**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 Shennon\_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:

* Констуктор – 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>\* shennon, 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 Шенона-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(Nlog n)
* *get\_tree\_with\_code()* – O(N)
* *fill\_tree (Map<char, string>\* shennon, Node\* root, string cur)* – O(N)
* *get\_list\_symbol()* – O(1)
* *Decoding(Node\* root, string& coding\_str, string& decoding\_str, int& position)* – O(Nlog 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 |
|  |
|  |
|  |

Листинг

|  |
| --- |
| Shannon\_Fano.h |
| 1. #pragma once 2. #include"Map.h" 3. #include"List.h" 4. #include"Pair.h" 5. #include"string" 6. class Shennon\_Fano 7. { 8. class Node { 9. public: 10. Node(Pair<char, int> value = Pair<char, int>(), Node\* left = NULL, Node\* right = NULL) :value(value), left(left), right(right) {} 11. Pair<char, int> value; 12. Node\* left; 13. Node\* right; 14. }; 15. public: 16. Shennon\_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 символов иначе увеличиваем кол-во 21. map\_symbol->insert(str[i], 1); 22. else 23. map\_symbol->increment\_value(str[i]); 24. } 25. List<Pair<char, int>>\* list\_of\_pairs = map\_symbol->get\_pairs(); 26. map\_symbol->clear();//больше не нужен только занимает теперь память 27. list\_of\_pairs->sort();//сортируем для дальнейшего оборачивания 28. ///////////////////////////////////////////////////////////////////////////построение дерева кодирования 29. build\_tree(head, list\_of\_pairs); 30. /////////////////////////////////////////////////////////////////////////// 31. } 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) { 66. if (list->get\_size() == 1) { 67. list\_symbol->push\_back(list->at(0)); 68. root = new Node(list->at(0)); 69. } 70. else { 71. root = new Node(); 72. if (list->get\_size() > 1) { 73. List<Pair<char, int>>\* list1 = list; 74. List<Pair<char, int>>\* list2 = new List<Pair<char, int>>(); 75. absolute\_cut(list1, list2); 76. if (list->isEmpty()) { 77. root->left = build\_tree(root->left, list1); 78. root->right = build\_tree(root->right, list2); 79. } 80. } 81. } 82. return root; 83. } 84. } 85. void fill\_tree(Map<char, string>\* shennon, Node\* root, string cur) {//заполнение дерева кодов 86. if (head->left != NULL && head->right != NULL) { 87. if (root->left != NULL && root->right != NULL) { 88. fill\_tree(shennon, root->left, cur + '0'); 89. fill\_tree(shennon, root->right, cur + '1'); 90. } 91. else 92. { 93. shennon->insert(root->value.first, cur); 94. } 95. } 96. else { 97. shennon->insert(root->value.first, cur+'0'); 98. } 99. } 100. void absolute\_cut(List<Pair<char, int>>\*& list1, List<Pair<char, int>>\*& list2) {//разделение на два списка 101. int cost = 0; 102. for (int i = 0; i < list1->get\_size(); ++i) { 103. cost += list1->at(i).second; 104. } 105. list1->sort(); 106. list1->reverse(); 107. int cost\_list1 = cost; 108. int cost\_list2 = 0; 109. /////////////////////////////////////////////////////////////////////////////////////////////////////// 110. for (int i = 0; i < list1->get\_size() && list1->isEmpty(); ++i) { 111. if (list1->at(i).second + cost\_list2 <= cost / 2) { 112. cost\_list2 += list1->at(i).second; 113. list2->push\_back(list1->at(i)); 114. list1->remove(i); 115. } 116. } 117. cost\_list1 = cost\_list1 - cost\_list2; 118. if (list1->get\_size() == 0 || cost\_list2 == cost / 2) 119. for (int i = 0; i < list1->get\_size() && list1->isEmpty(); ++i) { 120. for (int j = 0; j < list2->get\_size() && list2->isEmpty(); ++j) { 121. if ((cost\_list2 - list2->at(j).second + list1->at(i).second > cost\_list2) && (cost\_list2 - list2->at(j).second + list1->at(i).second) <= cost / 2) { 122. cost\_list2 = cost\_list2 - list2->at(j).second + list1->at(i).second; 123. cost\_list1 = cost\_list1 + list2->at(j).second - list1->at(i).second; 124. Pair<char, int > cur1 = list1->at(i); 125. Pair<char, int > cur2 = list2->at(j); 126. list1->set(i, cur1); 127. list2->set(j, cur2); 128. } 129. } 130. if (cost\_list2 == cost / 2) 131. break; 132. } 133. ///////////////////////////////////////////////////////////////////////////////////////////////////////////////// 134. } 135. Node\* head; 136. List<Pair<char, int>>\* list\_symbol; 137. }; |

|  |
| --- |
| UnitTestitFano.cpp |
| 1. #include "pch.h" 2. #include "CppUnitTest.h" 3. #include<string> 4. #include"\Users\1\source\repos\Lab 2\Shennon\_Fano.h" 5. #include"\Users\1\source\repos\Lab 2\Map.h" 6. #include"\Users\1\source\repos\Lab 2\List.h" 7. using namespace Microsoft::VisualStudio::CppUnitTestFramework; 8. namespace UnitTestitFano 9. { 10. TEST\_CLASS(UnitTestitFano) 11. { 12. public: 13. TEST\_METHOD(TestCodingFano\_different\_symbol\_words) 14. { 15. string str = "Hello world";// 16. string coding\_str; 17. Shennon\_Fano\* Shennon\_Fano\_tree = new Shennon\_Fano(str); 18. Map<char, string>\* Fano = Shennon\_Fano\_tree->fill\_tree\_with\_code(); 19. int counter = 0; 20. List<Pair<char, int>>\* list\_symbol = Shennon\_Fano\_tree->get\_list\_symbol(); 21. for (int i = 0; i < str.size(); i++) 22. coding\_str += Fano->find(str[i]); 23. Assert::AreEqual(coding\_str, string("0010111001000100001110101011000000")); 24. } 25. TEST\_METHOD(TestCodingFano\_same\_symbol\_words) 26. { 27. string str = "qqqqq";// 28. string coding\_str; 29. Shennon\_Fano\* Shennon\_Fano\_tree = new Shennon\_Fano(str); 30. Map<char, string>\* Fano = Shennon\_Fano\_tree->fill\_tree\_with\_code(); 31. int counter = 0; 32. List<Pair<char, int>>\* list\_symbol = Shennon\_Fano\_tree->get\_list\_symbol(); 33. for (int i = 0; i < str.size(); i++) 34. coding\_str += Fano->find(str[i]); 35. Assert::AreEqual(coding\_str, string("00000")); 36. } 37. TEST\_METHOD(TestCodingFano\_text) 38. { 39. string str = "it is just string";// 40. string coding\_str; 41. Shennon\_Fano\* Shennon\_Fano\_tree = new Shennon\_Fano(str); 42. Map<char, string>\* Fano = Shennon\_Fano\_tree->fill\_tree\_with\_code(); 43. int counter = 0; 44. List<Pair<char, int>>\* list\_symbol = Shennon\_Fano\_tree->get\_list\_symbol(); 45. for (int i = 0; i < str.size(); i++) 46. coding\_str += Fano->find(str[i]); 47. Assert::AreEqual(coding\_str, string("0100000100010110100111011110000010011000000010101010001")); 48. } 49. }; 50. } |

Вывод

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