МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №4

По дисциплине: «***Алгоритмы и структуры данных***»

Тема: «***Построение кодов Хаффмана***»

**Выполнил:** Антонюк Н.А.

**Группа:** ПО-11

**Проверила:** Глущенко Т.А.

Брест 2023

***Задание.***

1. Дан текстовый файл размером не менее *5 кбайт*. Построить для данного текста *коды Хаффмана* (см. матермал лекции). Написать программу для кодировки и раскодировки заданного файла. Указать размеры файла до и после сжатия алгоритмом *Хаффмана*.   
   Для наглядности текстовый файл можно создать самостоятельно и меньшего размера.

#include <iostream>

#include <fstream>

#include <unordered\_map>

#include <queue>

#include <string>

using namespace std;

struct HuffmanNode {

char symbol;

int frequency;

HuffmanNode\* left;

HuffmanNode\* right;

HuffmanNode(char sym, int freq, HuffmanNode\* l = nullptr, HuffmanNode\* r = nullptr) : symbol(sym), frequency(freq), left(l), right(r) {}

};

struct Compare {

bool operator()(HuffmanNode\* a, HuffmanNode\* b) {

return a->frequency > b->frequency;

}

};

class HuffmanTree {

public:

HuffmanNode\* root;

HuffmanTree(const unordered\_map<char, int>& frequencies) {

buildTree(frequencies);

}

void buildTree(const unordered\_map<char, int>& frequencies) {

priority\_queue<HuffmanNode\*, vector<HuffmanNode\*>, Compare> priorityQueue;

for (const auto& kvp : frequencies) {

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

priorityQueue.push(node);

}

while (priorityQueue.size() > 1) {

HuffmanNode\* left = priorityQueue.top();

priorityQueue.pop();

HuffmanNode\* right = priorityQueue.top();

priorityQueue.pop();

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

priorityQueue.push(parent);

}

root = priorityQueue.top();

}

unordered\_map<char, string> buildCodes() {

unordered\_map<char, string> codes;

buildCodesRecursive(root, "", codes);

return codes;

}

void buildCodesRecursive(HuffmanNode\* node, string code, unordered\_map<char, string>& codes) {

if (node == nullptr)

return;

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

codes[node->symbol] = code;

}

buildCodesRecursive(node->left, code + "0", codes);

buildCodesRecursive(node->right, code + "1", codes);

}

};

unordered\_map<char, int> countFrequencies(const string& text) {

unordered\_map<char, int> frequencies;

for (char c : text) {

frequencies[c]++;

}

return frequencies;

}

string encode(const string& text, const unordered\_map<char, string>& codes) {

string encodedText;

for (char c : text) {

encodedText += codes.at(c);

}

return encodedText;

}

string decode(const string& encodedText, HuffmanTree\* huffmanTree) {

string decodedText;

HuffmanNode\* currentNode = huffmanTree->root;

for (char bit : encodedText) {

if (bit == '0') {

currentNode = currentNode->left;

}

else if (bit == '1') {

currentNode = currentNode->right;

}

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

decodedText += currentNode->symbol;

currentNode = huffmanTree->root;

}

}

return decodedText;

}

int main() {

string inputFilePath = "input.txt";

string compressedFilePath = "compressed.huff";

string decompressedFilePath = "decompressed.txt";

ifstream inputFile(inputFilePath);

string text((istreambuf\_iterator<char>(inputFile)), istreambuf\_iterator<char>());

cout << "Original File Size: " << text.length() << " bytes" << endl;

unordered\_map<char, int> frequencies = countFrequencies(text);

HuffmanTree\* huffmanTree = new HuffmanTree(frequencies);

unordered\_map<char, string> codes = huffmanTree->buildCodes();

string encodedText = encode(text, codes);

ofstream outputFile(compressedFilePath);

outputFile << encodedText;

outputFile.close();

cout << "Compressed File Size: " << encodedText.length() << " bytes" << endl;

string decodedText = decode(encodedText, huffmanTree);

ofstream decompressedFile(decompressedFilePath);

decompressedFile << decodedText;

decompressedFile.close();

cout << "Decompressed File Size: " << decodedText.length() << " bytes" << endl;

return 0;

}

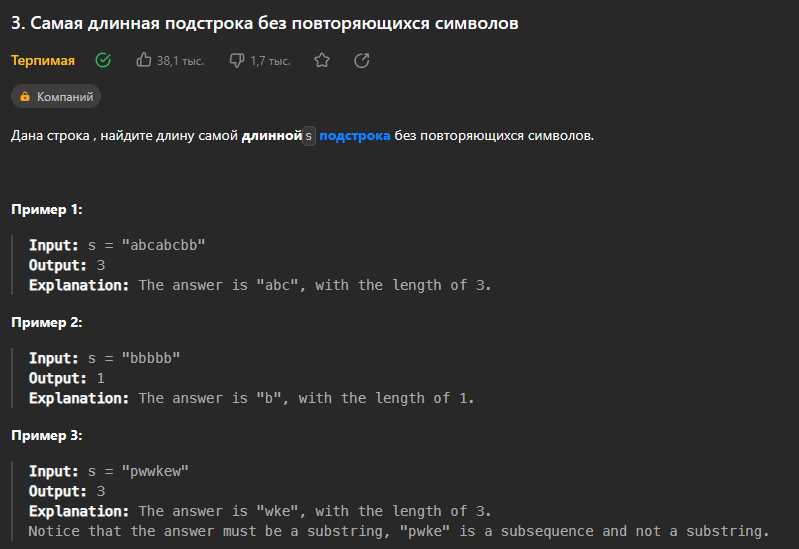
1. Письменно подробно ответить на вопросы

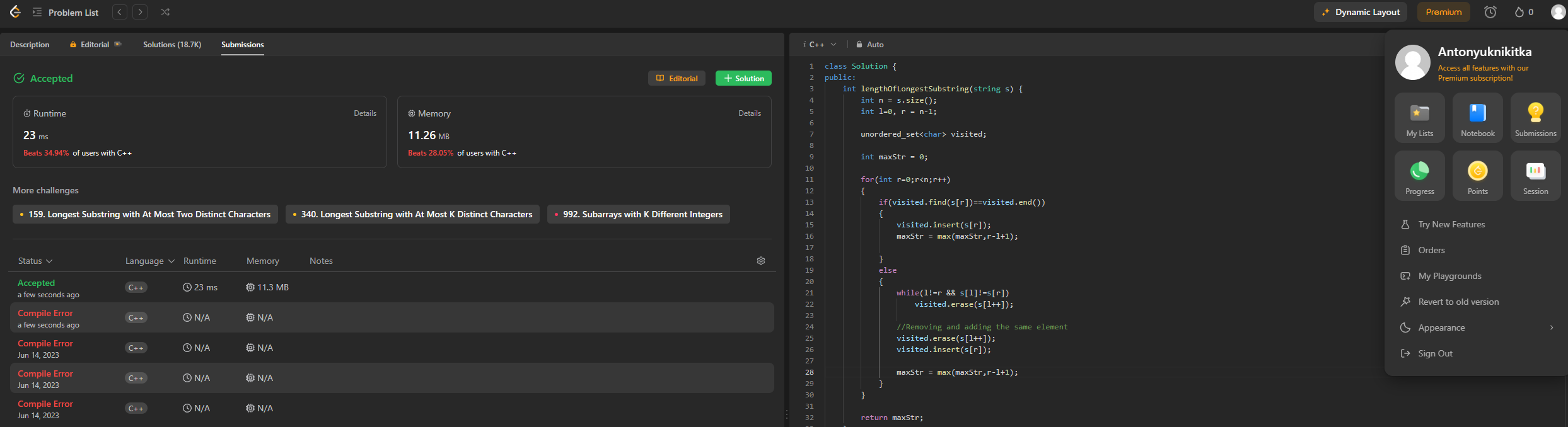
1. Что такое *сжатие без потерь*?

2. Опишите алгоритм построения кодов *Шеннона-Фано* для сжатия данных.

1. Что такое *префиксные коды*, являются ли в данных алгоритмах коды *префиксными* и для чего они используются?
2. Благодаря каким принципам происходит *сжатие* данных в указанных алгоритмах?
3. Укажите недостатки указанных кодов, средние коэффициенты сжатия для указанных алгоритмов.

**Задача на LEETCODE:**





**Алгоритм задачи:**

class Solution {

public:

    int lengthOfLongestSubstring(string s) {

        int n = s.size();

        int l=0, r = n-1;

        unordered\_set<char> visited;

        int maxStr = 0;

        for(int r=0;r<n;r++)

        {

            if(visited.find(s[r])==visited.end())

            {

                visited.insert(s[r]);

                maxStr = max(maxStr,r-l+1);

            }

            else

            {

                while(l!=r && s[l]!=s[r])

                    visited.erase(s[l++]);

                //Removing and adding the same element

                visited.erase(s[l++]);

                visited.insert(s[r]);

                maxStr = max(maxStr,r-l+1);

            }

        }

        return maxStr;

    }

};

1. **Сжатие без потерь:**

Сжатие без потерь — это метод сжатия данных, при котором информация восстанавливается полностью и точно после распаковки. Это означает, что не теряется никакая часть исходных данных, и сжатие происходит без искажения информации.

2. **Алгоритм построения кодов Шеннона-Фано:**

Алгоритм построения кодов Шеннона-Фано предполагает разделение множества символов на две части так, чтобы суммарные вероятности были приблизительно равны. Этот процесс рекурсивно повторяется для каждой части до тех пор, пока не достигнуты единичные символы. Каждый символ затем получает двоичный код, который строится путем присвоения одной половине символов 0, а другой — 1. Коды Шеннона-Фано не всегда являются префиксными.

3. **Префиксные коды:**

Префиксный код — это такой двоичный код, у которого ни одно из кодовых слов не является префиксом другого. Коды Шеннона-Фано и коды Хаффмана являются префиксными. Префиксные коды важны, поскольку они обеспечивают однозначное декодирование без необходимости использования разделителей между символами.

4. **Принципы сжатия данных**

- \*Частота встречаемости:\* Алгоритмы сжатия данных, такие как Шеннона-Фано и Хаффмана, используют информацию о частоте встречаемости символов в исходных данных для присвоения более коротких кодов часто встречающимся символам.

- \*Выделение паттернов:\* Алгоритмы пытаются выделить повторяющиеся паттерны или структуры в данных, чтобы представить их более компактно.

5. **Недостатки и коэффициенты сжатия:**

- *Шеннон-Фано*: Недостатки включают сложность в реализации и относительно низкую эффективность по сравнению с более современными методами. Коэффициент сжатия зависит от распределения частот символов в исходных данных.

- *Хаффман*: Недостатки включают некоторую сложность в построении дерева Хаффмана и неэффективность при работе с небольшими объемами данных. Коэффициент сжатия также зависит от распределения частот символов в исходных данных.