Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: Е. С. Пищик
Преподаватель:
Группа: М8О-206Б

Дата: 19.06.2021
Оценка:
Подпись:

Москва, 2021

Курсовой проект

Задача: Алгоритм сжатия lzw

Разработать программу, которая будет на вход получать текст, сжимать его алгоритмом сжатия lzw и выдавать векторное представление сжатого текста.

Формат входных данных: Текст или текстовый файл содержащий буквы a-z и пробел.

Формат результата: На первой строке вектор чисел, который является сжатым представлением нашего текста. На второй строке текст полученный из данного сжатого вектора.

1 Алогритм lzw

В 1984 г. Terry A. Welch опубликовал свою работу с модифицированным алгоритмом LZ78, получившим в дальнейшем название LZW. LZW позволяет достичь одну из наилучших степеней сжатия среди других существующих методов сжатия, при полном отсутствии потерь или искажений в исходных файлах. В настоящее время используется в файлах формата TIFF, PDF, а также в программе для сжатия ZIP.

Кодирование:

1. Все возможные символы заносятся в словарь и получают свой номер(например в unordered_map). Во входную фразу P заносится первый символ текста.
2. Считать символ текста C.
3. Если C - EOF, то записываем в вектор код для P, иначе
   - Если текст P+C есть в словаре, то присвоить входной фразе текст P+C и перейти к шагу 2.
   - Если текста P+C нет в словаре, то записать в вектор код для фразы P, добавить P+C в словарь, присвоить входной фразе значение C и перейти к шагу 2.

Декодирование:

1. Все возможные символы заносятся в словарь и получают свой номер(например в unordered_map). Во входную фразу P заносится первый код из вектора сжатого текста.
2. Считать код C из вектора сжатого текста.
3. Если C - последний код, то выдать символ для кода P, иначе
   - Если фразы с кодом P+C нет в словаре, то вывести фразу для кода P, а фразу с кодом P+C занести в словарь и перейти к шагу 2.
   - Если фраза с кодом P+C есть в словаре, то присвоить входной фразе код P+C и перейти к шагу 2.

2 Код

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <string>
#include <unordered_map>
#include <chrono>

std::pair<std::unordered_map<std::string, size_t>, size_t> init_dict_for_compress()
{
    std::unordered_map<std::string, size_t> m;
    size_t ind = 1;
    for(char symb = 'a'; symb <= 'z'; ++symb)
    {
        std::string s(sizeof(char), symb);
        m[s] = ind;
        ++ind;
    }
    std::string space(sizeof(char), ' ');
    m[space] = ind;
    ++ind;
    return std::pair(m, ind);
}

std::pair<std::unordered_map<size_t, std::string>, size_t> init_dict_for_decompress()
{
    std::unordered_map<size_t, std::string> m;
    size_t ind = 1;
    for(char symb = 'a'; symb <= 'z'; ++symb)
    {
        std::string s(sizeof(char), symb);
        m[ind] = s;
        ++ind;
    }
    std::string space(sizeof(char), ' ');
    m[ind] = space;
    ++ind;
    return std::pair(m, ind);
}

std::pair<std::vector<size_t>, size_t> compress()
{
    std::pair<std::unordered_map<std::string, size_t>, size_t> dict =
        init_dict_for_compress();
    std::vector<size_t> res;
    std::string p = "";
    size_t count = 0;
    char c = '0';
```

```
47      while(std::cin.get(c))
48      {
49          ++count;
50          if(dict.first.find(p+c) != dict.first.end())
51              p += c;
52          else
53          {
54              res.push_back(dict.first[p]);
55              dict.first[p+c] = dict.second;
56              ++dict.second;
57              p = c;
58          }
59      }
60      std::pair<std::vector<size_t>, size_t> result(res, count);
61      return result;
62  }
63
64  std::string decompress(std::vector<size_t> const& vct)
65  {
66      std::pair<std::unordered_map<size_t, std::string>, size_t> dict =
            init_dict_for_decompress();
67      size_t prev = vct[0];
68      size_t cw;
69      std::string res = dict.first[prev];
70      std::string c = "";
71      c += res[0];
72      std::string out = res;
73      for(size_t i = 0; i < vct.size()-1; ++i)
74      {
75          cw = vct[i+1];
76          if(dict.first.count(cw) == 0)
77          {
78              res = dict.first[prev];
79              res += c;
80          }
81          else
82              res = dict.first[cw];
83          out += res;
84          c = "";
85          c += res[0];
86          dict.first[dict.second] = dict.first[prev] + c;
87          ++dict.second;
88          prev = cw;
89      }
90      return out;
91  }
92
93  int main()
94  {
```

```
95      std::chrono::time_point<std::chrono::system_clock> start, end;
96      uint64_t compress_time = 0;
97      uint64_t decompress_time = 0;
98      start = std::chrono::system_clock::now();
99      std::pair<std::vector<size_t>, size_t> res = compress();
100     end = std::chrono::system_clock::now();
101     double comp_size = res.first.size();
102     double text_size = res.second;
103     compress_time = std::chrono::duration_cast<std::chrono::microseconds>(end - start).
            count();
104     std::cout << "Compressed file:\n";
105     for(auto& el: res.first)
106         std::cout << el << " ";
107     std::cout << "\n";
108     std::cout << "Decompressed vector:\n";
109     start = std::chrono::system_clock::now();
110     std::cout << decompress(res.first) << "\n";
111     end = std::chrono::system_clock::now();
112     decompress_time = std::chrono::duration_cast<std::chrono::microseconds>(end - start
            ).count();
113     std::cout << "Compress time: " << compress_time << " microseconds\n";
114     std::cout << "Decompress time: " << decompress_time << " microseconds\n";
115     std::cout << "Compress coefficient: " << comp_size / text_size << "\n";
116     return 0;
117 }
```

3 Результаты работы

Коэффициент сжатия = размер сжатого сообщения / размер исходного текста, время измерил при помощи библиотеки chrono.

Тесты:

1. 1000 рандомных букв. Compress time: 11715 microseconds. Decompress time: 9518 microseconds. Compress coefficient: 0.72.
2. 100 тысяч рандомных букв. Compress time: 608538 microseconds. Decompress time: 400181 microseconds. Compress coefficient: 0.38751.
3. 1 миллион рандомных букв. Compress time: 5533366 microseconds. Decompress time: 3346934 microseconds. Compress coefficient: 0.313447.
4. 10 миллионов рандомных букв. Compress time: 54484956 microseconds. Decompress time: 35338798 microseconds. Compress coefficient: 0.257739.

4 Выводы

Выполнив курсовой проект я изучил алгоритм сжатия lzw, который является одним из самых эффективных алгоритмов сжатия, коэффициент сжатия данного алгоритма на большом количестве данных сильно лучше, чем на маленьких данных. Плюсом данного алгоритма является абсолютное отсутствие потерь или искажений при сжатии. Время работы данного алгоритма (зависит еше от того как запрограммировать, здесь используется вариант с использованием unordered_map в качестве словаря, что дает хорошее преимущество во времени) растет почти линейно при увеличении объема данных, что является большим плюсом(1 млн. символов = 5 млн. микросекунд, 10 млн. сиволов = 50 млн. микросекунд). Также несомненным плюсом данного алгоритма является простота программирования на любом ЯП.