Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект
По курсу «Дискретный анализ»
4 семестр

Студент: И. О. Ильин
Преподаватель: С. А. Сорокин
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Задача: Требуется разработать программу, которая должна читать входные данные из стандартного потока ввода и выводить ответ на стандартный поток вывода. Даны входные файлы двух типов.

Первый тип:

```
compress
<text>
```

Текст состоит только из малых латинских букв. В ответ на него необходимо вывести коды, которыми будет закодирован данных текст.

Второй тип:

```
decompress
<text>
```

Вам даны коды, в которые был сжат текст из малых латинских букв. В ответ на него вам нужно вывести коды, которыми будет закодирован данных текст.

Начальный словарь выглядит следующим образом:

```
a -> 0
b -> 1
c -> 2
...
x -> 23
y -> 24
z -> 25
EOF -> 26
```

2 Описание

Требуется написать реализацию алгоритма LZW. Данный алгоритм является модификацией алгоритма LZ-78, имея 2 важные особенности. Первая — словарь изначально инициализирован всеми символами алфавита. Вторая — после нахождения максимального префикса поиск следующего префикса начинается не с символа после неподходящего, а с самого неподходящего символа, что позволяется выводить только код фразы, а сам символ выяснять прямо из словаря.

Перед началом работы, все символы текста заносятся в словарь.

Кодирование производится следующим путем: во входную фразу заносим первый символ X, затем считываем очередной символ Y, если XY уже содержится в словаре, то присваиваем входной фразе значение XY, переходим к считыванию следующего символа, иначе выводим код для входной фразы X, заносим XY в словарь, присваиваем входной фразе значение Y, переходим к считыванию следующего символа.

Декодирование производится следующим путем: во входную фразу заносим первый код X, затем считываем очередной код Y, если XY уже содержится в словаре, то присваиваем входной фразе код XY, переходим к считыванию следующего кода, иначе выводим символы для кода X, заносим XY в словарь, переходим к считыванию следующего символа.

3 Исходный код

В файле *lzw.h* напишем определение класса *TLZW*, который будет содержать статические функции кодирования и декодирования. Для сжатия получаем строку, возвращаем массив кодов, при расжатии наоборот. Также определим приватные функции заполнения словарей символами текста.

```
#pragma once
#include <string>
#include <vector>
#include <unordered_map>

namespace NAlgo {
    class TLZW {
    public:
        static std::vector<size_t> Encode(const std::string& text);
        static std::string Decode(const std::vector<size_t>& codes);
    private:
        static void InitDict(std::unordered_map<std::string, size_t>& dict);
        static void InitDict(std::vector<std::string>& dict);
    };
} // namespace NAlgo
```

В файле *lzw.cpp* напишем реализации вышеописанных функций. В качестве словаря при кодировании используется *std::unordered_ map*, поскольку необходимо множество раз искать код по строке (которая является ключом). При декодировании же используется *std::vector*, в виду того, что очередная фраза добавляется в конец словаря по коду, равному размеру словаря, таким образом мы можем по индексу (коду) получать расшифрованную фразу за $O(1)$.

```
#include "lzw.h"

namespace NAlgo {

    void TLZW::InitDict(std::unordered_map<std::string, size_t>& dict) {
        dict.clear();
        dict = {
                {"a", 0},
                {"b", 1},
                {"c", 2},
                {"d", 3},
                {"e", 4},
                {"f", 5},
                {"g", 6},
                {"h", 7},
                {"i", 8},
                {"j", 9},
                {"k", 10},
                {"l", 11},
```

```
            {"m", 12},
            {"n", 13},
            {"o", 14},
            {"p", 15},
            {"q", 16},
            {"r", 17},
            {"s", 18},
            {"t", 19},
            {"u", 20},
            {"v", 21},
            {"w", 22},
            {"x", 23},
            {"y", 24},
            {"z", 25}
        };
        // a b c d e f g h i j k l m n o p q r s t u v w x y z
    }

    void TLZW::InitDict(std::vector<std::string>& dict) {
        dict.clear();
        dict = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "
            o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"};
    }

    std::vector<size_t> TLZW::Encode(const std::string& text) {
        std::unordered_map<std::string, size_t> dict;
        InitDict(dict);

        std::vector<size_t> result;
        std::string str;
        for (char ch : text) {
            auto search = dict.find(str + ch);
            if (search != dict.end()) {
                str += ch;
            } else {
                result.emplace_back(dict[str]);
                dict[str + ch] = dict.size();
                str = ch;
            }
        }
        if (str.size()) {
            result.emplace_back(dict[str]);
        }

        return result;
    }

    std::string TLZW::Decode(const std::vector<size_t>& codes) {
        std::vector<std::string> dict;
```

```
        InitDict(dict);

        size_t prevCode = codes.front();
        if (prevCode >= dict.size()) {
            throw std::runtime_error("Invalid code");
        }
        std::string entry, result;
        entry = result = dict[prevCode];

        for (auto it = std::begin(codes) + 1; it != std::end(codes); ++it) {
            size_t currCode = *it;
            if (currCode < dict.size()) {
                entry = dict[currCode];
            } else if (currCode == dict.size()){
                entry += entry[0]; // ?
            } else {
                throw std::runtime_error("Invalid code");
            }

            result += entry;
            dict.push_back(dict[prevCode] + entry[0]);
            prevCode = currCode;
        }

        return result;
    }
} // namespace NAlgo
```

В *main.cpp* считываем команду, затем считываем либо строку, либо кода до конца строки, выводим соответсвующий результат.

```
#include <iostream>
#include <sstream>
#include "lzw.h"

int main() {
    try {
        std::string cmd;
        std::cin >> cmd;
        if (cmd == "compress") {
            std::string text;
            std::cin >> text;
            std::vector<size_t> encodedText = NAlgo::TLZW::Encode(text);
            for (auto it = std::begin(encodedText); it != std::end(encodedText); ++it)
                {
                std::cout << *it;
                if (it != std::end(encodedText) - 1) {
                    std::cout << " ";
                }
            }
```

```
            std::cout << std::endl;
        } else if (cmd == "decompress") {
            std::vector<size_t> codes;
            size_t code;
            while (std::cin >> code) {
                codes.push_back(code);
            }
            std::cout << NAlgo::TLZW::Decode(codes);
        } else {
            throw std::runtime_error("Invalid command");
        }
    } catch (const std::exception& ex) {
        std::cerr << ex.what();
    }

    return 0;
}
```

В *unit_tests.cpp* напишем пару юнит тестов, которые покрывают весь необходимый функционал кодирования, также проверяем корректную обработку последнего кода в зашифрованном сообщении, который также является последним в словаре (тот, для которого на момент дешифровки не существует запись в словаре в виду запаздывания декодирования на один шаг по сравнению с кодированием).

```
#include <gtest/gtest.h>
#include <vector>
#include <string>
#include "../src/lzw.h"

std::vector< std::pair< std::string, std::vector<size_t> > > encodedText = {
        { // 1
                "abc",
                {0, 1, 2}
        },
        { // 2
                "acagaatagaca",
                {0, 2, 0, 6, 0, 0, 19, 28, 26, 0}
        },
        { // 3
                "acagaatagaga",
                {0, 2, 0, 6, 0, 0, 19, 28, 33}
        },
        { // 4
                "abababa",
                {0, 1, 26, 28}
        },
        { // 5
                "abababac",
                {0, 1, 26, 28, 2}
```

```
26          }
27  };
28
29  TEST(LZWSuite, encodeTest) {
30
31      for (size_t i = 0; i < encodedText.size(); ++i) {
32          const std::string& text = encodedText[i].first;
33          std::vector<size_t> expected = encodedText[i].second;
34          std::vector<size_t> result = NAlgo::TLZW::Encode(text);
35
36          ASSERT_EQ(expected, result) << "FAILED test #" << i + 1;
37      }
38  }
39
40  TEST(LZWSuite, decodeTest) {
41
42      for (size_t i = 0; i < encodedText.size(); ++i) {
43          const std::string& expected = encodedText[i].first;
44          std::vector<size_t> codes = encodedText[i].second;
45          std::string result = NAlgo::TLZW::Decode(codes);
46
47          ASSERT_EQ(expected, result) << "FAILED test #" << i + 1;
48      }
49  }
50
51  int main(int argc, char **argv) {
52      testing::InitGoogleTest(&argc, argv);
53      return RUN_ALL_TESTS();
54  }
```

4 Консоль

Для тестирования были написаны скрипт на python для генерации тестов (n файлов с рандомными строками), а также скрипт на bash, который компилирует программу, выполняет юнит тесты, генерирует тестовые файлы и кодирует каждый файл, затем декодирует и сравнивает исходную строку с полученной.

```
MacBook-Pro:cp_da mr-ilin$ ./wrapper.sh 10 10
[2021-06-30 22:02:32] [INFO] Compiling...
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c src/main.cpp -o src/main.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c src/lzw.cpp -o src/lzw.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable src/main.o src/lzw.o -o
solution
[2021-06-30 22:02:33] [INFO] Making unittest...
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c test/unit_tests.cpp
-o test/unit_tests.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable test/unit_tests.o src/lzw.o
-o unit_test -lgtest -lgtest_main
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from LZWSuite
[ RUN      ] LZWSuite.encodeTest
[       OK ] LZWSuite.encodeTest (0 ms)
[ RUN      ] LZWSuite.decodeTest
[       OK ] LZWSuite.decodeTest (0 ms)
[----------] 2 tests from LZWSuite (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
[2021-06-30 22:02:34] [INFO] Generating tests...
[2021-06-30 22:02:35] [INFO] No failed tests,hooray
MacBook-Pro:cp_da mr-ilin$ ./solution
compress
googleisgoogle
6 14 14 6 11 4 8 18 26 28 30
MacBook-Pro:cp_da mr-ilin$ ./solution
decompress
6 14 14 6 11 4 8 18 26 28 30
googleisgoogle
```

5 Выводы

В ходе выполнения данного курсового проекта познакомился с алгоритмами сжатия данных, конкретно с алгоритмом LZW. Он достаточно прост в реализации, при этом обладает хорошим коэффициентом сжатия и некогда использовался повсеместно. Благодаря знаниям, полученным в ходе выполнения лабораторных работ, написание кода по конспекту лекций не составило большого труда. В ходе тестирования была выявлена ошибка при декодировании 3-х подряд идущих одинаковых символов в начале строки.

В качестве улучшений данной версии алгоритма, можно зарезервировать один код для очистки словаря, что пригодится в реальных условиях, когда размер словаря становится слишком большим. Также можно не опираться на малые латинские буквы, а либо принимать на вход алфавит, либо делать дополнительный проход по тексту.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Алгоритм LZW — Вики ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_LZW` (дата обращения: 10.06.2021).