Курсовая работа по курсу дискретного анализа: Архиватор LZ-77

Выполнил студент группы М80-308Б-20 Зубко Дмитрий Валерьевич.

Условие

Кратко описывается задача:

Ваша программа должна читать входные данные из стандартного потока ввода и выводить ответ на стандартный поток вывода.
Вам будут даны входные файлы двух типов.
Первый тип:

compress
<text>

Текст состоит только из малых латинских букв. В ответ на него вам нужно вывести тройки, которыми будет закодирован данный текст.
Второй тип:

decompress
<triplets>

Вам даны тройки (<offset, len, char>) в которые был сжат текст из малых латинских букв, вам нужно его разжать.

Метод решения

Декодирование текста заключается в декодировании последовательности троек. Мы находимся в какой-то позиции нашей строки, считываем тройку. Смещаемся на величину, равную shift, берем size букв. Дописываем character из тройки.

Кодирование заключается в сжатии строки в тройки. Есть закодированная и необработанная части. Сжатие заключается в том, что мы ищем максимальную подстроку в незакодированной части, которая начинается с необработанной части строки. Сжатие происходит в экземпляр структуры LZ77Triples: <shift, size, character>.

Описание программы

```
#include <iostream>
#include <vector>
#include <string>


using namespace std;


const char END = 'E';


struct LZ77Triples {
    int shift;
    int size;
    char character;
};


ostream& operator << (ostream &os, const LZ77Triples &triple) {
    os << triple.shift << ' ' << triple.size;
    os << ' ' << triple.character;
    return os;
}


string decoder(vector<LZ77Triples>& triples) {
    string buffer;
    for (LZ77Triples &triple: triples) {
        int position = int(buffer.size()) - triple.shift;
        for (int i = 0; i < triple.size; ++i) {
            buffer += buffer[position + i];
        }
        if (triple.character != ' ') {
            buffer += triple.character;
        }
    }
```

```
    return buffer;
}


vector<LZ77Triples> encoder(string& text) {
    vector<LZ77Triples> triples;
    int border = 0;
    while (border < text.size()) {
        int max_shift = 0;
        int max_size = 0;
        char next_character = text[border];
        for (int i = 0; i < border; ++i) {
            if (text[i] == text[border]) {
                int current_shift = border - i;
                int current_size = 1;
                char current_next_character = text[border + 1];
                for (int j = 1; j + i < text.size() and text[i + j] == text[border + j];
++j) {
                    ++current_size;
                    current_next_character = text[border + current_size];
                }
                if (current_size >= max_size) {
                    max_shift = current_shift;
                    max_size = current_size;
                    next_character = (border + max_size == text.size()) ? ' ' :
current_next_character;
                }
            }
        }
        triples.push_back({max_shift, max_size, next_character});
        border += max_size + 1;
    }
    return triples;
}


int main() {
    string command;
    cin >> command;
```

```
    if (command == "compress") {
        string text;
        cin >> text;
        vector<LZ77Triples> triples = encoder(text);
        for (LZ77Triples &triple: triples) {
            cout << triple << endl;
        }
    } else if (command == "decompress") {
        vector<LZ77Triples> triples;
        int shift;
        int size;
        char next_character;
        while (cin >> shift >> size) {
            if (cin >> next_character) {
                triples.push_back({shift, size, next_character});
            } else {
                triples.push_back({shift, size, ' '});
            }
        }
        cout << decoder(triples) << endl;
    }
}
```

Дневник отладки

Были проблемы с обработкой ввода, поскольку я ожидал, что в конце будет тройка, где отсутствует символ. Чтобы исправить ошибку, я перенес вызов функции декодирования после обработки всего вывода.

Тест производительности

Функция сжатия для строк длины N, где N = 10000, 20000, 30000, 500000.

1) N = 5000: 26 ms
2) N = 10000: 67 ms
3) N = 20000: 263 ms
4) N = 50000: 1399 ms
5) N = 100000: 4891 ms

Из тестов видно, что время сильно увеличивается при возрастании N.

Функция декодирования для строк длины N, где N = 5000, 10000, 20000, 50000, 100000.

1) N = 5000: 26 ms
2) N = 10000: 16 ms
3) N = 20000: 36 ms
4) N = 50000: 142 ms
5) N = 100000: 1204 ms

Время возрастает в примерно 10 раз, увеличивая строку строки с 50000 до 10000. Это совпадает со сложностью реализации – O(N^3).

Недочёты

Программа работает корректно только для правильных входных данных.

Выводы

Данный упрощенный курсовой проект по дискретному анализу познакомил меня с алгоритмом архивации LZ-77, разработанным в 1977 году. В своей реализации я использовал наивную реализацию алгоритма за O(N^3). Используя суффиксное дерево, можно было бы добиться сложности O(N^3).