Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий

Кафедра ИС

ОТЧЁТ

по лабораторной работе №6

АЛГОРИТМЫ СЖАТИЯ ДАННЫХ

Выполнил:

ст. гр. ИС/б-21-2-о

Шевелёв К. С.

Проверил:

Минкин С. И.

Севастополь

2024

1. ЦЕЛЬ РАБОТЫ

Изучить основные виды и алгоритмы сжатия данных и научиться решать задачи сжатия данных по методу Хаффмана и с помощью кодовых деревьев.

1. ПОСТАНОВКА ЗАДАЧИ
   1. На основании приведенных в лекции сведений реализуйте алгоритмы сжатия по методу Хаффмана через префиксные коды и на основе кодовых деревьев.
   2. Алфавит содержит 7 букв, которые встречаются с вероятностями 0,4; 0,2; 0,1; 0,1; 0,1; 0,05; 0,05. Осуществите кодирование по методу Хаффмана.
   3. Закодируйте по алгоритму Хаффмана строку с вашим именем, отчеством, фамилией, датой и местом рождения (например, "Иванова Наталья Николаевна, 1 января 1990 года, город Тверь"). При кодировании не округляйте частоты менее, чем четыре знака после запятой – сокращение точности понижает эффективность кодирования. Подсчитайте коэффициент сжатия.
   4. При кодировании по методу Фано все сообщения записываются в таблицу по степени убывания вероятности и разбиваются на две группы примерно (насколько это возможно) равной вероятности. Соответственно этой процедуре из корня кодового дерева исходят два ребра, которым в качестве весов присваиваются полученные вероятности. Двум образовавшимся вершинам приписывают кодовые символы 0 и 1. Затем каждая из групп вероятностей вновь делится на две подгруппы примерно равной вероятности. В соответствии с этим из каждой вершины 0 и 1 исходят по два ребра с весами, равными вероятностям подгрупп, а вновь образованным вершинам приписывают символы 00 и 01, 10 и 11. В результате многократного повторения процедуры разделения вероятностей и образования вершин приходим к ситуации, когда в качестве веса, приписанного ребру бинарного дерева, выступает вероятность одного из данных сообщений. В этом случае вновь образованная вершина оказывается листом дерева, т.к. процесс деления вероятностей для нее завершен. Задача кодирования считается решенной, когда на всех ветвях кодового бинарного дерева образуются листья. Закодируйте по алгоритму Фано данные текстового файла.
2. ХОД РАБОТЫ
   1. Алгоритм сжатия по методу Хаффмана.

Листинг 1 – Код программы

#include <iostream>

#include <queue>

#include <unordered\_map>

#include <windows.h>

using namespace std;

struct HuffmanNode {

char data;

int frequency;

HuffmanNode\* left;

HuffmanNode\* right;

HuffmanNode(char data, int frequency) : data(data), frequency(frequency), left(nullptr), right(nullptr) {}

};

struct CompareNodes {

bool operator()(HuffmanNode\* lhs, HuffmanNode\* rhs) {

return lhs->frequency > rhs->frequency;

}

};

HuffmanNode\* buildHuffmanTreePrefix(const unordered\_map<char, string>& prefixCodes) {

priority\_queue<HuffmanNode\*, vector<HuffmanNode\*>, CompareNodes> pq;

for (const auto& pair : prefixCodes) {

HuffmanNode\* node = new HuffmanNode(pair.first, pair.second.length());

pq.push(node);

}

while (pq.size() > 1) {

HuffmanNode\* left = pq.top(); pq.pop();

HuffmanNode\* right = pq.top(); pq.pop();

HuffmanNode\* parent = new HuffmanNode('\0', left->frequency + right->frequency);

parent->left = left;

parent->right = right;

pq.push(parent);

}

return pq.top();

}

void generatePrefixCodes(HuffmanNode\* root, const string& codePrefix, unordered\_map<char, string>& prefixCodes) {

if (root == nullptr) return;

if (root->data != '\0') prefixCodes[root->data] = codePrefix;

generatePrefixCodes(root->left, codePrefix + "0", prefixCodes);

generatePrefixCodes(root->right, codePrefix + "1", prefixCodes);

}

string compressUsingPrefixCodes(const string& data, const unordered\_map<char, string>& prefixCodes) {

string compressedData;

for (char c : data) compressedData += prefixCodes.at(c);

return compressedData;

}

string decompressUsingPrefixCodes(const string& compressedData, HuffmanNode\* root) {

string decompressedData;

HuffmanNode\* current = root;

for (char c : compressedData) {

current = (c == '0') ? current->left : current->right;

if (current->left == nullptr && current->right == nullptr) {

decompressedData += current->data;

current = root;

}

}

return decompressedData;

}

int main() {

SetConsoleOutputCP(1251);

string data = "AAAAABBBBBCCCCCDDDDDDEEEEE";

unordered\_map<char, string> prefixCodes = {

{'A', "0"},

{'B', "10"},

{'C', "110"},

{'D', "1110"},

{'E', "1111"}

};

cout << "Исходные данные: " << data << endl;

string compressedData = compressUsingPrefixCodes(data, prefixCodes);

cout << "Сжатые данные: " << compressedData << endl;

HuffmanNode\* root = buildHuffmanTreePrefix(prefixCodes);

string decompressedData = decompressUsingPrefixCodes(compressedData, root);

cout << "Декомпрессированные данные: " << decompressedData << endl;

return 0;

}

На рис. 1 представлены исходные данные, которые подаются в программу, сжатые данные и восстановленные данные на основе сжатия.

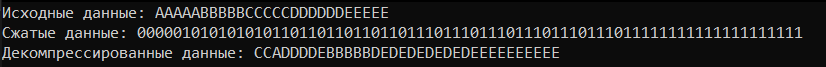


Рисунок 1 – Результат работы программы

* 1. Кодирование алфавита по методу Хаффмана

Листинг 2 – Код программы

#include <iostream>

#include <queue>

#include <unordered\_map>

#include <string>

#include <windows.h>

struct HuffmanNode {

char data;

int frequency;

HuffmanNode\* left;

HuffmanNode\* right;

HuffmanNode(char data, int frequency) : data(data), frequency(frequency), left(nullptr), right(nullptr) {}

};

struct CompareNodes {

bool operator()(HuffmanNode\* lhs, HuffmanNode\* rhs) {

return lhs->frequency > rhs->frequency;

}

};

HuffmanNode\* buildHuffmanTree(const std::unordered\_map<char, double>& probabilities) {

std::priority\_queue<HuffmanNode\*, std::vector<HuffmanNode\*>, CompareNodes> pq;

for (const auto& pair : probabilities) {

HuffmanNode\* node = new HuffmanNode(pair.first, pair.second \* 1000);

pq.push(node);

}

while (pq.size() > 1) {

HuffmanNode\* left = pq.top(); pq.pop();

HuffmanNode\* right = pq.top(); pq.pop();

HuffmanNode\* parent = new HuffmanNode('\0', left->frequency + right->frequency);

parent->left = left;

parent->right = right;

pq.push(parent);

}

return pq.top();

}

void generatePrefixCodes(HuffmanNode\* root, const std::string& codePrefix, std::unordered\_map<char, std::string>& prefixCodes) {

if (root == nullptr) return;

if (root->data != '\0') prefixCodes[root->data] = codePrefix;

generatePrefixCodes(root->left, codePrefix + "0", prefixCodes);

generatePrefixCodes(root->right, codePrefix + "1", prefixCodes);

}

int main() {

SetConsoleOutputCP(1251);

std::unordered\_map<char, double> probabilities = {

{'A', 0.4},

{'B', 0.2},

{'C', 0.1},

{'D', 0.1},

{'E', 0.1},

{'F', 0.05},

{'G', 0.05}

};

std::cout << "Исходные вероятности символов:\n";

for (const auto& pair : probabilities) {

std::cout << "Символ: " << pair.first << ", Вероятность: " << pair.second << "\n";

}

HuffmanNode\* root = buildHuffmanTree(probabilities);

std::unordered\_map<char, std::string> prefixCodes;

generatePrefixCodes(root, "", prefixCodes);

std::cout << "\nПрефиксные коды для каждого символа:\n";

for (const auto& pair : prefixCodes) {

std::cout << "Символ: " << pair.first << ", Код: " << pair.second << "\n";

}

return 0;

}

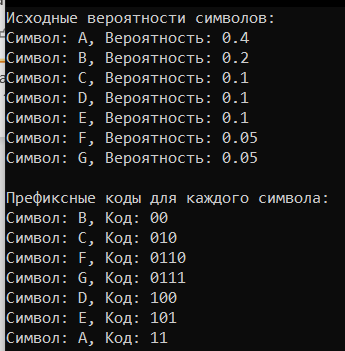


Рисунок 2 – Результат работы программы

* 1. Кодирование своих данных

Листинг 3 – Код программы

#include <iostream>

#include <unordered\_map>

#include <queue>

#include <string>

#include <iomanip>

#include <locale>

#include <windows.h>

struct HuffmanNode {

char data;

double frequency;

HuffmanNode\* left;

HuffmanNode\* right;

HuffmanNode(char data, double frequency) : data(data), frequency(frequency), left(nullptr), right(nullptr) {}

};

struct CompareNodes {

bool operator()(HuffmanNode\* lhs, HuffmanNode\* rhs) {

return lhs->frequency > rhs->frequency;

}

};

HuffmanNode\* buildHuffmanTree(const std::unordered\_map<char, double>& probabilities) {

std::priority\_queue<HuffmanNode\*, std::vector<HuffmanNode\*>, CompareNodes> pq;

for (const auto& pair : probabilities) {

HuffmanNode\* node = new HuffmanNode(pair.first, pair.second \* 1000);

pq.push(node);

}

while (pq.size() > 1) {

HuffmanNode\* left = pq.top(); pq.pop();

HuffmanNode\* right = pq.top(); pq.pop();

HuffmanNode\* parent = new HuffmanNode('\0', left->frequency + right->frequency);

parent->left = left;

parent->right = right;

pq.push(parent);

}

return pq.top();

}

void generatePrefixCodes(HuffmanNode\* root, const std::string& codePrefix, std::unordered\_map<char, std::string>& prefixCodes) {

if (root == nullptr) return;

if (root->data != '\0') prefixCodes[root->data] = codePrefix;

generatePrefixCodes(root->left, codePrefix + "0", prefixCodes);

generatePrefixCodes(root->right, codePrefix + "1", prefixCodes);

}

int main() {

SetConsoleOutputCP(1251);

std::string inputString = "Шевелёв Кирилл Станиславович, 29 июня 2003 года, город Новотроицк";

std::unordered\_map<char, int> charFrequencies;

for (char c : inputString) {

charFrequencies[c]++;

}

std::unordered\_map<char, double> probabilities;

for (const auto& pair : charFrequencies) {

char c = pair.first;

int frequency = pair.second;

double probability = static\_cast<double>(frequency) / inputString.length();

probabilities[c] = probability;

}

HuffmanNode\* root = buildHuffmanTree(probabilities);

std::unordered\_map<char, std::string> prefixCodes;

generatePrefixCodes(root, "", prefixCodes);

std::string encodedString;

for (char c : inputString) {

encodedString += prefixCodes[c];

}

std::cout << "Префиксные коды:\n";

for (const auto& pair : prefixCodes) {

std::cout << "'" << pair.first << "': " << pair.second << "\n";

}

std::cout << "Закодированная строка: " << encodedString << "\n";

int originalBits = inputString.length() \* 8;

int encodedBits = encodedString.length();

double compressionRatio = static\_cast<double>(originalBits) / encodedBits;

std::cout << "Коэффициент сжатия: " << std::fixed << std::setprecision(2) << compressionRatio << "\n";

return 0;

}

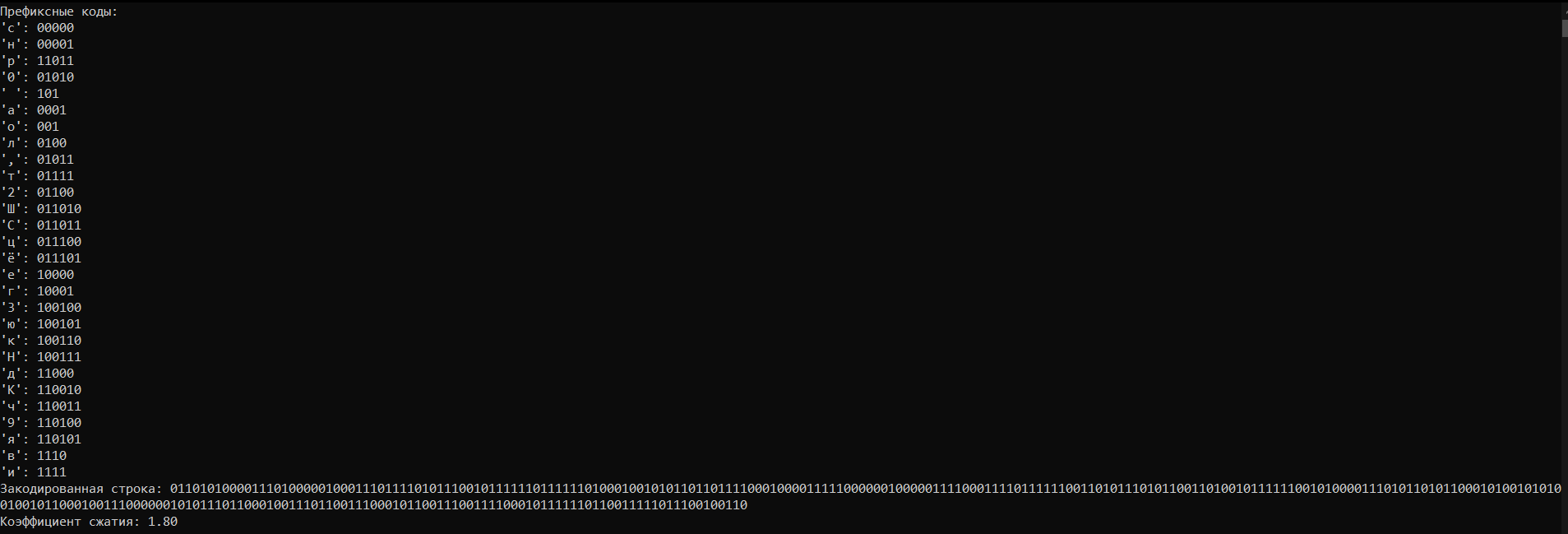


Рисунок 3 – Результат работы программы

* 1. Алгоритм Фано

Листинг 4 – Код программы

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_map>

#include <fstream>

#include <windows.h>

using namespace std;

struct FanoNode {

char symbol;

double probability;

FanoNode\* left;

FanoNode\* right;

FanoNode(char sym, double prob) : symbol(sym), probability(prob), left(nullptr), right(nullptr) {}

};

struct FanoNodeComparator {

bool operator()(const FanoNode\* a, const FanoNode\* b) const {

return a->probability < b->probability;

}

};

FanoNode\* buildFanoTree(const unordered\_map<char, double>& probabilities) {

priority\_queue<FanoNode\*, vector<FanoNode\*>, FanoNodeComparator> pq;

for (const auto& entry : probabilities) {

FanoNode\* node = new FanoNode(entry.first, entry.second \* 1000);

pq.push(node);

}

while (pq.size() > 1) {

FanoNode\* left = pq.top(); pq.pop();

FanoNode\* right = pq.top(); pq.pop();

FanoNode\* parent = new FanoNode('\0', left->probability + right->probability);

parent->left = left;

parent->right = right;

pq.push(parent);

}

return pq.top();

}

void generateFanoCodes(FanoNode\* root, string code, unordered\_map<char, string>& codes) {

if (!root)

return;

if (root->symbol != '\0') {

codes[root->symbol] = code;

return;

}

generateFanoCodes(root->left, code + "0", codes);

generateFanoCodes(root->right, code + "1", codes);

}

void encodeFile(const string& inputFilename, const string& outputFilename) {

unordered\_map<char, double> probabilities;

ifstream inputFile(inputFilename);

char ch;

int totalSymbols = 0;

while (inputFile.get(ch)) {

probabilities[ch]++;

totalSymbols++;

}

inputFile.close();

for (auto& entry : probabilities) {

entry.second /= totalSymbols;

}

FanoNode\* root = buildFanoTree(probabilities);

unordered\_map<char, string> codes;

generateFanoCodes(root, "", codes);

cout << "Префиксные коды:\n";

for (const auto& pair : codes) {

cout << pair.first << ": " << pair.second << "\n";

}

ofstream outputFile(outputFilename);

inputFile.open(inputFilename);

while (inputFile.get(ch)) {

outputFile << codes[ch];

}

inputFile.close();

outputFile.close();

}

int main() {

SetConsoleOutputCP(1251);

string inputFilename = "input.txt";

string outputFilename = "output.txt";

encodeFile(inputFilename, outputFilename);

return 0;

}

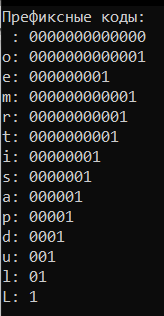


Рисунок 4 – Вывод закодированных префиксных кодов

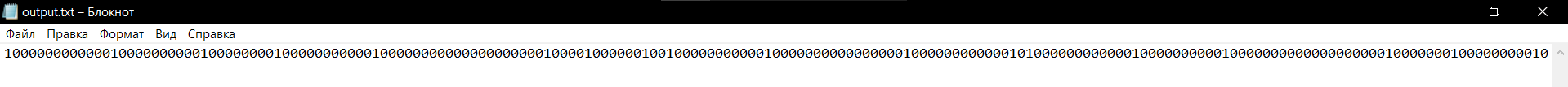


Рисунок 5 – Вывод в файл закодированных данных методом Фано

ВЫВОД

В ходе выполнения лабораторной работы были изучены основные виды и алгоритмы сжатия данных, а также решены задачи сжатия данных по методу Хаффмана и с помощью кодовых деревьев.

Контрольные вопросы

1. При кодировании каких данных можно использовать сжатие данных с потерями? Ответ обоснуйте.

Сжатие данных с потерями обычно применяется для данных, в которых допустима потеря части информации без существенного влияния на их восприятие или использование. Примерами таких данных могут быть аудио- и видеофайлы, изображения с высокой степенью сжатия (например, форматы JPEG), данные сенсоров и т. д. В этих случаях потеря некоторых деталей может быть приемлемой, если обеспечивается достаточно высокое качество воспроизведения или воспроизведение основной информации.

2. В чем преимущества и недостатки статических методов и словарного сжатия?

Преимущества статических методов сжатия:

* Простота реализации и применения.
* Быстрое сжатие и распаковка, поскольку алгоритм использует фиксированный набор правил.
* Отсутствие необходимости в создании и хранении словаря или других вспомогательных структур данных.
* Недостатки статических методов сжатия:
* Менее эффективное сжатие по сравнению со словарными методами.
* Отсутствие адаптивности к конкретным данным или структуре данных.
* Преимущества словарного сжатия:
* Более высокий уровень сжатия по сравнению со статическими методами.
* Адаптивность к конкретным данным, что позволяет получить лучшую эффективность сжатия.

Недостатки словарного сжатия:

* Больший объем памяти, необходимый для хранения словаря.
* Дополнительное время, требуемое для создания и обновления словаря.

3. Каким образом кодирование по алгоритму Хаффмана через префиксный код гарантирует минимальную длину кода?

Алгоритм Хаффмана через префиксный код гарантирует минимальную длину кода благодаря использованию префиксных кодов. Префиксный код – это код, в котором нет ни одного кодового слова, являющегося префиксом другого кодового слова. В алгоритме Хаффмана каждому символу присваивается код, который является префиксным кодом. Таким образом, ни одно кодовое слово не является префиксом другого, что обеспечивает однозначность декодирования и минимальную длину кода.

4. За счет чего в методе Хаффмана поддерживается однозначность соответствия кода кодируемому символу?

В методе Хаффмана однозначность соответствия кода кодируемому символу обеспечивается путем построения дерева Хаффмана. В дереве каждый символ представлен листовым узлом, а пути от корня до листов представляют коды символов. Поскольку нет двух символов с одинаковыми кодами, соответствие кода и символа остается однозначным.

5. Почему алгоритм Хаффмана малоэффективен для файлов маленьких размеров?

Алгоритм Хаффмана имеет некоторые накладные расходы на создание и хранение дерева Хаффмана и словаря символов, которые могут быть существенными для файлов маленьких размеров. Кроме того, для небольших файлов сложно достичь значительного сжатия, так как вероятности символов могут быть близкими, и построенное дерево Хаффмана может не дать значительного выигрыша в эффективности сжатия.

1. Выполните кодирование по методу Хаффмана через префиксный код символов, которые встречаются с вероятностями 0,3; 0,2; 0,1; 0,1; 0,1; 0,05; 0,05; 0,04; 0,03; 0,03. Сравните полученный результат с данными программной реализации.

7. Докажите, что метод Хаффмана кодирует информацию без потерь.

Доказательство основывается на двух ключевых свойствах метода Хаффмана:

а) Префиксный код: Коды Хаффмана являются префиксными, то есть ни один код не является префиксом другого кода. Это обеспечивает однозначность декодирования: нет возможности двусмысленного интерпретирования закодированной последовательности.

б) Восстановление дерева Хаффмана: При раскодировании используется та же структура дерева Хаффмана, что и при кодировании. Дерево можно восстановить из декодированных данных, так как оно сохраняет информацию о структуре и кодах символов.