Implementacja Funkcjonalna Kompresora

Bartosz Dybowski Jakub Klenkiewicz

11.04.2023

# Cel projektu

Celem projektu jest wykonanie kompresora w języku C. Program ten bazuje na algorytmie Huffmana. Umożliwia on kompresję w trzech poziomach od wyboru: ośmiobitowym, dwunastobitowym i szesnastobitowym oraz dekompresję.

# Dane wejściowe

Do swojego działania program potrzebuje wprowadzenia pliku wejściowego i wyjściowego. Czyta on zawartość pliku wejściowego, a następnie przystępuje do kompresji.

# Dane wyjściowe

Po udanej kompresji program zapisuje w pliku wyjściowym inicjały, informację o zaszyfrowaniu, rodzaj kompresji, ogonek (ile bitów z ostatniego zapisanego bajta jest istotnych), cale zakodowane drzewo, skompresowany kod w pliku wejściowym i cyfrę kontrolną oraz wypisuje komunikat o zakończeniu kompresji. W przypadku dekompresji odczytuje zakodowany tekst za pomocą drzewa i zapisuje go w drugim pliku.

# Argumenty wywołania programu

Wywołujemy program wpisując ./run oraz nazwy plików wejściowego i wyjściowego (zostaje wtedy wykonana podstawowa kompresja ośmiobitowa lub dekompresja, jeśli program rozpozna, że plik był już skompresowany).

Można również skorzystać z poniższych argumentów:

-e kompresja ośmiobitowa

-t kompresja dwunastobitowa

-s kompresja szesnastobitowa

-p [password] szyfrowanie

-i wyświetlanie dodatkowych informacji odnośnie do przebiegu programu

-h, h, --help, -help, help, -?, ? instrukcja obsługi programu

Przykładowe wywołanie programu:

./run we.txt wy.txt -p hasło -i -s

Wynikiem będzie kompresja szesnastobitowa (-s) pliku we.txt do pliku wy.txt, zaszyfrowanego hasłem „hasło” (-p hasło). Wypisane zostaną również etapy kompresji (-i).

# Teoria

**Ogólne działanie programu:**

Po uruchomieniu programu otwierane są pliki. Jeśli w pliku wejściowym odczytane zostaną inicjały, to wykonuje się dekompresja. W innym przypadku program przechodzi do czytania flag. Pierwszeństwo mają flagi nie dotyczące kompresji (hasło itd.). Następnie flagi dotyczące poziomu kompresji. Jeśli występuje jedna z trzech możliwych (-e -t -s), program przystępuje do kompresji wybraną metodą. W przypadku niepodania żadnej z powyższych flag program domyślnie kompresuje ósemkowo.

**Działanie kompresji**

1. Zlicza wystąpienia danych ciągów bitowych w pliku wejściowym (np. występowanie danych liter w kompresji ośmiobitowej)

2. Zlicza ilość unikalnych ciągów bitowych (potrzebne do drzewa)

3. Tworzy drzewo

4. Tworzy słownik, czyli przypisuje kod, wyczytany z drzewa Huffmana, dla każdego unikalnego ciągu bitowego

5. Przechodzi do zapisywania informacji

Jeśli jest hasło to xoruje zmienną, początkowo równą zero po każdej jego literze i następnie zapisuje znaki xorowane przez ta zmienna

5.1 Nagłówka

Inicjały

Informacja czy zaszyfrowany (0 lub 1)

Rodzaj kompresji (8,12,16)

„Ogonek”, czyli ile bitów z ostatniego zapisanego bajta jest istotnych (są rzeczywistym kodem, który należy zdekompresować, ogonek jest różny od zera dla długości kodu huffmana niepodzielnego przez 8)

Następnie cale drzewo w formacie :

Indeks w oryginalnej tablicy używanej przy kompresji (dla 8,12 zajmuje zawsze 2 bajty, dla 16 zajmuje zawsze 3 bajty)

Czy liść (0 lub 1)

Jeżeli liść :

Bity jakim odpowiada liść (dla 8 jeden bajt, dla 12,16 dwa bajty)

Jeżeli nie liść :  
 Indeks lewego dziecka

Indeks prawego dziecka (zajmują tyle samo co indeksy opisane wyżej)

5.2 Skompresowanego kodu reprezentującego oryginalny przekaz

Otwiera oryginalny plik, czyta znak, znajduje kod, dodaje kod do zmiennej tymczasowej i zapisuje ja do pliku wyjściowego jako znak, gdy zapełni się w niej wszystkie 8 bitów i tak, aż się skończą znaki w oryginalnym

5.3 Cyfry kontrolnej

Cyfra kontrolna zapisana na samym końcu jest zmienną początkowo ustawioną na wartość równą 0, a następnie xorowana z każdym znakiem zawierającym kod (5.2) (bez nagłówka) zapisanym do pliku wyjściowego

**Działanie dekompresji**

Wczytujemy wszystkie informacje z nagłówka po kolei, jeśli plik jest zaszyfrowany, to prosimy o hasło. Następnie odbudowujemy drzewo z zawartych informacji. Odczytujemy w tej funkcji również cyfrę kontrolną występującą na samym końcu pliku. Następnie w funkcji odczytyjącej kod, przy użyciu drzewa, czytamy kolejno bity skompresowanego kodu, przesuwając się po drzewie aż do momentu, gdy dojdziemy do liścia, wtedy zapisujemy do pliku wyjściowego odpowiadającą mu sekwencję bitową. Przy zapisie xorujemy wcześniej odczytaną cyfrę kontrolna, jeśli wyniesie ona zero po zapisaniu wszystkich sekwencji, to poprawnie zdekompresowaliśmy plik.

# Komunikaty błędów

Program pokazuje następujące komunikaty:

* *Brak argumentow, uzyj -h, aby otrzymac pomoc* – Komunikat występuje podczas wywoływania programu bez argumentów
* *Nie podano nazwy pliku* – Plik nie został podany podczas wywołania programu
* *Nie udalo sie otworzyc pliku* – Problem z otwarciem pliku
* *Jeden z parametrow potrzebuje argumentu* – Nie podanie potrzebnego argumentu po wpisaniu flagi
* *Nieznany parametr* – Wypisanie niewystępującej flagi
* *Blad, nie udalo sie zdekompresowac pliku!* – Komunikat pojawia się, gdy podczas dekompresji znaleziony błąd powodujący zatrzymanie procesu
* *Nie udalo sie zarezerwowac pamieci na :*
  + *tablice nodes* – Brak miejsca w pamięci na tablicę przechowującą znaki
  + *tablice uzywana do zliczania wystapien ciagow bitow*
  + *tablice zawierajaca klucze slownika*
  + *zero jedynkowy kod huffmana dla danego klucza w slowniku*
  + *tablice znakow w funkcji pomocniczej add\_zeros\_to\_left()*
  + *tablice znakow tworzaca zapis binarny liczby dziesietnej (funkcja dec\_to\_bin\_string())*
* *Blad, nie udalo sie wczytac znaku z pliku! – użytkownik próbuje odczytać plik do którego nie ma odpowiednich uprawnień*
* *Blad, nie udalo sie zapisac znaku do pliku! - użytkownik próbuje zapisać do pliku nie posiadająć odpowiednich uprawnień*
* *Podano nieprawidlowe haslo – przy dekompresji zaszyfrowanego pliku wprowadzono niepoprawne hasło*