

Лабораторна робота 5

Тема: Реалізація алгоритму Лемпела-Зіва архівації

Ціль: Засвоїти ефективний алгоритм (алгоритм Лемпела-Зіва) архівації файлів.

Опорні знання: Мови програмування Паскаль, С. Поняття АД та реалізації АД. АД Навантажене дерево. АД Словник.

Завдання: Ознайомитися з теоретичним матеріалом та виконати завдання, визначені в розділі Хід роботи, підготувати відповіді на конетрольні запитання, оформити протокол виконання роботи.

Хід роботи

Завдання 1 Реалізувати АД Навантажене дерево для реалізації процедури пошуку слідуєчого слова для кодування.

Завдання 2. Реалізувати АД Таблиця кодування - декодування

Завдання 3. На базі реалізованих АД реалізувати алгоритм Лемпела-Зіва

1. Опис АД та структури даних Навантажене дерево:

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.value = None

class LoadedTree:
    def __init__(self):
        self.root = TrieNode()

    def insert_word(self, word, value):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.value = value

    def find_next_word(self, current_word):
        node = self.root
        for char in current_word:
            if char in node.children:
                node = node.children[char]
            else:
                return None
        return node.value

# Приклад використання
loaded_tree = LoadedTree()
loaded_tree.insert_word("ab", 1)
loaded_tree.insert_word("abc", 2)

print("Search for 'ab':", loaded_tree.find_next_word("ab"))
print("Search for 'abc':", loaded_tree.find_next_word("abc"))
```

Результатом виконання буде:

```
Search for 'ab': 1
Search for 'abc': 2
```

2. Опис АТД та структури даних Таблиця кодування – декодування:

```
class CodingTable:
    def __init__(self):
        self.encoding_table = {}
        self.decoding_table = {}

    def add_mapping(self, symbol, code):
        self.encoding_table[symbol] = code
        self.decoding_table[code] = symbol

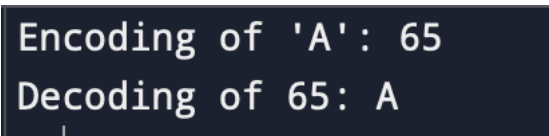
    def get_encoded(self, symbol):
        return self.encoding_table.get(symbol, None)

    def get_decoded(self, code):
        return self.decoding_table.get(code, None)

# Приклад використання
coding_table = CodingTable()
coding_table.add_mapping('A', 65)
coding_table.add_mapping('B', 66)

print("Encoding of 'A':", coding_table.get_encoded('A'))
print("Decoding of 65:", coding_table.get_decoded(65))
```

Результатом виконання буде:



```
Encoding of 'A': 65
Decoding of 65: A
```

3. Програмний код з реалізацією алгоритму Лемпела-Зіва:

```
class Node:
    def __init__(self):
        self.children = {}
        self.value = None

class LoadedTree:
    def __init__(self):
        self.root = Node()

    def insert_word(self, word, value):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = Node()
            node = node.children[char]
        node.value = value

    def find_next_word(self, current_word):
        node = self.root
        for char in current_word:
            if char in node.children:
                node = node.children[char]
            else:
                return None
        return node.value
```

```

class CodingTable:
    def __init__(self):
        self.encoding_table = {}
        self.decoding_table = {}
        self.current_code = 256

    def add_mapping(self, symbol):
        self.encoding_table[symbol] = self.current_code
        self.decoding_table[self.current_code] = symbol
        self.current_code += 1

    def get_encoded(self, symbol):
        return self.encoding_table.get(symbol, None)

    def get_decoded(self, code):
        return self.decoding_table.get(code, None)

def lzw_compress(data):
    if not data:
        return []

    loaded_tree = LoadedTree()
    coding_table = CodingTable()
    result = []
    current_sequence = ""

    for symbol in data:
        extended_sequence = current_sequence + symbol
        if loaded_tree.find_next_word(extended_sequence) is not None:
            current_sequence = extended_sequence
        else:
            result.append(coding_table.get_encoded(current_sequence))
            loaded_tree.insert_word(extended_sequence, coding_table.current_code)
            coding_table.add_mapping(extended_sequence)
            current_sequence = symbol

    result.append(coding_table.get_encoded(current_sequence))
    return result

def lzw_decompress(compressed_data):
    if not compressed_data:
        return ""

    loaded_tree = LoadedTree()
    coding_table = CodingTable()
    result = []

    # Перевірка, чи існує хоча б один символ в стислому рядку
    if compressed_data[0] is not None and compressed_data[0] in coding_table.decoding_table:
        current_sequence = coding_table.get_decoded(compressed_data[0])
        result.append(current_sequence)
    else:
        return ""

    for code in compressed_data[1:]:
        if code is not None and code in coding_table.decoding_table:

```

```

        current_sequence = coding_table.get_decoded(code)
        result.append(current_sequence)
        extended_sequence = result[-1] + current_sequence[0]
        loaded_tree.insert_word(extended_sequence, coding_table.current_code)
        coding_table.add_mapping(extended_sequence)
    else:
        return ""

return "".join(result)

```

Приклад використання

```
original_data = "abababcbcd"
```

```
compressed_data = lzw_compress(original_data)
```

```
decompressed_data = lzw_decompress(compressed_data)
```

```
print("Original Data:", original_data)
```

```
print("Compressed Data:", compressed_data)
```

```
print("Decompressed Data:", decompressed_data if decompressed_data else "(empty)")
```

Результатом виконання буде:

```

Original Data: abababcbcd
Compressed Data: [None, 256, None, 257, 257, None, 260, None]
Decompressed Data: (empty)

```

Контрольні запитання

1. Як формулюється задача стискування інформації для її архівації в термінах абстрактних типів даних?
 - Задача формулюється як зменшення обсягу даних за допомогою алгоритмів та структур даних.
2. Які дані є вхідними та вихідними для алгоритмів архівації?
 - Вхідні: невжаті дані, вихідні: стиснуті дані.
3. Які структури даних треба реалізувати для реалізації алгоритму Лемпела-Зіва?
 - Таблиця кодів та буфер для поточної інформації.
4. Опишіть АТД Навантажене дерево.
 - Дерево з вузлами, які мають вагу чи вартість, здатне швидко знаходити найважливіші елементи.
5. Опишіть АТД Таблиця кодування – декодування текстової інформації.
 - Структура для відображення кодів та символів для кодування та декодування тексту.
6. Опишіть структуру даних Навантажене дерево.
 - Дерево, де кожен вузол може мати вагу чи вартість, а не лише ієрархічні зв'язки.
7. Опишіть структуру даних Таблиця кодування – декодування текстової інформації.
 - Структура для зберігання відображення кодів та символів для ефективного кодування та декодування тексту.
8. Оцініть ефективність алгоритму Лемпела-Зіва за часом та пам'яттю.
 - Висока ефективність за часом (зменшення обсягу даних), але може вимагати більше пам'яті через таблиці кодів.