МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА
ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНОЇ РОБОТИ № 9

з навчальної дисципліни
«Алгоритми та методи обчислень»

Тема «Стиснення даних. Кодування і декодування Гафмена»

Студент гр. КІ-24-1, Варакута О.О
Викладач, Сидоренко В. М

Кременчук 2025

Тема. Стиснення даних. Кодування і декодування Гафмена

Мета: набути практичних навичок застосування алгоритму оптимального кодування Гафмена.

Хід роботи

Варіант 5:

Завдання:Маємо текст AABAABABABABCBBCBCAAADEE. Закодувати текст, використовуючи алгоритм Гафмена. Побудувати двійкове дерево. Оцінити ефект від кодування у порівнянні з неоптимальним випадком.

Ручний розрахунок:

1Частоти символів:

Довжина тексту N = 24

A: 10
B: 8
C: 3
D: 1
E: 2

4.Побудова коду Гафмена (кроки об’єднання):

Беремо 2 найменші частоти і об’єднуємо:

1. D(1) + E(2) = 3 → вузол (D,E)
2. C(3) + (D,E)(3) = 6 → вузол (C,(D,E))
3. 6 + B(8) = 14 → вузол ((C,(D,E)),B)
4. A(10) + 14 = 24 → корінь (A,((C,(D,E)),B))
5. Двійкове дерево (0 — ліворуч, 1 — праворуч)

Корінь (24)
0 → A(10)
1 → (14)
0 → (6)
0 → C(3)
1 → (3)
0 → D(1)
1 → E(2)
1 → B(8)

5.Таблиця кодів (за деревом)

A = 0
B = 11
C = 100
D = 1010
E = 1011

5.Закодований текст

Початковий текст: AABAABABABABCBBCBCAAADEE

Бітовий рядок (47 біт):
00110011011011011100111110011100000101010111011

Перевірка довжини:
$10 \cdot 1 + 8 \cdot 2 + 3 \cdot 3 + 1 \cdot 4 + 2 \cdot 4$
$= 10 + 16 + 9 + 4 + 8$
= 47 біт

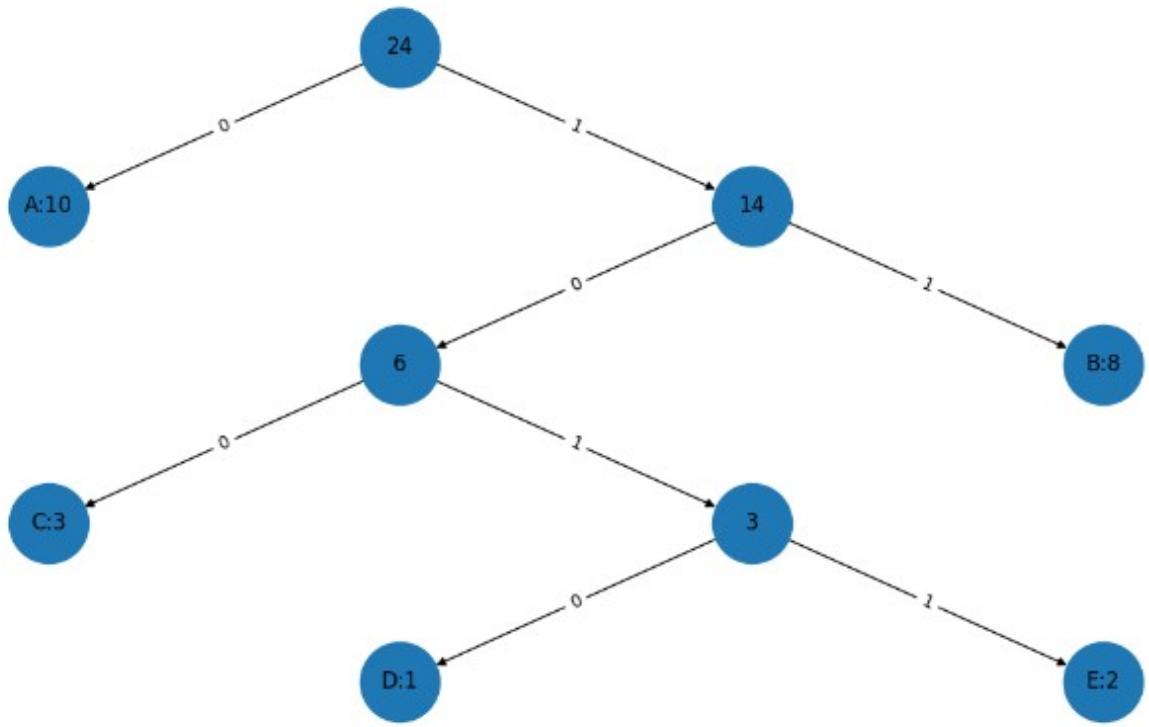
Середня довжина коду:
$47 / 24 \approx 1.958$ біт/символ

6.Оцінка ефекту порівняно з неоптимальним випадком

Неоптимальний (фіксована довжина для 5 символів):
потрібно ceil(log2(5)) = 3 біти/символ
Отже $24 \cdot 3 = 72$ біти

Порівняння:
Було 72 біти, стало 47 біт
Економія: $72 - 47 = 25$ біт
Відносне зменшення: $25 / 72 \approx 34.7\%$
Коефіцієнт стиснення: $72 / 47 \approx 1.53$ рази

Реалізація Python:




```
from collections import Counter
import heapq
import networkx as nx
import matplotlib.pyplot as plt

class Node:
    def __init__(self, freq, symbol=None, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right

    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(text):
    freq = Counter(text)
    heap = []

    for symbol, f in freq.items():
```

```
        heapq.heappush(heap, Node(f, symbol))

    while len(heap) > 1:
        n1 = heapq.heappop(heap)
        n2 = heapq.heappop(heap)
        heapq.heappush(heap, Node(n1.freq + n2.freq, None, n1, n2))

    return heap[0], freq

def generate_codes(node, prefix="", codes=None):
    if codes is None:
        codes = {}
    if node.symbol is not None:
        codes[node.symbol] = prefix
        return codes
    generate_codes(node.left, prefix + "0", codes)
    generate_codes(node.right, prefix + "1", codes)
    return codes

def encode_text(text, codes):
    return "".join(codes[ch] for ch in text)

def add_nodes_edges(graph, node, parent=None, edge_label="", x=0, y=0,
pos=None):
    if pos is None:
        pos = {}

    node_id = id(node)
    label = f"{node.symbol}:{node.freq}" if node.symbol else f"{node.freq}"

    graph.add_node(node_id, label=label)
    pos[node_id] = (x, y)

    if parent:
        graph.add_edge(parent, node_id, label=edge_label)

    if node.left:
        add_nodes_edges(graph, node.left, node_id, "0", x - 1, y - 1, pos)
    if node.right:
        add_nodes_edges(graph, node.right, node_id, "1", x + 1, y - 1, pos)

    return pos


def draw_huffman_tree(root):
```

```
    G = nx.DiGraph()
    pos = add_nodes_edges(G, root)

    labels = nx.get_node_attributes(G, "label")
    edge_labels = nx.get_edge_attributes(G, "label")

    plt.figure(figsize=(10, 6))
    nx.draw(G, pos, with_labels=False, node_size=2000)
    nx.draw_networkx_labels(G, pos, labels)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
    plt.title("Двійкове дерево Гафмена")
    plt.show()


def main():
    text = "AABAABABABABCBBCBCAAADEE"

    root, freq = build_huffman_tree(text)
    codes = generate_codes(root)
    encoded = encode_text(text, codes)

    print("Частоти:")
    for k, v in freq.items():
        print(k, ":", v)

    print("\nКоди Гафмена:")
    for k, v in codes.items():
        print(k, "=", v)

    print("\nЗакодований текст:")
    print(encoded)
    print("Довжина:", len(encoded), "біт")

    fixed_bits = len(text) * 3
    print("\nНеоптимальне кодування:", fixed_bits, "біт")
    print("Економія:", fixed_bits - len(encoded), "біт")
    print("Коефіцієнт стиснення:", round(fixed_bits / len(encoded), 2))

    draw_huffman_tree(root)


main()
```

Виведення результатів роботи алгоритма :
Частоти:

A : 10
B : 8
C : 3
D : 1
E : 2

Коди Гафмена:
A = 0
C = 100
D = 1010
E = 1011
B = 11

Закодований текст:
00110011011011011100111110011100000101010111011
Довжина: 47 біт

Неоптимальне кодування: 72 біт
Економія: 25 біт
Коефіцієнт стиснення: 1.53
Висновок:

У ході виконання практичної роботи було реалізовано алгоритм Гафмена для кодування тексту. На основі частот символів побудовано двійкове дерево кодування та згенеровано префіксні двійкові коди. Проведено порівняння ефективності алгоритму Гафмена з неоптимальним фіксованим кодуванням. Результати показали зменшення загальної довжини закодованого повідомлення та підвищення ефективності подання інформації. Таким чином, алгоритм Гафмена дозволяє здійснювати ефективне стиснення даних за рахунок використання статистичних властивостей тексту.

Контрольні Питання

1.Що таке кодування Гафмена та як воно працює?

Кодування Гафмена — це алгоритм безвтратного стиснення даних, який ґрунтується на частоті появи символів у повідомленні. Суть алгоритму полягає у присвоєнні коротших двійкових кодів символам, що зустрічаються частіше, та довших кодів — символам з меншою частотою. Алгоритм працює шляхом побудови двійкового дерева, у якому кожен лист відповідає символу, а шлях від кореня до листа визначає його код.

2.Як визначається оптимальний двійковий код Гафмена для стиснення даних?

Оптимальний двійковий код Гафмена визначається на основі частот символів. Алгоритм послідовно об’єднує два символи або вузли з найменшими частотами, формуючи новий вузол з сумарною частотою. Цей процес повторюється до утворення одного кореневого вузла. Отримане дерево гарантує мінімальну середню довжину коду серед усіх можливих префіксних двійкових кодів.

3.Які переваги має кодування Гафмена над іншими методами стиснення даних?

Кодування Гафмена забезпечує безвтратне стиснення даних, має просту реалізацію та не потребує додаткової інформації для декодування, окрім самого дерева. Воно гарантує оптимальність серед префіксних кодів і ефективно працює для даних з нерівномірним розподілом частот символів. Алгоритм широко застосовується в архіваторах і форматах збереження даних.

4.Як відбувається декодування даних, закодованих за допомогою кодування Гафмена?

Декодування здійснюється шляхом проходження по двійковому дереву Гафмена відповідно до бітів закодованого повідомлення. Починаючи з кореня дерева, кожен біт визначає перехід ліворуч або праворуч. При досягненні листа відповідний символ відновлюється, після чого процес починається знову з кореня для наступних бітів. Завдяки префіксній властивості кодів процес декодування є однозначним.

5.Які є можливі недоліки кодування Гафмена?

До недоліків кодування Гафмена належить необхідність збереження або передачі дерева кодування разом із стисненими даними. Алгоритм менш ефективний для дуже коротких повідомлень або для даних з рівномірним розподілом частот символів. Також класичне кодування Гафмена не завжди забезпечує максимальний рівень стиснення порівняно з більш складними алгоритмами.

6.Для чого використовується побудова дерева в кодуванні Гафмена?

Побудова дерева використовується для формування префіксних кодів символів та забезпечення однозначного декодування. Двійкове дерево відображає структуру кодів і дозволяє як виконувати кодування, так і декодування даних. Саме дерево гарантує мінімальну середню довжину коду та ефективність алгоритму.