**Московский авиационный институт (национальный исследовательский университет)**

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

**Курсовой проект по предмету "Дискретный анализ"**

Студент: Кострюков Е.С.

Преподаватель: Макаров Н.К.

Группа: М8О-307Б-22

Дата: 20.12.2024

Оценка:

Подпись:

**Оглавление**

[Цель работы 3](#__RefHeading__738_2004454639)

[Постановка задачи 3](#__RefHeading__740_2004454639)

[Общий алгоритм решения 3](#__RefHeading__744_2004454639)

[Реализация 5](#__RefHeading__746_2004454639)

[Пример работы 9](#__RefHeading__740_2004454639)

[Вывод 9](#__RefHeading__750_2004454639)

# **Цель работы**

Реализуйте алгоритм LZW. Начальный словарь выглядит следующим образом:

*a -> 0 b -> 1 c -> 2 ... x -> 23 y -> 24 z -> 25 EOF -> 26*

# **Постановка задачи**

**Формат ввода**

Вам будут даны тесты двух типов. Первый тип: *compress <text>*

Текст состоит только из малых латинских букв. В ответ на него вам нужно вывести коды, которыми будет закодирован данный текст.

Второй тип: *decompress <codes>*

Вам даны коды в которые был сжат текст из малых латинских букв, вам нужно его разжать.

**Формат вывода**

В ответ на тест первого типа вам нужно вывести коды, которыми будет закодирован данный текст через пробел.

В ответ на тест второго типа выведите разжатый текст.

## Общий алгоритм решения

В коде алгоритма есть две основные части: сжатие текста и разжатие текста. Рассмотрим каждую из них.

**1. Сжатие текста (compress)**

**Входные данные:**

- Текст, состоящий из строчных латинских букв.

**Выходные данные:**

- Последовательность чисел (кодов), представляющих сжатый текст.

**Алгоритм:**

1. Инициализация словаря:

- Создаётся словарь, где каждому символу (*a* до *z*) сопоставлен его код (*0* до *25*).

- Добавляется специальный код *26* для обозначения конца текста (*EOF*).

2. Сжатие текста:

- Обрабатываем текст посимвольно, начиная с пустой строки *current*.

- На каждом шаге добавляем следующий символ к строке *current*, образуя новую строку *next*.

- Если строка *next* есть в словаре, то обновляем *current* как *next* и продолжаем.

- Если строки *next* нет в словаре:

- Добавляем код строки *current* в результат.

- Добавляем строку *next* в словарь с новым кодом.

- Устанавливаем *current* равным текущему символу.

3. Добавление оставшихся данных:

- Если после завершения цикла *current* не пуст, добавляем его код в результат.

- В конце добавляем код *EOF* (*26*) в результат.

**2. Разжатие текста (decompress)**

**Входные данные:**

- Последовательность чисел (кодов), представляющих сжатый текст.

**Выходные данные:**

- Разжатый текст.

**Алгоритм:**

1. Инициализация словаря:

- Создаётся словарь, где каждому коду (*0* до *25*) сопоставлен соответствующий символ (*a* до *z*).

- Добавляется специальная строка *EOF* для кода *26*.

2. Разжатие кодов:

- Инициализируем пустую строку *previous* для хранения предыдущей декодированной строки.

- Проходим по каждому коду в последовательности:

- Если код — *26*, завершаем обработку (*EOF*).

- Если код есть в словаре, берём соответствующую строку.

- Если кода нет в словаре (редкий случай), формируем строку, добавляя первый символ предыдущей строки к самой себе.

- Добавляем текущую строку в результат.

- Если *previous* не пуст, добавляем новую строку в словарь, объединив *previous* и первый символ текущей строки.

- Обновляем *previous* текущей строкой.

# **Реализация**

1. #include <vector>
2. #include <string>
3. #include <iostream>
4. #include <unordered\_map>
5. using namespace std;
6. vector<int> compress(const string& text) {
7. // Ключ — строка, значение — её код
8. unordered\_map<string, int> dictionary;
9. for (char c = 'a'; c <= 'z'; ++c) {
10. dictionary[string(1, c)] = c - 'a';
11. }
12. dictionary["EOF"] = 26;
13. vector<int> result;
14. string current;
15. for (char c : text) {
16. string next = current + c;
17. // Если строка next уже есть в словаре, значит, она встречалась ранее
18. if (dictionary.count(next)) {
19. current = next;
20. } else {
21. result.push\_back(dictionary[current]);
22. dictionary[next] = dictionary.size();
23. current = string(1, c);
24. }
25. }
26. // После выхода из цикла может остаться непустая подстрока current
27. if (!current.empty()) {
28. result.push\_back(dictionary[current]);
29. }
30. result.push\_back(dictionary["EOF"]);
31. return result;
32. }
33. string decompress(const vector<int>& codes) {
34. // Ключ — код, значение — строка
35. vector<string> dictionary(27);
36. for (char c = 'a'; c <= 'z'; ++c) {
37. dictionary[c - 'a'] = string(1, c);
38. }
39. dictionary[26] = "EOF";
40. string result;
41. string previous;
42. for (int code : codes) {
43. if (code == 26) break;
44. string entry;
45. if (code < dictionary.size()) {
46. entry = dictionary[code];
47. } else if (!previous.empty()) {
48. entry = previous + previous[0];
49. }
50. result += entry;
51. if (!previous.empty()) {
52. dictionary.push\_back(previous + entry[0]);
53. }
54. previous = entry;
55. }
56. return result;
57. }
58. int main() {
59. cin.tie(0);
60. ios::sync\_with\_stdio(false);
62. string command;
63. cin >> command;
64. if (command == "compress") {
65. string text;
66. cin >> text;
67. vector<int> compressed = compress(text);
68. for (size\_t i = 0; i < compressed.size(); ++i) {
69. if (i > 0) cout << " ";
70. cout << compressed[i];
71. }
72. cout << endl;
73. } else if (command == "decompress") {
74. vector<int> codes;
75. int code;
76. while (cin >> code) {
77. codes.push\_back(code);
78. }
79. string decompressed = decompress(codes);
80. cout << decompressed << endl;
81. }
82. return 0;
83. }

## Пример работы

**Ввод:**

compress

banana

**Вывод:**

1 0 13 0 27 29 26

**Ввод:**

decompress

1 0 13 0 27 29 26

**Вывод:**

banana

# **Вывод**

Во время выполнения лабораторной работы по применению алгоритма *LZW* я изучил его основные принципы, включая построение динамического словаря для сжатия текстов и обратное восстановление текста при разжатии. Я разобрался, как алгоритм обрабатывает подстроки, добавляет новые записи в словарь, и как это влияет на эффективность работы. На практике я освоил реализацию словаря с использованием *unordered\_map* и *vector*, что позволило эффективно выполнять операции поиска и добавления элементов. Также я научился обрабатывать текстовые данные, преобразовывая их в числовые коды и обратно, с учетом специальных символов, таких как *EOF*. Эта работа помогла мне глубже понять, как алгоритмы сжатия данных могут применяться в реальных задачах, например, для экономии памяти или сокращения объема передаваемой информации.