Specyfikacja implementacyjna programu realizującego kompresję plików wykorzystującego algorytm Huffmana

Autorzy: Adrian Chmiel, Mateusz Tyl

08.04.2023

Historia zmian dokumentu:

Autor:	Data:	Opis zmiany:	Wersja dokumentu
Mateusz Tyl	08.04.2023	Dodanie pierwszego szkicu	1.0
Mateusz Tyl	08.04.2023	Dodanie informacji o plikach	2.0
Adrian Chmiel	10.04.2023	Drobne poprawki	2.1
Mateusz Tyl	11.04.2023	Dodatkowe informacje do plików	2.2

Cel dokumentu

Celem tego dokumentu jest przedstawienie informacji o sposobie działania programu od strony technicznej. Zostanie omówiony każdy plik kodu programu oraz pliki Makefile i CMake.

Informacje ogólne

Program został napisany w całości w języku C. Wykorzystano także język powłoki bash oraz skrypty Makefile i CMake do implementacji funkcjonalności pomocniczych.
Program jest przystosowany do pracy i kompilacji na systemach Unixowych.
Do dyspozycji programisty przygotowano szeroki zakres testów do wykonania w celu sprawdzenia poprawności działania programu. W programie zaimplementowano kompresor oraz dekompresor służący do przetestowania poprawności działania kompresora.

Omówienie kodu źródłowego

Program oferuje kilka modułów, które współpracują ze sobą. Niektóre moduły pracują niezależnie od innych.

Folder "gen"

- gen.c -> Program pomocniczy służący do generowania gotowych testów do tego kompresora. Program generuje losowe znaki, zależnie od ustawień podanych przez użytkownika i zapisuje je w pliku.
  Lista argumentów: [nazwa pliku wyjściowego] [ilość znaków] [ilość różnych znaków, które mogą wystąpić w outpucie] [seed]
- testhufftree.c -> Program pomocniczy odpowiedzialny za wygenerowanie testu w wyniku którego możliwe było przeciążenie pamięci przez tworzenie drzewa Huffmana. Jedynym argumentem jest nazwa pliku wyjściowego
- testmax.c -> Program pomocniczy odpowiedzialny za wygenerowanie testu o wielkości ok. 1.5 MB. Argumentem jest nazwa pliku wyjściowego
- testo3.c -> Program pomocniczy odpowiedzialny za wygenerowanie testu testo3.in zawierającego wszystkie możliwe znaki/słowa dla wszystkich możliwych poziomów kompresji

Folder "src"

- alloc.* -> zawiera funkcje służące do bezpiecznej alokacji pamięci. Zwraca błędy jeżeli alokacja się nie powiedzie.
- bits_analyze.* -> zawiera funkcje służące do analizy kolejnych bitów zawartych w kolejnych znakach przy dekompresji. Sprawdza, czy alokacja pamięci się powiodła i wypisuje stosowny komunikat w przypadku błędu. Analizuje zapisany słownik w pliku. Funkcje korzystają z pomocniczej struktury dtree odwzorowującej drzewo binarne. W pliku nagłówkowym zdefiniowany jest enum, który pełni funkcję zmiennej przechowującej aktualny tryb analizy aktualnych bitów. W zależności od odczytywanych bitów zmienna przyjmuje odpowiedni tryb i przekazuje programowi, co ma robić. Tryby są następujące: dictRoad - tryb uaktywnia się, gdy przemieszczamy się w drzewie, dictWord - uaktywniany, gdy znajdziemy się w liściu w celu odczytania znaku/słowa, bitsToWords - służy do odczytywania skompresowanych danych po odczytaniu całości słownika
- countCharacters.* -> zawiera funkcje odpowiedzialne za zliczanie wystąpień wszystkich znaków w pliku wejściowym. W pliku nagłówkowym zdefiniowano strukturę count_t służącą do przechowywania ilości wystąpień każdego ze znaków oraz drzewa Huffmana. Struktura zawiera w sobie zmienną zliczającą znaki, zmienną przechowującą dany znak oraz trzy wskaźniki na struktury odpowiedzialne za realizację drzewa Huffmana. Plik źródłowy zawiera funkcje do dodawania nowych znaków do listy, sortowanie, zwalnianie pamięci, wyświetlanie listy.
- decompress.* -> zawiera funkcje odpowiedzialne za dekompresję. Funkcja compress jest główną funkcją dekompresującą, przyjmuje dwa argumenty -

nazwa pliku wejściowego, nazwa pliku wyjściowego. Funkcja compareBuffer sprawdza, czy aktualny fragment kodu w buforze odpowiada jakiejś literze. Jeżeli tak, to zapisuje tę literę do podanego pliku.

- dtree.* -> zawiera typ pomocniczego pseudodrzewa do dekompresji oraz związane z nim funkcje
- huffman.* -> zawiera główne funkcje realizujące algorytm Huffmana. Funkcje odpowiedzialne są za generację kodów Huffmana dla każdego znaku w pliku wejściowym. Tworzone drzewo Huffmana wykorzystuje strukturę count_t. Zaimplementowano tworzenie słownika na bazie drzewa Huffmana oraz ogólnej struktury kompresowanego pliku. Więcej szczegółów na ten temat w specyfikacji funkcjonalnej.
- list.* -> zawiera implementację listy liniowej służącej do przechowywania kodów znaków oraz funkcje służące do jej obsługi.
- main.c -> odpowiada za sprawdzanie danych wejściowych oraz sterowanie całym programem
- noCompress.* -> zawiera funkcje obsługujące plik w wypadku, gdy nie zachodzi żadna kompresja
- output.* -> zawiera funkcje realizujące zapis skompresowanego pliku w zależności od wybranego poziomu kompresji przez użytkownika.
- utils.* -> zawiera pomocnicze typy oraz funkcje, obsługę komunikatów wyświetlanych użytkownikowi. Zdefiniowany jest tutaj union pack pack_t służący jako bufor dla znaków.

Inne pliki w głównym katalogu

test -> katalog zawierający przykładowe pliki do przetestowania, w tym pliki dźwiękowe i graficzne

- CMake.sh -> skrypt automatyzujący kompilację przy pomocy CMake
- CMakeLists.txt -> zawiera komendy do kompilacji przy użyciu CMake
- compare.c -> służy do porównywania, czy dwa podane pliki są jednakowe (weryfikacja, czy otrzymaliśmy plik początkowy po wykonaniu kompresji, a następnie dekompresji) Użycie: ./compare <file1> <file2>
- encoding.sh -> skrypt sprawdzający na wiele sposobów kodowanie danego pliku (może wymagać instalacji dodatkowych pakietów). Informacje na temat instalacji wymaganych pakietów są zawarte w tym pliku.
- Makefile -> zawiera komendy do standardowej kompilacji oraz przykładowe testy programu
- README.md -> plik do wstawienia informacji na github

CMake

Przekazujemy do dyspozycji programisty skrypt CMake służący do automatyzacji procesu kompilacji. W katalogu głównym programu znajdują się do tego przeznaczone pliki: CMake.sh i CMakeLists.txt.

Makefile

W pliku Makefile znajdują się różnego rodzaju testy do sprawdzenia poprawności działania programu. Testów jest 14, każdy z nich jest oznaczony w następujący sposób: testX, gdzie X to numer testu od 1 do 14.
Listę testów wraz z ich opisem przedstawiono poniżej:

- test1 - Przypadek, gdy plik wejściowy jest pusty
- test2 - Próba dekompresji uszkodzonego pliku skompresowanego
- test3 - Test kompresji całego tekstu Pana Tadeusza
- test4 - Test kompresji i dekompresji z różnymi szyframi
- test5 - Test kompresji i dekompresji muzyki z szyfrowaniem
- test6 - Test kompresji i dekompresji zdjęcia z szyfrowaniem
- test7 - Test wygenerowany przy pomocy gen/gen.c (kompresja 8-bit)
- test8 - Test wygenerowany przy pomocy gen/gen.c (kompresja 12-bit)
- test9 - Test wygenerowany przy pomocy gen/gen.c (kompresja 16-bit)
- test10 - Test na rozbudowane drzewo Huffmana
- test11 - Wymuszenie dekompresji dla pliku nieskompresowanego
- test12 - Przypadek, gdy plik nie istnieje
- test13 - Test na brak kompresji i brak szyfrowania
- test14 - Test na brak kompresji z szyfrowaniem

Przeprowadzenie wszystkich testów za jednym razem możemy wykonać komendą `make test`