Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Temat:
Kompresja plików metodą kodowania Huffmana

Autor	Piotr Hasiec
Prowadzący	Dr Inż. Bożena Wieczorek
Rok akademicki	2019/2020
Kierunek	Informatyka
Rodzaj studiów	SSI
Semestr	2
Sekcja	12
Termin oddania sprawozdania	2020-09-09

1. Treść zadania

Napisać program, dokonujący bezstratnej kompresji plików metodą kodowania kodem prefiksowym Huffmana.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-k jeśli chcemy skompresować plik,

-d jeśli chcemy rozpakować plik,

-i <NAZWA.ROZSZERZENIE> plik wejściowy

-o <NAZWA.ROZSZERZENIE> plik wyjściowy

2. Analiza zadania

2.1. Struktury danych

W programie wykorzystane są struktury drzewiaste. Pierwszą ze struktur jest drzewo będące wynikiem działania funkcji `node* add_to_tree(node* leafs);` będące drzewem kodów Huffmanna, drugą jest struktura kopca typu min zaimplementowana w celu użycia jej jako kolejki priorytetowej.

2.2. Algorytmy

Użyte w programie algorytmy to algorytm budowy drzewa kodów Huffmana będący algorytmem zachłannym oraz algorytmy słownikowe kopca typu min.

3. Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego i wyjściowego po odpowiednich przełącznikach (odpowiednio: -i, -o) oraz procedurę wykonywaną na tych plikach – kompresję lub rozpakowywanie (odpowiednio: -k, -d)

```
Program.exe –i dane_wejsciowe.txt –o wynik -k
Program.exe –o wyjście.txt –i dane - d
```

Pliki wejściowy jak i plik wyjściowy może posiadać dowolne rozszerzenie. Nalży jednak zwrócić uwagę żeby przy dekompresji pliku podać odpowiednie rozszerzenie.

Podanie niewłaściwych nazw pliku powoduje otrzymanie komunikatów:

```
Nie udało się uzyskać dostępu do co najmniej jednego pliku.
```

W każdym przypadku podania niewłaściwych danych, program kończy swoje działanie.

4. Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym.

4.1. Ogólna struktura programu

W przypadku funkcjonalności kompresji pliku program w pierwszej kolejności zlicza liczbę wystąpień każdego z 256 możliwych bajtów. Złożoność tej procedury wynosi $O(n)$. Następnie na podstawie tych danych budowana jest kolejka priorytetowa typu min, drzewo kodów Huffmana oraz słownik kodów. Złożoność czasowa tycch trzech algorytmów wynosi $O(1)$. Nie jest to prawda w ogólności ponieważ: złożoność procedury budowy kopca wynosi: $O(s)$ gdzie s jest liczbą elementów tablicy. Złożoność budowy drzewa Huffmana wynosi $O(s \lg s)$, a procedury `void codematrix(node* A, int** code, int i)` polegającej na przejściu całego drzewa - $O(s)$. Jednak we wszystkich przypadkach $s = 256$ zatem złożoności redukują się do czasu stałego. Ostatnią procedurą funkcjonalności kodowania jest przetłumaczenie nie zakodowanego pliku. Operacja ta wykonuje się w czasie liniowym.

W przypadku funkcjonalności dekompresji pliku program wczytuje tablicę częstości występowania bajtów zapisaną w pliku - $O(1)$, buduje drzewo kodów Huffmana - $O(1)$, wczytuje X bajtów, tłumaczy je na ciąg 0 i 1 zapisanych w przeznaczonej do tego tablicy i na jego podstawie dekoduje plik. Dekodowanie pliku odbywa się poprzez n-krotne przejście drzewa Huffmana którego średnia (ważona częstością występowania bitów) głębokość zależy od struktury pierwotnego pliku. Dolnym ograniczeniem wartości n jest liczba bajtów w pierwotnym (nie zakodowanym pliku). W rzeczywistości jest ona większa ponieważ czasem wczytany ciąg 0 i 1 będzie w sobie zawierał niepełne słowo kodowe. Takie zdarzenie jednak może nastąpić jedynie A/X razy (A – liczba bajtów w zakodowanym pliku, X – liczba wczytywanych za jednym razem bajtów). Zatem złożoność procedury dekodowania pliku jest linowa.

4.2. Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji znajduje się w załączniku na końcu dokumentu.

5. Testowanie

Program został przetestowany na różnych typach plików – graficznych, dźwiękowych, wykonawczych, tekstowych. W trakcie testowania (po kompresji i rozpakowaniu) nie zaobserwowano zmiany, ubytku żadnego bajta w pliku.

Największy plik dla jakiego przetestowano program ma 3,01 GB, kompresja pliku trwałą ok 16 m, rozpakowywanie 40m 46s. Program wykazuje liniową złożoność czasową względem wielkości pliku. Plik o rozmiarze 1,5 GB rozpakowywał się około 20m, a o rozmiarze 0,3 GB 4m.

Maksymalny rozmiar pliku zależy od zależy od kompilatora (typ int może być realizowany jako zmienna dwu- lub czterobajtowa).

6. Wnioski

Program kompresji plików metodą kodowania Huffmana jest programem operującym na plikach binarnych. Pisanie go pozwoliło mi lepiej zrozumieć sposób strukturę plików. Nie jest ona skomplikowana jednak po raz pierwszy miałem większy projekt związany właśnie z ich obsługą i to właśnie to było najtrudniejsze w projekcie.

Źródła

Ron Rivest i Thomas H. Cormen „Wprowadzenie do algorytmów”

Huffman

4

Wygenerowano przez Doxygen 1.8.17

Rozdział 1

Indeks struktur danych

1.1 Struktury danych

Tutaj znajdują się struktury danych wraz z ich krótkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

Rozdział 3

Dokumentacja struktur danych

3.1 Dokumentacja struktury min_heap

< Struktura kopca (statycznego)/ Kolejka priorytetowa typu min

`#include <sources.h>`

Pola danych

- **node** ∗ **tab** [256]
  tablica przechowująca wierzchołki drzewa
- int **lengh**
  długość aktualnej kolejki

3.1.1 Opis szczegółowy

< Struktura kopca (statycznego)/ Kolejka priorytetowa typu min

Dokumentacja dla tej struktury została wygenerowana z pliku:

- **sources.h**

3.2 Dokumentacja struktury node

< struktura węzła drzewa

`#include <sources.h>`

Pola danych

- unsigned int **frequency**

  częstość występowania bitu
- int **bajt**

  bit którego dotyczy powyższa częstość lub -1 jeśli w węźle nie znajduje się żaden
- struct **node** ∗ **left_n**

  wskaźnik na lewego potomka
- struct **node** ∗ **right_n**

  wskaźnik na prawego potomka

3.2.1 Opis szczegółowy

< struktura węzła drzewa

Dokumentacja dla tej struktury została wygenerowana z pliku:

- **sources.h**

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku Main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "sources.h"
```

Funkcje

- int **main** (int argc, char ∗argv[ ])

4.2 Dokumentacja pliku sources.c

```
#include <stdlib.h>
#include <stdio.h>
#include "sources.h"
```

Funkcje

- void **delete_tree** (**node** ∗root)
- void **min_heapify** (**min_heap** ∗A, int i)
- void **heap_insert** (**min_heap** ∗A, **node** ∗key)
- **min_heap build_min_heap** (**node** ∗A, int n)
- **min_heap build_min_heap_i** (**node** ∗A, int n)
- **node** ∗ **extract_min** (**min_heap** ∗A)
- **node** ∗ **add_to_tree** (**node** ∗leafs)
- **node** ∗ **read_file_uncoded** (char ∗nazwa, **node** ∗tab)
- void **codematrix** (**node** ∗A, int ∗∗code, int i)

  zwraca tablice(słownik) kodów huffmana na podstawie drzewa huffmna
- int **save_to_file** (char ∗iname, char ∗oname, int ∗∗codevector, **node** ∗leafs_huffman_tree)
- void **number_to_string** (char number, char string_n[1280], int ∗w)
- int **find_eqivalent** (**node** ∗root, char code[1280], int ∗w)
- int **read_file_coded** (char ∗iname, char ∗oname)
- int **compress** (char ∗iname, char ∗oname)
- void **readme** ()

4.2.1 Dokumentacja funkcji

4.2.1.1 add_to_tree()

```
node* add_to_tree (
            node * leafs )
```

Buduje drzewo kodów Huffmna

Parametry

in,out	*leafs*	Tablica 256 liści drzewa

Zwraca

Wskaźnik na korzeń drzewa

4.2.1.2 build_min_heap()

```
min_heap build_min_heap (
             node * A,
            int n )
```

Buduje kopiec typu min z podanej elementowej tablicy

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*n*	ilość elementów tablicy

Zwraca

zwraca kolejkę typu min

4.2.1.3 build_min_heap_i()

```
min_heap build_min_heap_i (
             node * A,
            int n )
```

Buduje kopiec typu min z podanej elementowej tablicy

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*n*	ilość elementów tablicy

Zwraca

zwraca kolejkę typu min∗ zwraca kolejkę typu min

4.2.1.4 codematrix()

```
void codematrix (
              node * A,
             int ** code,
             int i )
```

zwraca tablice(słownik) kodów huffmana na podstawie drzewa huffmna

Funkcja tworzy słownik w którym dla i-tego wiersza występuje kod bajtu o wartości i, kod jest ciągiem 0 i 1 zakończonym liczbą 2

Parametry

in	*A*	Wskaźnik na korzeń drzewa modów Huffmana
in,out	*code*	Tablica 257x256 w której zwracany będzie słownik
in	*i*	Zmienna pomocnicza, domyślnie 0

4.2.1.5 compress()

```
int compress (
             char * iname,
             char * oname )
```

Funkcja tworzy obsługująca funkcjonalność kompresji pliku

Parametry

in	*iname*	Nazwa pliku wejściowego (niezakodowanego)
in	*oname*	Nazwa pliku wyjściowego

4.2.1.6 delete_tree()

```
void delete_tree (
              node * root )
```

Funkcja usuwa drzewo bez liści

Parametry

in,out	*root*	Wskaźnik na korzeń drzewa

4.2.1.7 extract_min()

```
node* extract_min (
              min_heap * A )
```

Zciąga z kopca element o najmniejszej częstości występowania

Parametry

in,out	*A*	Wskaźnik na kolejkę

Zwraca

zwraca element o najmniejszej częstości występowania

4.2.1.8 heap_insert()

```
void heap_insert (
              min_heap * A,
              node * key )
```

Funkcja wstawia klucz do kopca

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*key*	Klucz który należy wstawić do kolejki

4.2.1.9 min_heapify()

```
void min_heapify (
```

```
 min_heap * A,
int i )
```

Funkcja przywraca własności kopca typu min

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*i*	element tablicy dla którego wywołujemy funkcję

4.2.1.10 read_file_coded()

```
int read_file_coded (
            char * iname,
            char * oname )
```

Funkcja tworzy obsługująca funkcjonalność dekompresji pliku

Parametry

in	*iname*	Nazwa pliku wejściowego (niezakodowanego)
in	*oname*	Nazwa pliku wyjściowego

Zwraca

0 w przypadku poprawnego wykonania funkcji

4.2.1.11 read_file_uncoded()

```
node* read_file_uncoded (
            char * nazwa,
             node * tab )
```

Funkcja uzupełnia dane w tablicy liści dotyczące częstości występowania bajtów na podsawie pliku

Parametry

in,out	*tab*	Tablica 256 liści drzewa
in	*nazwa*	Nazwa pliku dla którego należy zrobić statystykę

Zwraca

Wskaźnik na przyjętą tablicę

4.2.1.12 readme()

```
void readme ( )
```

Funkcja tworzy plik readme

4.2.1.13 save_to_file()

```
int save_to_file (
            char * iname,
            char * oname,
            int ** codevector,
             node * leafs_huffman_tree )
```

Funkcja tworzy zakodowany plik razem z danymi potrzebnymi do jego odczytu

Parametry

in	*iname*	Nazwa pliku wejściowego (niezakodowanego)
in	*oname*	Nazwa pliku wyjściowego
in	*codevector*	słownik kodów Huffmana
in	*leafs_huffman_tree*	Tablica liści drzewa huffmana

Zwraca

0 w przypadku poprawnego wykonania funkcji

4.3 Dokumentacja pliku sources.h

Struktury danych

- struct **node**

  < *struktura węzła drzewa*
- struct **min_heap**

  < *Struktura kopca (statycznego)/ Kolejka priorytetowa typu min*

Definicje typów

- typedef struct **node** **node**

  < *struktura węzła drzewa*
- typedef struct **min_heap** **min_heap**

  < *Struktura kopca (statycznego)/ Kolejka priorytetowa typu min*

Funkcje

- void **delete_tree** (**node** ∗root)
- void **min_heapify** (**min_heap** ∗A, int i)
- void **heap_insert** (**min_heap** ∗A, **node** ∗key)
- **min_heap build_min_heap** (**node** ∗A, int n)
- **min_heap build_min_heap_i** (**node** ∗A, int n)
- **node** ∗ **extract_min** (**min_heap** ∗A)
- **node** ∗ **add_to_tree** (**node** ∗leafs)
- **node** ∗ **read_file_uncoded** (char ∗nazwa, **node** ∗tab)
- void **codematrix** (**node** ∗A, int ∗∗code, int i)

  zwraca tablice(słownik) kodów huffmana na podstawie drzewa huffmna
- void **number_to_string** (char number, char ∗string_n, int ∗w)
- void **readme** ()
- int **compress** (char ∗iname, char ∗oname)
- int **save_to_file** (char ∗iname, char ∗oname, int ∗∗codevector, **node** ∗leafs_huffman_tree)
- int **read_file_coded** (char ∗iname, char ∗oname)

4.3.1 Dokumentacja funkcji

4.3.1.1 add_to_tree()

```
node* add_to_tree (
            node * leafs )
```

Buduje drzewo kodów Huffmna

Parametry

in,out	*leafs*	Tablica 256 liści drzewa

Zwraca

Wskaźnik na korzeń drzewa

4.3.1.2 build_min_heap()

```
min_heap build_min_heap (
             node * A,
            int n )
```

Buduje kopiec typu min z podanej elementowej tablicy

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*n*	ilość elementów tablicy

Zwraca

zwraca kolejkę typu min

4.3.1.3 build_min_heap_i()

```
min_heap build_min_heap_i (
              node * A,
             int n )
```

Buduje kopiec typu min z podanej elementowej tablicy

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*n*	ilość elementów tablicy

Zwraca

zwraca kolejkę typu min∗ zwraca kolejkę typu min

4.3.1.4 codematrix()

```
void codematrix (
              node * A,
             int ** code,
             int i )
```

zwraca tablice(słownik) kodów huffmana na podstawie drzewa huffmna

Funkcja tworzy słownik w którym dla i-tego wiersza występuje kod bajtu o wartości i, kod jest ciągiem 0 i 1 zakończonym liczbą 2

Parametry

in	*A*	Wskaźnik na korzeń drzewa modów Huffmana
in,out	*code*	Tablica 257x256 w której zwracany będzie słownik
in	*i*	Zmienna pomocnicza, domyślnie 0

4.3.1.5 compress()

```
int compress (
            char * iname,
            char * oname )
```

Funkcja tworzy obsługująca funkcjonalność kompresji pliku

Parametry

in	*iname*	Nazwa pliku wejściowego (niezakodowanego)
in	*oname*	Nazwa pliku wyjściowego

4.3.1.6 delete_tree()

```
void delete_tree (
             node * root )
```

Funkcja usuwa drzewo bez liści

Parametry

in,out	*root*	Wskaźnik na korzeń drzewa

4.3.1.7 extract_min()

```
node* extract_min (
             min_heap * A )
```

Zciąga z kopca element o najmniejszej częstości występowania

Parametry

in,out	*A*	Wskaźnik na kolejkę

Zwraca

zwraca element o najmniejszej częstości występowania

4.3.1.8 heap_insert()

```
void heap_insert (
              min_heap * A,
              node * key )
```

Funkcja wstawia klucz do kopca

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*key*	Klucz który należy wstawić do kolejki

4.3.1.9 min_heapify()

```
void min_heapify (
              min_heap * A,
             int i )
```

Funkcja przywraca własności kopca typu min

Parametry

in,out	*A*	Wskaźnik na kolejkę
in	*i*	element tablicy dla którego wywołujemy funkcję

4.3.1.10 number_to_string()

```
void number_to_string (
             char number,
             char * string_n,
             int * w )
```

Pomocnicza funkcja zamiany liczb na ciąg znaków 0 i 1 zakończony 2 jako znakiem końca ciągu

Parametry

in	*number*	Liczba którą należy zmienić
in,out	*string_n*	Zwracany ciąg
in,out	*w*	Pierwszy niezapisany element ciągu

4.3.1.11 read_file_coded()

```
int read_file_coded (
            char * iname,
            char * oname )
```

Funkcja tworzy obsługująca funkcjonalność dekompresji pliku

Parametry

in	*iname*	Nazwa pliku wejściowego (niezakodowanego)
in	*oname*	Nazwa pliku wyjściowego

Zwraca

0 w przypadku poprawnego wykonania funkcji

4.3.1.12 read_file_uncoded()

```
node* read_file_uncoded (
            char * nazwa,
             node * tab )
```

Funkcja uzupełnia dane w tablicy liści dotyczące częstości występowania bajtów na podsawie pliku

Parametry

in,out	*tab*	Tablica 256 liści drzewa
in	*nazwa*	Nazwa pliku dla którego należy zrobić statystykę

Zwraca

Wskaźnik na przyjętą tablicę

4.3.1.13 readme()

```
void readme ( )
```

Funkcja tworzy plik readme

4.3.1.14 save_to_file()

```
int save_to_file (
             char * iname,
             char * oname,
             int ** codevector,
              node * leafs_huffman_tree )
```

Funkcja tworzy zakodowany plik razem z danymi potrzebnymi do jego odczytu

Parametry

in	*iname*	Nazwa pliku wejściowego (niezakodowanego)
in	*oname*	Nazwa pliku wyjściowego
in	*codevector*	słownik kodów Huffmana
in	*leafs_huffman_tree*	Tablica liści drzewa huffmana

Zwraca

0 w przypadku poprawnego wykonania funkcji

Indeks