

Politechnika Śląska w Gliwicach

Wydział Automatyki, Elektroniki i Informatyki



Laboratorium Programowania Komputerów

LZ77

autor:	Jakub Łagódka
użytkownik laboratoryjny:	Jakub-Lagodka-gr22
prowadzący:	dr. inż. Adam Gudyś
rok akademicki:	2016/2017
kierunek:	informatyka
rodzaj studiów:	SSI
semestr:	2
termin laboratorium / ćwiczeń	czwartek 10:00-11:30
grupa:	2
sekcja:	2
termin oddania sprawozdania:	2017-09-31
data oddania sprawozdania:	2017-07-06
ścieżka:	C:\Users\KubineX\Desktop\PK\Jakub-Lagodka-gr22-
repo\Projekt\LZ77\LZ77	

1. Treść zadania

Napisać program kompresujący i dekompresujący dane wykorzystując algorytm LZ77.

2. Analiza, projektowanie

2.1. Algorytmy, struktury danych, ograniczenia specyfikacji

Algorytm LZ77 jest metodą strumieniowej słownikowej kompresji danych. Metoda LZ77 wykorzystuje fakt, że w danych powtarzają się ciągi bajtów (np. w tekstach naturalnych będą to słowa, frazy lub całe zdania) – kompresja polega na zastępowaniu powtórzonych ciągów o wiele krótszymi liczbami wskazującymi, kiedy wcześniej wystąpił ciąg i z ilu bajtów się składał; z punktu widzenia człowieka jest to informacja postaci "taki sam ciąg o długości 15 znaków wystąpił 213 znaków wcześniej". Metoda LZ77 korzysta z bufora (okna), który logicznie podzielony jest na dwie części:

- **bufor słownikowy** (słownik), przechowujący określoną ilość ostatnio przetwarzanych symboli (sufiks);
- **bufor wejściowy** (lub bufor kodowania), przechowujący określoną ilość symboli do zakodowania.

Stopień kompresji LZ77 w dużej mierze zależy od długości słownika oraz długości bufora wejściowego (bufora kodowania). Dekompresja danych w metodzie LZ77 jest o wiele prostsza i szybsza niż kompresja. Do zdekodowania wymagane jest istnienie bufora (okna) o dokładnie takich samych parametrach jak przy kodowaniu – bufor podzielony jest na część słownikową obejmującą k pierwszych pozycji i bufor wyjściowy zajmujący n kolejnych pozycji.

Do napisania programu wykorzystano tablice znaków typu char o dynamicznie przydzielonej pamięci, do której wczytano zawartość pliku wejściowego. Taka struktura umożliwia zaalokowanie dowolnej ilości pamięci. Jednak wymaga dużej ostrożności w użyciu, ponieważ mogą wystąpić wycieki pamięci. Natomiast po użyciu tablicy należy dealokować pamięć.

Dla uproszenia wprowadzania do programu rozmiaru słownika i bufora wejściowego w bajtach, podaje się te wartości - parametry jako wykładniki potęgi o podstawie 2. Teoretycznie można podać dowolną wielkość słownika i bufora wejściowego, jednak przy rozmiarze słownika większym niż 2^{15} B kompresja większej ilości danych zajmuje programowi bardzo dużo czasu, a poza tym jeżeli rozmiar ten przekroczy rozmiar kompresowanych danych, to dalsze zwiększanie go nie przyniesie żadnego efektu.

3. Specyfikacja zewnętrzna

3.1 Obsługa programu

Program jest uruchamiany wraz z odpowiednimi parametrami z linii poleceń. Parametry mogą być podane w dowolnej kolejności. Parametry niezbędne do wywołania programu:

-i nazwa pliku wejściowego

-o nazwa pliku wyjściowego

-t tryb działania: kompresja/dekompresja

-s rozmiar słownika (w B, podawany jest wykładnik potęgi liczby 2)

-b rozmiar bufora wyjściowego (w B, podawany jest wykładnik potęgi liczby 2)

Poniżej został załączony zrzut ekranu z przykładowym wywołaniem programu:

Command	\$(TargetPath)
Command Arguments	-i wyjscie.txt -o wyjscie3.txt -t dekompresja -s 2 -b 2
Working Directory	\$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

W celu wyświetlenia pomocy, należy podać jedynie nazwę pliku bez żadnych parametrów lub jedynie z parametrem -h.

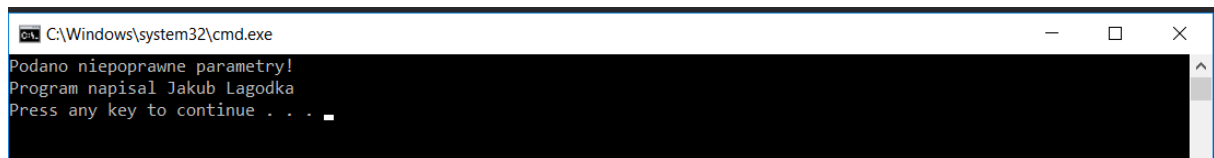
3.2 Format danych wejściowych

Program należy wywoływać z podaniem wszystkich wymienionych parametrów (i żadnych innych). Ich liczba jest sprawdzana przy uruchamianiu programu. W przypadku pliku wejściowego program sprawdza czy udało się znaleźć i otworzyć plik o podanej nazwie. Jeśli nie program wyświetli odpowiedni komunikat i zatrzyma działanie. Podobnie dla pliku wyjściowego program sprawdza czy udało go się stworzyć i zapisać w nim dane. Jeśli chodzi o tryb pracy, możliwe są jedynie 2 tryby – kompresja i dekompresja. Jeżeli użytkownik poda w tym miejscu cokolwiek innego, program wyświetli odpowiedni komunikat i nastąpi zatrzymanie jego pracy. Rozmiary słownika i bufora wejściowego powinny być podane jako dodatnie liczby naturalne.

3.3 komunikaty


W przypadku prawidłowego działania programu na ekranie nie powinien wyświetlić się żaden komunikat (poza podpisem autora).

1. Jeżeli zostanie podana niewłaściwa liczba parametrów lub zostaną niewłaściwie podane przełączniki, program wyświetli następujący komunikat:



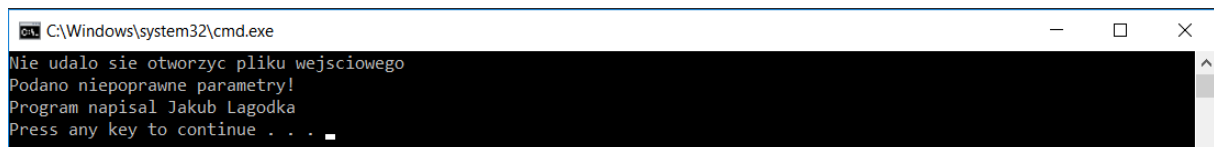
```
C:\Windows\system32\cmd.exe
Podano niepoprawne parametry!
Program napisal Jakub Lagodka
Press any key to continue . . .
```

2. Jeżeli zostanie podana niewłaściwa nazwa trybu pracy programu (inna niż kompresja lub dekompresja), program wyświetli następujący komunikat:



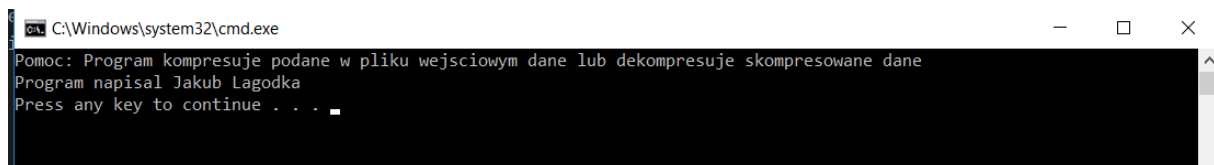
```
C:\Windows\system32\cmd.exe
podano zly tryb przetwarzania
Program napisal Jakub Lagodka
Press any key to continue . . .
```

3. Jeżeli nie udało się otworzyć pliku wejściowego lub nie znaleziono go, program wyświetli następujący komunikat:



```
C:\Windows\system32\cmd.exe
Nie udalo sie otworzyc pliku wejsciowego
Podano niepoprawne parametry!
Program napisal Jakub Lagodka
Press any key to continue . . .
```

4. Natomiast jeżeli wybrano pomoc (nie podano żadnego parametru lub -h), to program wyświetli następujący komunikat:



```
C:\Windows\system32\cmd.exe
Pomoc: Program kompresuje podane w pliku wejsciowym dane lub dekompresuje skompresowane dane
Program napisal Jakub Lagodka
Press any key to continue . . .
```

4. Specyfikacja wewnętrzna

W głównej funkcji programu `main()`, najpierw program sprawdza poprawność parametrów przekazanych do programu. Jeżeli zostały prawidłowo podane i odczytane oraz udało się odczytać plik wejściowy i zapisać plik wyjściowy, to następuje odczytanie rozmiaru pliku wejściowego, alokacja pamięci dynamicznej dla łańcucha znaków, skopiowanie zawartości pliku do tego łańcucha znaków i wywołanie odpowiedniej funkcji dla wybranego przez użytkownika trybu pracy. Powyższe część kodu została zaimplementowana w następujący sposób:

```
fseek(in, 0, SEEK_END);           //przesuwam wskaźnik na koniec pliku
                                rozmiar = ftell(in);           //pobieram rozmiar pliku
```

```

na początek                fseek(in, 0, SEEK_SET);                //powracam ze wskaźnikiem

                                bufor = malloc(sizeof(char)*(2 * rozmiar + 1));
                                //alokuje pamięć
                                bufor[0] = 0;
                                fread(bufor, sizeof(char), rozmiar, in); //kopiuje zawar-
tość pliku to zmiennej bufor
                                bufor[rozmiar] = 0; //na koniec dopisuje znak końca

                                if (!strcmp(tryb, "kompresja"))
                                    Kompresja(bufor, rozmiar, out, slownik_char, bu-
for_char);

                                else if (!strcmp(tryb, "dekompresja"))
                                    Dekompresja(bufor, rozmiar, out, slownik_char, bu-
for_char);

                                else
                                    printf("podano zly tryb przetwarzania\n");

                                free(bufor);
                                fclose(in);
                                fclose(out);
                                }
                                else
                                    printf("Podano niepoprawne parametry!\n");

```

Funkcja kompresja została zaimplementowana w następujący sposób:

```

//ta funkcja kompresuje wczytaną zawartość danych wejściowych przekazanych do funkcji
za pomocą parametrów *bufor i rozmiar
//parametry funkcji:
//char *bufor - łańcuch znaków zawierający zawartość pliku wejściowego
//int rozmiar - rozmiar bufora czyli zawartości pliku wejściowego
//FILE *out - wskaźnik na strukture danych, która umożliwia operowanie na pliku wyj-
ściowym (w tym przypadku potrzebna jest
//możliwość zapisu do pliku)
//char *slownik_char - łańcuch znaków zawierający rozmiar słownika
//char *bufor_char - łańcuch znaków zawierający rozmiar bufora wejściowego (kodowania)
//funkcja nie zwraca żadnej wartości
void Kompresja(char *bufor, int rozmiar, FILE *out, char *slownik_char, char
*bufor_char);
{
    char *okno, znak_bufora, znak_slownika, *pomoc;
    unsigned long long licznik = 0, j, pozycja, max_dlugosc = 0, przesuniecie = 0,
p;
    int rozmiar_slownika, rozmiar_bufora, ile = 100;                //definiuje po-
trzebne zmienne
    short czy_znak = 0;

    rozmiar_slownika = pow(2, atoi(slownik_char));                //konwertuje pobrany jako
parametr wejściowy rozmiar słownika
    rozmiar_bufora = pow(2, atoi(bufor_char));                //konwertuje pobra-
ny jako parametr wejściowy rozmiar bufora

    if (rozmiar_slownika > rozmiar)                //jeżeli rozmiar słownika jest większy
niż rozmiar pliku, to zmniejszam go do rozmiaru
        rozmiar_slownika = rozmiar;                //pliku

```

```

        if (rozmiar_bufora > rozmiar)           //jeżeli rozmiar bufora jest większy
niż rozmiar pliku, to zmniejszam go do rozmiaru
            rozmiar_bufora = rozmiar;           //pliku

    pomoc = malloc(sizeof(char)*(rozmiar_bufora + rozmiar_slownika + 1));
    //alokuje pamięć
    okno = malloc(sizeof(char)*(rozmiar_bufora + rozmiar_slownika + ile*rozmiar +
1));
    //alokuje pamięć

    okno[0] = 0;                                //zeruje łańcuch znaków

    for (j = 0; j < rozmiar_slownika+2; j++)
        okno[j] = dane[0];                      //wypełniam okno pierwszym znakiem

    okno[j] = 0; //zeruje łańcuch znaków

    strncat(okno, dane, rozmiar_bufora);         //następnie doklejam do słownika
odpowiednią ilość znaków z pliku

    fwrite(dane, sizeof(char), 1, out);          //wyprowadzam do pliku pierwszy
znak

    while (przesuniecie < rozmiar-1) // pracuję w pętli, której ilość wywołań jest
zależna od dł_slownika i rozmiaru
    {
        max_dlugosc = 0;    //zeruje zmienne pomocnicze
        pozycja = 0;

        znak_bufora = okno[przesuniecie + rozmiar_slownika]; //zapisuje znak
bufora

        for (j = przesuniecie; j <przesuniecie + rozmiar_slownika + 2; j++) //w
kolejnej pętli sprawdzam znak po znaku czy znaki
        {
            //się powtarzają
            znak_slownika = okno[j]; //kopiuje ze słownika i bufora znaki do
zmiennych pomocniczych by łatwiej
//na nich operować
            if (licznik != 0)           //Jeżeli pierwszy znak się po-
krywa, to sprawdzam ile kolejnych znaków
            {                           //się pokrywa
                znak_bufora = okno[licznik + przesuniecie + roz-
miar_slownika];

                if (znak_bufora == znak_slownika && licznik < roz-
miar_slownika - 1 && j < przesuniecie + rozmiar_slownika + 1
                    && j < rozmiar - 1)
                    licznik++; //jeżeli znaki są równe i nie przekro-
czono rozmiaru słownika

            else //jeśli znaki są różne, to zapisuje pozycje i
długość tych, co się pokrywały
            {
                if (licznik > max_dlugosc) //w sytuacji gdy ciąg był
większy niż poprzednie w słowniku
                {
                    max_dlugosc = licznik; //to zapamiętuje
                    pozycja = j - licznik - przesuniecie;
                }
                licznik = 0; //zeruje licznik
                znak_bufora = okno[przesuniecie + rozmiar_slownika];

```

```

    }
    }
    if (licznik == 0 && j >= przesuniecie + rozmiar_slownika || j >=
rozmiar) //jeżeli sprawdziłem już dane w słowniku, to wychodzę z pętli
        break;
    if (licznik == 0 && znak_bufora == znak_slownika)
        licznik++; //jeżeli znak z bufora się pokrywa ze
znakiem ze słownika zwiększam licznik
    }
    for (j = 0; j < max_dlugosc + 1; j++)
    {
        if (dane[przesuniecie + rozmiar_bufora + j] < 0) //tutaj
sprawdzam czy pojawiły się znaki nienależące do
        {
            //standartu ascii
            czy_znak++;
            break;
        }
    }
    if (czy_znak) //jeśli tak, to muszę je skopiować za pomocą poniż-
szej funkcji, ponieważ po przyrównaniu //uległyby zamianie na znaki standarto-
we ascii
    {
        strncat(okno, &dane[przesuniecie + rozmiar_bufora], max_dlugosc +
1); //przesuwam okno
        czy_znak = 0;
    }
    else //jeżeli nie to mogę użyć zwykłego przyrównania/skopiowania zna-
ków, dzięki temu program wykonuje się szybciej
    {
        p = strlen(okno);
        for (j = 0; j < max_dlugosc + 1; j++)
            okno[p + j] = dane[przesuniecie + rozmiar_bufora + j];

        okno[p + j] = 0;
    }
    sprintf(pomoc, "%d", pozycja); //konwertuje pozycję powtarzają-
cych się znaków na char

    fwrite(pomoc, sizeof(char), strlen(pomoc), out);
    //wyprowadzam do pliku pozycję skompresowanych znaków

    if (rozmiar_bufora > 9 || rozmiar_slownika > 9)
        fwrite(",", sizeof(char), 1, out); //wyprowadzam do
pliku przecinek, aby odseparować pozycję i długość ciągu

    sprintf(pomoc, "%d", max_dlugosc); //konwertuje długość powtarzają-
cych się znaków na char

    fwrite(pomoc, sizeof(char), strlen(pomoc), out);
    //wyprowadzam do pliku długość skompresowanych znaków

    if (rozmiar_bufora > 9 || rozmiar_slownika > 9)
        fwrite(",", sizeof(char), 1, out); //wyprowadzam do pliku przeci-
nek, aby odseparować długość ciągu i następny znak

    fwrite(&okno[max_dlugosc + przesuniecie + rozmiar_slownika], size-
of(char), 1, out);
    //wyprowadzam do pliku następny znak

    przesuniecie += max_dlugosc + 1; //zapisuje przesunięcie okna
}

```

```

        free(pomoc);
        free(okno);          //dealokuje pamięć
    }

```

Funkcja dekompresja została zaimplementowana w następujący sposób:

```

//ta funkcja dekompresuje odczytaną zawartość danych wejściowych (skompresowanych)
//przekazanych do funkcji za pomocą parametrów *bufor i rozmiar
//parametry funkcji:
//char *bufor - łańcuch znaków zawierający zawartość pliku wejściowego
//int rozmiar - rozmiar bufora czyli zawartości pliku wejściowego
//FILE *out - wskaźnik na strukturę danych, która umożliwia operowanie na pliku wyj-
ściowym (w tym przypadku potrzebna jest
//możliwość zapisu do pliku)
//char *sownik_char - łańcuch znaków zawierający rozmiar słownika
//char *bufor_char - łańcuch znaków zawierający rozmiar bufora wejściowego (kodowania)
//funkcja nie zwraca żadnej wartości
void Dekompresja(char *wejscie, int rozmiar, FILE *out, char *sownik_char, char
*bufor_char)
{
    unsigned long long i, j, pozycja, dlugosc, przesuniecie = 0, p, ile = 100;
    //deklaruje zmienne pomocnicze
    short czy_znak = 0, pierwsze_wywołanie = 1;

    int rozmiar_sownika, rozmiar_bufora;
    char *bufor;

    rozmiar_sownika = pow(2, atoi(sownik_char));          //konwertuje pobrany jako
parametr wejściowy rozmiar słownika
    rozmiar_bufora = pow(2, atoi(bufor_char));             //konwertuje pobra-
ny jako parametr wejściowy rozmiar bufora

    if (rozmiar_sownika > rozmiar)                        //jeżeli rozmiar słownika jest większy
niż rozmiar pliku, to zmniejszam go do rozmiaru
        rozmiar_sownika = rozmiar;                        //pliku
    if (rozmiar_bufora > rozmiar)                          //jeżeli rozmiar bufora jest większy
niż rozmiar pliku, to zmniejszam go do rozmiaru
        rozmiar_bufora = rozmiar;                        //pliku

    bufor = malloc(sizeof(char)*(ile*rozmiar + 1));       //alokuje pamięć

    bufor[0] = 0;          //zeruje łańcuch znaków
    p = rozmiar_sownika;

    for (i = 0; i < rozmiar_sownika; i++)
        bufor[i] = wejscie[0];          //wypełniam bufor(okno) pierwszym znakiem

    bufor[i] = 0;

    for (i = 1; i < rozmiar; i++)          //w pętli analizuje znak po znaku dane wej-
ściowe
    {
        pozycja = wejscie[i] - '0'; //wczytuje pierwszą cyfrę pozycji zna-
ku

        while ((rozmiar_bufora > 9 || rozmiar_sownika > 9) && wejscie[i +
1] != ',')
        {
            pozycja *= 10;          //jeżeli liczba ma więcej cyfr, to
wczytuję kolejne i od razu konwertuję na int
            i++;

```



```

        pozycja += wejscie[i] - '0';
    }
    if ((rozmiar_bufora > 9 || rozmiar_slownika > 9))
        i++; //pomijam przecinek

    i++; //przechodzę do długości, następnej liczby

    dlugosc = wejscie[i] - '0'; //wczytuje pierwszą cyfrę

    if (pierwsze_wywołanie) //dla pierwszego wywołania mogę
odczytać tylko 1 znak
    {
        dlugosc = 1;
        pierwsze_wywołanie = 0;
    }

    while ((rozmiar_bufora > 9 || rozmiar_slownika > 9) && wejscie[i +
1] != ',')
    {
        dlugosc *= 10; //jeżeli liczba ma więcej cyfr, to
wczytuję kolejne i od razu konwertuje na int
        i++;
        dlugosc += wejscie[i] - '0';
    }
    if ((rozmiar_bufora > 9 || rozmiar_slownika > 9))
        i++; //pomijam przecinek

    i++; //przechodzę do następnego znaku

    for (int j = 0; j < dlugosc; j++)
    {
        if (bufor[przesuniecie + pozycja + j] < 0) //tutaj
sprawdzam czy pojawiły się znaki nienależące do
    {
        //standartu ascii
        czy_znak++;
        break;
    }
    }
    if (czy_znak) // jeśli tak, to muszę je skopiować za pomocą poniż-
szej funkcji, ponieważ po przyrównaniu
    { //uległyby zamianie na znaki
standartowe ascii
        strncat(bufor, &bufor[przesuniecie + pozycja], dlugosc);
//kopiuje odpowiednie znaki z bufora
        strncat(bufor, &wejscie[i], 1); //następnie dodaje
kolejny znak z pliku
        czy_znak = 0;
    }
    else //jeżeli nie to mogę użyć zwykłego przyrównania/skopiowania
znaków, dzięki temu program wykonuje się szybciej
    {
        for (j = 0; j < dlugosc; j++)
            bufor[p + j] = bufor[przesuniecie + pozycja + j];
//kopiuje odpowiednie znaki z bufora

        bufor[p + j] = wejscie[i]; //następnie dodaje kolej-
ny znak z pliku
        bufor[p + j + 1] = 0;
    }
}

```

```

        fwrite(&bufor[p], sizeof(char), dlugosc + 1, out);    //zapisuję
odtworzoną część skompresowanych danych

        przesuniecie += dlugosc + 1;    //przesuwam bufor(okno)
        p += dlugosc + 1;

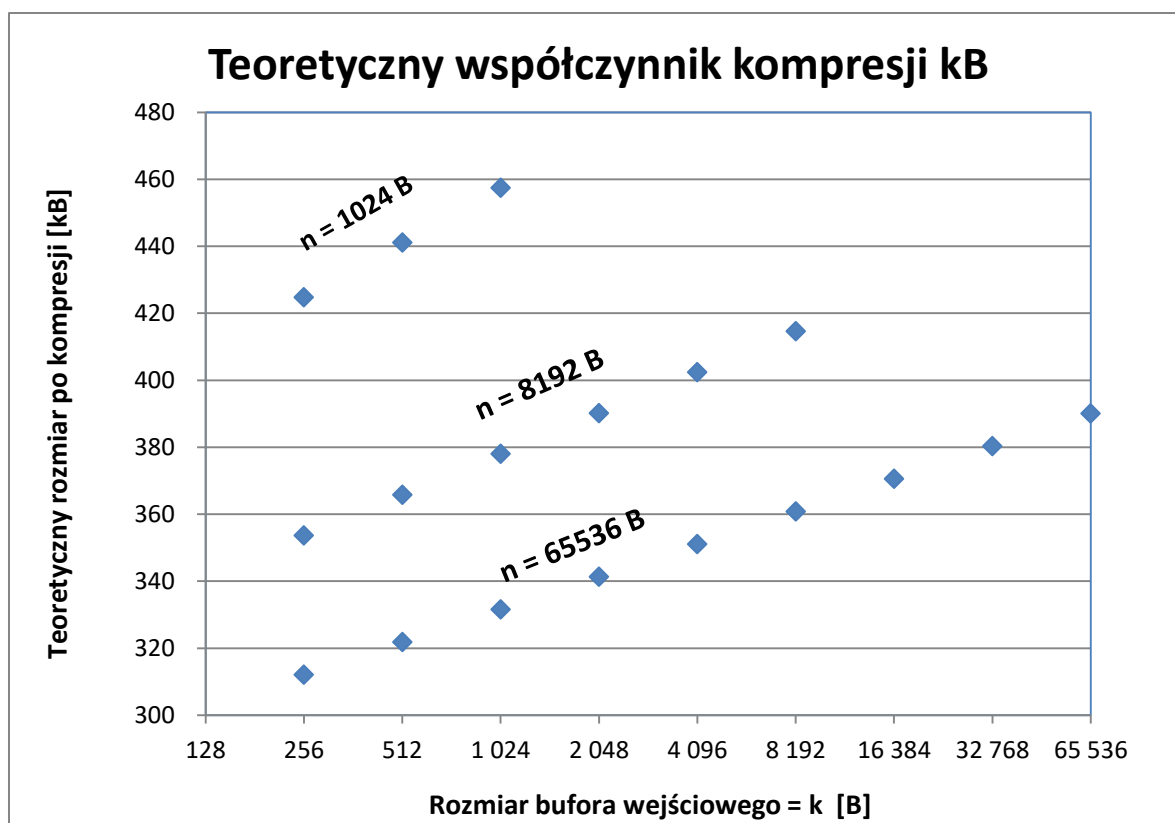
    }
    free(bufor);    //dealokuje pamięć
}

```

5. Testowanie

Program był testowany na różnych większych i mniejszych plikach. Testowanie plików o rozmiarze większym niż 500 kB było jednak problematyczne, gdyż program długo przetwarzał (kompresował/dekompresował) dane. Jednak program poprawnie wykonywał powierzone mu zadanie, jedynie dla pliku tekstowego. Poniżej wstawiam dane testowe i wykresy teoretycznego współczynnika kompresji dla powieści Pan Tadeusz.txt

rozmiar słownika	rozmiar bufora	teoretyczny współczynnik kompresji
1024	256	424820
1024	512	441159
1024	1024	457499
8192	256	353629
8192	512	365823
8192	1024	378017
8192	2048	390212
8192	4096	402406
8192	8192	414600
65536	256	312092
65536	512	321844
65536	1024	331597
65536	2048	341350
65536	4096	351103
65536	8192	360856
65536	16384	370609
65536	32768	380362
65536	65536	390115



6. Wnioski

Program był dla mnie bardzo trudny do wykonania. Choć nie wymagał znajomości żadnych nietypowych struktur danych, to algorytm kompresji był dość skomplikowany. Jednak zdecydowanie najwięcej czasu zajęło mi poprawianie kodu programu w taki sposób, by działał on dla różnych rozmiarów słownika i bufora wejściowego, a także próba zmniejszenia czasu jego działania, gdyż dla pliku o wielkości ok 370 kB kompresja zajmuje przynajmniej kilka sekund, a dla większego rozmiaru słownika (w okolicy wielkości pliku), trwała ona ponad minutę. Doszedłem do wniosku, iż główną przyczyną tak długiej pracy jest wielokrotne wywoływanie funkcji `strncat(..)`. Jednak przy zastąpieniu tej funkcji prostym przyrównywaniem (kopiowaniem) pojedynczego znaku, po dekompresji bezpowrotnie traciły się wszystkie polskie znaki i inne nienależące do standardu Ascii. Dlatego zarówno przy kompresji i dekompresji sprawdzam czy w części danych kompresowanych/dekompresowanych występują jakieś znaki nienależące do ascii (czy znak < 0). Jeżeli nie, to wykonuje wielokrotnie szybsze przyrównywanie, a jeżeli są to używam funkcji `strncat(...)`. Dzięki takiemu rozwiązaniu program wykonuje się wielokrotnie szybciej, szczególnie dekompresja, a jednocześnie nie następuje utrata żadnych znaków w tekście. Niestety po kompresji i dekompresji bitmapy (plików z rozszerzeniem `bmp`), nie da się już odtworzyć tej bitmapy.