Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования.

Курсовой проект по курсу «Дискретный анализ»

Студент: П. А. Харьков

Преподаватель: С. А. Сорокин

Группа: М8О-206Б-19

Дата: _____

Оценка: _____

Подпись: _____

Москва, 2021

Введение

Задача: реализовать алгоритм сжатия данных LZ-77 на языке программирования C++.

Алгоритм сжатия без потерь LZ-77 был опубликован израильскими математиками Авраамом Лемпелем и Яаковым Зива в 1977 году. Этот алгоритм относится к словарным методам сжатия, то есть к тем, которые не кодируют каждый символ по отдельности, а кодируют по подстрокам текста.

В данном алгоритме используется «скользящее окно», то есть учитывает информацию, которая до этого уже была закодирована на определенном расстоянии. Закодированный текст представляет из себя набор троек: смещение относительно найденной подстроки в окне и текущей, длину текущей подстроки и символ, несовпавший с символом подстроки в окне.

Для того, чтобы поиск в «скользящем окне» имел линейную сложность, я строю для него суффиксное дерево с помощью алгоритма Укконена и после успешно найденной подстроки в нем, сохраняю указатель на текущее состояние поиска в окне, чтобы поиск занимал константное время. Благодаря этому, сложность сжатия текста линейная.

Алгоритм работы программы

Алгоритм сжатия текста:

- Считывается символ из стандартного потока ввода.
- Символ добавляется в суффиксное дерево, то есть в «скользящее окно».
- Затем ищется индекс начала подстроки в окне, которое совпадает с текущей подстрокой.
    - Если он находится, то переходим к п. 1.
    - Если же он не находится, то записываем в результаты кодирования индекс, длину подстроки без последнего символа и последний считанный символ.
- Если длина окна превысила ограничение, то очищаем суффиксное дерево.

Алгоритм восстановление текста:

1. Считываем закодированную тройку.
2. Сдвигаем указатель влево на смещение по уже восстановленному тексту.
3. Копируем все символы, начиная с поставленного указателя, считанной длины в конец восстановленного текста.
4. Добавляем в конец восстановленного текста считанный несовпавший символ.

Код программы

kp.cpp

```cpp
#include <iostream>
#include <string>
#include "lz77/lz77.hpp"

using namespace std;
int main(){
    string type;
    cin >> type;
    if(type == "compress"){
        LZ77::CompressText();
    }else if(type == "decompress"){
        LZ77::DecompressText();
    }

    return 0;
}
```

lz77.hpp

```cpp
#pragma once
namespace LZ77
{
    void CompressText();
    void DecompressText();

};
```

lz77.cpp

```cpp
#include <iostream>
#include <string>
#include "lz77.hpp"
#include "sufftree/tree.hpp"

int Buffer = 32768;

namespace LZ77{
void CompressText(){
    NSuffTree::TTree tree;
    int textLen = 0;
    int prevRes = 0;
    int curLen = 0;

    char c;
    while(cin >> c){
        int curRes;
        tree.Add(c);
        textLen++;
        if((curRes = tree.FindLastChar()) != -1){
            curLen++;
            prevRes = curRes;
            continue;
        }
        if (curLen == 0) {
            cout << 0 << " " << 0 << c;
        }else {
            cout << (textLen - 1) - prevRes << " " << curLen << c;
        }
        curLen = 0;
        prevRes = 0;
        if(textLen > Buffer){
```

```
            tree.Deinit();
            tree.Init();
            textLen = 0;
        }
    }

    if (curLen != 0) {
        cout << textLen - prevRes << " " << curLen - 1 << c;
    }

}

void DecompressText(){
    string text;
    int offset;
    int len;
    char c;

    int textLen = 0;
    while (cin >> offset >> len >> c) {
        for (int i = 0; i < len; i++) {
            text += text[textLen - offset];
            textLen++;
        }

        text += c;
        textLen++;
        if(textLen > Buffer){
            cout << text;
            textLen = 0;
            text = "";
        }
    }
    if(text != ""){
        cout << text;
    }

  }
}
```

tree.hpp

```
#pragma once

#include <string>
#include <unordered_map>

using namespace std;

namespace NSuffTree{

    struct TNode{
        int Lpos, Rpos;
        TNode* Link;
        unordered_map<char, TNode*> Next;

        TNode(int lpos, int rpos):Lpos(lpos), Rpos(rpos){
            Link = nullptr;
        }
    };

    struct Pos{
        TNode* CurNode;
        int CurLen;
        int CurEdge;
    };

    class TTree{
```

```
        private:
            TNode* Root;
            int End, Rem;
            Pos pos;
            Pos findpos;

            TNode* FindNode(TNode* NodeToFind, char c);
            int EdgeLen(TNode* node);
            void DeinitNode(TNode* node);
        public:
            string Text;
            void Init();
            void Deinit(){
                DeinitNode(Root);
            }
            void Add(char c);
            int FindLastChar();

            TTree(){
                Init();
            }
            ~TTree(){
                DeinitNode(Root);
            }
    };

}
```

tree.cpp

```
#include "tree.hpp"
const char SENTI = '$';

namespace NSuffTree{
    void TTree::Init(){
        Text = "";
        Root = new TNode(0, 0);
        Root->Link = Root;
        findpos.CurNode = Root;
        findpos.CurLen = 0;
        findpos.CurEdge = 0;
        pos.CurNode = Root;
        pos.CurLen = 0;
        pos.CurEdge = 0;
        Rem = 0;
        End = 0;
    }

    TNode* TTree::FindNode(TNode* NodeToFind, char c){
        auto found = NodeToFind->Next.find(c);
        if(found != NodeToFind->Next.end()){
            return found->second;
        }
        return nullptr;
    }

    int TTree::EdgeLen(TNode* node){
        if (node->Rpos == -1) {
            return End - node->Lpos + 1;
        }
        return node->Rpos - node->Lpos;
    }


    void TTree::Add(char c){
        Text += c;
        TNode* nodeForSuff = nullptr;
```

```
        Rem++;

        if (Rem == 1) {
            pos.CurEdge = End;
        }

        while (Rem > 0){
            TNode* nextNode = FindNode(pos.CurNode, Text[pos.CurEdge]);
            if (nextNode == nullptr){
                pos.CurNode->Next[Text[pos.CurEdge]] = new TNode(End, -1);
                if (nodeForSuff != nullptr){
                    nodeForSuff->Link = pos.CurNode;
                }
                nodeForSuff = pos.CurNode;
            }else{
                int edgeLen = EdgeLen(nextNode);

                if (pos.CurLen >= edgeLen){
                    pos.CurNode = nextNode;
                    pos.CurLen -= edgeLen;
                    pos.CurEdge += edgeLen;
                    continue;
                }

                if (Text[nextNode->Lpos + pos.CurLen] == Text[End]){
                    pos.CurLen++;
                    if (nodeForSuff != nullptr){
                        nodeForSuff->Link = pos.CurNode;
                    }
                    break;
                }

                TNode* midNode = new TNode(nextNode->Lpos, nextNode->Lpos + pos.CurLen);
                pos.CurNode->Next[Text[pos.CurEdge]] = midNode;
                midNode->Next[Text[End]] = new TNode(End, -1);
                nextNode->Lpos += pos.CurLen;
                midNode->Next[Text[nextNode->Lpos]] = nextNode;

                if (nodeForSuff != nullptr){
                    nodeForSuff->Link = midNode;
                }
                nodeForSuff = midNode;
            }


            if (pos.CurNode == Root && pos.CurLen > 0){
                pos.CurEdge++;
                pos.CurLen--;
            }else if (pos.CurNode != Root){
                pos.CurNode = pos.CurNode->Link;
            }

            Rem--;
        }

        End++;
    }

    int TTree::FindLastChar(){
        if(findpos.CurLen == 0){
            findpos.CurEdge = End - 1;
        }

        TNode* nextNode = FindNode(findpos.CurNode, Text[findpos.CurEdge]);
        if (nextNode == nullptr){
            findpos.CurNode = Root;
            findpos.CurLen = 0;
        }else{
            int edgeLen = EdgeLen(nextNode);
```

```
            if (findpos.CurLen == edgeLen){
                findpos.CurNode = nextNode;
                findpos.CurLen -= edgeLen;
                findpos.CurEdge += edgeLen;
                nextNode = FindNode(findpos.CurNode, Text[findpos.CurEdge]);
            }

            if (nextNode != nullptr && Text[nextNode->Lpos + findpos.CurLen]
                           == Text[End - 1] && nextNode->Lpos + findpos.CurLen != End - 1) {
                findpos.CurLen++;
                return nextNode->Lpos + findpos.CurLen;
            }
            else {
                findpos.CurNode = Root;
                findpos.CurLen = 0;
            }
        }

        return -1;

    }


    void TTree::DeinitNode(TNode* node){
        for(auto next: node->Next){
            DeinitNode(next.second);
        }
        delete node;
    }
}
```

Результаты работы программы

В результате работы программы для строки *ababbbbbaaaaaaaaaaaaaaabababbbbaaaaaaabab* мы получили такие тройки:

- 0 0 a
- 0 0 b
- 2 2 b
- 2 3 a
- 1 14 b
- 22 5 a
- 12 7 b

Так как каждый латинский символ можем кодировать 1 байтом, то для хранения нашей начальной строки нам нужно 38 байт. В результате кодирования же мы могли получить 24 символа, если не считать пробелы и переводы строк. Однако для большей экономии памяти я храню эти тройки в таком формате: 0 0a0 0b2 2b2 3a1 14b22 5a12 7b, то есть, для закодированного текста нам понадобился 31 байт.

Также я провел тест для 2 размеров скользящего окна: максимальная длина скользящего окна в первом тесте равна 16384 символам. Во втором тесте – 32768. Для обоих тестов используется один и тот же текст, состоящий из 100 тысяч символов ‘a’ и ‘b’. Размером файла после сжатия я называю его размер файла, в который записывается вывод моей программы.

Результаты первого теста:

1. Время сжатия текста: 105 миллисекунд.
2. Размер файла после сжатия: 56816 байт, то есть коэффициент сжатия 44%.

Результаты второго теста:

1. Время сжатия текста: 101 миллисекунда.
2. Размер файла после сжатия: 55637 байт, то есть коэффициент сжатия 45%.

Время восстановления текст после первого и второго теста занимает около 1 миллисекунды.

Третий тест состоит из 1 миллиона символов ‘a’ и ‘b’ и длина скользящего окна равна 32768 символам:

1. Время его сжатия: 1158 миллисекунд.
2. Размер файла после сжатия: 554 тысяч байт, то есть коэффициент сжатия так же 45%.

Также я провел тест, состоящий из 100 тысяч символов и всех букв латинского алфавита, но тогда сжатие, к сожалению, не уменьшало размер файл. Для того, чтобы оно уменьшало можно преобразовать текст, к примеру, методом Барроуза Уилера.

Выводы

Выполнив данный курсовой проект, я научился реализовывать алгоритм LZ-77 для сжатия текстов. Также во время написания кода программы я нашел решения для лучшей степени сжатия: первое решение – это не полное обнуление суффиксного дерева, а удаление только листьев, индексы которых удалены на длину скользящего окна относительно текущего индекса. Это позволяет с большей вероятностью кодировать подстроки, так как скользящее окно всегда будет существовать. Второе решение – это увеличивать длину скользящего окна, так как тогда мы сможем кодировать подстроки большей длины, что увеличивает степень сжатия, но увеличивает затраты по памяти и сложность алгоритма. Алгоритм LZ-77 может не быть самым оптимальным, так как не анализирует весь текст, а смотрит только набор предыдущих символов. Однако преимуществом является то, что нет необходимости передавать декодеру какие-либо данные, текст можно восстановить только с помощью кода.

Список литературы

1. Гасфилд Дэн. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология / Пер. с англ. И.В. Романовского. – СПб.: Невский диалект; БХВ-Петербург, 2003. – 654 с.: ил.
2. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2003. – 384 с.