Specyfikacja Implementacyjna

programu file-compressor

Jan Gębal i Jan Brzozowski

06.04.2023

1. Spis Treści

2. Informacje Ogólne

Program ma za zadanie zakodować plik na podstawie algorytmu Huffmana w celu zmniejszenia rozmiaru pliku. Program ma również za zadanie odkodować tak zapisany plik. Program uruchamiany jest następująco:

./compressor -x input-file -o output-file -compress-rate [-c password -v -h]

gdzie :

- -x jest flagą odpowiadającą za plik wejściowy, a input-file to przykładowy plik wejściowy
- -o jest flagą odpowiadającą za plik wyjściowy, a output-file to przykładowy plik wyjściowy
- -compress-rate jest to jedna z flag odpowiedzialnych za stopień kompresji, wyróżniamy następujące :
    - -1 kompresja 8-bitowa
    - -2 kompresja 12-bitowa
    - -3 kompresja 16-bitow

W nawiasie kwadratowym znajdują się flagi opcjonalne, możemy wyróżnić następujące:

- -c jest flagą odpowiadającą za hasło, a password jest to przykładowe hasło
- -v jest flagą odpowiadającą za wyświetlanie dodatkowych informacji dotyczących kompresji oraz dekompresji
- -h jest flagą odpowiadającą za wyświetlanie pomocy

Szczegółowe wyjaśnienie funkcjonalności trybów, formatu plików, struktury folderu oraz składni programu znajduje się w specyfikacji funkcjonalnej projektu file-compressor.

3. Opis modułów

File-compressor składa się z 9 modułów, z czego jeden to plik main.c, zawierający główną funkcję sterującą programem, natomiast reszta składa się z dwóch plików. Jednym z nich jest plik źródłowy .c, a drugi to plik nagłówkowy .h. :

main - odpowiada za sterowanie całym programem i wywołuje potrzebne funkcje z odpowiednich modułów.

eight – odpowiada za analizę pliku wejściowego w formacie 8-bitowym oraz zapisuje taki format kompresji do pliku wyjściowego.

twelve – odpowiada za analizę pliku wejściowego w formacie 12-bitowym oraz zapisuje taki format kompresji do pliku wyjściowego.

sixteen – odpowiada za analizę pliku wejściowego w formacie 16-bitowym oraz zapisuje taki format kompresji do pliku wyjściowego.

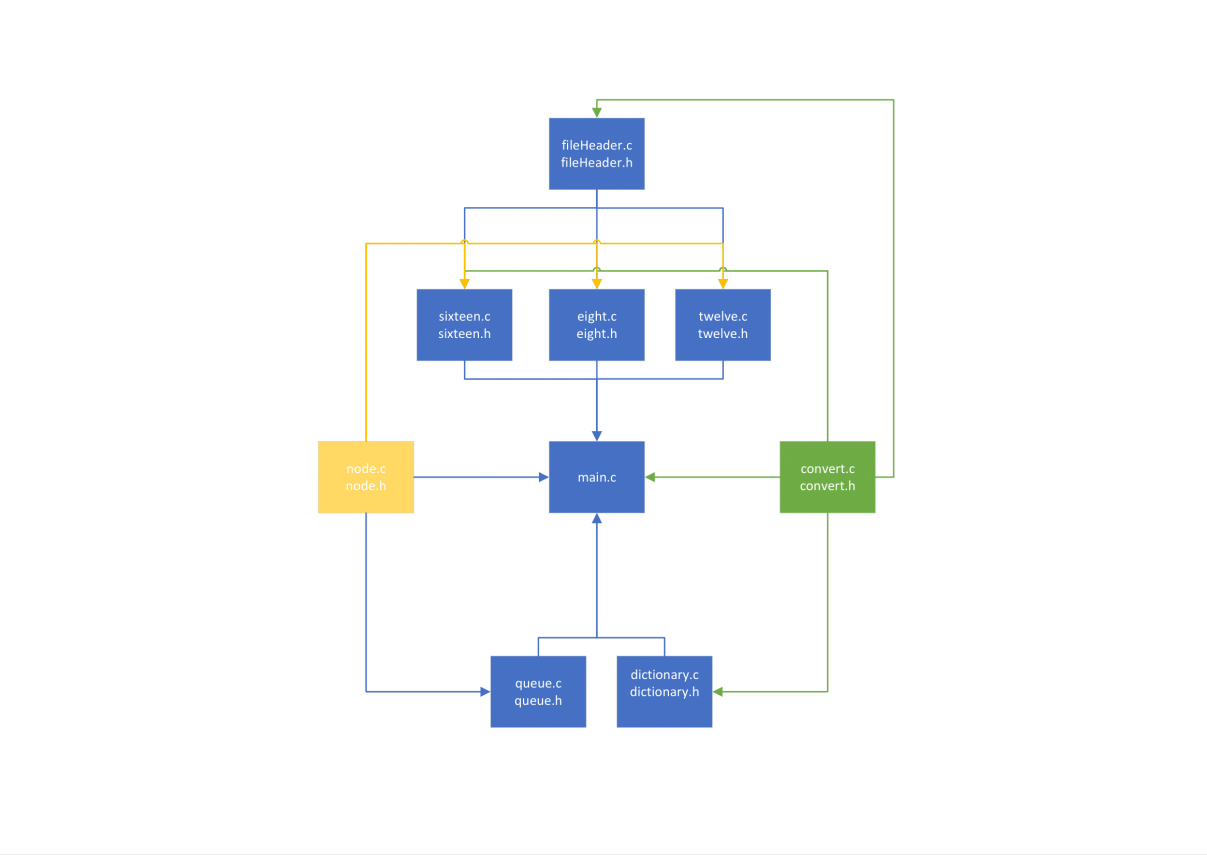
node – zawiera strukturę węzła drzewa oraz funkcję odpowiadającą za tworzenie węzłów.

queue – odpowiada za tworzenie drzewa.

convert – zawiera funkcje zamieniające liczby binarne na dziesiętne oraz dziesiętne na binarne.

dictionary – odpowiada za zapis słownika do pliku wyjściowego.

fileHeader – odpowiada za tworzenie nagłówków w pliku wyjściowym.



Rysunek 1. Diagram modułów

4. Wykorzystane Struktury

W programie użyliśmy struktury odpowiadającej węzłowi

```
typedef struct node{
   unsigned short *value;
   int freq;
   bool used;
   struct node *left;
   struct node *right;
}node;
```

5. Wykorzystane Funkcje

5.1 Moduł eight:

int eightAnalyzeInput(FILE* in, int* charcounter, int uniqueCounter) – funkcja zajmuje się zliczeniem wszystkich oraz różnych znaków(sekwencji 8-bitowych) w pliku wejściowym.

void eightOutputGenerator(FILE* in, int uniqueCounter, unsigned short** codes, FILE *out, char password, char *remainingChar, int remainingLen, unsigned char *crc) – funkcja zajmuje się zapisem zkompresowanego pliku wejściowego do pliku wyjściowego na podstawie słownika utworzonego w funkcji **dictionary** w formacie 8 bitowym.

5.2 Moduł twelve :

int twelveAnalyzeInput (FILE* in, int* charcounter, int uniqueCounter) – funkcja zajmuje się zliczeniem wszystkich oraz różnych sekwencji 12-bitowych w pliku wejściowym.

void twelveOutputGenerator(FILE* in, int uniqueCounter, unsigned short** codes, FILE *out, char password, char *remainingChar, int remainingLen, unsigned char *crc) – funkcja zajmuje się zapisem zkompresowanego pliku wejściowego do pliku wyjściowego na podstawie słownika utworzonego w funkcji **dictionary** w formacie 12 bitowym.

5.3 Moduł sixteen :

int sixteenAnalyzeInput (FILE* in, int* charcounter, int uniqueCounter) – funkcja zajmuje się zliczeniem wszystkich oraz różnych sekwencji 16-bitowych w pliku wejściowym.

void sixteenOutputGenerator(FILE* in, int uniqueCounter, unsigned short** codes, FILE *out, char password, char *remainingChar, int remainingLen, unsigned char *crc) – funkcja zajmuje się zapisem zkompresowanego pliku wejściowego do pliku wyjściowego na podstawie słownika utworzonego w funkcji **dictionary** w formacie 16 bitowym.

5.4 Moduł node:

node *makeNode(unsigned short *character, int frequency, bool used, node *left, node *right) – funkcja odpowiada za utworzenie węzła.

5.5 Moduł queue:

bool notFull(node **queue, int size) - funkcja sprawdza czy tablica nie jest pełna.

node *findMinNotUsedNode(node **queue, int queueSize) – funkcja znajduje węzły z najniższą częstotliwością bez przypisanego rodzica.

void addNewNodeToQueue(node **queue, int queueSize) – łączy dwa węzły znalezione przy pomocy funkcji **findMinNotUsedNode** oraz stworzone przy pomocy funkcji **makeNode.**

void readCodes(node *root, int size, unsigned short **codes, unsigned short *tmp, int level) – funkcja przypisuje kody poszczególnym znakom utworzone przy pomocy drzewa Huffmana(Tworzy słownik).

5.6 Moduł convert:

int binToDec(char *binary) – zmienia liczbę binarną na dziesiętną.

char *DectoBin(unsigned short decimal, int bitnumber) – zmienia liczbę dziesiętną na binarną.

5.7 Moduł dictionary:

int dictionary(unsigned short **codes, FILE *out, int uniqueCounter, int inputSize, char *bufor, unsigned char *crc, char password) – funkcja zapisuje słownik do pliku wyjściowego.

5.8 Moduł fileHeader:

void header(FILE *out, int inputSize, int zeroCounter, unsigned char crc) – Zapisuje odpowiedni nagłówek do pliku wyjściowego w zależności od wybranych formatów kompresji, trybów uruchamiania.

6. Wykorzystane Algorytmy

Kodowanie Huffmana

Kodowanie Huffmana– jedna z najprostszych i łatwych w implementacji metod kompresji bezstratnej. Algorytm polega na określeniu częstotliwości występowania dla każdego symbolu ze zbioru S. Następnie należy utworzyć listę drzew binarnych które w węzłach przechowują pary: symbol, częstotliwość występowania. Na początku drzewa składają się wyłącznie z korzenia.

Dopóki na liście jest więcej niż jedno drzewo, należy powtarzać następujące kroki:

- Usunąć z listy dwa drzewa o najmniejszej częstotliwości.
- Wstawić nowe drzewo , w którego korzeniu jest suma prawdopodobieństw usuniętych drzew, natomiast one same stają się jego lewym i prawym poddrzewem. Korzeń drzewa nie przechowuje symbolu.

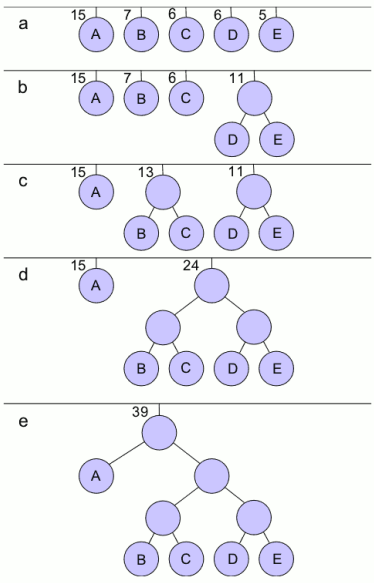
Drzewo, które pozostanie na liście, jest nazywane drzewem Huffmana.

Algorytm Huffmana jest algorytmem niedeterministycznym, ponieważ nie określa, w jakiej kolejności wybierać drzewa z listy, jeśli mają takie samo prawdopodobieństwo. Nie jest również określone, które z usuwanych drzew ma stać się lewym bądź prawym poddrzewem. Jednak bez względu na przyjęte rozwiązanie średnia długość kodu pozostaje taka sama.

Na podstawie drzewa Huffmana tworzone są słowa kodowe; algorytm jest następujący:

- Każdej lewej krawędzi drzewa przypisz 0, prawej 1 (można oczywiście odwrotnie).
- Przechodź w głąb drzewa od korzenia do każdego liścia (symbolu):
- Jeśli skręcasz w prawo, dopisz do kodu bit o wartości 1.
- Jeśli skręcasz w lewo, dopisz do kodu bit o wartości 0.

Długość słowa kodowego jest równa głębokości symbolu w drzewie, wartość binarna zależy od jego położenia w drzewie.



Rysunek 2. Konstruowanie drzewa Huffmana, źródło: https://en.wikipedia.org/wiki/Huffman_coding

7. Testowanie

Dostępne pliki testowe:

- easy.txt 11B - 5 różnych znaków
- file.txt 268B – wiele różnych znaków
- one-letter.txt 1B – 1 znak
- one-type.txt 8B – 1 znak
- alphabet.txt 1MB – wielkie litery alfabetu
- gen.txt 128KB – wszystkie możliwe kombinacje par znaków
- lotr.txt 3,15MB - treść “Lord of the Rings”
- pan-tadeusz.txt 482KB - treść “Pana Tadeusza”
- test-jpg.jpg 826KB – plik formatu .jpg
- test-pdf.pdf 4,22MB – plik formatu .pdf
- test-png.png 2,26MB – plik formatu .png

Program można testować w sposób półautomatyczny, dzięki formułom z pliku “Makefile”. By przetestować:

- Kompresję 8-bitową na małych plikach tekstowych – make test-8-small
- Kompresję 8-bitową na dużych plikach .txt, .pdf, .jpg, .png – make test-8-big
- Kompresję 8-bitową na wszystkich dostępnych testach – make test-8-all
- Kompresję 12-bitową na małych plikach tekstowych – make test-12-small
- Kompresję 12-bitową na dużych plikach .txt, .pdf, .jpg, .png – make test-12-big
- Kompresję 12-bitową na wszystkich dostępnych testach – make test-12-all
- Kompresję 16-bitową na małych plikach tekstowych – make test-16-small
- Kompresję 16-bitową na dużych plikach .txt, .pdf, .jpg, .png – make test-16-big
- Kompresję 16-bitową na wszystkich dostępnych testach – make test-16-all
- Kompresję 8-, 12- i 16-bitową na małych plikach tekstowych – make test-small
- Kompresję 8-, 12- i 16-bitową na dużych plikach .txt, .pdf, .jpg, .png – make test-big
- Kompresję 8-, 12- i 16-bitową na wszystkich dostępnych testach – make test-all
- Dostępne opcje (kompresja 8-, 12- i 16-bitowa przy szyfrowaniu hasłem “password”; tryb “verbose” podczas kompresji oraz dekompresji; wyświetlanie komunikatu “help”) - make test-features