МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе № 2

по дисциплине «Алгоритмы и структуры данных»

Вариант №1

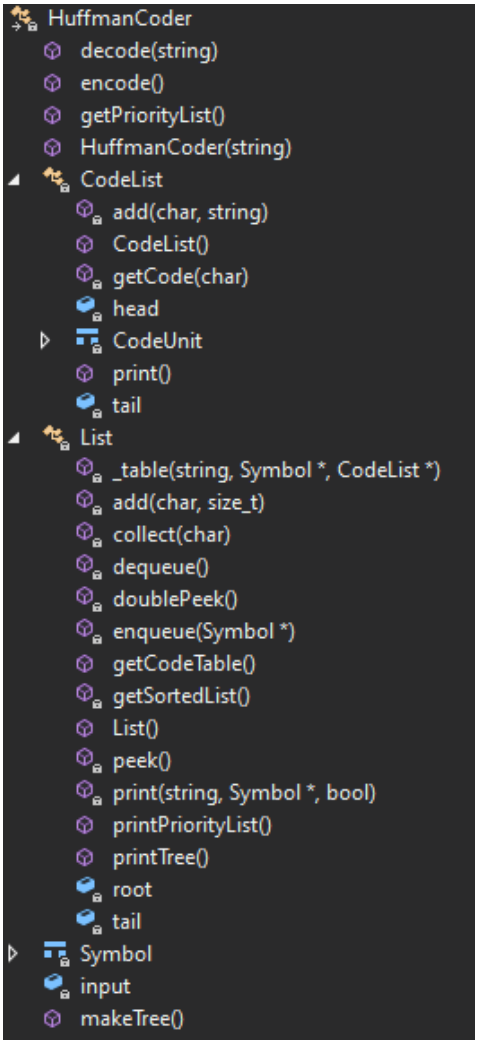
Студент гр. 8301 _____ Урбан М.Ф.
Преподаватель _____ Тутуева А.В.

Санкт-Петербург
2020

Цель работы.

Реализовать кодирование и декодирование по алгоритму Хаффмана входной строки, вводимой через консоль

Описание реализуемого класса и методов



Класс `HuffmanCoder` содержит приватные поля:

- `string input` – входная строка
- `struct Symbol` – универсальная структура хранения символа и его частоты в дереве или списке
- `class CodeList` – класс списка кодов для символов

Публичные поля:

- `HuffmanCoder(string)` – конструктор, принимающий ввод
- `List getPriorityList()` – возвращает список весов каждого символа (можно вывести в консоль методом `printPriorityList()`)
- `List makeTree()` – возвращает дерево Хаффмана (можно вывести в консоль методом `printTree()`, а также можно вывести таблицу кодов методами `getCodeTable().print()`)
- `string decode()` — декодирует и возвращает строку, которая была закодирована по правилам сжатия входной строки
- `string encode()` – кодирует входную строку, выводит её, таблицу кодов, исходный и конечный размеры и их соотношение

Описания остальных методов и полей находятся в коде.

Оценка временных сложностей

Основные методы:

`getPriorityList()` – O(n) + `aux`

`makeTree` – O(log n) + `aux`

`encode()` – O(n) + `aux`

`decode()` – O(log n) + `aux`

*`aux` – временные сложности вспомогательных методов. Описаны в коде.

Описание реализованных Unit-тестов.

Тесты тестируют всё, что можно протестировать. Ну или не всё, но тестируют. К остальным методам невозможно подобрать проверяемые утверждения.


```
TEST_METHOD(getPriorityListTest)
{
    // The only useful thing of publicity of this method is printing.
    HuffmanCoder("boop beep beer").getPriorityList().printPriorityList();
}

TEST_METHOD(makeTreeTest) {
    // Same story
    HuffmanCoder("boop beep beer").makeTree().printTree();
}

TEST_METHOD(decodeTest) {
    string _test = HuffmanCoder("AAKKTb ").decode("011100010011000100011101100");
    string _expect = "A KAK KAKATb";
    for (size_t i = 0; _test[i] != '\0'; i++) {
        Assert::AreEqual(_test[i], _expect[i]);
    }
}

TEST_METHOD(encodeTest) {
    string _test = HuffmanCoder("AAKKTb ").encode();
    string _expect = "0101000010110011";
    for (size_t i = 0; _test[i] != '\0'; i++) {
        Assert::AreEqual(_test[i], _expect[i]);
    }
}
```

Пример работы программы

Кодировщик вызывается из тела программы одной строкой. На изображении ниже показан вариант для бесконечного вызова кодировщика к вводимой из консоли строки.

```
while (true) {
    string _string, _decode;
    cout << "[INPUT]: ";
    getline(cin, _string);

    HuffmanCoder(_string).encode();

    _getch();
    system("cls");
}
```

Вот как это работает:

```
[INPUT]: сжатие сжимает сжимаемое

[CODE TABLE]:
  : 111
о : 1101
т : 1100
с : 101
ж : 100
а : 011
и : 010
м : 001
е : 000

[ENCODED TEXT]:
10110001111000100001111011000100010110001100111101100010001011000001110100 0

[INITIAL STRING SIZE]: 192 bits (24 bytes)
[COMPRESSED STRING SIZE]: 75 bits (9.375 bytes)
[COMPRESSION RATIO]: 2.56
```

Декодирование происходит на основе таблицы кодов. В моей реализации эта таблица генерируется из входной строки.

```
while (true) {
    string _string, _decode;
    cout << "[KEY]: ";
    getline(cin, _string);
    cout << "[DECODE]: ";
    getline(cin, _decode);

    cout<<HuffmanCoder(_string).decode(_decode); // Single-Line call!!!

    _getch();
    system("cls");
}
```

```
[KEY]: кккккккуууууушшааанонпилпю
[DECODE]: 01110111101010011100101110111101000101111001011100001000000101001110010100110000001001011000100001
кукушка кукушнку купила капюшон
```

(ну, руками «0» и «1» писать не так просто, поэтому «О» пропустил случайно)

Также можно построить дерево Хаффмана для заданной строки

```
HuffmanCoder(_string).makeTree().printTree();
```

```
[INPUT]: дерево


[CODE TREE]:
'---0*
    |---1*
    |   |---1 [в]
    |   '---0  [о]
    '---0*
        |---1 [е]
        '---0*
            |---1 [д]
            '---0  [р]
```

(Т.к. при таком выводе 0 и 1 привязаны к узлам, у корня стоит 0. В реальном пути его нет. Баг вывода дерева с прошлого семестра)

Доступен вывод таблицы кодов:

```
HuffmanCoder(_string).makeTree().getCodeTable().print();
```

```
[INPUT]: POOPY-DI SCOOP SCOOP-DIDDY-WHOOP WHOOP-DI-SCOOP-DI-POOP

[CODE TABLE]:
O : 11
D : 101
H : 1001
  : 1000
- : 011
S : 0101
C : 0100
P : 001
I : 0001
Y : 00001
W : 00000
```

И частотный анализ:

```
HuffmanCoder(_string).getPriorityList().printPriorityList();
```

```
[INPUT]: POOPY-DI SCOOP SCOOP-DIDDY-WHOOP WHOOP-DI-SCOOP-DI-POOP

[FREQUENCY ANALYSIS]:
[Y : 2] [W : 2] [H : 2] [  : 3] [S : 3] [C : 3] [I : 4] [D : 6] [- : 7] [P : 9]
[O : 14]
```

Код доступен на GitHub