Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Дискретный анализ»

Алгоритм LZW

Студент: Л. В. Ивенкова
Преподаватель: С. А. Сорокин
Группа: М8О-208Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Курсовая работа, базовый вариант №1.5

Задача: Требуется разработать программу, читающую входные данные из стандартного потока ввода и выводящую 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 нужно вывести коды, которыми будет закодирован данный текст с помощью алгоритма LZW.

Второй тип:

```
decompress
<text>
```

Даны коды в которые был сжат текст из малых латинских букв, нужно его разжать.

Начальный словарь состоит выглядит следующим образом:

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

1 Теория

Требуется написать реализацию алгоритма сжатия LZW.

Алгориитм Лемпеля — Зива — Уэлча (Lempel-Ziv-Welch, LZW) — это универсальный алгоритм сжатия данных без потерь, созданный Авраамом Лемпелем (англ. Abraham Lempel), Яаковом Зивом (англ. Jacob Ziv) и Терри Велчем (англ. Terry Welch). Он был опубликован Велчем в 1984 году в качестве улучшенной реализации алгоритма LZ78, опубликованного Лемпелем и Зивом в 1978 году[2]. (Включение явного символа в вывод после каждой фразы часто является расточительным. В LZW этого избегают с помощью инициализации списка фраз, включающего все символы исходного алфавита.)

Данный алгоритм при кодировании (сжатии) сообщения динамически создаёт словарь фраз: определённым последовательностям символов (фразам) ставятся в соответствие группы битов (коды). Словарь инициализируется всеми 1-символьными фразами.

Формально данный алгоритм можно описать следующей последовательностью шагов:

1. Инициализация словаря всеми возможными односимвольными фразами. Инициализация входной фразы W первым символом сообщения.
2. Если КОНЕЦ_СООБЩЕНИЯ, то выдать код для W и завершить алгоритм.
3. Считать очередной символ K из кодируемого сообщения.
4. Если фраза WK уже есть в словаре, то присвоить входной фразе W значение WK и перейти к Шагу 2.
5. Иначе выдать код W, добавить WK в словарь, присвоить входной фразе W значение K и перейти к Шагу 2.

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

1. Инициализация словаря всеми возможными односимвольными фразами. Инициализация входного кода C первым числом сообщения.

2. Если КОНЕЦ_СООБЩЕНИЯ, то выдать фразу для C и завершить алгоритм.

3. Выдать фразу W, соответствующую C.

4. Считать очередное число C2 из сообщения.

5. Если фраза, образованная слиянием фраз кодов C и C2 = WK, уже есть в словаре, то присвоить входной фразе W значение WK и перейти к Шагу 2.

6. Иначе добавить WK в словарь, присвоить входной фразе W значение K и перейти к Шагу 2.

7. Если же мы встречаем код, не принадлежащий текущему, восстановленному нами алфавиту, то надо сконкатенировать предудущую строку и её первый символ.

2 Исходный код

В начале кодирования мы создаём словарь с ключом-фразой и значением-числом, и заполняем его начальными символами. В начале декодирования создаём тот же словарь, но уже с ключом-числом - так как мы будем уже искать фразу по коду, а не наоборот.

Алгоритмы кодирования и декодирования были подробно описаны в разделе Теория.

Листинг кода:

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

using namespace std;

void Encoding() {

   unordered_map<string, int> Dictionary;
    for (int i = 0; i < 26; ++i) {
        string s;
        s.push_back('a' + i);
        Dictionary.insert(pair<string, int>(s, i));
        Dictionary[s] = i;
    }
    Dictionary.insert(pair<string, int>("EOF", 26));

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

```
36
37  void Decoding() {
38
39      unordered_map<int, string> CodeDictionary;
40      for (int i = 0; i < 26; ++i) {
41          string s;
42          s.push_back('a' + i);
43          CodeDictionary.insert(pair<int, string>(i, s));
44          CodeDictionary[i] = s;
45      }
46      CodeDictionary.insert(pair<int, string>(26, "EOF"));
47
48      int code;
49      string pattern;
50      while(cin >> code) {
51          string tmp = pattern;
52          if(CodeDictionary.count(code) != 0) {
53              pattern = CodeDictionary[code];
54
55              if(pattern == "EOF") {
56                  cout << "\n";
57                  break;
58              }
59
60              cout << pattern;
61              if(tmp != "") {
62                  pattern = tmp + pattern.front();
63                  int size = CodeDictionary.size();
64                  CodeDictionary.insert(pair<int, string>(size, pattern));
65                  pattern = CodeDictionary[code];
66              }
67          }
68          else {
69              pattern = tmp + tmp.front();
70              int size = CodeDictionary.size();
71              CodeDictionary.insert(pair<int, string>(size, pattern));
72              cout << pattern;
73          }
74      }
75  }
76
77  int main() {
78
79      string command;
80      getline(cin, command);
81      if(command == "compress") Encoding();
82      else if(command == "decompress") Decoding();
83      else cout << "Incorrect command!";
84
```

```
85      return 0;
86  }
```

3 Консоль

```
parsifal@DESKTOP-3G70RV4:~/DA/Curs$ make
g++ -std=c++17 -O3 -Wextra -Wall -pedantic main.cpp -o Curs
parsifal@DESKTOP-3G70RV4:~/DA/Curs$ cat test.txt
compress
acagaatagaga
parsifal@DESKTOP-3G70RV4:~/DA/Curs$ ./Curs <test.txt
0 2 0 6 0 0 19 29 34
parsifal@DESKTOP-3G70RV4:~/DA/Curs$ cat test.txt
decompress
0 2 0 6 0 0 19 29 34
parsifal@DESKTOP-3G70RV4:~/DA/Curs$ ./Curs <test.txt
acagaatagaga
```

4 Коэффициент сжатия, время работы

Для тестирования программы напишем генератор текстов из малых латинских букв, где мы можем задавать длинну повторений и их количество.

```
import random
import string


def generate_random_string(length):
    letters = string.ascii_lowercase
    rand_string = ''.join(random.choice(letters) for i in range(length))
    rand_string = rand_string*100
    print(rand_string)

print("compress")
generate_random_string(50000)
```

И проверяем время работы архиватора на текстах различной длинны (будем изменять параметр функции generate_random_string() rand_string = rand_string*200).

Количество символов	Время работы архиватора (мс)
1000	12
10.000	14
100.000	34
1.000.000	417
10.000.000	5023
100.000.000	69761

Коэффициент сжатия (при rand_string = rand_string*100):

Количество символов	Размер начального файла(Кб)	Размер конечного файла(Кб)	Коэффициент сжатия
1000	1	1	1
10.000	10	6	1,67
100.000	98	64	1,53
1.000.000	977	723	1,35
10.000.000	9766	7962	1,23
100.000.000	97657	86387	1,13

5 Выводы

Выполнив курсовую работу по курсу «Дискретный анализ», я изучила алгоритм сжатия LZW - научилась с его помощью сжимать и разжимать тексты, а также анализировать, насколько эффективно прошло сжатие.

Список литературы

[1] Ватолин Д., Ратушняк А., Смирнов М., Юкин В.. *Методы сжатия данных. Устройство архиваторов,сжатие изображений и видео.*

[2] *Алгоритм Лемпеля — Зива — Велча — Википедия.*
URL: `http://https://ru.wikipedia.org/wiki/Алгоритм_Лемпеля_-_Зива_-_Велча`
(дата обращения: 7.06.2021).