Курсовая работа по курсу дискретного анализа: Алгоритм LZW.

Выполнила студент группы М8О-307-20 МАИ *Михеева Кристина.*

Условие

Реализуйте алгоритм LZW.

Начальный словарь выглядит следующим образом: a -> 0 b -> 1 c -> 2 ... x -> 23 y -> 24 z -> 25 EOF -> 26

Формат ввода

Вам будут даны тесты двух типов. Первый тип: compress <text> Текст состоит только из малых латинских букв. В ответ на него вам нужно вывести коды, которыми будет закодирован данный текст. Второй тип: decompress <codes> Вам даны коды в которые был сжат текст из малых латинских букв, вам нужно его разжать.

Формат вывода

В ответ на тест первого типа вам нужно вывести коды, которыми будет закодирован данный текст через пробел. В ответ на тест второго типа выведите расжатый текст.

Метод решения

LZW-это алгоритм сжатия без потерь на основе словаря. Словарные алгоритмы сканируйте файл на наличие последовательностей данных, которые встречаются более одного раза. Затем эти последовательности сохраняются в словаре, а внутри сжатого файла помещаются ссылки на те места, где когда-либо встречались повторяющиеся данные.

Сначала заводится словарь, в котором инициализируется все возможные односимвольные слова. Считывается буква и проверяется есть ли она в словаре. Если слова нет в словаре, то для последнего слова, которое было, записывается его код, а в словарь добавялется предыдущее слово и первая буква, которую не смогли считать.

Алгоритму декодирования на входе требуется только закодированный текст: соответствующий словарь фраз легко воссоздаётся посредством имитации работы алгоритма кодирования.

Также заводится словарь, в котором ницилизируется все возможные односимвольные слова. Если конец фразы, то выдать фразу и завершить алгоритм. Если фраза, образованная слиянием фраз кодов, уже есть в словаре, то присвоить входной фразе новое значение и завершить алгоритм. Иначе добавить новую в словарь, присвоить входной фразе новое значение. Если же мы встречаем код, не принадлежащий текущему, восстановленному нами алфавиту, то надо соединить предудущую строку и её первый символ.

Главное особенность алгоритма в том, что декодирование идет с запаздыванием на один шаг от кодирования.

Описание программы

Программа состоит из одного файла LZW.cpp. Создаем две функции **Encoding** и **Decoding** для кодирования и декодирования соответственно.

Encoding:

Создаем словарь CodeDictionary, где будут хранится буквы с их кодами в виде хеш-таблицы [string]:[int]. Циклом проходимся по всем буквам в дипозоне от 0 до 26 т.к у нас даны только маленькие латинские буквы. Создаем строку str, в которую в конец строки мы добавляем новую букву, код которой равен коде буквы ('a' + i), имеет вид 97-0, 98-1, 99-2... так как используется код ASCII. В словарь добавляются методом insert пара, состоящие из значение и кода. Затем присваиваем значение словарю со значением соответвующий код. Когда мы прошлись по всему циклу, то записывается в словарь пара, состаящая из конца строки("") и ее кода(26).

Реализация самого алгоритма кодирования. Заводим две переменные буквы(c) и слова(word). Пока считываем буквы выполняем операции. Далее заводится временная переменная, которой присвается word, считывается буква записывается в слово и так далее. Если считали все буквы, то в слово кладется значение tmp и выводится 26. Смотрим если такое слова нет в словаре, то вывод кода от tmp, где tmp слово word на предыдущем шаге. Определяется размер словаря. Добавляем пару (новое слово, размер словаря на предыдущем шаге) и кладем новое слово.

Dencoding:

Такой же словарь создается и для декодирования. Различия только в том, что в словарь добавляется пара состоящая из кода и значения.

Реализация самого алгоритма декодирования. Заводим две переменные код(code) и слово(word). Пока вводится код буквы заводится временная переменная которой присвается word. Если в словаре есть считаный code, то в word мы кладем значение, которое лежит в словаре. Если оно - пустота, то заканчиваем цикл, иначе выводим word и если tmp(значение word на предыдущем шаге) не равно пустой строке, то делаем новое слово из tmp и первой буквы word и кладем его в словарь. Если code встречается в словаре, то делаем новое слово из tmp и первой буквы tmp и кладем его в словарь.

Сложность алгоритма получилась **O(N)** как и для кодирования так и для декожирования.

Листинг программы

```
#include <iostream>
#include <map>
#include <unordered_map>

using namespace std;

void Encoding() {
```

```
   unordered_map<string, int> CodeDictionary;
    for (int i = 0; i <= 25; ++i) {
        string str;
        str.push_back('a' + i);
        CodeDictionary.insert({str, i});
        CodeDictionary[str] = i;
    }
    CodeDictionary.insert({"\0", 26});

    char c;
    string word;
    while((c = getchar())) {
        string tmp = word;
        word += c;
        if(c == EOF) {
            word = tmp;
            cout << 26 << "\n";
            break;
        }
        if(CodeDictionary.count(word) == 0) {
            cout << CodeDictionary[tmp] << " ";
            int size = CodeDictionary.size();
            CodeDictionary.insert({word, size});
            word = c;
        }
    }
}

void Decoding() {
    unordered_map<int, string> DecodeDictionary;
    for (int i = 0; i <= 25; ++i) {
        string str;
        str.push_back('a' + i);
        DecodeDictionary.insert({i, str});
        DecodeDictionary[i] = str;
    }
        DecodeDictionary.insert({26, "\0"});

    int code;
    string word;
        string tmp = word;
        if(DecodeDictionary.count(code) != 0) {
```

```
            word = DecodeDictionary[code];

            if(word == "EOF") {
                cout << "\n";
                break;
            }

            cout  << word;
            if(tmp != "") {
                word = tmp + word.front();
                int size = DecodeDictionary.size();
                DecodeDictionary.insert({size, word});
                word = DecodeDictionary[code];
            }
        }
        else {
            word = tmp + tmp.front();
            int size = DecodeDictionary.size();
            DecodeDictionary.insert({size, word});
            cout  << word;
        }
    }


int main() {

    string com;
    getline(cin, com);
    if(com == "compress") Encoding();
    else if(com == "decompress") Decoding();
    else cout << "Incorrect command!";

    return 0;
}
```

Вывод из консоли

```
kristina@kristina-KLVL-WXXW:~/DA/KP$ g++ LZW.cpp -o main
kristina@kristina-KLVL-WXXW:~/DA/KP$ ./main
compress
adadas
0 3 27 0 18 26
```

```
kristina@kristina-KLVL-WXXW:~/DA/KP$ g++ LZW.cpp -o main
kristina@kristina-KLVL-WXXW:~/DA/KP$ ./main
decompress
0 3 27 0 18 26
adadas
kristina@kristina-KLVL-WXXW:~/DA/KP$ g++ LZW.cpp -o main
kristina@kristina-KLVL-WXXW:~/DA/KP$ ./main
compress
asdfgdadsdadadsfadada
0 18 3 5 6 3 0 3 28 33 33 18 5 36 32
kristina@kristina-KLVL-WXXW:~/DA/KP$ g++ LZW.cpp -o main
kristina@kristina-KLVL-WXXW:~/DA/KP$ ./main
decompress
0 18 3 5 6 3 0 3 28 33 33 18 5 36 32
asdfgdadsdadadsfadada
kristina@kristina-KLVL-WXXW:~/DA/KP$ g++ LZW.cpp -o main
kristina@kristina-KLVL-WXXW:~/DA/KP$ ./main
compress
alibaba
0 11 8 1 0 30 0 26
kristina@kristina-KLVL-WXXW:~/DA/KP$ g++ LZW.cpp -o main
kristina@kristina-KLVL-WXXW:~/DA/KP$ ./main
decompress
0 11 8 1 0 30 0 26
alibaba
```

Дневник отладки

Возникла ошибка с выводом кода для конца строки, программа брала код первой буквы. Для решения пришлось отдельно прописать определенное значение 26 для конца строки.

Тест производительности

Количество символов	Кодирование
100	8ms.
1000	13ms.
10000	16ms.

Количество символов	Декодирование
100	9ms.
1000	12ms.
10000	17ms.

Выводы

В курсовой работе по дисретному анализу был реализован алгоритм LZW, в котором выполнялись как команда кодирования, так и декодирования. С помощью данного алгоритма оказалось очень легко сжимать и расжимать тексты, а также был апроведен анализ программы, который позволил онять насколько эффективно происхоит сжатие и расжатие текстов.