

МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра САПР

ОТЧЕТ
по лабораторной работе № 2
по дисциплине «Алгоритмы кодирования»

Студент гр. 8301

Готовский К.В.

Преподаватель

Тутева А.В.

Санкт-Петербург

2020

Цель работы

Реализовать кодирование и декодирование по алгоритму Шеннона-Фано.

Описание реализуемого класса

Класс `Shannon_Fano`, содержит вложенный класс `Node`, который в свою очередь имеет три публичных поля: указатели на `Node *Left` (Указатель на левый элемент), `Node *Right` (Указатель на правый элемент) и переменную `value` класса `Pair` (вспомогательный класс, состоящий из двух публичных шаблонных полей). А сам класс имеет поля `Node* head` (вершина дерева) и `List<Pair<char,int>>*` `list_symbol` (список символов в дереве). Сам класс подразумевает собой дерево кодирования построенное на основе алгоритма кодирования Шеннона-Фано. Класс содержит следующие методы:

- Конструктор – который получает на вход строку, и строит список типа `Node` символов и их частоты появления в строке. Потом этот список передаётся в функцию `build_tree`.
- Деструктор – реализован деструктор, который вызывает метод `clear` (на основе обычного удаления двоичного дерева).
- `Node* build_tree(Node*& root, List<Pair<char, int>>*& list)` – функция построения дерева кодирования. Само добавление проходит по следующему алгоритму: проверяем, есть ли хотя бы один элемент списка для создания дерева; если элемент один, делаем его корнем и всё; иначе разделяем на 2 список и уходим в рекурсию до тех пор, пока листьями не будут односимвольные элементы.
- `Map<char,string>*& get_tree_with_code()` – функция получения ассоциативного массива символов и их кодов. Для этого вызывается функция `fill_tree`.
- `void fill_tree (Map<char, string>*& shannon, Node* root, string cur)` – функция заполнения ассоциативного массива, ключом которого является символ, присутствующий в строке, а значением код символа (каждый символ – это элемент, не имеющий сыновей). Методом Шеннона-Фано мы определяем код каждого элемента и записываем его в ассоциативный массив.
- `List<Pair<char, int>>*& get_list_symbol()` – функция получения списка пар символов и их частот появления в строке.
- `void Decoding(Node* root, string& coding_str, string& decoding_str, int& position)` – функция декодирования закодированной строки. Мы идём по закодированной строке и, если встречаем 0, то идём влево, иначе – вправо, и, если встречаем символ, то возвращаемся в корень, не меняя позиции продвижения по строке, тем самым декодируем закодированную строку.

Оценка временной сложности алгоритмов

- `build_tree (List<Node>*& list_for_build_tree)` – $O(N \log n)$
- `get_tree_with_code()` – $O(N)$
- `fill_tree (Map<char, string>*& shannon, Node* root, string cur)` – $O(N)$
- `get_list_symbol()` – $O(1)$
- `Decoding(Node* root, string& coding_str, string& decoding_str, int& position)` – $O(N \log n)$

Описание реализованных unit-тестов

Реализованные мною тесты проверяют правильность кодирования информации методом Шенона-Фано. В них мы рассматриваем такие случаи, как предложение, слова из разных символов и слова из повторяющихся символов.

Пример работы программы

Пример

Консоль отладки Microsoft Visual Studio

```
Hello world
{d}=={0000} с частотой: 1
{ }=={0001} с частотой: 1
{H}=={001} с частотой: 1
{o}=={010} с частотой: 2
{e}=={011} с частотой: 1
{l}=={100} с частотой: 3
{r}=={101} с частотой: 1
{w}=={11} с частотой: 1
старая строка: Hello world
новая строка: 0010111001000100001110101011000000
кол-во символов: 34
Коэффициент сжатия: x2.58824
C:\Users\1\Downloads\учёба\АиСД\2 лаба\Debug\AISDLab2var2.exe (процесс 128) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" -
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

Консоль отладки Microsoft Visual Studio

```
qqqqq
{q}=={0} с частотой: 5
старая строка: qqqqq
новая строка: 00000
кол-во символов: 5
Коэффициент сжатия: x8
C:\Users\1\Downloads\учёба\АиСД\2 лаба\Debug\AISDLab2var2.exe (процесс 128) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" -
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

Консоль отладки Microsoft Visual Studio

```
it is just string
{t}=={0000} с частотой: 3
{g}=={0001} с частотой: 1
{r}=={001} с частотой: 1
{i}=={010} с частотой: 3
{u}=={011} с частотой: 1
{ }=={100} с частотой: 3
{n}=={101} с частотой: 1
{s}=={110} с частотой: 3
{j}=={111} с частотой: 1
старая строка: it is just string
новая строка: 0100000100010110100111011110000010011000000010101010001
кол-во символов: 55
Коэффициент сжатия: x2.47273
C:\Users\1\Downloads\учёба\АиСД\2 лаба\Debug\AISDLab2var2.exe (процесс 128) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" -
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

Листинг

Shannon_Fano.h

```
1. #pragma once
2. #include "Map.h"
3. #include "List.h"
4. #include "Pair.h"
5. #include "string"
6. class Shannon_Fano
7. {
8.     class Node {
9.     public:
10.         Node(Pair<char, int> value = Pair<char, int>(), Node* left = NULL, Node* right = NULL) :value(value) {}
11.         Pair<char, int> value;
12.         Node* left;
13.         Node* right;
14.     };
15. public:
16.     Shannon_Fano(string str) {
17.         Map<char, int>* map_symbol = new Map<char, int>();
18.         list_symbol = new List<Pair<char, int>>();
19.         for (int i = 0; i < str.size(); i++) {
20.             if (!map_symbol->find_is(str[i]))//если символа нет в списке тогда добавляем значение в карту
21.             {
22.                 map_symbol->insert(str[i], 1);
23.             }
24.             else
25.                 map_symbol->increment_value(str[i]);
26.         }
27.         List<Pair<char, int>>* list_of_pairs = map_symbol->get_pairs();
28.         map_symbol->clear();//больше не нужен только занимает теперь память
29.         list_of_pairs->sort();//сортируем для дальнейшего оборачивания
30.         build_tree(head, list_of_pairs);
31.         ///////////////////////////////////////////////////построение дерева кодирования
32.         List<Pair<char, int>>* get_list_symbol() {
33.             return list_symbol;
34.         }
35.         Map<char, string>* fill_tree_with_code() {
36.             Map<char, string>* shennon = new Map<char, string>();
37.             string cur;
38.             fill_tree(shennon, head, cur);
39.             return shennon;
40.         }
41.         string Decoding_shennon_tree(string& coding_str) {
42.             string decoding_str;
43.             int pos = 0;
44.             Decoding(head, coding_str, decoding_str, pos);
45.             return decoding_str;
46.         }
47. private:
48.         void Decoding(Node* root, string& coding_str, string& decoding_str, int& position) {
49.             if (coding_str.size() > position) {
50.                 while (root->right != NULL && root->left != NULL) {
51.                     if (coding_str[position] == '0')
52.                         root = root->left;
53.                     else
54.                         root = root->right;
55.                     position++;
56.                 }
57.                 decoding_str += root->value.first;
58.                 if (head->left == NULL && head->right == NULL)
59.                     position++;
60.                 Decoding(head, coding_str, decoding_str, position);
61.             }
62.         }
63.         Node* build_tree(Node*& root, List<Pair<char, int>>*& list) { //построение самого дерева
```

```

64.     if (list->get_size() >= 1) {
65.
66.
67.         if (list->get_size() == 1) {
68.             list_symbol->push_back(list->at(0));
69.             root = new Node(list->at(0));
70.         }
71.         else {
72.             root = new Node();
73.             if (list->get_size() > 1) {
74.                 List<Pair<char, int>>* list1 = list;
75.                 List<Pair<char, int>>* list2 = new List<Pair<char, int>>();
76.                 absolute_cut(list1, list2);
77.                 if (list->isEmpty()) {
78.                     root->left = build_tree(root->left, list1);
79.                     root->right = build_tree(root->right, list2);
80.                 }
81.             }
82.         }
83.         return root;
84.     }
85. }
86. void fill_tree(Map<char, string>* shennon, Node* root, string cur) { //заполнение дерева кодов
87.     if (head->left != NULL && head->right != NULL) {
88.         if (root->left != NULL && root->right != NULL) {
89.             fill_tree(shennon, root->left, cur + '0');
90.             fill_tree(shennon, root->right, cur + '1');
91.         }
92.         else
93.         {
94.             shennon->insert(root->value.first, cur);
95.         }
96.     }
97.     else {
98.         shennon->insert(root->value.first, cur+'0');
99.     }
100.
101. }
102. void absolute_cut(List<Pair<char, int>>*& list1, List<Pair<char, int>>*& list2) { //разделение н
103.     int cost = 0;
104.     for (int i = 0; i < list1->get_size(); ++i) {
105.         cost += list1->at(i).second;
106.     }
107.     list1->sort();
108.     list1->reverse();
109.     int cost_list1 = cost;
110.     int cost_list2 = 0;
111.     //////////////////////////////////////
112.     for (int i = 0; i < list1->get_size() && list1->isEmpty(); ++i) {
113.         if (list1->at(i).second + cost_list2 <= cost / 2) {
114.             cost_list2 += list1->at(i).second;
115.             list2->push_back(list1->at(i));
116.             list1->remove(i);
117.         }
118.     }
119.     cost_list1 = cost_list1 - cost_list2;
120.     if (list1->get_size() == 0 || cost_list2 == cost / 2)
121.         for (int i = 0; i < list1->get_size() && list1->isEmpty(); ++i) {
122.             for (int j = 0; j < list2->get_size() && list2->isEmpty(); ++j) {
123.                 if ((cost_list2 - list2->at(j).second + list1->at(i).second > cost_list2) && (c
124.                     at(j).second + list1->at(i).second) <= cost / 2) {
125.                     cost_list2 = cost_list2 - list2->at(j).second + list1->at(i).second;
126.                     cost_list1 = cost_list1 + list2->at(j).second - list1->at(i).second;
127.                     Pair<char, int> cur1 = list1->at(i);
128.                     Pair<char, int> cur2 = list2->at(j);
129.                     list1->set(i, cur1);
130.                     list2->set(j, cur2);

```

```

130.         }
131.     }
132.     if (cost_list2 == cost / 2)
133.         break;
134. }
135. //////////////////////////////////////////////////
136. }
137. Node* head;
138. List<Pair<char, int>>* list_symbol;
139. };

```

UnitTestitFano.cpp

```

1. #include "pch.h"
2. #include "CppUnitTest.h"
3. #include<string>
4. #include"\Users\1\source\repos\Lab 2\Shannon_Fano.h"
5. #include"\Users\1\source\repos\Lab 2\Map.h"
6. #include"\Users\1\source\repos\Lab 2>List.h"
7.
8. using namespace Microsoft::VisualStudio::CppUnitTestFramework;
9.
10. namespace UnitTestitFano
11. {
12.     TEST_CLASS(UnitTestitFano)
13.     {
14.     public:
15.
16.         TEST_METHOD(TestCodingFano_different_symbol_words)
17.         {
18.             string str = "Hello world";//
19.             string coding_str;
20.             Shannon_Fano* Shannon_Fano_tree = new Shannon_Fano(str);
21.             Map<char, string>* Fano = Shannon_Fano_tree->fill_tree_with_code();
22.             int counter = 0;
23.             List<Pair<char, int>>* list_symbol = Shannon_Fano_tree->get_list_symbol();
24.             for (int i = 0; i < str.size(); i++)
25.                 coding_str += Fano->find(str[i]);
26.             Assert::AreEqual(coding_str, string("0010111001000100001110101011000000"));
27.         }
28.         TEST_METHOD(TestCodingFano_same_symbol_words)
29.         {
30.             string str = "qqqqq";//
31.             string coding_str;
32.             Shannon_Fano* Shannon_Fano_tree = new Shannon_Fano(str);
33.             Map<char, string>* Fano = Shannon_Fano_tree->fill_tree_with_code();
34.             int counter = 0;
35.             List<Pair<char, int>>* list_symbol = Shannon_Fano_tree->get_list_symbol();
36.             for (int i = 0; i < str.size(); i++)
37.                 coding_str += Fano->find(str[i]);
38.             Assert::AreEqual(coding_str, string("00000"));
39.         }
40.         TEST_METHOD(TestCodingFano_text)
41.         {
42.             string str = "it is just string";//
43.             string coding_str;
44.             Shannon_Fano* Shannon_Fano_tree = new Shannon_Fano(str);
45.             Map<char, string>* Fano = Shannon_Fano_tree->fill_tree_with_code();
46.             int counter = 0;
47.             List<Pair<char, int>>* list_symbol = Shannon_Fano_tree->get_list_symbol();
48.             for (int i = 0; i < str.size(); i++)
49.                 coding_str += Fano->find(str[i]);
50.             Assert::AreEqual(coding_str, string("0100000100010110100111011110000010011000000010101010001"));
51.         }
52.
53.

```

```
54.     };  
55. }  
56.
```

Вывод

В данной лабораторной работе я ознакомился с методом кодирования Шенона-Фано, а также закрепил свои навыки в объектно-ориентированном программировании.