

Specyfikacja funkcjonalna programu realizującego kompresję plików wykorzystującego algorytm Huffmana

Autorzy: Adrian Chmiel, Mateusz Tyl

10.06.2023

Historia zmian dokumentu:

| Autor: | Data: | Opis zmiany: | Wersja dokumentu |
|---------------|------------|---------------------------|------------------|
| Adrian Chmiel | 10.06.2023 | Dodanie pierwszego szkicu | 1.0 |

Cel projektu

Ten projekt jest kontynuacją projektu poprzedniego realizującego kompresję algorytmem Huffmana w języku C. Tym razem zadaniem było utworzenie programu w języku Java oferującego następujące funkcjonalności:

- Dekompresja plików pochodzących z wcześniej napisanego kompresora w języku C
- Sprawdzanie poprawności pliku poprzez m.in. sumę kontrolną
- Wizualizacja słownika poszczególnych znaków dla konkretnego pliku

Ze względu na ograniczoną czytelność drzewa obrazującego słownik dla wyższych poziomów kompresji, wizualizacja jest realizowana jedynie dla plików skompresowanych 8-bitowo.

Teoria

Algorytm Huffmana jest jednym z najprostszych i łatwych do zaimplementowania algorytmów wykorzystujących bezstratną kompresję danych. Jego działanie opiera się na tworzeniu zmiennodługościowych kodów binarnych dla poszczególnych znaków. Kody są generowane w oparciu o drzewo binarne tworzone na podstawie częstotliwości wystąpień znaków. Im częściej występuje dany znak, tym krótszy jest przypisany mu kod, a dla znaków występujących rzadziej kod jest dłuższy. W procesie kompresji, każdy znak w pliku zostaje zamieniony na odpowiadający mu kod i zapisany w formie ciągu binarnego.

Na przykładzie pliku zawierającego tekst "alamakota" możemy zobaczyć, jak działa ten algorytm. Dla tego konkretnego przypadku zestaw kodów wyglądałby następująco:

Znak 'k' otrzymuje kod: 1111
Znak 'o' otrzymuje kod: 1110
Znak 'l' otrzymuje kod: 1101
Znak 'm' otrzymuje kod: 1100
Znak 't' otrzymuje kod: 101
Znak '\n' otrzymuje kod: 100
Znak 'a' otrzymuje kod: 0

Po zastosowaniu kompresji, plik zostaje zapisany jako ciąg skompresowanych danych:

0110101100011111101010100

Następnie odczytując plik (zakładając, że jest to prawidłowy plik pochodzący z kompresora naszego autorstwa) należy zauważyć, że jest on podzielony na 3 sekcje:

- Nagłówek zawierający inicjały autorów, wszelkie flagi zawierające ustawienia kompresji oraz wynik sumy kontrolnej
- Zapis słownika dzięki któremu można dokonać późniejszej dekompresji
- Skompresowany zapis pliku

Na podstawie tych informacji możemy wywnioskować, że w dużym skrócie poprawna dekompresja pliku będzie przebiegać w sposób następujący:

1. Weryfikacja poprawności pliku na podstawie informacji z nagłówka
2. Odczytanie słownika dla danego pliku
3. Porównywanie kodów napotkanych przy analizie pliku z tymi, które znajdują się w słowniku
4. Zapisywanie do pliku wyjściowego odpowiedniego symbolu, jeżeli kody się pokrywają

Dane wejściowe

Program wymaga przekazania dowolnego pliku, który pochodzi z kompresora w języku C napisanego przez nas w ramach poprzedniego projektu. W przypadku podania innego pliku dekompresja się nie powiedzie, a program zwróci jeden z kodów błędu.

Uruchamianie programu

Program może być uruchomiony w jednym z dwóch trybów:

- **wsadowy** - jedynym wymagany parametrem jest nazwa pliku, który chcemy zdekompresować

Lista parametrów opcjonalnych:

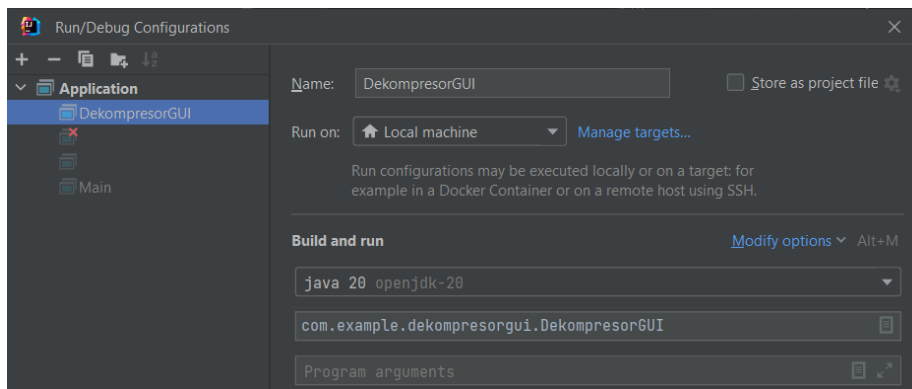
- * **-h** - wyświetl pomoc do programu
 - * **-c** - zaszyfruj wynik działania programu
 - * **-d** - wymuś dekompresję
- **graficzny** - wyświetla graficzne okienko umożliwiające wybór pliku oraz pożądaną opcję dekompresji pliku, a następnie w przypadku plików skompresowanych 8-bitowo wyświetlenie wizualizacji słownika

Uwaga! Przy uruchamianiu dekompresora w trybie wsadowym wizualizacja słownika nie zostanie wyświetlona.

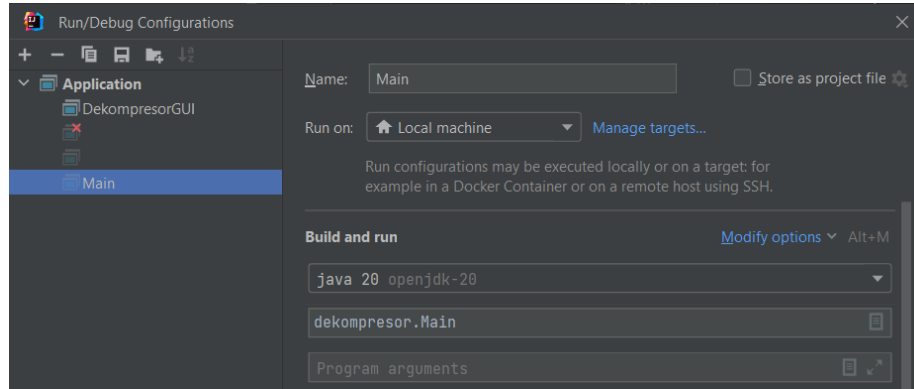
Uruchamianie programu przez IntelliJ IDEA

Niezwykle pomocnym narzędziem przy realizacji tego projektu był definitywnie IntelliJ IDEA od JetBrains dostępny za darmo w ramach licencji studenckiej. W celu uruchomienia tego programu bez przeszkód w tym środowisku, po pobraniu całego projektu najlepiej będzie najpierw ustawić katalog DekompresorGUI\src\main\java jako Source Root, a następnie katalog główny jako kolejny Source Root. Następnie możemy sobie przygotować dwie konfiguracje takie jak widoczne na poniższych zrzutach ekranu:

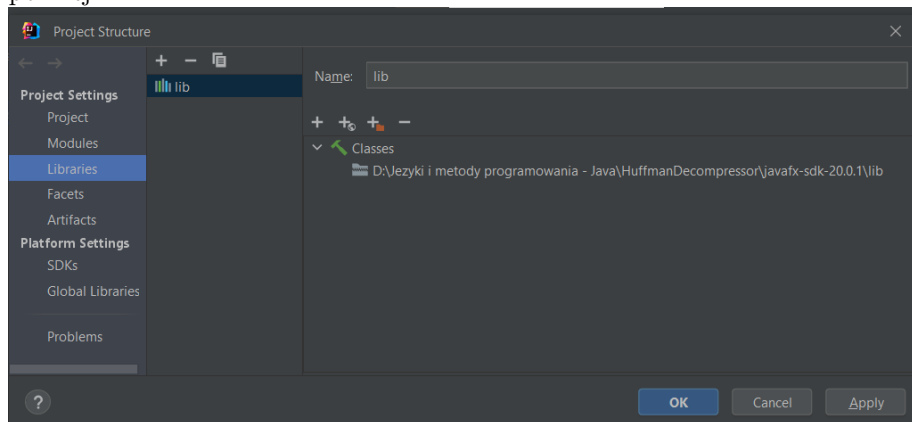
- tryb graficzny



- tryb wsadowy



Ostatnim krokiem będzie dodanie biblioteki JavaFX do struktury projektu, co możemy zrobić poprzez przejście w `File > Project Structure > Libraries`. Następnie klikamy na '+' i odnajdujemy ścieżkę z JavaFX, który powinien być dołączony razem z projektem. Koniecznie wybieramy folder **lib**. Po wykonaniu tego zadania całość powinna się prezentować mniej więcej tak, jak na zrzucie poniżej:



Przykłady wywołania programu

Przedstawione zostały jedynie przykłady dla trybu wsadowego przy założeniu, że znajdujemy się w głównym katalogu projektu (w innym przypadku wymagane może być podanie innej ścieżki do pliku *Dekompresor.jar*)

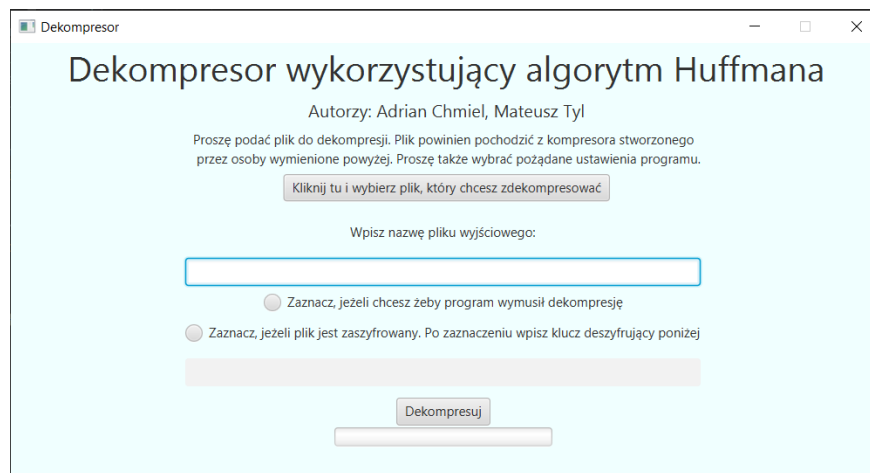
- **java -jar Dekompresor.jar test/8-bit/easy-text-test output**
Program wczyta plik o nazwie easy-text-test z katalogu test/8-bit i go zdekompresuje
- **java -jar Dekompresor.jar test/8-bit/cipher8 output -c**

Program wczyta plik o nazwie cipher8 z katalogu test/8-bit i zdekompresuje go do pliku output z włączonym szyfrowaniem używając domyślnego kodu "Politechnika_Warszawska"

- **java -jar Dekompresor.jar -h**
Zostanie wyświetlona pomoc do programu

W przypadku trybu graficznego GUI jest bardzo przejrzyste i intuicyjne, jak widać na załączonym niżej zdjęciu oraz jego obsługa nie powinna sprawiać najmniejszych problemów. Składa się ono z następujących elementów:

- przycisk do wyboru pliku, który chcemy zdekompresować
- pole tekstowe do wpisania względnej ścieżki pliku wyjściowego
- pola wyboru, w których możemy dostosować ustawienia
- pole tekstowe do wpisania szyfru inny niż domyślny (*dostępne jedynie po zaznaczeniu opcji szyfrowania*)
- przycisk **Dekompresuj** uruchamiający faktyczną część programu
- pasek ładowania przedstawiający obecny stan dekompresji



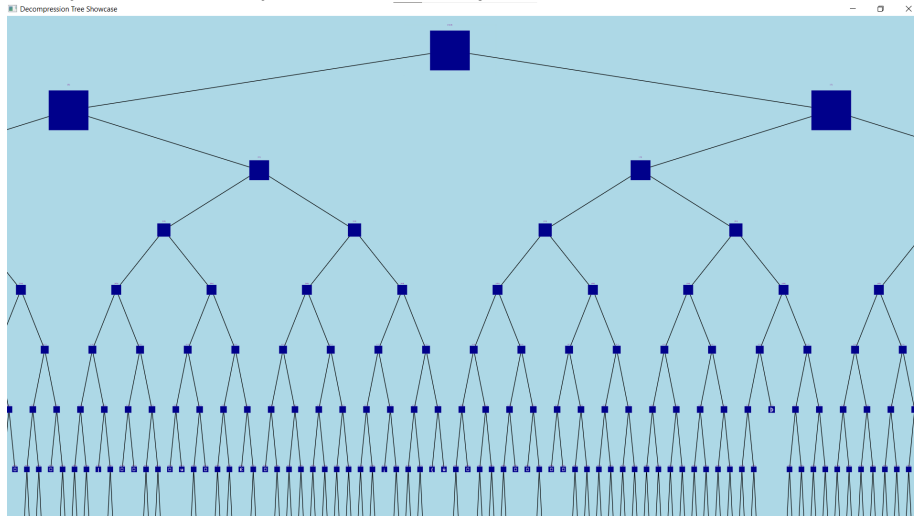
Wizualizacja słownika

Wizualizacja słownika w postaci drzewa wyświetla się zaraz po ukończeniu dekompresji (*lecz jedynie, gdy plik wejściowy został skompresowany 8-bitowo*). Domyślnie jest mocno zbliżona do korzenia i zdecydowana większość drzewa może być niewidoczna na pierwszy rzut oka, dlatego zaimplementowana została możliwość przeciągania po okienku przy pomocy lewego klawisza myszy oraz przybliżania i oddalania widoku przy pomocy rolki. Wizualizacja składa się z węzłów tj. kwadratów połączonych w odpowiedni sposób przy pomocy linii. Węzły zawierać mogą poszczególne dane:

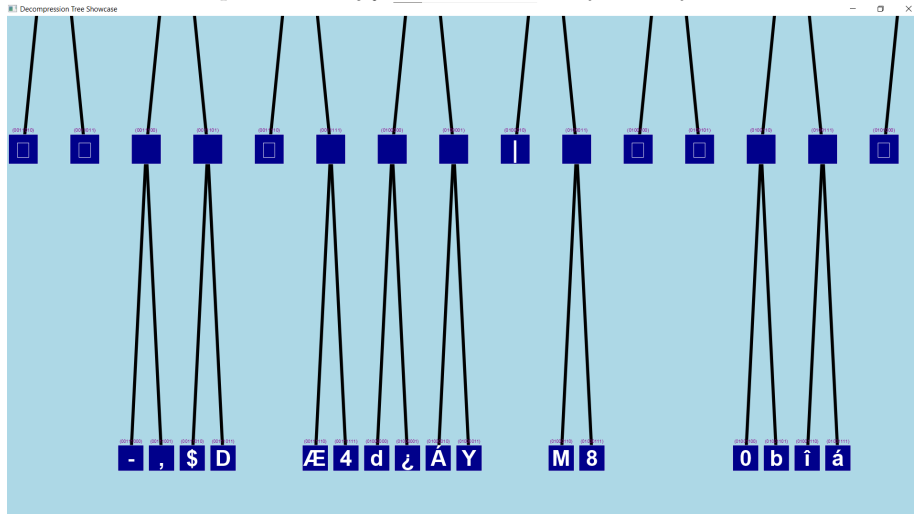
- kod odpowiadający danemu węzłowi znajdujący się bezpośrednio nad nim
- symbol odpowiadający danemu kodowi (*jedynie, gdy jest to liść*)

Przykładowe zrzuty ekranu wizualizacji zawarte są poniżej:

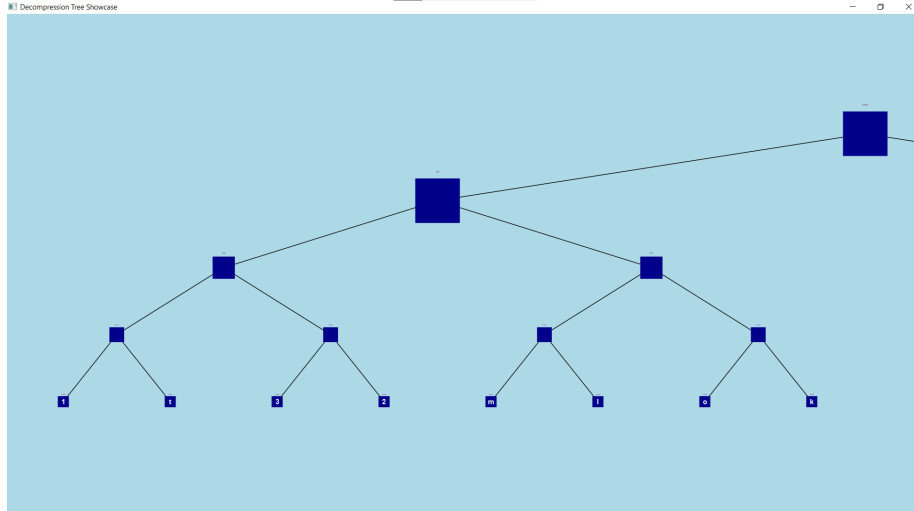
- Maksymalnie oddalony widok wizualizacji



- Zbliżenie na liście przedstawiające możliwość odczytania symboli i kodów



- Przykładowy fragment wizualizacji dla prostszego pliku



Struktura pliku wejściowego

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

0101100010000001011001000101100001100101

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.

0101100010000001011001000101100001100101

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.

Struktura pliku wyjściowego

Plikiem wyjściowym jest po prostu zdekompresowany plik, który jest identyczny z plikiem wcześniej skompresowanym przy użyciu kompresora naszego autorstwa w języku C.

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 niezidentyfikowanych argumentów: `Unknown argument! (ignoring...)`
5. Podany szyfr nie umożliwia pomyślnej dekompresji: `Decompression encryption failure!`
6. Napotkano błąd przy zapisie pliku wyjściowego: `Output file error!`
7. Napotkano błąd przy zapisie pliku pomocniczego do GUI: `Data file error!`

8. Napotkano błąd przy zapisie pliku realizującego wizualizację słownika:
`Tree file error!`

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 przy operacjach na pliku wejściowym
- 3 - Błąd przy operacjach na pliku wyjściowym
- 4 - Pusty plik wejściowy
- 5 - Podano plik, który nie może zostać zdekompresowany
- 6 - 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.