Specyfikacja funkcjonalna 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	25.03.2023	Dodanie pierwszego szkicu	1.0
Mateusz Tyl	08.04.2023	Dodanie informacji o nowych funkcjonalnościach, poprawki	2.0
Mateusz Tyl	08.04.2023	Dodanie informacji o debugowaniu	2.1
Adrian Chmiel	10.04.2023	Drobne poprawki	2.2
Mateusz Tyl	11.04.2023	Dodanie opisu słownika	2.3

Cel projektu

Przeznaczeniem progamu jest kompresja plików takich jak pliki tekstowe, graficzne, dźwiękowe z wykorzystaniem *algorytmu Huffmanna*.
Program oferuje różnego rodzaju funkcjonalności na które składają się:

- Kompresja danych w trybach 8, 12, 16 bitowym
- Możliwość zaszyfrowania danych wyjściowych
- Odpowiednio dostosowana struktura pliku wyjściowego w celu łatwego zaimplemetowania dekompresora
- Obsługa sum kontrolnych w celu sprawdzania poprawności pliku, opcjonalnie wymuszenie kompresji.

Teoria

Algorytm Huffmana wykorzystany w tym programie to przykład jednego z najprostszych i najłatwiejszych w implementacji algorytmów wykorzystujących kompresję bezstratną.
Zasada działania opiera się na utworzeniu dla każdego znaku w pliku nieskompresowanym kodu binarnego o zmiennej długości. Kody tworzy się za pomocą

drzewa binarnego, badając częstotliwości wystąpień każdego ze znaków. Dla częściej występujących znaków otrzymamy krótsze kody, dla rzadziej występujących dłuższe. Kompresując plik zamieniamy każdy znak na odpowiadający mu kod i zapisujemy kolejno po sobie w pliku wyjściowym.
Przykładowo, dla pliku zawierającego tekst `alamakota` otrzymamy następujący zestaw kodów:
Character: k, Code: 1111
Character: o, Code: 1110
Character: l, Code: 1101
Character: m, Code: 1100
Character: t, Code: 101
Character: \n, Code: 100
Character: a, Code: 0
Zatem po kompresji otrzymamy następujący ciąg danych:
011010110001111111010100
Plik skompresowany tworzony przez program zawiera jeszcze na początku nagłówek, słownik oraz ewentualnie na końcu dodatkowe zera wypełniające brakujące bity. Więcej o tym w kolejnych rozdziałach.

Dane wejściowe

Program wymaga przekazania dowolnego pliku, który ma zostać skompresowany. Program obsługuje prawidłowo pliki tekstowe, dźwiękowe, graficzne.
Dalsze instrukcje zostaną przedstawione w następnym rozdziale.

Argumenty wywołania programu

Program działa w trybie wsadowym. Wymaganymi parametrami są kolejno: nazwa pliku do skompresowania, nazwa pliku wyjściowego.
Lista parametrów opcjonalnych:

- `-o0` - brak kompresji
- `-o1` - kompresja 8 bitowa
- `-o2` - kompresja 12 bitowa
- `-o3` - kompresja 16 bitowa
- `-h` - wyświetl pomoc do programu
- `-c` - zaszyfruj wynik działania programu
- `-x` - wymuś kompresję
- `-d` - wymuś dekompresję

W przypadku niepodania argumentów opcjonalnych program automatycznie stwierdzi, czy plik nadaje się do ewentualnej dekompresji, a następnie wykona

kompresję (jeżeli dekompresja jest niemożliwa) lub dekompresję.

Przykład wywołania programu

- ./program input output
  Program wczyta plik o nazwie input i skompresuje go do pliku output z domyślnymi ustawieniami
- ./program input output -c
  Program wczyta plik o nazwie input i skompresuje go do pliku output z domyślnymi ustawieniami oraz włączonym szyfrowaniem
- ./program input output -o3 -c
  Program wczyta plik o nazwie input i skompresuje go do pliku output wykorzystując kompresję 16 bitową i włączone szyfrowanie
- ./program lub ./program -h
  Zostanie wyświetlona pomoc do programu
- ./program input output -x
  Program wczyta plik o nazwie input i skompresuje go do pliku output wykorzystując kompresję (umożliwia to np. ponowną kompresję pliku już skompresowanego)

Struktura pliku wyjściowego

Pierwsze cztery bajty pliku wyjściowego zarezerwowane są na nagłówek.
Pierwsze dwa bajty nagłówka to pierwsze litery nazwisk autorów - CT.
Kolejny bajt to maska, z której można odczytać szczegółowe informacje o pliku.
Ostatnim bajtem nagłówka jest wynik wyliczonej sumy kontrolnej.
Następnie pojawia się słownik, a na końcu znajdują się kolejno po sobie skompresowane dane w postaci kodów Huffmana o zmiennej długości.
W przypadku, gdy po zapisaniu wszystkich danych ilość bitów w ostatnim bajcie jest niekompletna, zostanie wykonane dopełnienie zerami. Jeżeli użytkownik wybierze zarówno brak kompresji, jak i brak szyfrowania, to otrzymuje on dokładnie ten sam plik, jak ten podany jako wejściowy.

Struktura maski w nagłówku pliku

Szablon bitowy: 0bKKSZCEEE

- K - sposób kompresji: 00 - brak, 01 - 8-bit, 10 - 12-bit, 11 - 16-bit
- S - szyfrowanie: 0 - nie, 1 - tak
- Z - zapisanie informacji, czy konieczne będzie usunięcie nadmiarowego znaku \0 z końca pliku podczas dekompresji

- C - dodatkowe sprawdzenie, czy ten plik jest skompresowany: 0 - nie, 1 - tak
- E - ilość niezapisanych bitów kończących (tj. dopełniających ostatni bajt) zapisana binarnie

Słownik

Kod Huffmana każdego znaku wraz z odpowiadającym mu znakiem można odczytać ze słownika, który znajduje się od razu po nagłówku. Do odczytania słownika będzie potrzebny odpowiedni algorytm.
Algorytm prezentuje się następująco:
Tworzymy pomocnicze drzewo binarne. Rozpoczynamy analizę bitów składających się na słownik. Znajdujemy się w korzeniu drzewa. W zależności na co napotkamy analizując kolejne bity (analizujemy kolejno po dwa) robimy to co następuje:

- 00 - przechodzimy po drzewie w dół do lewego syna, jeżeli ten jest nieodwiedzony, w przeciwnym razie do prawego syna
- 01 - to samo co 00, ale po tym przejściu znajdziemy się w liściu - wtedy otrzymujemy kod znaku w całości, kolejne 8/12/16 (w zależności od poziomu kompresji) bitów to znak, którego kod otrzymaliśmy.
- 10 - wycofanie się do ojca
- 11 - koniec słownika

W przypadku napotkania na 01 możemy z drzewa odczytać cały kod Huffmana dla konkretnego znaku. Kod czytamy od liścia w stronę korzenia. Z kolejnych 8/12/16 (w zależności od poziomu kompresji) bitów możemy odczytać kodowany znak.
Uwaga - zawsze po odczytaniu znaku (tzn. odczytaniu 8/12/16 bitów) w drzewie należy wykonać cofnięcie z aktualnej pozycji do ojca!

Przykład (dla kompresji 8 bitowej):
Nasz przykładowy słownik:

`0101100010000001011001000101100001100101`

Analizujemy pierwsze dwa bity

`01`0110001000000101100100010110000110010 1

Zatem przechodzimy do lewego syna. Znajdujemy się w liściu, co oznacza że właśnie otrzymaliśmy znak. Odczytujemy z drzewa kod Huffmana dla aktualnego znaku. W tym przypadku jest to 0. Lewą gałąź w drzewie oznaczamy zerem, prawą jedynką. Odczytujemy kolejne 8 bitów. Te bity to znak, którego kod Huffmana właśnie otrzymaliśmy.

01011000100000010110010001011000011001 01

Otrzymaliśmy zatem pierwszy znak i odpowiadający mu kod Huffmana.
Znak w postaci binarnej: 01100010, w postaci dziesiętnej: 98
Odpowiadający mu kod Huffmana to: 0
Dalej postępujemy tak samo, pamiętając, że jeżeli napotkamy na `11` to słownik zostaje zakończony. W przypadku braku kompresji słownik nie jest wcale zapisywany.

Szyfrowanie

Szyfrowanie pliku odbywa się za pomocą *Szyfrowania Vigenère'a*. Domyślnie kluczem szyfrowania jest: `Politechnika_Warszawska`

W celu zmiany klucza szyfrowania możemy skorzystać z argumentu `-c "klucz"` np.
`-c "Huffman"` ustawi klucz szyfrowania na `Huffman`

Ten rodzaj szyfrowania polega na przesuwaniu kolejnych zapisywanych znaków o wartości ASCII kolejnych znaków znajdujących się w kluczu szyfrowania. W momencie, gdy podczas szyfrowania znajdziemy się na końcu klucza, należy po prostu wrócić na jego początek i kontynuować szyfrowanie przechodząc po nim kolejny raz.
Szyfrowaniu podlega słownik oraz skompresowany tekst, a więc wszystko oprócz nagłówka w postaci czterech pierwszych bajtów pliku.

Komunikaty błędów

1. Błąd podczas wczytywania pliku wejściowego: `Input file could not be opened!`
2. Plik wejściowy jest pusty: `Input file is empty!`
3. Błąd podczas wczytywania pliku wyjściowego: `Output file could not be opened!`
4. Pominięcie niezindentyfikowanych argumentów: `Unknown argument! (ignoring...)`
5. Wystąpił błąd podczas alokowania pamięci: `Failed memory allocation!`
6. Napotkano na błąd zarządzania pamięcią podczas kompresji: `Compression memory failure!`
7. Napotkano na błąd zarządzania pamięcią podczas dekompresji: `Decompression memory failure!`

8. Podany szyfr nie umożliwia pomyślnej dekompresji: `Decompression encryption failure!`

Zwracane wartości

Program po zakończeniu pracy zwraca wartość typu całkowitego, która może być użyteczna w przypadku identyfikacji różnego rodzaju niepowodzeń:

- 0 - Program zakończył się pomyślnie
- 1 - Podano za mało argumentów
- 2 - Błąd podczas wczytywania pliku wejściowego
- 3 - Błąd podczas wczytywania pliku wyjściowego
- 4 - Pusty plik wejściowy
- 5 - Wymuszono dekompresję dla pliku, który nie może zostać zdekompresowany
- 6 - Błąd przy alokowaniu/dealokowaniu pamięci
- 7 - Podano nieprawidłowy szyfr przy dekompresji

Należy jednak pamiętać, że podanie nieprawidłowego szyfru może się również zakończyć powodzeniem, lecz uzyskany plik wynikowy nie będzie zgodny z oryginałem tj. plikiem przed kompresją.

Tryb debugowania

Tryb debugowania włączony poprzez podanie opcjonalnego parametru -DDEBUG podczas kompilacji programu, pozwala prześledzić działanie programu oraz wyświetlić niektóre komunikaty o błędach, niewidoczne w standardowym trybie. Makefile daje możliwość kompilacji w trybie debugowania poprzez komendę `make debug`. Obejmuje on następujące możliwości:

1. Wyświetlenie komunikatów informujących o nieprawidłowym alokowaniu/realokowaniu pamięci
2. Wyświetlenie aktualnych ustawień kompresji pliku
3. Wyświetlenie wygenerowanych kodów Huffmana dla każdego znaku
4. Wyświetlenie obliczonej sumy kontrolnej
5. Wyświetlenie aktualnego klucza szyfrowania
6. Wyświetlenie, ile bitów końcowych zostało dopełnionych w ostatnim bajcie
7. Wyświetlenie statystyk na temat skompresowanego pliku

W Makefile znajduje się też dodatkowy tryb pełnego debugowania możliwy do uzyskania komendą `make dbfull`, który oprócz wszystkich powyższych informacji wyświetla również na bieżąco każdy symbol drukowany do pliku przy kompresji.