Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: И. А. Мазин
Преподаватель: А. А. Журавлёв
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Постановка задачи

Задача: Необходимо написать программу для сжатия файлов методами Хаффмана и LZW.

Формат запуска должен быть аналогичен формату запуска программы gzip, должны быть поддержаны следующие ключи: -c, -d, -k, -l, -r, -t, -1, -9. Должно поддерживаться указание символа дефиса в качестве стандартного ввода.

1 Описание

Требуется написать реализацию алгоритмов сжатия Хаффмана и LZW.

Определение

1. Алгоритм Хаффмана.

   Суть алгоритма состоит в том, чтобы построить коды символов размером 8 бит в зависимости от их частоты появления в тексте — чем больше частота, тем меньше код.

2. Алгоритм LZW.

   Этот алгоритм основывается не на частоте, а на запоминании последовательностей символов в словаре «код-символ» или «символ-код» в зависимости от режима работы алгоритма (кодирование/декодирование). Каждой последовательности символов сопоставляется код некоторой длины. Опишем процесс декодирования.

   На каждом шаге мы имеем возможно пустую последовательностей символов, полученную на предыдущих шагах. Эта последовательность точно уже есть в словаре. При чтении очередного символа мы проверяем, получается ли новое слово при добавлении его в последовательность. Если нет, то обновляем последовательность и переходим к следующему шагу. Если получается, что пишем код последовательности, сбрасываем её, запоминаем очередной символ и переходим к следующему шагу.

   Процесс декодирования аналогичен за исключением того нюанса, что этот процесс отстаёт на один шаг от кодирования — декодер может получить код, которого не знает. Это станет более ясным после описания процесс декодирования.

   После прочтения очередного кода мы получим новое слово, если прибавим к слову под предыдущим кодом первую букву слова под текущим кодом. Однако если очередной код был новым кодом, то этого когда нет в словаре, так как мы его ещё не добавили! Это значит только одно — новое слово это текущее слово с дополнительным первым символом на конце, так как следующее слово мы добавить можем только на этом шаге при помощи текущего слова. Таким образом, алгоритм декодирования отстаёт от алгоритма кодирования на один шаг.

Свойства и идея

Алгоритм Хаффмана

Мной был выбран алгоритм Хаффмана на адаптивной модели, а именно — алгоритм Виттера. Выбор адаптивной модели объясняется тем, что читать два раза файл, который можно читать один раз — плохо, да и в целом эта модель выигрывает у статической и полустатической.

Стоит отметить важность значения **Sibling Property** для алгоритмов Хаффмана на адаптивной модели. Это свойство, согласно которому:

1. У каждой внутренней вершины есть две дочерние вершины.
2. Частота вершины равна сумме частот её дочерних вершин.

Алгоритм Виттера является улучшением алгоритма FGK, как пишет сам Виттер (идёт речь о версии FGK, в которой нумерация символов идёт в порядке связвания узлов снизу вверх). Если представить дерево как уровни, то легко представить две неприятные ситуации:

1. В случае FGK дерево может получиться недостаточно эффективным в том случае, если частоты символов обновляются равномерно. Допустим, что сначала в файле идёт весь алфавит. Уже получаем высоту дерева 256. Далее, пусть частоты символов увеличиваются так, что на каждом шаге частота любого символа не превышает частоту символа уровнем выше. В такой ситуации дерево не изменится, так как **Sibling Property** никогда не будет нарушено. Легко видеть, что в подобной ситуации алгоритм FGK только увеличит размер исходного файла.
2. Пусть есть два типа перемещения узла с его детьми: на один уровень вверх $\uparrow$ и на один уровень вниз $\downarrow$ (большие перемещения могут происходить только в том случае, если узел будет замещать ART-символ). Перемещения типа $\downarrow$ возможны, так как на следующем шаге FGK всё рвно перейдёт к родителю этого узла и перенесёт его наверх, чтобы соблюсти **Sibling Propety**. Очевидно, что такие перемещения не имеют смысла.

Введём понятие **External path length**. Это

$$\sum_{i}^{n} l_i,$$

где l_i — длина кода i-ого символа. Понятно, что чем меньше эта величина, тем оптимальнее сжатие. Из минимальности этой величины следует минимальность максимум длин кодов $\max l_i$.

Таким образом, можно выделить 2 минуса алгоритма FGK:

1. Несбалансированность дерева, то есть неоптимальность получающегося **External path length**.
2. Неоптимальное количество перестановок узлов.

Алгоритм Виттера призван избавиться от этих минусов. Он минимизирует величину **External path length**, $\max l_i$ и количество перестановок узлов. Рассмотрим этот алгоритм.

Представим дерево в виде последовательности блоков узлов. Блоки обладают следующими свойствами:

1. Каждый блок содержит узлы веса w (частота).
2. Каждый блок содержит узлы типа t (два типа: внутренние узлы — in, и листья — l).
3. Блок веса w и типа l всегда предшествует блоку веса w и типа in.

Пусть блоки расположены в порядке возрастания. Тогда изначально в дереве один блок веса 0 и типа l, в котором лежит узел с меткой ART.

Также необходимо ввести понятие **лидер блока**. Это самый правый элемент в блоке (если блоки расположены в порядке возрастания).

Также наше дерево обладает векторами **Labels** и **RevLabels** — первый отвечает за соответствие «код-узел», а второй — за «узел-код». В качестве супремума количества узлов в дереве Виттера можно взять число 2^{10}, так как высота может быть равна 10, если в дерево добавлены все символы длины 8 бит. Однако по факту это число гораздо меньше.

В составных вещах алгоритмы FGK и Виттера практически аналогичны. Особого внимания заслуживают методы **Update(a)** и **SlideAndIncrement(p)**, где a — это очередной символ, а p — ссылка на узел.

Рассмотрим эти методы по псевдокоду из Википедии:

```
1  Update(a) {
2      leaf_to_increment = -1 // костыль,покрывающий случай,когда
3                             // узел,в котором лежит очередной символ,
4                             // брат ART-символу
5
6      p = указатель на лист,содержащий очередной символ a
7
8      IF (p это -1) THEN // этот символ не появлялся ранее
9
10        Расширить p в его блоке,добавив ему две дочерние вершины
11        // Левый ребёнок становится новым ART-символом
12        // Правый становится листом,содержащим очередной символ
13        p = Родитель нового листа,соедржащего оч. символ
14        leaf_to_increment = Правый ребёнок p
15
16     ELSE
17
18        Обменть метками узел p с лидером блока,в котором он находится
19        p = лидер блока
20        IF (p это брат ART-символа) THEN
21              leaf_to_increment = p
22              p = Родитель узла p
23
24     WHILE (p != -1)
25        SlideAndIncrement(p)
26     IF (leaf_to_increment != -1)
27        SlideAndIncrement(leaf_to_increment)
28 }
```

```
1  SlideAndIncrement(p) {
2      previous_p = Родитель p
3      w = Вес p
4      IF (p это внутренний узел) THEN
5           Прослайдить p в дереве выше чем листья веса w + 1
6           // указатель p возможно изменился (зависит от реализации)
7           Увеличить вес p на 1
8           p = previous_p
9      ELSE
10          Прослайдить p в дереве выше чем внутренние узлы веса w
11          Увеличить вес p на 1
```

```
12          p = Родитель p
13 }
```

По сути каждый узел характеризуется тремя величинами, а не двумя — тип, вес и высота. Чем правее узел в блоке, тем больше его высота.

Чтобы определить операцию "Прослайдить нужно определить атомарную оперцию "Слайд". "Слайд— это операция обмена узлов в блоке таким образом, что:

1. Узлы меняются местами в блоке
2. Узлы обмениваются связями с родителями

Тогда функцию **SlideAndIncrement(p)** можно определить как выполнение операции Слайд по всему блоку так, чтобы первым в обмене всегда выступал изначальный узел **p**.

Таким образом, выполняется правило "Чем правее узел в блоке, тем больше его высота".

В итоге получаем минимизированный **External path length** и отсутствие перемещений узлов вниз по дереву.

Что касается реализации, то сначала была реализована версия с блоками. Догадавшись о том, что блоков много, а элементов в них мало, был сделан вывод о том, что время на работу в с блоками намного превышают выгоду их использования, поэтому было решено переписать код.

Блоки стали условными абстракциями над группами узлов в векторе. Узлы в векторе находятся в порядке убывания веса, поэтому правила, связанные с блоками, незначительно поменялись, не нарушив при этом результаты работы алгоритма.

Пространственная сложность: $O(2^k)$, где $k = \lceil \log(\sum + 1) \rceil + 1$, $\sum$ — мощность алфавита.

Временная сложность: $O(N \log \sum)$, где N — количество входных символов.

Алгоритм LZW

В случае LZW выбирать не пришлсоь.

Главными вопросами были:

1. Контроль длины кода

   Было решено увеличивать длины посылаемого когда на 1 по мере необходимости. Также ключами -1..9 можно регулировать максимальное количество кодов таблице, при достижении которого таблица полностью сбрасывается (остаются только символы алфавита).

2. Словари для кодирования/декодирования

   Выбор способа хранения словарей отталкивался лишь от скорости.

   Для кодирования в итоге используется несжатый **trie**, который позволяет добавлять символы в последнюю посещённую или произвольную вершину. Можно было бы использовать сжатый **trie**, однако при определённых данных с такой структурой можно и проиграть по времени.

   Для декодирования используется вектор со строками. Ничего хитрого. Можно было бы использовать тот же несжатый **trie**, дополненный таблицей соответствия кодов узлам, но, опять же, вопрос о памяти стоял не так строго, чтобы это хотелось реализовать.

Пространственная сложность: $O(N)$

Временная сложность: $O(N)$

2 Исходный код

Этапы написания кода:

1. THuff — реализация алгоритма Виттера.
2. TLZW — реализация алгоритма LZW.
3. TBitIO — реализация ввода/вывода произвольного количества бит.
4. TPrefTree — реализация несжатого **trie** для TLZW.
5. TGZIP — реализация класса, который отвечает за использование вышеуказанных алгоритмов сжатия на произвольном количестве файлов.
6. main.cpp — расфасовка данных из командной строки.

THuff

```
#pragma once

#include <iostream>
#include <fstream>
#include <unordered_map>
#include <string>
#include <vector>
#include <sstream>
#include <filesystem>
#include <algorithm>
#include <list>
#include "TBitIO.hpp"
#include <exception>
#include <memory>

class THuff {
    public:
        typedef int TCodeType;

        THuff();
    void PrepForWork(const std::string &sName, const std::string &rName, char keys);
    void Clear();
    void SetUncompSize(std::size_t size);
        void Encode(TBitIO &Source);
        void Decode(TBitIO &Result);
    std::size_t UncompSize();
    std::size_t CompSize();
    std::string InfoName();
        ~THuff();
    private:
    struct THData {
      THData *Parent;
      THData *LeftChild;
      THData *RightChild;

      bool Type; // 0 - leaf, 1 - internal
      std::size_t Weight;
      int Index;

      THData(THData *parent, bool type, std::size_t weight, int index);
      bool IsLeft();
      bool IsSiblingToNYT();
      ~THData() {}

      static void Slide(int ia, int ib);
    };

```

```
48      friend THData;
49
50      struct THAData : public std::vector<THData*> {
51        THData* IChangeWLeader(int i);
52        THData* Slide(int i);
53      };
54
55      struct THAAllocator : public std::vector<THData> {
56        int Used;
57        THData* Get();
58      };
59
60      static THAAllocator Emptys;
61      static THAData Data;
62
63          TCodeType ART;
64          char Keys;
65
66      std::string SName;
67          TBitIO Result;
68          TBitIO Source;
69          std::fstream Fin;
70          std::fstream Fout;
71
72      unsigned int Mask;
73      unsigned int Buffer;
74
75          static std::vector<int> Labels;
76          static std::vector<TCodeType> RevLabels;
77
78      void Upd(TCodeType a);
79      void SlideAndInc(THData* &p);
80          void EncSym(TCodeType a);
81          void AddNode(TCodeType a);
82          TCodeType DecSym();
83          TCodeType ReceiveSym();
84      void SendEOF();
85
86      void Print();
87      void PrintHelper(THData *node, unsigned long long space);
88
89      std::size_t Compressed;
90          std::size_t Uncompressed;
91      static const int CHECK;
92      std::string IName;
93  };
```

TLZW

```
#pragma once

#include <iostream>
#include <fstream>
#include <unordered_map>
#include <string>
#include <sstream>
#include <filesystem>
#include "TBitIO.hpp"
#include <exception>
#include "TPrefTree.hpp"

class TLZW {
    public:
        typedef int TCodeType;
        TLZW();
    void PrepForWork(const std::string &sName, const std::string &rName, char keys);
        void Encode(TBitIO &Result);
        void Decode(TBitIO &Source);
    void Clear();
        unsigned long long UncompressedSize();
    ~TLZW();
    private:

        class EncodeTable : public TPrefTree {
            private:
                unsigned long long GivenSize;
            public:
                void SetSize(const unsigned long long &givenSize);
                void AddWord(std::string &str);
        void AddWord(std::string &str, std::size_t nodeNum);
        };

        class DecodeTable : public std::vector<std::string> {
            private:
                unsigned long long GivenSize;
            public:
                void SetSize(const unsigned long long &givenSize);
                void AddCode(std::string &str);
                void Clear();
        std::size_t Size();
        };

    void SendBits(TCodeType bufToSend, TBitIO &Result);
    TCodeType ReceiveBits(TBitIO &Source);
    void EncCheckCodeLength();
    bool IncCodeLength();
```

```
48
49          int ToSR;
50      unsigned long long Sup;
51
52          static const unsigned long long Fast;
53          static const unsigned long long Best;
54          char Keys;
55          EncodeTable CodeTable;
56          DecodeTable WordTable;
57      std::ostream *Result;
58          std::istream *Source;
59      std::fstream Fin;
60          std::fstream Fout;
61
62          unsigned long long Uncompressed;
63  };
```

TBitIO

```
#pragma once

#include <iostream>
#include <string>

// Send bits from string
// Receive 1 bit
// Receive n bits
// Send n bits

class TBitIO {
  private:
    std::istream *In;
    std::ostream *Out;
    unsigned char Buffer;
    int Used;
    bool __EOF;
  public:
    TBitIO() {}
    TBitIO(std::istream *ptr);
    TBitIO(std::ostream *ptr);
    void seekp(std::streamoff pos, std::ios_base::seekdir way);
    std::streampos tellp();
    void read(char* ptr, std::size_t n);
    void write(char* ptr, std::size_t n);
    void reopen(std::istream *ptr);
    void reopen(std::ostream *ptr);
    bool good();
    bool Good();
    TBitIO& operator<<(std::string &str);
    void Send(int from, unsigned char n);
    void Send(unsigned char from, unsigned char n);
    void Send(std::size_t from, unsigned char n);
    unsigned char Recv(char n);
    int Recv(unsigned char n);
    std::size_t Recv(int n);
    void Close();
};
```

TPrefTree

```
#pragma once

#include <map>
#include <vector>
#include <string>

struct SearchRes {
  int Code;
  std::size_t NodeNum;
};

class TPrefTree {
  private:
    class TPNode {
      friend TPrefTree;
      private:
        std::vector<std::size_t> NextPoints;
        int Code;
      public:
        TPNode();
        TPNode(const TPNode &right);
        TPNode(TPNode &&right);
        TPNode& operator=(const TPNode &right);
        TPNode& operator=(TPNode &&right);
        std::size_t find(unsigned char c);
        std::size_t add(unsigned char c, int point);
    };
    std::vector<TPNode> Points;
    int NextCode;
  public:
    TPrefTree();
    void Clear();
    void Add(std::string &str);
    void Add(std::size_t nodeNum, char c);
    SearchRes Get(std::string &str);
    SearchRes Get(std::size_t nodeNum, char c);
    SearchRes Get(std::size_t nodeNum);
    int Size();
};
```

TGZIP

```
#pragma once

#include <iostream>
#include <string>
#include <unordered_map>
#include "THuff.hpp"
#include "TLZW.hpp"
#include <vector>
#include <filesystem>
#include <fstream>
#include <sstream>
#include <cmath>
#include "TBitIO.hpp"

class TGZIP {
    struct TGZData {
    std::string Name;
        std::size_t Compressed;
        std::size_t Uncompressed;
    };
    private:
    THuff Huff;
    TLZW LZW;

        char Key;
        std::vector<TGZData> Results;
        void PrintInfo();
    void Encode();
    void Decode();
    void DumpFromIn();
    void DumpToOut();
    public:
        TGZIP(std::string &name, std::string &resName, char key);
    ~TGZIP();
};
```

main.cpp

```cpp
#include <iostream>
#include <string>
#include "TGZIP.hpp"

// /-t/-r/-l/-k/-d/-/-c/-1 -9/

char GetKeys(std::string S) {
    char key = 0;
    if (S.size() == 1)
        return 1 << 2;
    for (int i = 1; i < S.size(); ++i)
        switch (S[i]) {
            case '9':
                key |= 1;
                break;
            case '8':
                key |= 1;
                break;
            case '7':
                key |= 1;
                break;
            case '6':
                key |= 1;
                break;
            case '5':
                key |= 1;
                break;
            case 'c':
                key |= (1 << 1);
                break;
            case 'd':
                key |= (1 << 3);
                break;
            case 'k':
                key |= (1 << 4);
                break;
            case 'l':
                key |= (1 << 5);
                break;
            case 'r':
                key |= (1 << 6);
                break;
            case 't':
                key |= (1 << 7);
                break;
        }
    return key;
```

```
}

int main(int argc, char const *argv[]) {
  std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int flag = 0;
    char key = 0;
    std::string name;
    for (int i = 1; i < argc; ++i)
        if (argv[i][0] == '-')
            key |= GetKeys(std::string(argv[i]));
        else if (argv[i][0] == '>' || argv[i][0] == '<')
            continue;
        else if (!flag) {
            name = std::string(argv[i]);
            flag = 1;
        }
    std::string resName = name;

  if (key & ((1 << 5) | (1 << 7)))
    key |= ((1 << 3) | (1 << 4));

    if (!name.empty() && !(key & 1 << 6)) {
    if (key & 1 << 3) {
      if (name.compare(name.size() - 3, 3, ".gz") != 0)
        name += ".gz";
      else
        resName.erase(resName.begin() + resName.size() - 3, resName.end());
    } else {
      if (name.compare(name.size() - 3, 3, ".gz") == 0)
            resName.erase(resName.begin() + resName.size() - 3, resName.end());
         else
            resName += ".gz";
    }
    }

  try {
      TGZIP Arch(name, resName, key);
  }
  catch (const std::runtime_error&) {}

  return 0;
}
```

3 Консоль

```
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ touch filename
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ nano filename
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ head filename
me spin me win
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ./prog -c -<filename >filename.gz
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l filename.gz
-rw-rw-r--1 a3nippo a3nippo 54 июн 18 19:45 filename.gz
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ./prog -c -d -<filename.gz
me spin me win
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$
```

4 Тест производительности

Входные данные : 230мб повторений строки длинной 250 (t1), 5мб pdf-файла (t2), 2гб бинарного файла (возможно файл уже сжат) (t3).

```
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ time ./prog testfile1

real 1m36.582s
user 1m36.130s
sys 0m0.456s
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l testfile1.gz
-rw-rw-r--1 a3nippo a3nippo 693867 июн 18 19:57 testfile1.gz
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l t1
-rw-rw-r--1 a3nippo a3nippo 277348352 июн 18 19:55 t1
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ time ./prog -d testfile1

real 0m1.027s
user 0m0.308s
sys 0m0.719s
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ cmp testfile1 t1
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ time ./prog testfile2.pdf

real 0m4.289s
user 0m3.255s
sys 0m1.031s
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l testfile2.pdf.gz
-rw-rw-r--1 a3nippo a3nippo 4715498 июн 18 19:59 testfile2.pdf.gz
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l t2
-rw-rw-r--1 a3nippo a3nippo 4769871 июн 18 19:55 t2
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ time ./prog -d testfile2.pdf

real 0m0.959s
user 0m0.923s
sys 0m0.036s
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ cmp testfile2.pdf t2
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ time ./prog testfile3

real 26m21.558s
user 23m45.288s
sys 2m36.132s
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l testfile3.gz
```

```
-rw-rw-r--1 a3nippo a3nippo 2486603285 июн 18 20:27 testfile3.gz
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ ls -l t3
-rw-rw-r--1 a3nippo a3nippo 2134804438 июн 18 19:55 t3
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ time ./prog -d testfile3

real 8m47.425s
user 8m38.393s
sys 0m8.935s
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$ cmp testfile3 t3
a3nippo@mopsy:~/LABS/DA_LABS/KP/KP$
```

5 Выводы

1. Не хватает аккуратности.
2. Нужно увеличить время на обдумывание кода.
3. Впервые написал алгоритм по документации. Тяжко.

В итоге код был написан так, чтобы работал, а не так, как бы я хотел. А хотелось бы, чтобы:

1. ...у классов архиваторов был общий виртуальный класс с методами, которые в данной версии есть у обоих архиваторов.
2. ...всей работой с какими-либо файлами и потоками занимался класс TGZIP. В данной версии этот класс занимается только потоками и манипулированием архиваторами.

После проделанной работы хочется написать КП по всем остальным темам.

Список литературы

[1] *JEFFREY SCOTT VITTER «Design and Analysis of Dynamic Huffman Codes»* URL:`http://www.ittc.ku.edu/~jsv/Papers/Vit87.jacmACMversion.pdf` (дата обращения: xx.05.2019).

[2] *Juha Nieminen «An efficient LZW implementation»* URL:`http://warp.povusers.org/EfficientLZW/` (дата обращения: xx.05.2019).