Politechnika Poznańska

Wydział Elektryczny
Informatyka

Teoria Informacji i Kodowanie

Dokumentacja Projektu

Autorzy:
Michał Majka
Nr albumu: 112679
Piotr Parysek
Nr albumu: 106100

Prowadzący:
dr inż. Ewa Idzikowska

17 listopada 2015

1 Wstęp

Zadaniem projektowym była implementacja algorytmu kompresji bezstratnej ***Run-Length Encoding*** (**RLE**).

Zadanie zrealizowaliśmy w środowisku programistycznym Qt Creator 5.5.1[4] korzystając z kompilatora GCC 4.9.1[5].
Do kontroli wersji oraz plików źródłowych wykorzystaliśmy oprogramowanie Git[6], projekt hostowaliśmy w repozytorium GitHub[7].
Dokumentacja została wykonana w LaTeX[1] w programie Texmaker 4.5 [2] oraz edytora online: ShareLaTeX[3].

2 Algorytm

2.1 Historia

Run-Length Encodings, również znane jako **Golomb Codings**, swoje „podwaliny" powstania wiąże z pracami XVII-wiecznego francuskiego matematyka *Blaise'a Pascal'a* związanych z probabilistyką[9]. Koncepcja kodowania powtarzających się znaków była używana od początków istnienia teorii informacji (Shannon 1949, Laemmel 1951), jednakże metodę oraz sposób kodowania wynalazł i opracował *Solomon Wolf Golomb*[8].

2.2 Zasada działania

Algorytm jest relatywnie prosty → przedstawia powtarzające się wartości jako dany znak i licznik powtórzeń. Na przykład ciąg znaków:

ppppppppuuuuutttttt.....ppppooooooozzzzznnnnnannnnn....pppppplllll

Zostaje przedstawiony w ciąg postaci:

$p7u5t5.6p4o6z5n4an5.4p6l5$

Ciąg znaków	Liczba
ppppppppuuuuutttttt.....ppppooooooozzzzznnnnnannnnn....pppppplllll	64
$p7u5t5.6p4o6z5n4an5.4p6l5$	26
	$^{26}/_{64} \approx 0.41$

Tabela 1: Przykładowa kompresja znakowa

3 Opis implementacji

Program wykonano w języku C++ z metodami dostarczonymi z Qt.

3.1 Kodowanie

Ustalenie znaku kodowania

Rysunek 1: Schemat blokowy wyszukiwania „znaku”.

Do wyszukania „znaków” wykorzystano kontener QMap<quint, int>, gdzie zmienna quint8 (unsigned byte) wskazuje na poszczególne możliwe wartości bajta, a zmienna int wskazuje ilość wystąpień

danego bajta w opracowywanym pliku.

Kod 1: Deklaracja wraz z inicjalizacją kontenera QMap¡quint8, int¿

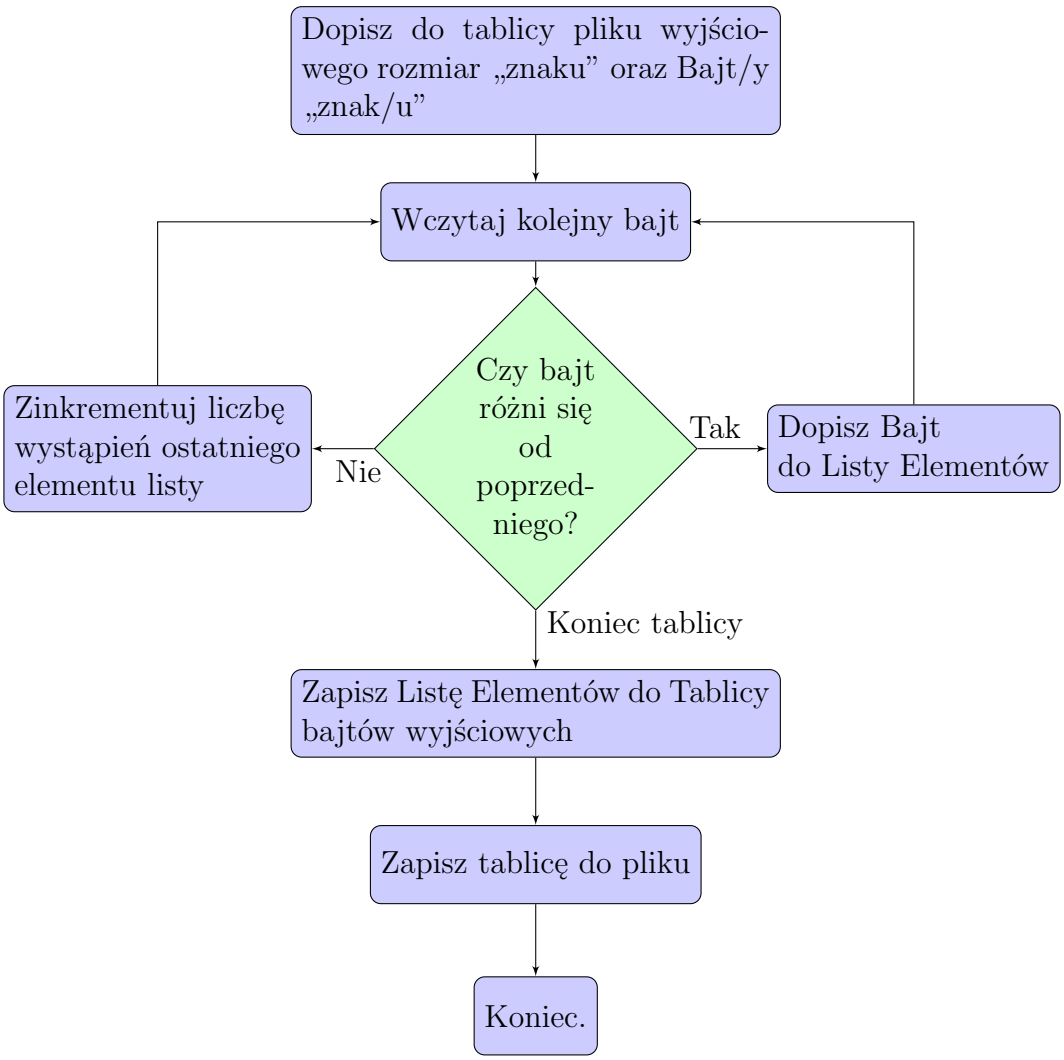
```
1 QMap<quint8, int> IIMap;
2 for (quint8 i = 0; i < 255; i++) {
3     IIMap.insert(i, 0);
4 }
```

W przypadku zdarzenia, że w pliku występują wszystkie możliwe kombinacje bajta, zamiast jedno bajtowego „znaku" zostaje wprowadzony znak dwu bajtowy:

Kod 2: Deklaracja dwu bajtowej zmiennej znakowej

```
1 QPair<quint8, quint8> SIGN16;
```

Kodowanie



Rysunek 2: Schemat blokowy kodowania pliku.

Do sprawnego wczytania, zliczenia i zakodowania poszczególnych bajtów stworzono kontener QList struktury Element. Struktura Element posiada dwa pola: quint8 item → oznaczające dany bajt oraz quint32 value → oznaczające ilość wystąpień (Założono, że dany bajt nie powtórzy się więcej jak 4294967295 razy).

Kod 3: Główne struktury danych kodowania

```
1 struct Element {
2     quint8 item;
3     quint32 value;
4 };
5 QList<Element> Elements;
6 quint8 CurrentByte;
```

Zapisanie danych do pliku odbywa się z pośrednictwem tablicy bajtów QByteArray. Analiza zapisanych znaków odbywa się poprzez przejście przez wcześniej wspomnianą listę: QList< Element> i odpowiednią interpretację wartości wystąpień danego znaku.

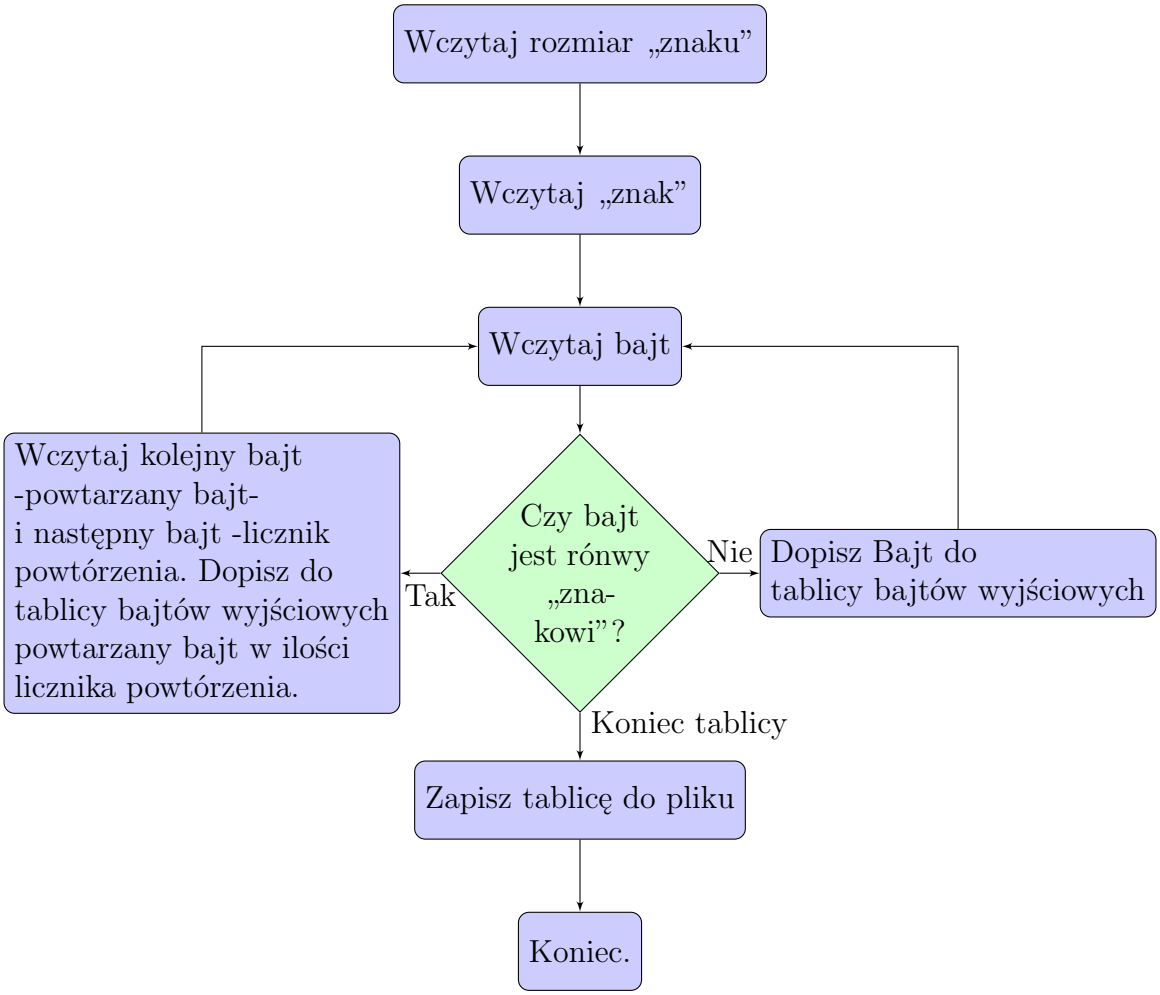
Jeżeli wartość powtórzenia danego znaku nie jest większa niż minimalna długość jego zastąpienia, który ma postać „znak" inicjujący, bajt powtarzany, ilość powtórzeń, to wpisywana jest „pierwotna postać".

W przypadku, gdy wartość ilości powtórzeń bajtu jest większa niż maksymalna wartość jaką może osiągnąć bajt - 255 - to postać zastąpienia przybiera postać: czterech bajtów „znaku", bajt powtarzany i cztery bajty licznika.

Kod 4: Zapisanie i zakodowania danych skompresowanych do pliku

```
1  if (e.value < 256) {
2      OutByteArray.append(SIGN);
3      OutByteArray.append(e.item);
4      OutByteArray.append(e.value);
5  } else {
6      OutByteArray.append(SIGN);
7      OutByteArray.append(SIGN);
8      OutByteArray.append(SIGN);
9      OutByteArray.append(SIGN);
10     OutByteArray.append(e.item);
11     QByteArray TempArray;
12     TempArray = RLE::IntToHex(e.value);
13     OutByteArray.append(TempArray);
14 }
```

3.2 Dekodowanie



Rysunek 3: Schemat blokowy dekodowania pliku.

Dekodowanie odbywa się z pomocą analogicznych struktur / zasad / funkcji co zostały użyte podczas procesu kodowania.

4 Użytkowanie programu

5 Testy

6 Porównanie kompresji

Literatura

[1] Kurs LaTeXw π^e minut `http://www.fuw.edu.pl/~kostecki/kurs_latexa.pdf`.

[2] Program Texmaker 4.5 `http://www.xm1math.net/texmaker/`.

[3] ShareLaTeX is an online LaTeX editor `https://www.sharelatex.com/`.

[4] Qt `http://www.qt.io/`.

[5] GCC, the GNU Compiler Collection `https://gcc.gnu.org/`.

[6] Git `http://git-scm.com/`.

[7] GitHub `https://github.com/`.

[8] Run-length encodings - S. W. Golomb (1966); IEEE Trans Info Theory 12(3):399 http://urchin.earth.li/~twic/Golombs_Original_Paper/.

[9] Variable-length codes for data compression / David Salomon, London : Springer, 2007.

[10] Ikony http://www.flaticon.com/.

Dokument wykonany w LaTeX[1]
w programie Texmaker 4.5 [2]