Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: Д. Д. Стрыгин
Преподаватель: С. А. Сорокин
Группа: М8О-306Б-19

Дата: 02.12.2021
Оценка:
Подпись:

Москва, 2021

Курсовой проект

Задача: Необходимо реализовать программу, которая в качестве аргументов должна принимать ключ и имя файла и, в зависимости от ключа, кодировать/декодировать файл.
Ключи:

1. -c(--compress) - кодирует файл, создовая новый формат *.cmp*
2. -d(--decompress) - декодирует файл, создовая новый формат *.dcmp*
3. -t(--test) - тестирует и анализирует выполнение программы, сравнивая размер входного файла на кодирование и выходного после декодирования
4. -h(--help) - выводит справочную информацию об использовании программы

Требований к входным данным нет.

1 Описание

Требуется написать реализацию алгоритма LZ-78 для кодирования текста.
Алгоритм LZ-78 использует словарный подход к кодированию данных. Во время кодирования он генерирует временный словарь, в котором хранит фразы, запоминаемые в процессе обработки файла. Изначально словарь пуст, а алгоритм пытается закодировать первый символ. На каждой итерации мы пытаемся увеличить кодируемый префикс, пока такой префикс есть в словаре. Кодовые слова такого алгоритма будут состоять из двух частей — номера в словаре самого длинного найденного префикса и символа, который идет за этим префиксом. При этом после кодирования такой пары префикс с приписанным символом добавляется в словарь, а алгоритм продолжает кодирование со следующего символа. Для хранения словаря используется map, а для пар «префикс, символ» реализован класс TId. В процессе выполнения кодирования считается итоговое отношение между размером входящего файла и обработанного. Оно демонстрирует эффективность сжатия данных, и если строго больше, чем единица, то сжатие произошло. Этот показатель необходим для тестирования алгоритма на разном наборе текстовых данных.

2 Ход работы

Разобьем процесс написания кода на несколько этапов

1. Реализация алгоритма LZ-78
2. Реализация интерфейса программы
3. Реализация генератора файлов для тестирования
4. Тестирование
5. Отладка

3 Исходный код

archiver.cpp:

```
#include "LZ78.hpp"

using namespace std;
using namespace NComp;

//
void usage () {
    cout << "Usage: ./kp [option] file.."
    << "\nOptions:"
    << "\n\t-h(--help) Display all information about program"
    << "\n\t-c(--compress) [filename] - Only compress file and give new extension \"[filename].cmp\""
    << "\n\t-d(--decompress) [filename] - Only decompress file and give extension \"[filename].dcmp\""
    << "\n\t-t(--test) [filename] - Compress, decompress and then analyze efficiency\n";
}

int main (int argc, char *argv[]) {
    uint64_t sizeOld, sizeNew;
    try {
        //   -
        if (
            argc < 2 ||
            argc > 3 ||
            ((string(argv[1]) == "-h" || string(argv[1]) == "--help") && argc != 2) ||
            ((string(argv[1]) != "-h" && string(argv[1]) != "--help") && argc != 3)
        ) {
            throw -1;
        }
        //
        string command(argv[1]);
        if (command == "-h" || command == "--help") {
            usage();
        } else if (command == "-c" || command == "--compress") {
            sizeOld = Compress(argv[2]);
        } else if (command == "-d" || command == "--decompress") {
            sizeNew = Decompress(argv[2]);
        } else if (command == "-t"|| command == "--test") {
            sizeOld = Compress(argv[2]);
            sizeNew = Decompress(string(argv[2]) + ".cmp");
            if (sizeNew == sizeOld) {
                cout << "\nongratulation\n\n";
            }
            else {
                cerr << "\nFaild and Pain\n\n";
```

```
            }
        } else {
            throw -2;
        }
    //
    } catch (int a) {
        string error = "For more information use my program with key \"-h\" or \"--help
            \".\n";
        if (a == -1) {
            cerr << "./kp: error: Incorrect count of arguments.\n" << error;
        } else {
            cerr << "./kp: error: Invalid command.\n" << error;
        }
    }
    return 0;
}
```

LZ78.cpp:

```
#include "LZ78.hpp"

namespace NComp {
    ///
    TId::TId () : pos(0), next('\0') {}

    TId::TId (uint16_t pos, char next) : pos(pos), next(next) {}

    TId::~TId () {}

    uint64_t TId::SizeOfNode () {
        return sizeof(pos) + sizeof(next);
    }
    //
    std::ifstream &operator>> (std::ifstream &input, TId &node) {
        input.read(reinterpret_cast<char*>(&node.pos), sizeof(node.pos));
        input.read(reinterpret_cast<char*>(&node.next), sizeof(node.next));
        return input;
    }

    std::ofstream &operator<< (std::ofstream &output, const TId &node) {
        output.write(reinterpret_cast<const char*>(&node.pos), sizeof(node.pos));
        output.write(reinterpret_cast<const char*>(&node.next), sizeof(node.next));
        return output;
    }

    uint64_t Compress (const std::string &filename) {
        //
        std::ifstream file(filename);
        if (!file) {
            throw std::runtime_error("Can't open file \"" + filename + "\"");
```

```
32          }
33          //
34          std::ofstream output(filename + ".cmp", std::ios::binary);
35
36          uint64_t oldSize = 0, newSize = 0, toRead = 0;
37
38          file.seekg(0, std::ios::end);
39          //
40          oldSize = file.tellg();
41          file.seekg(0);
42
43          std::string str, buffer;
44          str.resize(SIZE_OF_TEXTPART);
45
46          char last_char;
47          std::map<std::string, uint64_t> dict;
48          //   (    )
49          toRead = oldSize > SIZE_OF_TEXTPART ? SIZE_OF_TEXTPART : oldSize;
50
51          file.read(&str[0], toRead);
52          uint64_t j = 0;
53          for (uint64_t i = 0; i < oldSize; ++i) {
54              //
55              if (j == str.size()) {
56                  j = 0;
57                  toRead = oldSize - i > SIZE_OF_TEXTPART ? SIZE_OF_TEXTPART : oldSize - i
                        ;
58                  file.read(&str[0], toRead);
59              }
60              //
61              if (dict.count(buffer + str[j]) && i != oldSize - 1) {
62                  buffer += str[j];
63              } else {
64                  uint16_t pos = dict[buffer] == 0 ? 0 : dict[buffer] - 1;
65                  output << TId(pos, str[j]);
66                  if (dict.size() != DICT_SIZE) {
67                      dict[buffer + str[j]] = dict.size();
68                  }
69                  buffer.clear();
70                  newSize += TId::SizeOfNode();;
71              }
72              ++j;
73          }
74          if (buffer != "") {
75              last_char = buffer.back();
76              buffer.pop_back();
77              output << TId(dict[buffer], last_char);
78              newSize += TId::SizeOfNode();
79          }
```

```
80
81          std::cout << "Input filename: " << filename << "\nInput size: " << oldSize
82          << "\n\nOutput filename: " << filename + ".cmp" << "\nOutput size: " << newSize
83          << "\n\nEfficiency: " << (round((double)oldSize / (double)newSize * 10) / 10)
                << "\n\n";
84
85          file.close();
86          output.close();
87
88          return oldSize;
89      }
90
91      uint64_t Decompress (const std::string &filename) {
92          //
93          std::ifstream file(filename, std::ios::binary);
94          if (!file) {
95              throw std::runtime_error("Can't open file \"" + filename + "\"");
96          }
97          //
98          std::string decompfilename = filename;
99          decompfilename = std::regex_replace(decompfilename, std::regex(".cmp"), ".dcm")
                ;
100         std::ofstream output(decompfilename + "p");
101
102         TId node;
103         std::string ans, word;
104         std::vector<std::string> dict;
105         uint64_t inputSize = 0, outputSize = 0;
106         dict.push_back("");
107
108         file.seekg(0, std::ios::end);
109         inputSize = file.tellg();
110         file.seekg(0);
111
112         while (file >> node) {
113             // ,
114             word = dict[node.pos] + node.next;
115             ans += word;
116             if (dict.size() != DICT_SIZE) {
117                 dict.push_back(word);
118             }
119             if (ans.size() > SIZE_OF_TEXTPART) {
120                 outputSize += ans.size();
121                 output << ans;
122                 ans.clear();
123             }
124         }
125         if (ans.size() > 0) {
126             outputSize += ans.size();
```

```
127             output << ans;
128         }
129 
130         std::cout << "Input Filename: " << filename << "\nInput size: " << inputSize
131         << "\n\nOutput Filename: " << decompfilename + "p" << "\nOutput size: " <<
                outputSize << "\n";
132 
133         file.close();
134         output.close();
135 
136         return outputSize;
137     }
138 }
```

LZ78.hpp:

```
1  #ifndef LZ78_HPP
2  #define LZ78_HPP
3  
4  #include <iostream>
5  #include <fstream>
6  #include <string>
7  #include <regex>
8  #include <cmath>
9  #include <map>
10 #include <vector>
11 
12 // ,
13 const uint64_t SIZE_OF_TEXTPART = (1 << 10) * 64;
14 //
15 const uint64_t DICT_SIZE = 1 << 16;
16 
17 namespace NComp {
18     //,    ,
19     class TId {
20     public:
21         uint16_t pos;
22         char next;
23 
24         TId ();
25         TId (uint16_t pos, char next);
26         ~TId ();
27 
28         static uint64_t SizeOfNode ();
29 
30         friend std::ifstream &operator>> (std::ifstream &input, TId &node);
31         friend std::ofstream &operator<< (std::ofstream &output, const TId &node);
32     };
33     uint64_t Compress (const std::string &filename);
34     uint64_t Decompress (const std::string &filename);
```

```
}

#endif
```

generator.py

```
import random

TEXT_LENGTH = 10000

def generate_random_text(length):
    alphabet = " abcdefghij"
    rand_text = ''.join(random.choice(alphabet) for i in range(length))
    return rand_text

text = generate_random_text(TEXT_LENGTH)
with open("text1", "w") as file:
    file.write(text)
```

4 Консоль

```
Тест эффективности на файле с алфавитом из 11 символов
den@DESKTOP-1B5EV3F:/mnt/c/Users/danst/OneDrive/Документы
/GitHub/DA/kp$ ./archiver -t text1
Input filename: text1
Input size: 10000

Output filename: text1.cmp
Output size: 8475

Efficiency: 1.2

Input Filename: text1.cmp
Input size: 8475

Output Filename: text1.dcmp
Output size: 10000

Congratulation

Тест эффективности на файле с алфавитом из 6 символов
den@DESKTOP-1B5EV3F:/mnt/c/Users/danst/OneDrive/Документы
/GitHub/DA/kp$ ./archiver -t text1
```

```
Input filename: text1
Input size: 10000

Output filename: text1.cmp
Output size: 6888

Efficiency: 1.5

Input Filename: text1.cmp
Input size: 6888

Output Filename: text1.dcmp
Output size: 10000

Congratulation

Тест эффективности на файле с алфавитом из одного символа
den@DESKTOP-1B5EV3F:/mnt/c/Users/danst/OneDrive/Документы
/GitHub/DA/kp$ ./archiver -t text1
Input filename: text1
Input size: 10000

Output filename: text1.cmp
Output size: 423

Efficiency: 23.6

Input Filename: text1.cmp
Input size: 423

Output Filename: text1.dcmp
Output size: 10000

Congratulation
```

5 Выводы

Выполнив курсовой проект по курсу «Дискретный анализ», я познакомился с алгоритмами кодирования LZ. Научился реализовывать кодом LZ-78. Протестировав его работу на файлах различного алфавита, выявил, что эффективность алгоритма уменьшается с увеличением алфавита входных данных. Так же была замечена низкая эффективность алгоритма на маленьких объёмах данных.

Список литературы

[1] *Алгоритмы сжатия.*
URL: http://mf.grsu.by/UchProc/livak/po/comprsite/theory_lz78.html

[2] *Алгоритмы LZW, LZ77 и LZ78.*
URL: https://habr.com/ru/post/132683/