sama dengan jumlah frekuensi mereka.
   - Tambahkan node baru ke antrian.
4. Node yang tersisa dalam antrian adalah node akar dari Huffman Tree
5. Telusuri Huffman Tree dari akar hingga daun, menetapkan 0 atau 1 ke setiap tepi tergantung pada apakah itu ke kiri atau kanan, masing-masing.
6. Kode biner untuk setiap karakter diperoleh dengan menggabungkan semua 0 dan 1 sepanjang jalurnya dari akar ke daun.

```
Buatlah antrian prioritas Q yang terdiri dari setiap karakter yang unik.
Urutkan karakter-karakter tersebut secara menaik berdasarkan frekuensinya.
Untuk setiap karakter yang unik:
    buatlah sebuah node baru
    ekstrak nilai minimum dari Q dan berikan ke leftChild dari newNode
    ekstrak nilai minimum dari Q dan berikan ke rightChild dari newNode
    hitung jumlah dari dua nilai minimum tersebut dan berikan ke value dari newNode
sisipkan newNode ke dalam pohon
kembalikan rootNode
```

C. Program Huffman Trees and Codes

Berikut adalah implementasi program Huffman coding dalam bahasa Python menggunakan kelas:

```
# Huffman Coding in python

string = 'BCAADDDCCACACAC'


# Creating tree nodes
class NodeTree(object):

    def __init__(self, left=None, right=None):
        self.left = left
        self.right = right
```

```
    def children(self):
        return (self.left, self.right)

    def nodes(self):
        return (self.left, self.right)

    def __str__(self):
        return '%s_%s' % (self.left, self.right)


# Main function implementing huffman coding
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.children()
    d = dict()
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString +
'1'))
    return d


# Calculating frequency
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1],
reverse=True)

nodes = freq

while len(nodes) > 1:
    (key1, c1) = nodes[-1]
    (key2, c2) = nodes[-2]
    nodes = nodes[:-2]
    node = NodeTree(key1, key2)
    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1],
reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])

print(' Char | Huffman code ')
print('----------------------')
for (char, frequency) in freq:
```

```
    print(' %-4r |%12s' % (char, huffmanCode[char]))
```

D. **Analisis Kebutuhan Waktu**

1. Analisis menyeluruh dengan memperhatikan operasi/instruksi yang di eksekusi berdasarkan operator penugasan atau assignment (=, +=, *=, dan lainnya) dan operator aritmatika (+, %, * dan lainnya)

Dalam algoritma Huffman Trees and Codes, terdapat beberapa operasi dan instruksi yang dijalankan untuk menghitung kode Huffman dan memperkecil jumlah bit yang digunakan untuk merepresentasikan data. Berikut adalah beberapa operasi dan instruksi yang penting dalam algoritma ini:

A..Operator penugasan (=)

Operator ini digunakan untuk menentukan nilai frekuensi setiap simbol dalam data. Misalnya, untuk setiap simbol dalam data, nilai frekuensi akan disimpan ke dalam sebuah variabel.

B. Operator aritmatika (+)

Operator ini digunakan untuk menjumlahkan frekuensi dua simpul saat simpul-simpul tersebut digabungkan untuk membentuk simpul baru. Misalnya, ketika dua simpul dengan frekuensi f1 dan f2 digabungkan, maka frekuensi simpul baru adalah f1 + f2.

C.Operator penugasan dan aritmatika gabungan (+=)

Operator ini digunakan untuk menambahkan frekuensi dua simpul saat simpul-simpul tersebut digabungkan untuk membentuk simpul baru. Misalnya, ketika dua simpul dengan frekuensi f1 dan f2 digabungkan, maka frekuensi simpul baru adalah f1 += f2.

D. Operator perbandingan (<)

Operator ini digunakan untuk membandingkan frekuensi dua simpul dalam heap, sehingga simpul-simpul dengan frekuensi terkecil dapat diambil untuk digabungkan.

E.Fungsi untuk memasukkan simpul ke dalam heap

Fungsi ini digunakan untuk memasukkan simpul ke dalam heap. Pada umumnya, algoritma menggunakan struktur heap minimum untuk mengambil simpul dengan frekuensi terkecil.

2.Analisis berdasarkan jumlah operasi abstrak atau operasi khas

Berikut adalah operasi abstrak atau operasi khas yang dilakukan oleh algoritma Huffman Trees and Codes:

A. Membangun tabel frekuensi: Setiap simbol dalam data dihitung berapa kali kemunculannya dalam data, sehingga membentuk tabel frekuensi. Jumlah operasi abstrak dalam tahap ini adalah O(n), di mana n adalah panjang data.

B. Membangun Huffman tree: Algoritma Huffman membangun pohon biner dengan menempatkan dua simbol dengan frekuensi terkecil di bawah satu node baru. Operasi ini diulang sampai semua simbol ada di dalam pohon. Jumlah operasi abstrak dalam tahap ini adalah O(n log n), di mana n adalah jumlah simbol.

C. Membangkitkan kode Huffman: Kode Huffman dibentuk dengan menelusuri pohon dan menambahkan 0 atau 1 ke kode saat bergerak ke kiri atau kanan, secara

berurutan dari akar ke daun. Jumlah operasi abstrak dalam tahap ini adalah O(n), di mana n adalah jumlah simbol.

D. Kompresi data: Data asli dikonversi ke kode Huffman, sehingga mengurangi jumlah bit yang diperlukan untuk merepresentasikan data. Jumlah operasi abstrak dalam tahap ini adalah O(n), di mana n adalah panjang data.

Jumlah operasi abstrak atau operasi khas tergantung pada ukuran data dan

Levitin, Anany. 2012. Introduction to the Design and Analysis of Algorithms.
Edisi ketiga. Boston, MA: Pearson Education, Inc.

Trees and Codes memiliki kompleksitas waktu O(n log n) untuk membangun Huffman tree dan O(n) untuk menghasilkan kode Huffman dan melakukan kompresi data.

3.Analisis menggunakan pendekatan best-case (kasus terbaik), worst-case (kasus terburuk), dan average-case (kasus rata-rata)

A.Best-case: Kasus terbaik terjadi ketika semua simbol dalam data memiliki frekuensi yang sama. Dalam kasus ini, Huffman tree yang dibangun adalah pohon biner yang seimbang, di mana kedalaman setiap node sama dan tidak ada operasi penyesuaian yang diperlukan. Jumlah operasi abstrak yang diperlukan dalam kasus terbaik adalah O(n log n), di mana n adalah jumlah simbol dalam data.

B.Worst-case: Kasus terburuk terjadi ketika semua simbol dalam data memiliki frekuensi yang berbeda-beda, sehingga membuat Huffman tree dengan tinggi maksimum. Dalam kasus ini, Huffman tree yang dibangun adalah pohon biner dengan satu cabang yang sangat panjang dan cabang lain yang sangat pendek. Operasi penyesuaian harus dilakukan untuk memperbaiki struktur Huffman tree, yang mengakibatkan jumlah operasi abstrak menjadi O(n^2). Namun, kasus terburuk pada algoritma Huffman jarang terjadi pada data dunia nyata.

C. Average-case: Kasus rata-rata pada algoritma Huffman tergantung pada distribusi frekuensi simbol dalam data. Jika distribusi frekuensi simbol mendekati distribusi normal atau mendekati distribusi power law, maka Huffman tree yang dihasilkan kemungkinan besar akan seimbang dan jumlah operasi abstrak yang diperlukan akan mendekati O(n log n). Namun, jika distribusi frekuensi simbol sangat tidak seimbang atau terdapat beberapa simbol dengan frekuensi yang sangat tinggi, maka Huffman tree yang dihasilkan akan lebih tinggi dan memerlukan lebih banyak operasi penyesuaian.

E.Referensi

Levitin, Anany. 2012. Introduction to the Design and Analysis of Algorithms.
Edisi ketiga. Boston, MA: Pearson Education, Inc.
https://www.programiz.com/dsa/huffman-coding .Diakses pada 9 Maret 2023
https://www.javatpoint.com/huffman-coding-using-python Diakses pada 9 Maret 2023
https://favtutor.com/blogs/huffman-coding. Diakses pada 9 Maret 2023

F.Link Github
https://github.com/Reviza2308/Huffman-Trees-and-Codes