Politechnika Śląska

Wydział Matematyk Stosowanej

Kierunek Informatyka

Gliwice, 25.01.2025

Programowanie I

**projekt zaliczeniowy**

**"Kodowanie Huffmana"**

**Dawid Piotrowski gr. 3. lab. 6.**

**1. Opis projektu.**

Prosta implementacja algorytmu kodowania Huffmana, funkcjonująca zarówno jako program typu utility wiersza poleceń, jak i statyczna biblioteka języka C++.

**2. Funkcjonalności.**

* Funkcja encode(std::istream& input), która enkoduje dane z pliku lub wpisane przez użytkownika w konsoli. Zapisuje je do struktury, którą następnie można zserializować do pliku.
* Funkcja decode(const EncodedMessage& input, std::ostream& output), która dekoduje uprzednio zakodowaną wiadomość i wypisuje ją do konsoli lub do pliku.
* Interfejs wiersza konsoli poleceń, zrealizowany przez argumenty podawane przy wywołaniu programu. Dostępne komendy:
  + encode / e – koduje podany plik wejściowy i zapisuje wynik do pliku wyjściowego w zaprojektowanym przeze mnie formacie hff
  + decode / d – dekoduje podany plik hff i wypisuje wynik do konsoli
  + decode-to-file / df – dekoduje podany plik hff i zapisuje wynik do pliku wyjściowego
  + help – wypisuje listę dostępnych poleceń

**3. Przebieg realizacji.**

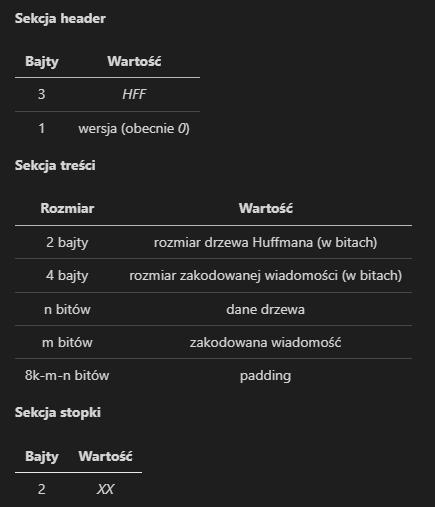
Algorytm Huffmana generuje drzewo binarne zawierające wszystkie znaki kodowanego tekstu. Na podstawie tego drzewa generuje unikatowe kody dla każdego znaku. Znaki częściej pojawiające się w tekście znajdują się „wyżej” na drzewie, co w konsekwencji przekłada się na krótsze kody.

W ramach jego implementacji stworzyłem następujące pliki źródłowe:

* huffman.cpp – zawiera funkcje kodujące i dekodujące tekst
* tree.cpp – definiuje klasę drzewa binarnego generowanego przez algorytm Huffmana
* buffer.cpp – definiuje klasę pomocniczą, przechowującą dane jako ciąg bitów
* message.cpp – definiuje strukturę, w której przechowywana jest zakodowana wiadomość w postaci bufora bitów i drzewo binarne wygenerowane przez algorytm; strukturę tą można zdekodować, aby otrzymać oryginalną wiadomość; implementuje ona również metody serializacji i deserializacji
* interface.cpp – plik ten znajduje się poza biblioteką huffman, służy wyłącznie do obsługi wejścia użytkownika i wykonania zadanej przez niego komendy
* main.cpp – wywołuje metodę zdefiniowaną w pliku interface.cpp oraz obsługuje ewentualne wyjątki

W projekcie starałem trzymać się dobrych praktyk C++. Zaimplementowałem system wyjątków dla każdej funkcji, której działanie może zakończyć się błędem. Starałem się korzystać ze smart pointerów w przypadkach, w których było to możliwe. Tworząc projekt przykładałem również wagę do jego skalowalności.

Ciekawym zadaniem była dla mnie implementacja serializacji i deserializacji zakodowanej wiadomości. Specyfikacja formatu hff wygląda następująco:



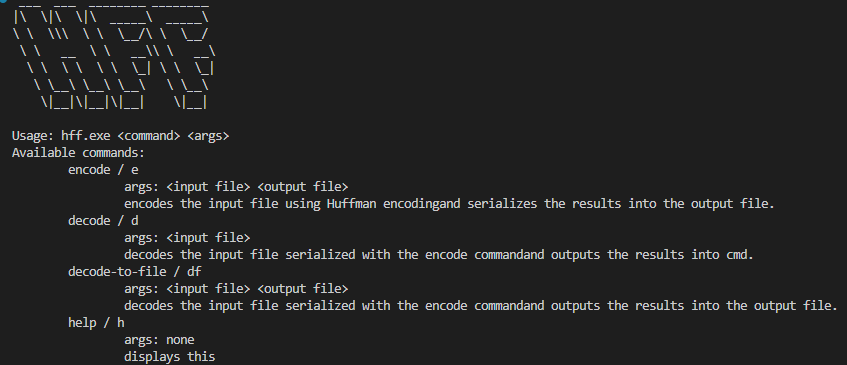
Stała wartość pierwszych trzech znaków zapewnia rozpoznawalność formatu pliku nawet przy niestandardowym rozszerzeniu. Zapisując wersję programu, w której plik był zakodowany, istnieje możliwość dalszego rozwoju formatu przy jednoczesnym zachowaniu kompatybilności wstecznej.

Ciekawie działa algorytm serializacji drzewa binarnego. Serializowanie samego drzewa zamiast słów kodowych pozwala oszczędzić miejsce. Serializacja drzewa polega na przejściu preorder, oznaczając liście przez bit 0 poprzedzający zakodowany w tym węźle znak. Każdemu z pozostałych wierzchołków odpowiada binarna jedynka. Jest to optymalny sposób zakodowania informacji niezbędnych do późniejszego odtworzenia drzewa.

Plik kończy się stałą wartością, którą można wykorzystać do sprawdzenia poprawności zapisanych danych, w szczególności rozmiaru sekcji treści. W przypadku wysyłania zserializowanych danych przez sieć stanowi również naturalny sposób na weryfikację poprawnego odebrania całego pliku.

**4. Instrukcja użytkownika.**

Program zawiera listę dostępnych komend wyświetlaną przez instrukcję „hff.exe help”. Wygląda ona następująco:



W przypadku wywołania programu bez podania komendy, program wyświetla komunikat sugerujący wywołanie instrukcji help. Jeśli użytkownik nieprawidłowo wywoła komendę (np. wpisze nieprawidłową nazwę lub poda złą liczbę argumentów), komenda nie zostanie wykonana, a zamiast niej pojawi się komunikat wskazujący przyczynę błędu.

Lista dostępnych komend w języku polskim jest opisana w sekcji „Funkcjonalności”.

**5. Podsumowanie i wnioski.**

Największą trudność sprawiła mi implementacja serializacji i deserializacji zakodowanej wiadomości. Wymagała ona zarówno zastosowania niebezpiecznych, niskopoziomowych operacji, jak i zwrócenia uwagi na wiele przypadków brzegowych, co utrudniało debugowanie kodu. Dalszy rozwój projektu mógłby polegać na jego dalszej optymalizacji (operacje nieoptymalne są obecnie oznaczone komentarzem) oraz na zaimplementowaniu kodowania ciągów jednego znaku (obecnie kodowanie takich ciągów nie jest wspierane, ponieważ program nie zapisuje ilości znaków w oryginalnej wiadomości). Mimo tych niedociągnięć, jestem zadowolony z ostatecznego wyniku. Sądzę, że program jest prosty w obsłudze i działa wystarczająco szybko, aby zadowolić przeciętnego użytkownika.

Program przetestowałem na treści „Pana Tadeusza” pobranej ze strony wolnelektury.pl. Kodowanie do pliku na moim komputerze zajmuje średnio ok. 500ms. Dekodowanie wyniku do pliku zajmuje ok. 650ms. Zakodowany plik zajmował na dysku ok. 64% pamięci (308 KB) zajmowanej przez plik oryginalny (482 KB).