Dokumentacja końcowa

Dekompresor plików z wykorzystaniem algorytmu Huffmana w języku Java

Łukasz Gumienniczuk, Paweł Rusak

4 czerwca 2023

Spis treści

1 Cel projektu

Celem projektu było stworzenie programu w języku Java17, który dekompresuje pliki skompresowane przy pomocy kompresora w języku C11. Program dekompresuje podany przez użytkownika skompresowany plik i zapisuje wynik dekompresji do podanego przez użytkownika pliku.

2 Flagi

Program obsługuje flagi zawarte w poniższej tabeli. Flagę lub flagi należy podać przy wywołaniu programu po pliku wejściowym i wyjściowym.
Dostępne flagi można można wyświetlić korzystając z flagi `-h` lub `-help`

Flaga	Opis flagi
`-h` lub `-help`	Program wyświetli instrukcje obsługi i zakończy działanie
`-v`	Program wyświetli dodatkowe informacje na temat kompresji pliku
`-t`	Program wyświetli strukturę drzewa w terminalu
`-s`	Program wyświetli słownik w terminalu

Poniżej przedstawiamy przykład użyciu flag `-v`, `-t` oraz `-s` dla pliku `.txt` skompresowanego pliku `.txt` o treści "Testowy plik".

Widok dodatkowych informacji na temat kompresji pliku

```
Naglowek: #LP#
Stopien kompresji: 8
Reszta1: 1
Reszta2: 0
Suma kontrolna: -30
```

Widok drzewa

```
Root:
   |-- Bit: 0
   |   |-- Bit: 0
   |   |   |-- Bit: 0
   |   |   |   |-- Bit: 0, Bit Representation: 108
   |   |   |    -- Bit: 1, Bit Representation: 112
   |   |    -- Bit: 1
   |   |       |-- Bit: 0, Bit Representation: 32
   |   |        -- Bit: 1, Bit Representation: 121
   |    -- Bit: 1
   |       |-- Bit: 0
   |       |   |-- Bit: 0, Bit Representation: 119
   |       |    -- Bit: 1, Bit Representation: 111
   |        -- Bit: 1
   |           |-- Bit: 0, Bit Representation: 116
   |            -- Bit: 1, Bit Representation: 115
    -- Bit: 1
      |-- Bit: 0
      |   |-- Bit: 0
      |   |   |-- Bit: 0, Bit Representation: 101
      |   |    -- Bit: 1, Bit Representation: 84
      |    -- Bit: 1, Bit Representation: 10
       -- Bit: 1
          |-- Bit: 0, Bit Representation: 107
           -- Bit: 1, Bit Representation: 105
```

Widok słownika

```
Code: 0000, Bit representation: 108
Code: 0001, Bit representation: 112
Code: 0010, Bit representation: 32
Code: 0011, Bit representation: 121
Code: 0100, Bit representation: 119
Code: 0101, Bit representation: 111
Code: 0110, Bit representation: 116
Code: 0111, Bit representation: 115
Code: 1000, Bit representation: 101
Code: 1001, Bit representation: 84
Code: 101, Bit representation: 10
Code: 110, Bit representation: 107
Code: 111, Bit representation: 105
```

3 Instrukcja użycia

Instrukcje obsługi programu można znaleźć w katalogu głównych programu w pliku `README.md`.

Poniżej prezentujemy sposób uruchomienia skompilowanego programu:

```
java -jar target\decompressor.jar {in_file} {out_file} <flags>
```

Przykłady:

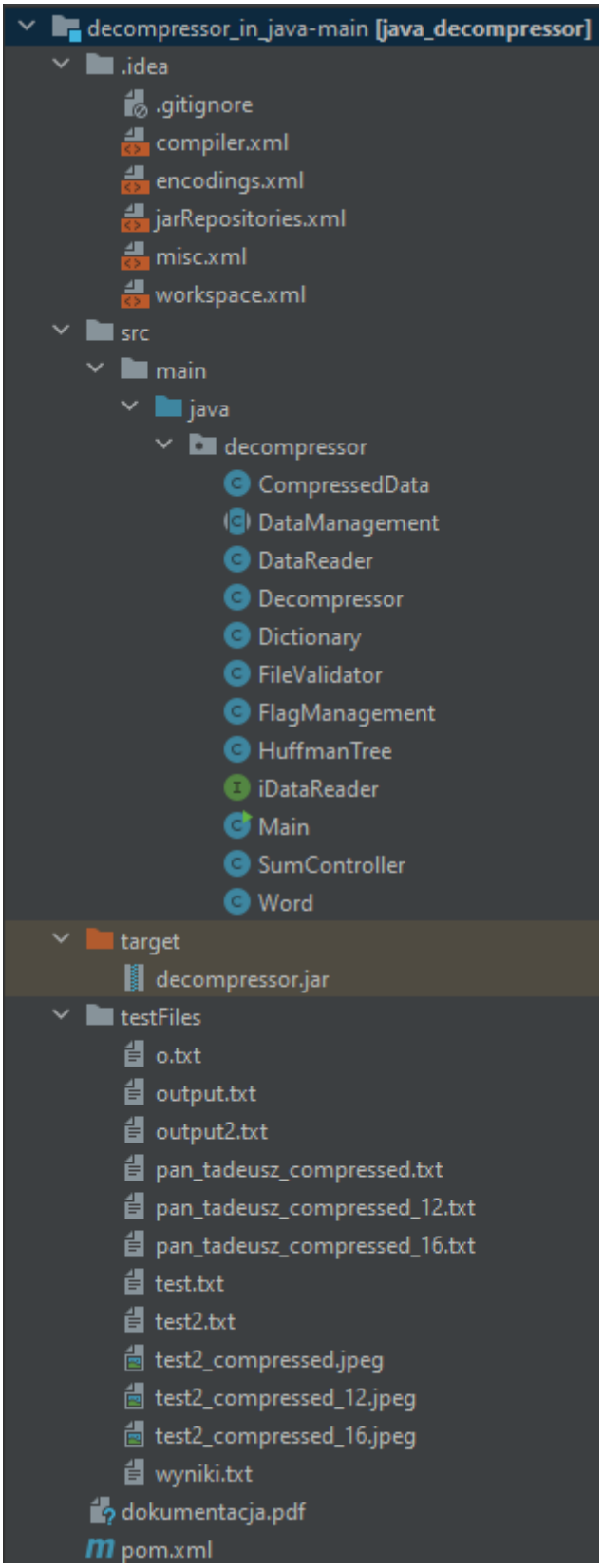
- `java -jar target\decompressor.jar .\testFiles\pan_tadeusz_compressed.txt .\testFiles\wyniki.txt`
- `java -jar target\decompressor.jar .\testFiles\pan_tadeusz_compressed_16.txt .\testFiles\pan_tadeusz_nowy.txt -v -t`
- `java -jar target\decompressor.jar .\testFiles\test2_compressed_12.jpeg .\testFiles\nasze_zdjecie.jpeg -t`

Uwaga:

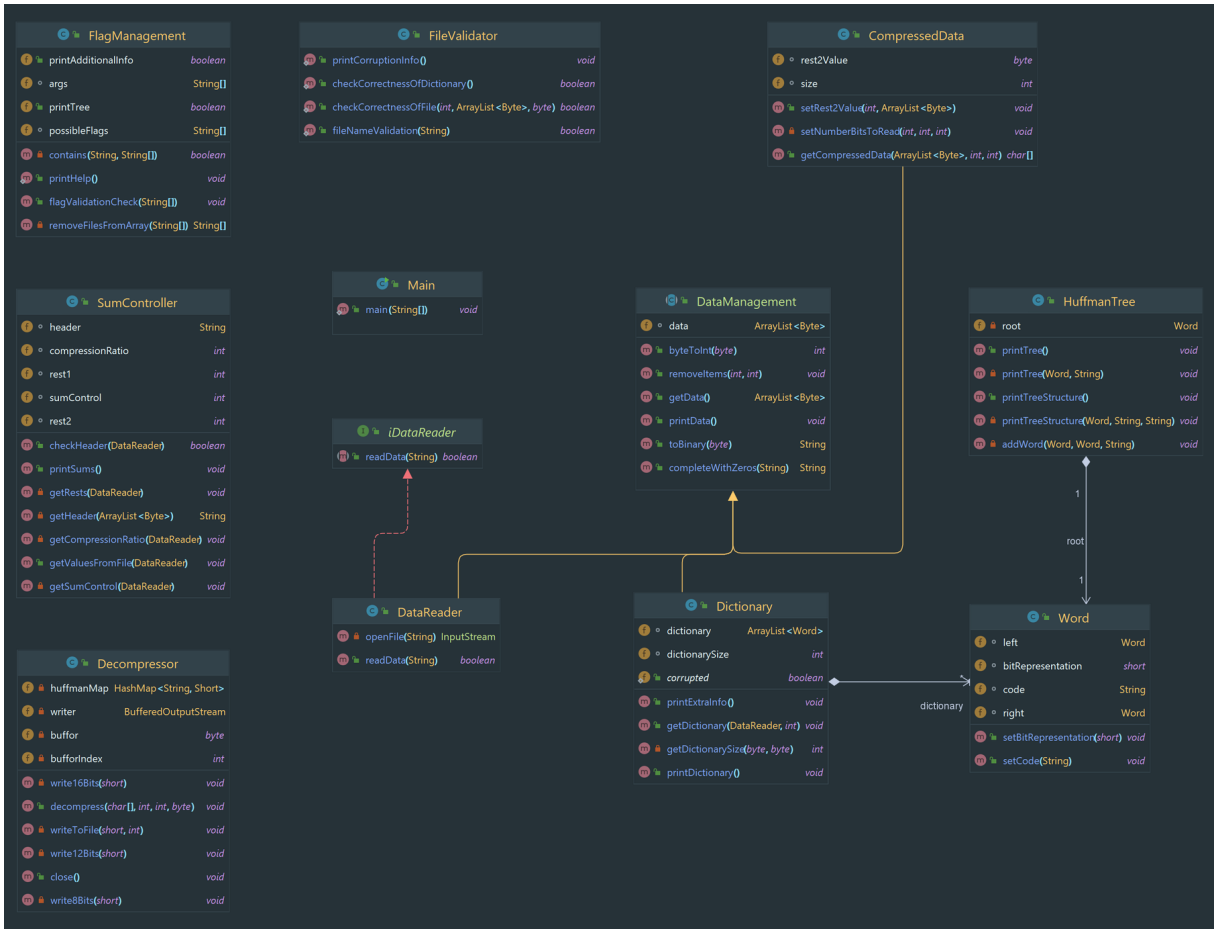
- Jeżeli plik jest jednocześnie plikiem wejściowym i wyjściowym to plik ten zostanie nadpisany wynikiem działania programu.
- Pojawienie się flagi `-h` lub `-help` skutkuje wyświetleniem pomocy użytkownika i zakończeniem działania programu.
- Pojawienie się nieznanej flagi skutkuje przerwaniem działania programu i wyświetleniem pomocy użytkownika.

4 Struktura programu

4.1 Struktura folderów



4.2 Struktura klas



5 Wykorzystane klasy

Poniżej prezentujemy najważniejsze klasy, które wykorzystujemy w naszym programie.

5.1 Main

Klasa `Main` w pakiecie `decompressor` jest punktem wejściowym dla aplikacji do dekompresji danych. Ta klasa wykonuje następujące działania:

- Weryfikuje, czy podano odpowiednią liczbę argumentów (dwa: nazwę pliku wejściowego i wyjściowego).
- Sprawdza poprawność nazwy pliku wyjściowego.
- Inicjalizuje obiekty różnych klas, takich jak `FlagManagement`, `DataReader`, `SumController`, `Dictionary`, `HuffmanTree` i `CompressedData`.
- Czyta dane z pliku wejściowego i weryfikuje, czy plik jest skompresowany.
- Oblicza i, jeśli to konieczne, wyświetla sumy kontrolne dla danych.
- Buduje słownik do dekompresji.
- Buduje drzewo Huffmana i, jeśli to konieczne, wyświetla jego strukturę.
- Próbuje zdekompresować dane i zapisać wyniki do pliku wyjściowego.

Wszystkie te operacje są zabezpieczone przez odpowiednie warunki, które sprawdzają poprawność danych i kończą program, jeśli coś jest nie tak. Kluczowymi elementami są operacje sprawdzania integralności danych i poprawności struktur danych, takich jak słownik czy drzewo Huffmana.

W razie wykrycia problemu, program informuje użytkownika o tym i zamyka się z kodem błędu. Podobnie, w przypadku problemów z operacjami I/O, wyjątki są odpowiednio obsługiwane i błędy są zwracane użytkownikowi.

5.2 DataReader

Klasa `DataReader` dziedziczy z `DataManagement` i implementuje interfejs `iDataReader`. Jej zadaniem jest odczytywanie danych z plików. Metoda `openFile` próbuje otworzyć plik, a metoda `readData` odczytuje z niego dane do bufora i dodaje je do `ArrayList`. Jeżeli odczyt danych przebiegnie pomyślnie, metoda `readData` zwraca `true`, a w przeciwnym przypadku `false`.

5.3 Dictionary

Klasa `Dictionary` dziedziczy z `DataManagement`. Zajmuje się zarządzaniem słownikiem używanym w dekompresji danych. Konstruktor klasy inicjalizuje puste słownik oraz ustawia flagę `corrupted` na wartość `false`. Metoda `getDictionarySize` pobiera rozmiar słownika na podstawie dwóch bajtów. Metoda `getDictionary` odczytuje dane ze słownika na podstawie kompresji, a następnie usuwa te dane z `DataReader`.

5.4 Word

Klasa `Word` reprezentuje pojedyncze słowo, które jest elementem drzewa Huffmana. Każde słowo ma reprezentację bitową, kod i referencje do swoich dzieci w drzewie (lewe i prawe). Konstruktor klasy inicjalizuje te pola, z wyjątkiem kodu, który jest ustawiany za pomocą metody `setCode`.

5.5 HuffmanTree

Klasa `HuffmanTree` reprezentuje drzewo Huffmana. Konstruktor klasy tworzy drzewo na podstawie słownika dostarczanego jako argument. Metoda `addWord` jest używana do dodawania słów do drzewa na podstawie ich kodów. Metody `printTree` i `printTreeStructure` są używane do wyświetlenia struktury drzewa.

6 Opis interfejsów oraz zmiennych publicznych

6.1 Klasa `CompressedData`

- `int size` – zmienna przechowująca rozmiar skompresowanych danych. Jest zainicjowana jako 0 w konstruktorze klasy.
- `byte rest2Value` – zmienna przechowująca wartość ostatniego bajtu danych.
- `CompressedData()` – konstruktor klasy, który inicjalizuje zmienną `size` jako 0.
- `public void setRest2Value(int rest2, ArrayList<Byte> data)` – metoda służy do ustawienia wartości zmiennej `rest2Value`.
- `public char[] getCompressedData(ArrayList<Byte> data, int rest1, int rest2)` – metoda zwraca skompresowane dane w formie tablicy znaków. Przyjmuje jako argumenty listę bajtów do kompresji oraz dwie wartości typu int: `rest1` i `rest2`, które reprezentują ilość bitów, które należy pominąć na początku i na końcu danych.

6.2 Klasa `DataManagement`

- `ArrayList<Byte> data` – zmienna przechowująca listę bajtów, które są danymi obsługiwanymi przez klasę.
- `public ArrayList<Byte> getData()` – metoda zwracająca listę bajtów przechowywaną w zmiennej `data`.
- `public void removeItems(int firstIndex, int lastIndex)` – metoda umożliwiająca usunięcie elementów z listy `data`. Usuwa elementy od indeksu `firstIndex` do `lastIndex`.
- `public void printData()` – metoda umożliwiająca wydrukowanie danych przechowywanych w zmiennej `data`.
- `public String toBinary(byte a)` – metoda konwertująca bajt na ciąg binarny. Wykorzystuje do tego metodę `byteToInt` i `Integer.toBinaryString`.
- `public int byteToInt(byte a)` – metoda konwertująca bajt na wartość całkowitą.
- `public String completeWithZeros(String binary)` – metoda uzupełniająca ciąg binarny zerami do długości 8 bitów.

6.3 Klasa `DataReader`

- `DataReader()` – konstruktor klasy. Inicjalizuje zmienną `data` jako nową listę bajtów.
- `public boolean readData(String fileName)` – metoda odpowiedzialna za odczyt danych z pliku o podanej nazwie. Używa do tego prywatnej metody `openFile`. Dane odczytane z pliku są dodawane do listy `data`. W przypadku udanego odczytu danych zwraca `true`, w przeciwnym przypadku `false`.

6.4 Klasa `Decompressor`

- `Decompressor(Dictionary dictionary, String fileName)` – konstruktor klasy, który tworzy mapę szybkiego wyszukiwania kodu Huffmana na jego binarną reprezentację. Używany jest także do inicjalizacji obiektu `BufferedOutputStream` do zapisu wyników dekompresji do pliku.
- `public void decompress(char[] dataToDecompress, int compressionRatio, int restControl, byte restValue)` – metoda jest odpowiedzialna za dekompresję danych. Wykorzystuje mapę Huffmana stworzoną podczas inicjalizacji klasy do przetwarzania kodów binarnych z powrotem na ich pierwotne reprezentacje. Na podstawie określonego współczynnika kompresji wywoływane są odpowiednie metody do zapisu danych.
- `public void close()` – metoda jest odpowiedzialna za zamykanie strumienia `BufferedOutputStream` używanego do zapisywania wyników dekompresji.

6.5 Klasa `Dictionary`

- `Dictionary()` – konstruktor klasy, który tworzy nową listę słów i ustawia rozmiar słownika na 0.
- `public void printDictionary()` – metoda drukująca wszystkie słowa (reprezentacje bitowe i kody) zawarte w słowniku.
- `public void printExtraInfo()` – metoda wyświetlająca dodatkowe informacje na temat słownika, w tym pełną zawartość słownika oraz jego rozmiar.
- `public void getDictionary(DataReader data, int compressionRatio)` – metoda jest odpowiedzialna za ekstrakcję słownika z danych odczytanych z pliku. Przetwarza dane wejściowe i tworzy odpowiednie obiekty `Word`, które dodaje do listy słownikowej.

6.6 Klasa `FileValidator`

- `public static boolean checkCorrectnessOfFile(int sumControl, ArrayList<Byte> data, byte rest2Value)` – metoda sprawdzająca poprawność pliku poprzez porównanie przekazanej wartości kontrolnej z wartością wyliczoną na podstawie danych.
- `public static boolean fileNameValidation(String fileName)` – metoda sprawdzająca poprawność nazwy pliku.
- `public static boolean checkCorrectnessOfDictionary()` – metoda sprawdzająca poprawność słownika.
- `public static void printCorruptionInfo()` – metoda wyświetlająca informację o uszkodzeniu pliku i kończąca działanie programu.

6.7 Klasa `FlagManagement`

- `public boolean printTree` – zmienna określa, czy struktura drzewa powinna być wyświetlona.
- `public boolean printAdditionalInfo` – zmienna określa, czy powinny być wyświetlone dodatkowe informacje.
- `public static void printHelp()` – metoda wyświetlająca informacje o dostępnych flagach i kończy działanie programu.
- `public void flagValidationCheck(String[] args)` – metoda sprawdzająca poprawność flag podanych przez użytkownika. Jeżeli są one niepoprawne, wywoływana jest metoda `printHelp`.

6.8 Klasa `HuffmanTree`

- `public HuffmanTree(Dictionary dictionary)` – konstruktor klasy, który tworzy nowe drzewo Huffmana na podstawie podanego słownika. Zaczyna od stworzenia korzenia z symulowanym węzłem, a następnie dodaje do niego słowa na podstawie kodu Huffmana zawartego w słowniku.
- `public void printTree()` – metoda służąca do wyświetlenia drzewa Huffmana. Wywołuje metodę pomocniczą `printTree` z korzeniem drzewa jako argumentem początkowym i pustym prefixem.
- `public void printTreeStructure()` – metoda służąca do wyświetlenia struktury drzewa Huffmana. Wywołuje metodę pomocniczą `printTreeStructure` z korzeniem drzewa jako argumentem początkowym, pustym wcięciem i znakiem "-" jako początkowym bitem.

6.9 Interfejf `iDataReader`

- `public boolean readData(String fileName)` – metoda wczytująca dane z pliku o nazwie `fileName`.

6.10 Klasa `Main`

- `public static void main(String[] args)` – główna metoda programu, która steruje całą procedurą dekompresji. Na początku waliduje argumenty wejściowe, a następnie korzysta z innych klas do przeprowadzenia procesu dekompresji.

6.11 Klasa `SumController`

- `public void getValuesFromFile(DataReader data)` – metoda odczytująca różne wartości z pliku, takie jak stopień kompresji, reszty i sumę kontrolną.
- `public boolean checkHeader(DataReader data)` – metoda sprawdzająca, czy nagłówek pliku jest poprawny.
- `public void printSums()` – metoda wyświetlająca różne informacje na temat pliku, takie jak nagłówek, stopień kompresji, reszty i sumę kontrolną.

6.12 Klasa `Word`

- `public short bitRepresentation` – zmienna przechowująca krótką binarną reprezentację danego słowa.
- `public String code` – zmienna przechowująca kod słowa.
- `public Word left` – zmienna wskazująca na lewe poddrzewo w drzewie Huffmana.
- `public Word right` – zmienna wskazująca na prawe poddrzewo w drzewie Huffmana.
- `Word(short bitRepresentation)` – konstruktor klasy tworzący nowy obiekt klasy Word z podaną krótką binarną reprezentacją. Lewe i prawe poddrzewo są inicjalizowane jako `null`.
- `public void setBitRepresentation(short bit)` – metoda pozwalająca ustawić binarną reprezentację danego słowa.
- `public void setCode(String code)` – metoda pozwalająca ustawić kod danego słowa.

7 Wykorzystane struktury danych

Mapa [1]

Mapy to kontenery lub tablice asocjacyjne, które tylko zawierają indeksy i wartości, które są nam potrzebne. W dodatku indeksem może być nie tylko liczba całkowita dodatnia, ale także liczba rzeczywista, ciąg znaków, ogromne liczby typu long long, pary i różne rodzaje struktur, które zawierają operator porównania. Podsumowując, elementem mapy jest para, która składa się z klucza i wartości. Elementy mapy są uporządkowane zgodnie z kluczem.

8 Testy

Nasz program przetestowaliśmy na różnych plikach różnych formatów. Program w każdym przypadku działał poprawnie i plik skompresowany na kompresorze w C11, a następnie zdekompresowany na dekompresorze w Javie17 był identyczny jak plik pierwotny.

9 Podział pracy

- Sprawdzenie nagłówka – Łukasz Gumienniczuk;
- Odczyt pliku jako binarny – Łukasz Gumienniczuk;
- Obsługa flag – Łukasz Gumienniczuk;
- Sumy kontrolne – Łukasz Gumienniczuk;
- Odtworzenia słownika – Łukasz Gumienniczuk;
- Odtworzenie drzewa – Paweł Rusak;
- Dekompresja pliku na podstawie kodów binarnych i skompresowanej części pliku – Paweł Rusak;
- Zapis do pliku – Łukasz Gumienniczuk, Paweł Rusak;
- Stworzenie pliku .jar – Łukasz Gumienniczuk;
- Testy – Łukasz Gumienniczuk, Paweł Rusak;
- Dokumentacja – Paweł Rusak, Łukasz Gumienniczuk.

Literatura

[1] https://www.algorytm.edu.pl/stl/mapa-tablica-asocjacyjna, dostęp na 4 czerwca 2023