

Seweryn Pastuch

Sprawozdanie Koder LT

Spis treści:

1. Abstrakt.....	2
2. Cele.....	2
2.1 Wstęp.....	2
2.2 Podział pliku tekstowego.....	3
2.3 Losowanie stopni.....	3
2.4 Zapis sumy kontrolnej.....	3
2.5 Uwagi.....	4
3. Kod, Funkcje.....	5
4. Dowody pracy kodera.....	18
4.1 Deterministyczne Losowanie stopni.	18
4.2 Dekodowanie ręczne przykładowych danych.....	22
4.3 Dowód poprawnego działania operacji XOR.....	30
4.4 Dowód poprawnego zapisu zakodowanych danych.....	31
5. Wnioski.....	31
6. Źródła.....	32
7. Kod kodera.....	32
8. Kod programu odczytującego(podgląd zakodowanych danych).....	41

1. Abstrakt

W tym sprawozdaniu chciałbym zaprezentować idee oraz logikę działania mojej implementacji kodera danych LT.

2. Cele

2.1 wstęp

Koder LT musi umieć odczytać dane z pliku tekstowego, podzielić je na równe części(tablice znaków), następnie w odpowiedni sposób wylosować stopień oraz indeksy symboli wejściowych, które będzie łączył ze sobą wykonując operację XOR kodów znaków symbolu wejściowego z tablicy ASCII, następnie trzeba dodać do symbolu kodu nagłówka, czyli informacje o tym, z czego został zrobiony oraz jego sumę kontrolną.

Jeżeli stopień symbolu kodu ==1 to wybieramy losowo jeden symbol wejściowy, kopujemy go do miejsca na dane w symbolu kodu, dodajemy nagłówek i zapisujemy w pliku wynikowym.

Ideą zapisania danych o indeksach i sumie kontrolnej (nagłówka zakodowanego symbolu) jest łączenie tablic char z tymi danymi zapisując je do nowej, większej.

Format danych:

[n znaków danych] [znak stopnia] [indeksy sąsiadów w ilości znaku stopnia][znaki sumy kontr.j]

np. 'a' 'b' 'c' 'd' 3 9 2 8 0 0 1

dane symbolu kodu: abcd

nagłówek: 3 9 2 8 0 0 1

stopień:3

indeksy symboli wejściowych: 9, 2, 8

suma kontrolna: 100

Ideą, celem takiego sposobu wysyłania danych jest to, że w praktyce szybciej odbierzemy dane wejściowe w oryginalnej postaci, gdy czasem będziemy przekazywać „niezrozumiałe” fragmenty danych, a czasem kopie danych wejściowych służących do zdekodowania tych nie zrozumiałych, niż jakbyśmy nieustannie wysyłali losowe kopie pakietów wejściowych, przewaga kodowania LT jest jeszcze bardziej widoczna, kiedy dane przesyłane są przez kanały stratne, a część przesyłanych pakietów jest utracona.

2.2 podział pliku tekstowego

Już na samym początku pracy z koderem napotkałem na ścianę, która na początku była dla mnie abstrakcją, a teraz w miarę swobodnie jestem w stanie przez nią przechodzić. Chcąc stworzyć koder dowolnych danych będziemy mieli dane różnych długości, które trzeba zapisać w odpowiednich zmiennych tablicowych. Aby zainicjować tabelę (tablica 2D) posiadającą rozmiar definiowany przez zmienną, będziemy musieli użyć tablicy wskaźników rozmiaru liczby pakietów wejściowych (indeksy tej tablicy to numery wierszy) do tablic char, również o rozmiarze definiowanym w trakcie działania programu, tym razem rozmiaru długości pakietu danych definiowanym przez użytkownika (wiersze tabeli). Jako, że ideą przy pracy była niezawodność dla różnych rozmiarów, idea dynamicznie alokowanej pamięci przetacza się przez cały koder danych.

2.3 Losowanie stopni

Idea deterministycznego losowania stopni na pierwszy rzut oka nie jest intuicyjna. Koncepcje obliczania prawdopodobieństwa wylosowania stopnia symbolu kodu pominę w tym momencie, jego interpretacja, algorytm liczenia zdefiniowany jest w kodzie, jednocześnie był on sprawdzany na kartce z długopisem, gdzie z reguły to kod udowadniał błędy w obliczeniach na kartce. Jeżeli mamy wyliczone prawdopodobieństwo na każdy indeks w tablicy zmiennych double rho[1]=0.40 ; rho[2]=0.50; rho[3] = 0.10 => rho[1]=0.40 ; rho[2]=0.90; rho[3] =1. źródła: [4][3]

To w tym momencie możemy losować liczbę z przedziału od 0 do 1 i jako wylosowaną liczbę od indeksu 1 do 3 brać ten indeks, którego wartość jako pierwsza sprawdzona jest mniejsza od naszej zmiennej wylosowanej 0;1 iterując od najmniejszego indeksu do największego tablicy prawdopodobieństwa. Przy testowaniu takiego losowania, im więcej liczb wylosujemy, tym bardziej proporcja wylosowania powiedzmy indeksu 1 do reszty będzie w przybliżeniu równa 0.4, czyt. prawdopodobieństwo wylosowania liczby 1 równa się w przybliżeniu 0.4 źródła: [4][3]

2.4 Zapis sumy kontrolnej

Przy obliczaniu sumy kontrolnej dużej części z generowanych symboli kodu, szczególnie tych o stopniu 1 będziemy dostawać wartości większe niż 255. Tych wartości nie jesteśmy w stanie zapisać na jednym znaku, więc musimy jakoś zakodować sumę kontrolną, tak aby udało się ją zamieścić w nagłówku symbolu kodu, w tablicy zmiennych unsigned char. Początkową ideą zapisywania sumy kontrolnej było konwertowanie jej na ciąg bitów i powiedzmy, w przypadkach przetwarzania bardzo dużych symboli wejściowych, gdy suma by wynosiła np. 65 535₁₀, to moglibyśmy ją podzielić na dwa znaki unsigned char i zapisać jako tablicę dwuelementową [255] [255]. Jednak implementacja takiego podziału na pierwszy rzut oka wydawała mi się

czasochłonna, lub nie znałem odpowiednich funkcji w cpp ułatwiających takie rozwiązanie. Wpadłem zatem na pomysł, aby taką sumę 432 zapisywać jako ciąg znaków o wartościach liczbowych [2] [3] [4], rozwiązanie te nie generuje dużej nadmiarowości kodu oraz okazało się być dość niezawodne. Mechanizm transformacji dużej liczby int na tablice znaków unsigned char/char okazuje się dość prosty wykorzystując operatory / oraz % na liczbach całkowitych int.

2.5 uwagi:

- A)** Ograniczeniem losowania stopnia, w tej interpretacji kodowania LT, jest maksymalny stopień możliwy do wylosowania, a zarazem maksymalna liczba pakietów wejściowych - 255 czyli maksymalna wartość jaką możemy zapisać na 8 bitach, czyt. maksymalny stopień możliwy do zapisania na jednym znaku, jednej zmiennej typu unsigned char, jako, że stopień kodujemy na jednym znaku w formacie danych.
- B)** Wszystkie dowody przedstawione w pracy, oraz przesłana implementacja kodera danych wczytuje dane wejściowe z pliku tekstowego .txt. Jest to element wygody wprowadzania i oglądania danych wejściowych, jednak na początku otwieranie plików, aby załadować je do pakietów wejściowych programu było realizowane przez otwieranie plików .pdf lub .jpg, co jest raczej bardziej formalną realizacją niezawodnego kodowania danych. Nic nie stoi na przeszkodzie zmiany typu otwieranych plików, jednak nic nie daje nam również powodów do takiego działania (brak aplikacji dekodującej dane)
- C)** Koder przy generowaniu liczby symboli kodu 300 razy większej niż liczba symboli wejściowych nie wyświetla dalej swojej pracy na ekranie(lecz prawdopodobnie nie zatrzymuje swojej pracy, funkcja main() nigdy nie zwraca wartości zero, proces tak naprawdę nic nie zwraca). Kodowanie przerywane jest w środku któregoś symbolu kodu powyżej tej liczby ograniczającej. Jest jednak to tak duża ilość kodu już wygenerowanego i zapisanego, że nawet przy modelu strat 90 procent pakietów danych, przy liczbie pakietów wejściowych np. 10 gdy mamy 3 000 wygenerowanych symboli kodu, dostajemy 300 symboli kodu zdolnych do przekazania informacji o symbolach wejściowych, co napewno będzie wystarczającą ilością do zdekodowania tych danych, jako dowód może posłużyć nam wzór na ilość symboli kodu potrzebnych do odzyskania danych zgodnie z założeniami procesu LT:

$$K = k \cdot \ln(k/\delta) // \text{średnia liczba symboli kodu potrzebna do odzyskania danych z prawdopodobieństwem } 1 - \delta$$

k=10

$$\delta = 0.01 // \text{prawdopodobieństwo nie powodzenia, tego że liczba potrzebnych symboli kodu będzie większa, niż wyliczona}$$

K =69,07

Dla $\delta = 0.99$, $K=23$

Bardzo ciężko jest mi teraz zlokalizować źródło tego problemu, jest cieś szansy, że może to być

problem z wielokrotnym wywołaniem polecenia cout. W trakcie wykonywania dowodu poprawnego losowania stopni, zauważylem, że na laptopie liczba możliwych do wygenerowania symboli kodu jest mniejsza niż 300x a wynosi ona około 250x więcej niż liczba pakietów wejściowych.

3. Kod, Funkcje:

1. void wczytajpakiety(int pakietln)

Funkcja ma na celu **załadowanie** do pamięci komputera **wszystkich symboli wejściowych**, które wytwarza, dzieląc plik tekstowy na równe części. Przyjmuje podstawowy parametr pakietln, który używa do obliczenia ilości pakietów do załadowania do pamięci oraz do zainicjowania rozmiaru tablic unsigned char na pakiety wejściowe. W ostatniej, podwójnej pętli programu czytamy kolejne symbole wejściowe, zapisujemy je do pamięci, jednocześnie znak po znaku wyświetlając je na ekranie dla późniejszej analizy.

```
void wczytajpakiety(int pakietln){  
    ifstream file("test.txt", ios::in | ios::binary);  
    if (!file) {cout << "Nie udało się otworzyć pliku!";}  
    liczbaPakietow = dlugosc(file, pakietln); // Obliczanie liczby pakietów za pomocą predefiniowanej funkcji  
    inputs = new char*[liczbaPakietow]; // Alokacja pamięci na tablice 2D char  
    for (int i = 0; i < liczbaPakietow; i++) {  
        inputs[i] = new char[pakietln];  
        memset(inputs[i], 0, pakietln);  
    }  
    for(int j=0; j<liczbaPakietow;j++) { // odczytywanie pakietów o długości pakietln z pliku test  
        file.read(inputs[j], pakietln);  
        cout << "paket nr " << (j+1) << ". ";  
        for( int d=0; d<pakietln; d++)  
            cout << inputs[j][d];  
        cout << endl;  
    }  
    file.close();  
}
```

Zmienne:

ifstream file // zmienna plikowa, dane wejściowe.

int liczbaPakietów // zmienna globalna obliczana przez funkcję int dlugosc(file, pakietln).

inputs = new char*[liczbaPakietow] // dynamiczna tablica wskaźników do znaków char o rozmiarze liczby pakietów do przechowania.(numery wierszy w klasycznej tabelce)

inputs[i] = new char[pakietln] // każdy ze wskaźników inputs wskazuje na tablice char długości

pakietIn(wiersze w klasycznej tabelce)

Funkcje składowe:

dlugosc(file, pakietIn) // liczy ile pakietów o długosci pakietIn będziemy mogli zainicjować

[7] memset(inputs[i], 0, pakietIn) // funkcja jest częścią biblioteki cstring. ma za zadanie ustawienie pewnych wartości we wskazanym obszarze pamięci. w naszym przypadku początek to indeks 0 każdej kolejnej zainicjowanej tablicy char, wartość każdego bajtu(od indeks 0 przez kolejne pakietIn bajtów) ustawiana jest na 0, aby wyeliminować "śmieci" z pamięci ram w inicjowanych tablicach.

file.read(inputs[j], pakietIn) // funkcja read odczytuje pakietIn znaków z pliku tekstowego i zapisuje je do kolejnych tablic char na które wskazuje inputs[].

file.close() // po zakończeniu zapisywania raz na zawsze żegnamy się z plikiem test.txt

2. int dlugosc(ifstream& c, int div)

Funkcja długość iteruje po każdym znaku pliku tekstowego, zliczając je, następnie dzieli liczbę znaków pliku przez pakietIn==div z zaokrągleniem w góre i zwraca wynik tego dzielenia. Przykład: liczba znaków=17 div=5 normalnie: 17/5 =(int)3[taka liczba pakietów 5 znakowych spowoduje krytyczną utratę danych juz przy zapisywaniu danych wejściowych do pamięci], ale w programie [patrz return]: (17 + 5 -1) / 5 = 21/5= (int)4 czyli jeżeli mamy symbole wejściowe 5 znakowe to ostatni będzie nie cały zapisany danymi, jednak nie będzie pominięty. Ostatnim, nieużywanym elementem funkcji jest ustawienie zmiennej globalnej kontrol na resztę z dzielenia znaków pliku przez długość pakietu.

```
int dlugosc(ifstream& c, int div) { // funkcja sprawdzająca ile pakietów zostanie wygenerowanych
    char znak;
    int liczbaznakow = 0;
    while (c.get(znak)) {
        liczbaznakow++;
    }
    c.clear(); // usuwanie flagi strumienia, np. o koncu pliku tekstowego
    c.seekg(0); // Resetowanie wskaznika na początek
    kontrol=liczbaznakow % div;
    return (liczbaznakow + div - 1) / div; // Zaokrąglenie w góre
}
```

Zmienne:

ifstream& c // referencja do zmiennej plikowej odpowiedź na ograniczenia środowiskowe - zmienna plikowa file z funkcji wczytującej pakiety nie może być przekazana jako parametr do funkcji dlugosc(), czyt. strumień danych nie może być od tak skopiowany jak zwykły int czy string przekazywany do funkcji. odpowiedzią jest wskazanie adresu w pamięci gdzie nasz strumień

danych się znajduje=> operujemy na tym strumieniu danych a nie jego kopii.

```
int div // długość pakietu
```

```
char znak // pojedynczy znak z pliku
```

```
int liczbaznakow // zmienna inkrementowana, gdy kolejne znaki z pliku zostają odczytane.
```

```
int kontrol // zmienna globalna, reszta z dzielenia znaków pliku przez długość pakietu wejściowego. Zmienna nie ma żadnej funkcjonalności w programie, jednak zostawiłem ją w razie konieczności rozpatrzenia skrajnych sytuacji w kodowaniu i dekodowaniu LT
```

Funkcje składowe:

```
c.get(znak) // odczytuje kolejny znak z pliku tekstowego, strumienia danych c.
```

```
c.clear // usuwa flagi ze strumienia, np. takie o skończeniu się pliku, odczytaniu już do końca, bo później na tym pliku będziemy dalej pracować w funkcji wczytajpakiety wyżej opisanej.
```

```
c.seekg(0) // Ustawiamy pozycję odczytu danych od pierwszego znaku.
```

3. void tablicasoliton(int inp)

Funkcja deklaruje tablice rho prawdopodobieństwa na każdy indeks symbolu wejściowego. Idea tego procesu została wyjaśniona wcześniej. liczba indeksów rho=255+1(MAX_K jest globalną stałą), maksymalna liczba symboli wejściowych obsługiwana przez program. wartości komórek rho[1] do rho[inp/liczbapakietow] są liczone na podstawie wzoru ideal Soliton distribution na końcu wartości tablicy rho[] zapisywane są w globalnej tablicy double rozklad[]

```
void tablicasoliton(int inp) {
    double rho[MAX_K + 1] = {0.0};
    rho[1] = 1.0 / inp;
    cout<<"pr kom 1: "<<rho[1]<<endl;
    for (int i = 2; i <= inp; ++i) {
        rho[i] = 1.0 / (i * (i - 1));
    }
    rozklad[1] = rho[1];
    for (int i = 2; i <= inp; ++i) {
        rozklad[i] = rho[i];
    }
}
```

Zmienne:

```
int inp // liczba pakietów wejściowych
```

```
double rho[] // tymczasowa tablica prawdopodobieństwa wylosowania każdego indeksu
```

```
double rozklad[] // globalna tablica prawdopodobieństwa
```

Funkcje składowe: brak.

4. `double calculateR(int inp)`

Funkcja zwraca parametr R, zależny od prawdopodobieństwa niepowodzenia się dekodowania D oraz liczby symboli wejściowych inp/liczba pakietów. Oryginalnie wartość zwrócona przez calculateR jest niezbędnym parametrem funkcji tablica_robust_soliton()

```
double calculateR(int inp ) {  
    cout<<"R: "<<(c*log ((inp/D))*sqrt(inp))<<endl;  
    return (c*log ((inp/D))*sqrt(inp)); }
```

Zmienne:

```
int inp // liczba pakietów wejściowych
```

```
double c // dowolna stała, większa od 0
```

```
double D // prawdopodobieństwo niepowodzenia się dekodowania, mniejsze wartości powodują większą ilość stopni 1 w losowaniu
```

Funkcje składowe:

```
sqrt(inp) // funkcja biblioteki cmath, potęga o wykładniku 2 zmiennej inp.
```

```
log(a) // funkcja biblioteki cmath, zwraca logarytm naturalny zmiennej a.
```

5. `void tablica_robust_soliton(int inp , double R)`

1 pętla for: Funkcja deklaruje tymczasową tablice rho o rozmiarze 256 indeksów(wyjaśnione w tablicasoliton). Korzystając ze wzoru Robust-Soliton distribution deklaruje prawdopodobieństwo na wylosowanie kolejnych indeksów symboli wejściowych.

2 pętla for: Funkcja do globalnej tablicy double rozkład[i](prawdopodobieństwa soliton) dodaje obliczone indeksy prawdopodobieństwa rho[i](prawdopodobieństwa robust).

3 pętla for: Funkcja normalizuje tablicę prawdopodobieństwa dzieląc każdy jej element przez sumę całej tablicy, gdyby tego nie wykonywać, suma prawdopodobieństw na każdy indeks wynosiła by więcej niż jeden, co nie ma sensu.

4 pętla for: Funkcja od 2 indeksu sumuje indeks obecny z poprzednim i zapisuje w obecnym.
Przykładowe dane: rho[1]=0.40 ; rho[2]=0.50; rho[3] = 0.10 => rho[1]=0.40 ; rho[2]=0.90; rho[3]=1. Element potrzebny do losowania deterministycznego liczby losowej całkowitej z przedziału <1; liczba pakietów>

```

void tablica_robust_soliton(int inp, double R) {
    double rho[MAX_K + 1] = {0.0};
    rho[1] = R / (inp);
    if (1 <= ((inp/R) - 1)) {
        for (int i=2;i<((inp/R)-1);i++) {
            if ((i== (inp/R)) && (i!=1))
                rho[i]=(R*log (R/D));
            else
                rho[i] = R / (i * inp);
        }
    }
    for (int i = 1; i <= inp; ++i) {
        rozklad[i]=rozklad[i]+rho[i];
        cout<<"komorka robust-soliton "<<rozklad[i]<<endl;
        suma=suma+rozklad[i];
    }
    cout<<"suma robust soliton: "<<suma<<endl;
    for (int i = 1; i <= inp; ++i) {
        rozklad[i]=rozklad[i]/suma;
        cout<<"pr komorki robust-soliton:"<<i<<". "<<rozklad[i]<<endl;
    }
    for (int i=2;i<=inp;i++) {
        rozklad[i]=rozklad[i]+rozklad[i-1];
        cout<<"koncowa tablica od indeksu 2: "<<rozklad[i]<<endl;
    }
}

```

Zmienne:

`int inp // liczba pakietów wejściowych`

`double R // parametr obliczany przez funkcję calculateR()`

`double D // prawdopodobieństwo niepowodzenia się dekodowania, mniejsze wartości powodują większą ilość stopni 1 w losowaniu`

`double rho[] // tymczasowa tablica prawdopodobieństwa wylosowania każdego indeksu`

`double suma // globalna suma wszystkich elementów tablicy rozklad[]`

Funkcje składowe: brak.

6. void degree(int pakietlength , int czas)

Funkcja zarządza dystrybucją stopni oraz tworzeniem tablicy zawierającej stopień symbolu kodu oraz indeksy sąsiadów symbolu kodu.

Kolejność działania: Losujemy ziarno generatora losowego.

Zapisujemy do zmiennej table stopień symbolu kodu wyliczony przez funkcje randrobustsoliton() opisana w pkt 7.

Dynamicznie tworzymy tablice char degre[] o wielkości stopnia symbolu kodu+1 , żeby pomieścić dane w stylu [stopień][liczba indeksów =stopień], np. (int)[3];[4];[2];[1]. Pierwszy element tablicy degre[] = stopień symbolu kodu.

Tworzymy tymczasowo używany symbol kodu, dynamiczną tablice char encodedpakiet[] do zapisywania symboli wejściowych po wykonaniu operacji XOR z resztą.

Pętla while: losujemy stopień symbolu kodu funkcją rand() % liczbaPakietow +1, w efekcie czego dostawac będziemy liczby całkowite przedziału <1;liczbaPakietow> powtarzamy tyle razy, ile wynosi zmienna int table, czyli stopień symbolu kodu. jednocześnie przy każdym kolejnym losowaniu sąsiada w while pętla for sprawdza, czy kolejny wylosowany sąsiad nie został już wylosowany wcześniej, czyt. eliminujemy powtórki przerywając to powtórzenie pętli.

Jeżeli brak powtórek zapisujemy wylosowanego sąsiada do tablicy degree indeksów.(if (! powtórka). Jeżeli jest to pierwszy wylosowany sąsiad to parametr aktywacja==true, wykonuje się funkcja memcpy(). W przeciwnym wypadku wysyłamy indeks sąsiada oraz długość pakietu danych do funkcji xorowanie, która wykonuje operacje XOR kolejnego sąsiada z już wybranym wcześniej.(patrz pkt 8)

Jest to może najmniej przejrzysty element programu z racji na rozpatrywanie kilku skrajnych przypadków losowania stopni i sąsiadów: 1. Jeżeli wylosowaliśmy pierwszego sąsiada to nie wykonujemy operacji XOR go z niczym tylko zapisujemy go do wynikowego pakietu, jeżeli losujemy kolejnych sąsiadów, to wykonujemy XOR ich danych z już wybranym pierwszym. Jeżeli wybrany sąsiad istnieje już w tabeli indeksów sąsiadów, to powtórka= true i losujemy sąsiada jeszcze raz.

```
void degree(int pakietlenght, int czas) {
    bool aktywacja = true;
    srand(czas); // Ziarno generatora losowego
    int table = randrobustsoliton(liczbaPakietow,time(0)); // Liczba sąsiadów
    if(table==1)
        licznikstopnia1+=1;
    licznikcalosciowy+=1;
    degré = new unsigned char[table+1];
    degré[0] = table; // Stopień zapisany na początku tablicy
    cout << "Stopień symbolu kodu: " << (int)degré[0] << endl;
    encodedpakiet = new unsigned char[pakietlenght]; // Rezerwacja pamięci dla pakietu wynikowego
    int i = 1; // Zmienna iterująca
    while (i <= table) {
        int temp = (rand() % liczbaPakietow) + 1;
        // Sprawdzanie powtórek
        bool powtorka = false;
        for (int j = 1; j <= i; j++) {
            if (degré[j] == temp) {
                powtorka = true;
                break;
            }
        }
        if (!powtorka) {
            degré[i] = temp;
            cout << "Sąsiad symbolu kodu: " << (int)degré[i]<< endl ;
            if (aktywacja) {
                memcpy(encodedpakiet, inputs[temp - 1], pakietlenght);
                aktywacja = false;
            } else {
                xorowanie(temp, pakietlenght);
            }
            i++; // Tylko gdy brak powtórek zwiększamy licznik
        }
    }
}
```

Zmienne:

```
int pakietlenght // długość symbolu wejściowego  
int czas // czas systemowy  
int table // stopień symbolu kodu  
int temp // obecnie sprawdzany, czy powtórka, wylosowany sąsiad symbolu kodu/indeks symbolu wejściowego  
int licznikstopnia1 // liczniuki używane do sprawdzania statystyk wygenerowanych symboli kodu  
int licznikcalosciowy // wyżej  
degree[] // dynamiczna unsigned char tabela przechowująca stopień oraz indeksy symbolu kodu  
bool powtorka // parametr używany do losowania kolejnych sąsiadów.
```

Funkcje składowe:

```
xorowanie(temp, pakietlenght) // funkcja wykonująca XOR każdego elementu z nowo zainicjowanego encodedpakiet[] oraz kolejnego wylosowanego symbolu wejściowego inputs[][]  
[1] memcpy(encodedpakiet, inputs[temp - 1], pakietlenght)// część biblioteki <cstring> kopiuje symbol wejściowy inputs( numerowane od zera dlatego -1) do encodedpakiet pakietlenght bajtów/znaków  
[2] rand() funkcja losująca liczbę losową z dość dużego przedziału, nie istotne dla nas my i tak wyznaczamy z niej odpowiednią resztę z dzielenia  
randrobustsoliton(liczbaPakietow,time(0))// funkcja zdefiniowana w pkt7.  
[2] srand(czas) // funkcja ustawia ziarno generatora losowego na pobrany wcześniej czas systemowy
```

7. **int randrobustsoliton(int inp , unsigned int a)**

Funkcja jest implementacją losowania deterministycznego liczby z dowolnego przedziału. Mechanizm działania jest prosty, skoro chcemy, żeby losować np 2 razy więcej cyfr 1 niż 2, to deklarujemy tablicę zmiennoprzecinkową z indeksami [1]; [2]. Wartości pod kolejnymi indeksami to 0,666 dla indeksu [1] oraz 1,000 dla indeksu [2]. Jeżeli będziemy losować liczbę z przedziału 0 do 1, to dużo częściej będziemy losować liczbę mniejszą od indeksu 1 niż mniejszą od indeksu [2] ale większą od indeksu [1]. W przybliżeniu w 2/3 przypadków spełniony będzie warunek 1. Realizacja tego założenia dla jakichkolwiek tablic w kontekście rozmiarów i wartości przedstawiona jest na poniższym zdjęciu.

(tablic tworzonych w funkcjach **void tablica_robust_soliton(int inp , double R)** oraz **void tablicasoliton(int inp)** (patrz punkty powyżej)).

```

int randrobustsoliton(int inp,unsigned int a) {
    static mt19937 gen(a);
    uniform_real_distribution<> dist(0.0, 1.0);
    double r = dist(gen); // Losowa liczba z przedzialu [0, 1]
    for (int i = 1; i <= inp; ++i) {
        if (r < rozklad[i]) {
            return i;
        }
    }
    return 0;
}

```

Zmienne:

`int inp` // liczba symboli wejściowych
`unsigned int a` // czas systemowy
`double r` // stopień symbolu kodu

Funkcje składowe:

`xorowanie(temp, pakietlenght)` // funkcja xorująca każdy element z nowo zainicjowanego `encodedpakiet[]` oraz kolejnego wylosowanego symbolu wejściowego `inputs[][]`

[3] `gen(a)`// argument biblioteki `<random>` użyty do zainicjowania generatora liczb pseudolosowych `mt19937` (tylko raz(static))

[3] `dist(gen)`// funkcja biblioteki `<random>` w efekcie wcześniej zdefiniowanych przedziałów itd funkcja generuje liczbę losową z przedziału 0 do 1

8. void xorowanie(int a , int pakietlenght)

Funkcja wykonuje działanie xor każdego znaku z obecnego, niepełnego symbolu kodu(bez indeksów i sumy kontrolnej) z kolejnym wylosowanym indeksem symbolu wejściowego.

```

void xorowanie(int a, int pakietlenght) {
    for(int l=0; l<pakietlenght;l++) {
        encodedpakiet[l]^=inputs[a-1][l];
    } cout<<endl;
}

```

Zmienne:

`int a` // kolejny wylosowany indeks sąsiada/symbolu wejściowego
`int pakietlenght` // długość pakietu danych symbolu wejściowego
`encodedpakiet[]` // dynamiczna tablica char produkowana przez funkcję `degre()`, patrz pkt 6.

```
inputs[][] // dynamiczna tablica char dynamicznych tablic char ;) symboli wejściowych, tworzona przez f.  
wczytajpakiety()
```

Funkcje składowe: brak

9. void combine(int len1) oraz void combine2(int len2 , int len1)

Funkcje combine oraz combine 2 należy omówić w jednym podpunkcie, z racji na ich bliźniacze zastosowanie. Funkcja combine inicjuje tabele temporary[] wielkości pakietIn/długość pakietu wejściowego + len2/wielkość tabeli zawierającej stopień i indeksy. następnie kopiuje każdy element z symbolu kodu po xorze do temporary[], a później każdy element z tablicy indeksów i stopni do odpowiednich indeksów temporary[].

Funkcja combine2 do długości len1/długość pakietu wejściowego dodaje długość tablicy z indeksami. Parametr len2 równa się wielkości tablicy zawierającej sumę kontrolną symbolu kodu w temporary[]. Funkcja usuwa nie potrzebną pamięć encodedpakiet[] oraz inicjuje ją na nowa z nowym rozmiarem len1+len2. Następnie kopiuje symbol kodu + indeksy (temporary[]) do encodedpakiet[], a na końcu w odpowiednie indeksy encodedpakiet[] zapisuje dane z tablicy zawierającej sumę kontrolną (sumach[])

```
void combine(int len1){ //funkcja laczaca symbol kodu i indeksy  
    int len2 = ((int)degre[0] +1);  
    temporary = new unsigned char[len1 + len2];  
    licznika=len1+len2;  
    for(int l=0; l<len1;l++){  
        temporary[l]=encodedpakiet[l];  
    }  
    for(int l=len1; l<len1+len2;l++){  
        temporary[l]=degre[l-len1];  
    }  
}  
void combine2(int len2,int len1){ //funkcja laczaca symbol kodu i indeksy  
    delete[] encodedpakiet;  
    len1 += ((int)degre[0] +1);  
    cout<<"dlugosc 3( zksorowany pakiet + indeksy ) "<<len1<<endl;  
    cout<<"dlugosc 4 ( suma kontrolna w charach ) "<<len2<<endl;  
    encodedpakiet = new unsigned char[len1 + len2];  
    licznika=len1+len2;  
    for(int l=0; l<len1;l++){  
        encodedpakiet[l]=temporary[l];  
    }  
    for(int l=len1; l<len1+len2;l++){  
        encodedpakiet[l]=sumach[l-len1];  
    }  
}
```

zmienne combine:

```
int len1 // długość sym,bolu wejściowego
```

```
int len2 // wielkość tablicy char zawierającej stopień plus indeksy
```

degre[] // dynamiczna tablica zawierająca indeksy symbolu kodu oraz jego stopień
encodedpakiet[] // dynamiczna tablica zawierająca symbol kodu bez indeksów

temporary[] // dynamiczna tablica zawierająca symbol kodu razem z indeksami

Funkcje składowe: brak

zmienne combine2:

int len1 // kolejny wylosowany indeks sąsiada/symbolu wejściowego

int len2 // kolejny wylosowany indeks sąsiada/symbolu wejściowego

degre[] // dynamiczna tablica zawierająca indeksy symbolu kodu oraz jego stopień

encodedpakiet[] // dynamiczna tablica zawierająca symbol kodu razem z indeksami i sumą kontrolną

temporary[] // dynamiczna tablica zawierająca symbol kodu razem z indeksami

Funkcje składowe:

delete[] // operator usuwa dynamicznie zainicjowaną tablicę przy pomocy **new**

10. **void calcsum(**int** pakietln)**

Prosta funkcja sumująca każdy znak symbolu kodu do zmiennej int, wywołuje następnie funkcję zamieniającą zmienną liczbową na tablicę znaków. Przyjmuje podstawowy parametr pakietln, który jest wielkością tablicy zawierającej symbol kodu oraz stopień+indeksy

```
void calcsum(int pakietln) {  
    int sumal=0; // suma kontrolna zakodowanego symbolu  
    for(int i=0; i<pakietln+degre[0]+1;i++) {  
        sumal+=temporary[i];  
    }  
    inttochar(sumal,pakietln);  
}
```

zmienne:

int pakietln // długość tablicy temporary[]

int suma1 // suma kontrolna zakodowanego symbolu

temporary[] // dynamiczna tablica zawierająca symbol kodu razem z indeksami

Funkcje składowe:

void inttochar(**int** suma , **int** pakietln) // funkcja zamieniająca sumę kontrolną np. (int) 123 na tablice char '3';'2';'1'

11. void inttochar(int suma , int pakietln)

Funkcja sumę kontrolną w postaci int zamienia na tablice znaków char. zasada działania: Pętla do...while() oblicza jak dużą tablicę znaków trzeba będzie zarezerwować na zapisanie sumy kontrolnej porównując i zliczając ile liczb po kolei można by teoretycznie zmieścić w sumie kontrolnej. np. licznik =1 suma=123, czy 10 się mieści? tak licznik=2; czy 100 się mieści? tak licznik=3, 1000 się nie mieści, wielkość tablicy zawierającej sumę kontrolną == 3.

W pętli for w kolejnych powtórzeniach do tablicy zawierającej sumę kontrolną zapisujemy reszty z dzielenia jedności liczby sumy kontrolnej, z każdym powtórzeniem ucinając jedności z tej liczby(123/10 = (int)12)

```
void inttochar(int suma, int pakietln){ // funkcja konwertujaca sume kontrolna, np. 3245 na tablice char : ['3' '2' '4' '5']
    int i=1;
    int counterofch=1;//dlugosc sumy kontrolnej
    do{
        i=i*10;
        if(i<suma)
            counterofch+=1;
    }while(i<suma);
    sumach = new unsigned char[counterofch];
    cout<<"wartosci tablicy zawierajacej sume kontrolna: ";
    for(int i=0;i<counterofch;i++){
        sumach[i]=suma%10;
        cout<<(int)sumach[i]<<" ";
        suma=suma/10;
    }
    cout<<endl;
    combine2(counterofch, pakietln);
}
```

zmienne:

```
int suma // suma kontrolna symbolu kodu
int pakietln // parametr przekazywany dalej
int counterofch // rozmiar tablicy sumach[], wielkość sumy kontrolnej w nowym formacie
sumach[] // suma w postaci dynamicznej tablicy char
```

Funkcje składowe:

combine2() //łączy symbol kodu oraz sumę kontrolną, opisana wyżej w pkt. 9

12. int main()

Funkcja wczytuje warunki początkowe kodowania a następnie wywołuje wszystkie niezbędne funkcje do kodowania danych.

```

int main()
{
    int pakietlenght;
    cout<<"podaj dlugosc pakietu"<<endl;
    cin>>pakietlenght;
    wczytajpakiety(pakietlenght);
    cout<<"podaj tolerancje na bledy, wartosci nizsze powoduja wieksza niezaowdnosc w klasifikowan
    cin>> D;
    cout<<"podaj stala C wieksza od zera, najlepiej rowna 1: ";
    cin>>c;cout<<endl;
    tablicasoliton(liczbaPakietow);
    tablica_robust_soliton(liczbaPakietow, calculateR(liczbaPakietow));
    float znakkk;
    cin>>znakkk;
    ofstream plik("example.txt", std::ios::out | std::ios::binary| std::ios::trunc ); // w pliku sa
    for(int i=0; i<(liczbaPakietow*5);i++){
        cout<<"======"<<endl;
        degree(pakietlenght, (time(0)+i));
        combine(pakietlenght);
        calcsum(pakietlenght);
        cout<<"wypisuje wynikowy pakiet ";
        for(int i=0; i<liczniaka; i++){//!!!!!!!!!!!!!
            plik<<encodedpakiet[i];
            cout<<(i+1)<<". "<<encodedpakiet[i]<<" "<<(int)encodedpakiet[i]<<" ";
        }cout<<endl;plik<<"";
        deletemem();
        deleteinputs();
        plik.close();
        cout<<"liczba jedynek wynosi: "<<licznikstopnia1<<endl;
        cout<<"liczba operacji wynosi: "<<licznikcalosciowy<<endl;
    }
    return 0;
}

```

kolejność działania:

```

int pakietlenght // wczytujemy długość pakietu wejściowego podaną przez użytkownika

wczytajpakiety() // zapisujemy pakiety danych z pliku tekstowego do pamięci ram

double D// wczytujemy tolerancje na błędy podaną przez użytkownika

float c // wczytujemy stałą skalowania podaną przez użytkownika, dla mnie zawsze równa1

tablicasoliton() // tworzymy tabelle prawdopodobieństwa solitonową

tablica_robust_soliton() // zastępujemy tabelle soliton tabelą prawdopodobieństwa robust-soliton

float znak // wczytujemy znak z klawiatury, aby kontynuować

ofstream plik(...) // tworzymy zmienną plikową do zapisywania danych, symboli kodu

for(int i=0; ...) // powtarzamy liczbatapakietów*dowolna liczba losowanie i tworzenie symbolu kodu

degree() //losowanie stopni symbolu kodu

combine() //łączenie stopni oraz symbolu kodu

calcum() //obliczanie sumy kontrolnej wywołuje inttochar() a ten wywołuje combine2()

```

```
for() // zapisywanie symbolu kodu do pliku  
deletemem()//usuwanie pamięci po zapisaniu symbolu kodu  
// zapisanie znaku ';' do pliku oddzielając symbole kodu  
//koniec pętli for//  
deleteinputs()//usuwanie symboli wejściowych po zakończeniu kodowania
```

13. void deletemem()

Funkcja usuwa pamięć, która przestaje być potrzebna po zapisaniu kolejnego symbolu kodu do pliku tekstowego. Wszystkie te tabele będą deklarowane na nowo, z nowymi sumami kontrolnymi, indeksami i danymi przy tworzeniu następnego symbolu kodu.

```
]void deletemem () {  
    delete [] degre; // tabela  
    delete [] encodedpakiet;  
    delete [] temporary; // pak  
    delete [] sumach;  
}
```

zmienne:

degre[] // tabela przechowująca stopień oraz indeksy sąsiadów symbolu kodu
encodedpakiet[] // symbol kodu LT
temporary[] // któryś, tymczasowy symbol kodu
sumach[] // suma kontrolna w charach

funkcje składowe:

delete[] // operator usuwa dynamicznie zainicjowaną tablicę przy pomocy new

14. void deleteinputs()

Funkcja iteruje po każdym wierszu tabeli inputs[] usuwając wszystkie zadeklarowane symbole wejściowe. Następnie usuwa wszystkie indeksy symboli wejściowych, tabelę inputs[] funkcja wywoływana jest po skończonym kodowaniu danych

```

void deleteinputs () {
    for (int k = 0; k < liczbaPakietow; k++) {
        delete[] inputs[k];
    }
    delete[] inputs;
}

```

zmienne:

inputs // dynamiczna tablica wskaźników do wskaźników do tablic znaków char o rozmiarze liczby pakietów do przechowania.(numery wierszy oraz wiersze w klasycznej tabelce);

funkcje składowe:

delete[] // operator usuwa dynamicznie zainicjowaną tablicę przy pomocy **new**

ZMIENNE GLOBALNE ORAZ BIBLIOTEKI:

```

#include <iostream>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <algorithm>
#include <cmath>
#include <iomanip>
#include <random>
using namespace std;
// liczniki do testowania działania kodowania, tj licznik ile pakietów zakodowanych wygenerowano oraz ile zakodowanych
int licznikstopnia=0;
int liczniklosciowy=0;
double R; // używane przy rozkładzie robust-soliton
float c;// stała do obliczania R
double D; //niezawodność
double rozklad[256];
const int MAX_K = 255;
double suma=0;
int kontrol;// zmienna, reszta z dzielenia liczby znaków w pliku tekstowym przez długość pakietu, może być przydatna przy
char** inputs; // tabela 2D przechowująca symbole wejściowe
int liczbaPakietow;
unsigned char* degré; //tabela char przechowująca stopień oraz indeksy "sasiadow" symbolu kodu
unsigned char* encodedpakiet;
unsigned char* temporary; //pakiet służący do podmieniania wartości encoded packiet
unsigned char* sumach; // zmienna przechowująca sumę kontrolną np 234 w postaci tablicy znaków o wartościach [2][3][4]
int licznika; // !! ta zmienna zrobić jako lokalna przewidywana długość pakietu po dodaniu czegos

```

4.Dowody pracy kodera:

4.1 Deterministyczne Losowanie stopni.

Kodowanie LT nie mogło być efektywne oraz skuteczne, jeżeli stopnie symboli kodu wybierane były by w sposób czysto losowy, jeżeli tak by było, nadmiarowość wysyłanych danych w skrajnych sytuacjach mógłaby być czasem niewielka, a czasem naprawdę duża, zanim dekodowanie danych zakończyło się pomyślnie. Losowanie deterministyczne stopni symboli kodu w teorii daje nam kontrolę nad spodziewaną nadmiarowością kodu, która będzie wygenerowana oraz nad częstotliwością wysyłanych stopni symboli kodu k danych wejściowych

od 1 do k. Poniżej przedstawiam wyliczony rozkład prawdopodobieństwa każdego stopnia symbolu kodu dla 10 pakietów wejściowych oraz dla porównania rozkład wyliczony przez koder oraz stosunek liczby wylosowanych „jedynek” symboli kodu do całości generowanych danych

Rozkład prawdopodobieństwa Robust-Soliton:

```
pakiet nr 7. a
pakiet nr 8. a
pakiet nr 9. a
pakiet nr 10. a
podaj tolerancje na bledy, wartosci nizsze powodują wieksza
Gdy wypisze tablice prawdopodobienstw, nacisnij znak z klaw
0.5
podaj stala C wieksza od zera, najlepiej rowna 1: 1

pr kom 1: 0.1
komorka robust-soliton 1.04733
komorka robust-soliton 0.5
komorka robust-soliton 0.166667
komorka robust-soliton 0.0833333
komorka robust-soliton 0.05
komorka robust-soliton 0.0333333
komorka robust-soliton 0.0238095
komorka robust-soliton 0.0178571
komorka robust-soliton 0.0138889
komorka robust-soliton 0.0111111
suma robust soliton: 1.94733
pr komorki robust-soliton:1. 0.53783
pr komorki robust-soliton:2. 0.256761
pr komorki robust-soliton:3. 0.0855871
pr komorki robust-soliton:4. 0.0427936
pr komorki robust-soliton:5. 0.0256761
pr komorki robust-soliton:6. 0.0171174
pr komorki robust-soliton:7. 0.0122267
pr komorki robust-soliton:8. 0.00917005
pr komorki robust-soliton:9. 0.00713226
pr komorki robust-soliton:10. 0.00570581
koncowa tablica od indeksu 2: 0.794591
koncowa tablica od indeksu 2: 0.880178
koncowa tablica od indeksu 2: 0.922972
koncowa tablica od indeksu 2: 0.948648
koncowa tablica od indeksu 2: 0.965765
koncowa tablica od indeksu 2: 0.977992
koncowa tablica od indeksu 2: 0.987162
koncowa tablica od indeksu 2: 0.994294
koncowa tablica od indeksu 2: 1
```

//Patrz pr komorki robust-soliton: 1...10

$$k = 10$$

$$\delta = 0,5$$

pr robust: $p(i) = \frac{R}{ik}$ dla $i = 1 \dots i = \frac{k}{R} - 1$
 $p(i) = R(C_0)(R/S)/k$ dla $i = \frac{k}{R}$
 $p(i) = 0$ dla reszty

$$R = C \ln\left(\frac{10}{0,5}\right) \cdot \sqrt{10}$$

pr robust:
dla $i = 1 \Rightarrow p(1) = \frac{1}{10}$
pr robust: dla reszty
 $p(1) = \frac{1}{10} = 0,1$
 $p(2) = \frac{1}{2} = 0,5$
 $p(3) = 0,1666 \dots$
 $p(4) = \frac{1}{4} = 0,25 \dots$
 $p(5) = \frac{1}{5} = 0,2 \dots$
 $p(6) = 0,1333 \dots$
 $p(7) = 0,0909 \dots$
 $p(8) = 0,07148571$
 $p(9) = 0,047619 \dots$
 $p(10) = 0,027777 \dots$

$$R = 2,995732 \cdot 3,162277$$

$$R = 9,47333440$$

pr robust: $p_1 = \frac{R}{ik} = 0,94733344$

$$p_2 = 0$$

$$p_3 = 0$$

$$\vdots$$

$$p_{10} = 0$$

$$\text{suma } p_1 \text{ r i } p_{10} = 1,04733344$$

reszta tak samo

$$p_1 = \frac{1,04733}{1,04733} = 0,978287 \quad \text{suma wszystkich: } \approx 1,94733323$$

$$p_2 = \frac{0,5}{10} = 0,256261872 \quad \text{podzielic wszystkie przez}$$

$$p_3 = 0,08558 \quad \text{sumę (normalizacja)}$$

$$p_4 = 0,04279$$

$$p_5 = 0,025646$$

$$p_6 = 0,014719$$

$$p_7 = 0,0122218$$

$$p_8 = 0,0091400$$

$$p_9 = 0,007124782$$

$$p_{10} = 0,0054054$$

poaktas wynikony

// obliczenia w programie się zgadzają z obliczeniami wykonanymi na kartce

```
Stopieñ symbolu kodu: 3
Sasiad symbolu kodu: 3
Sasiad symbolu kodu: 7

Sasiad symbolu kodu: 1

wartosci tablicy zawierajacej sume kontrolna: 1 ; 1 ; 1 ;
dlugosc 3( zksorowany pakiet + indeksy ) 5
dlugosc 4 ( suma kontrolna w charach ) 3
wypisuje wynikowy pakiet 1. a 97 2. ♥ 3 3. ♥ 3 4. 7 5. ♠ 1 6. ♠ 1 7. ♠ 1 8. ♠ 1
=====
Stopieñ symbolu kodu: 1
Sasiad symbolu kodu: 6
wartosci tablicy zawierajacej sume kontrolna: 4 ; 0 ; 1 ;
dlugosc 3( zksorowany pakiet + indeksy ) 3
dlugosc 4 ( suma kontrolna w charach ) 3
wypisuje wynikowy pakiet 1. a 97 2. ♠ 1 3. ♦ 6 4. ♦ 4 5. 0 6. ♠ 1
=====
Stopieñ symbolu kodu: 1
Sasiad symbolu kodu: 9
wartosci tablicy zawierajacej sume kontrolna: 7 ; 0 ; 1 ;
dlugosc 3( zksorowany pakiet + indeksy ) 3
dlugosc 4 ( suma kontrolna w charach ) 3
wypisuje wynikowy pakiet 1. a 97 2. ♠ 1 3. 9 4. 7 5. 0 6. ♠ 1
liczba jedynek wynosi: 1338
liczba operacji wynosi: 2500

Process returned 0 (0x0)    execution time : 98.640 s
Press any key to continue.
```

//Patrz dól zrzutu ekranu: liczba jedynek wynosi: oraz liczba operacji wynosi(liczba wytworzonych symboli kodu)

Obliczenia proporcji ilości wylosowanych stopni 1 do wytworzonych wszystkich stopni: $1338/2500=0,5352$ (próba testowa wynosi 2500 symboli kodu)

prawdopodobieństwo stopnia 1: 0,53783 (ze zrzutu ekranu na początku tego dowodu)

//proporcja w praktycznej realizacji kodwania różni się o 0,00583, 1,08 % (wyliczenia kontra praktyczna).

4.2 Dekodowanie ręczne przykładowych danych.

Ograniczeniem wykonania ręcznego dekodowania jest niskie skomplikowanie wejściowych danych, aby szukanie wartości znaków z tablicy Ascii było względnie szybkie, przykładowe dane wejściowe to będą: „aaaabbbbccccdddeeeeffffgggghhhiiiijjjj” a podział danych to będą 4 litery na każdy symbol wejściowy. Przy dekodowaniu danych, nie wprowadziłem symulacji utraty danych, więc każda odczytana suma kontrolna się zgadzała.

Algorytm działania dekodowania: **1.** odczytaj symbol kodu: odczytaj pierwsze 4 znaki jako dane symbolu kodu, odczytaj 5 znak jako stopień s. k., odczytaj tyle indeksów, kolejnych znaków jako zestaw indeksów symboli wejściowych tworzących ten symbol kodu, a następnie sprawdź, czy suma wartości ASCII poprzednio odczytanych znaków zgadza się z sumą zapisaną od końca indeksów aż do średnika. **2.** Jeżeli stopień symbolu kodu > 1 to zapisz ten symbol kodu w pamięci, sprawdź, czy któryś z indeksów poprzednio już odzyskanych symboli wejściowych (symboli kodu stopnia 1) pokrywa się z indeksami tego symbolu kodu, jeżeli tak, to wykonaj operacje XOR danych (pierwszych 4 liter) tego symbolu odczytanego z już poprzednio odzyskanym symbolem wejściowym 1. Zmniejsz stopień tego symbolu kodu o 1, usuń informacje o indeksie, który już został zXORowany z listy indeksów symbolu kodu **3.** Jeżeli stopień symbolu kodu == 1 to sprawdź czy indeks pakietu wejściowego tego symbolu kodu nie pokrywa się z którymś z indeksów już poprzednio wczytanych symboli kodu > 1, jeżeli tak, to wykonaj operacje XOR danych tego symbolu odczytanego z już poprzednio odczytanym ze stopniem większym niż 1. Zmniejsz stopień tego poprzednio odczytanego o 1, usuń informacje o indeksie, który już został zXORowany z listy indeksów symbolu kodu wcześniej odczytanego. Jeżeli stopień tego „większego” większego symbolu kodu równa się teraz 1, to wykonaj pkt 3.

Uwagi: w trakcie dekodowania na kartce błędnie odrzuciłem 11 odebrany symbol kodu, przez co liczba odczytanych symboli kodu, do momentu jak zdekodowałem dane była większa, niż w wersji zdigitalizowanej tego dekodowania, gdzie zauważałem ten błąd.

Nie jestem w stanie załączyć tu całej treści pracy programu generującego symbole, więc do kontroli zostawiam dwa pliki: „praca programu dekodowanie reczne.txt”, z tego pliku w łatwy sposób odczytywałem symbole kodu oraz „test dekodowania recznego.rtf”, wersja zdigitalizowana, bardziej czytelna dekodowania danych, którą polecam sprawdzić, są tam lepsze opisy tego, jaki symbol wejściowy udało się zdekodować z których danych jak i również dekodowanie przeprowadzone jest bardziej efektywnie, bez utraty / pominięcia 11 symbolu kodu.

2. in 1. aaaa (int) = 94
in 2. bbbb (int) = 98
in 3. cccc (int) = 92
in 4. dddol (int) = 100
in 5. eeee (int) = 101
in 6. ffff (int) = 102
in 7. gggg (int) = 103
in 8. hhhh (int) = 104
in 9. iiii (int) = 105
in 10. jjjj (int) = 106

zadeleranane

in 4. olololol (copy)

in 7. gg gg gg (copy)

in 1. aa aa aa (xor 7; 7;)

in 10. jj jj jj (copy)

in 5. eeee (copy)

in 3. cccc (XOR 3; 4;)

in 9. iiii (XOR 4; 9; 7; 3)

in 6. ffff (copy) 16 - kodu

in 2. bbbb (XOR 9; 2; 1)

in 8. hhhh (XOR 12345678 > 20)

1. kodu stopień 1 indeks 4

old old 1 4 5 0 4
└───┘ └───┘ └───┘ └───┘ └───┘

✓

$$2. \text{ kontrolna} = 100 \cdot 4 + 1 + 4 = 405, \text{ zgoda na sie}$$

2. kodu stopień 2 indeks 11 7

Pakryza rig 25

1. kashu

6 6 6 6

XOR 103 103 103 103

$$2. \text{ kontr. } 6 \cdot 4 + 2 + 1 + 7 = 34$$

$$\Rightarrow 9; 7; 9; 7; 9; 7; 1; 1;$$

✓

3. kodu stopień 10 indeks: 1; 8; 4; 9; 7; 5; 2; 3; 6; 10

→ 0; 1; 1; 1; 1; 1; 1; 1; 1; 1;

PK; 11; 11; 11; 11

$$11; 11; 11; 11; 10; 1; 8; 4; 9; 7; 5; 2; 3; 6; 10$$

Kashu

9; 0; 1

Pakryza rig 2 1 2. kashu indeks 9

2. kontr

$$\begin{array}{cccc} 11 & 11 & 1 & 1 \\ \hline \text{XOR} & 100 & 100 & 100 \end{array}$$

102

✓

111 111 111 111

stopień - 1 - 4

111 111 111 111 9 1; 2; 3; 5; 6; 7; 8; 9; 10;

pakryza rig 2 5 - 2 kashu indeks 7

stop - 1

$$\begin{array}{cccc} 111 & 111 & 111 & 111 \\ \hline \text{XOR} & 103 & 103 & 103 \end{array}$$

8; 8; 8; 8; 8; 8; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10;

8 8 8 8
Pakryza rig 2 (zakłoskowany) 2. kashu

$$\begin{array}{cccc} 8 & 8 & 8 & 8 \\ \hline \text{XOR} & 97 & 97 & 97 \end{array}$$

105; 105; 105; 105; 7; 8; 3; 5; 6; 8; 9; 10;

Pakryza rig 2 6. 2. kashu

d - 1 - 7 - 4 - 1

$$\begin{array}{cccc} 105 & 105 & 105 & 105 \\ \hline \text{XOR} & 106 & 106 & 106 \end{array}$$

76 - 1

3; 3; 3; 3; 3; 3; 6; 7; 2; 3; 5; 6; 8; 9;

3 3 3 3

$$\begin{array}{cccc} 101 & 101 & 101 & 101 \\ \hline \text{XOR} & 102 & 102 & 102 \end{array}$$

5; 2; 3; 6; 8; 9;

3- z. kashu iżg dobrzy

Palmyra zię z 10, zdekolorowym z. kashu

Palmyra zię z 4, zdekolorowanym z. kashu

$$\begin{array}{cccc} 102 & 102 & 102 & 102 \\ \text{in } 9. & 109 & 109 & 105 \end{array}$$

$$\begin{array}{cccc} \cancel{99} & 99 & 99 & 99 \\ \hline 108 & 108 & 108 & 108 \end{array} \quad \begin{array}{c} \cancel{10} - 2 \\ 3 \end{array} \quad \begin{array}{c} \cancel{8} - 3 \\ 1 \\ 2 \\ 6 \\ 8 \end{array}$$

palmyra zię z 16 symbolami kashu

$$\begin{array}{cccc} 108 & 108 & 108 & 108 \\ \text{XOR } 102 & 102 & 102 & 102 \end{array}$$

$$\begin{array}{cccc} 10 & 10 & 10 & 10 \\ ; & ; & ; & ; \end{array} \quad \begin{array}{c} \cancel{10} - 7 \\ 2 \end{array} \quad \begin{array}{c} \cancel{2} - 6 \\ 1 \\ 2 \\ 8 \end{array}$$

palmyra zię z 3 z. kashu i 2 reszty z

$$\begin{array}{cccc} 10 & 10 & 10 & 10 \\ \text{XOR } 98 & 98 & 98 & 98 \end{array} \quad \begin{array}{c} \cancel{10} - 1 \\ 1 \end{array} \quad \begin{array}{c} \cancel{9} - 2 \\ 1 \\ 8 \end{array}$$

4. krok stапінь 4 інв $\begin{matrix} 4,9 \\ \times 1 \\ \times 1 \\ \times \end{matrix}$

$\begin{matrix} 9,9,7,9,4,4 \\ 1,1,1,1,1,1 \end{matrix}$; $9,7,3,3,6$

фактична сума \geq 1. крок 1, конт = 63 ✓

999 9

XOR $\begin{array}{r} 100100100100 \\ 100100100100 \\ \hline 000000000000 \end{array}$ 109; 109; 109; 109; 3 3; 7; 9
109 109 109

стапінь-1, -4

5. krok стапінь

фактична сума \geq 5. розмір крохн інд 7

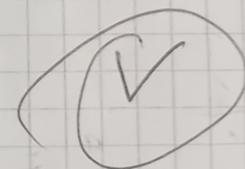
XOR $\begin{array}{r} 109 & 109 & 109 & 109 \\ 103 & 103 & 103 & 103 \\ \hline 10 & 10 & 10 & 10 \end{array}$

10; 10; 10; 10; 2; 3; 7
st-1 -4 -7

фактична сума \geq 10 7. крохн, залишковий

10 10 10 10

XOR $\begin{array}{r} 99 & 99 & 99 & 99 \\ \hline 105 & 105 & 105 & 105 \end{array}$ 1; 9



5. symb Koef. stopien 1 indeks 4

(V)

103; 103; 103; 103; 1; 7; 0; 2; 4

6. symb Koef. stopien 1 indeks 10

z. karta = 429 ✓

106; 106; 106; 106; 1; 10; 5; 3; 4; z. karta = 435 ✓

7. z. k. st. 4 inol. 4; 4; 2; 6;
7; 7; 7; 7; 4; 4; 7; 2; 6; 5; 3; 4; 7; 5; ✓
pokrywa się z 1. z. k. i 2. 5. z. koeff.

7 7 4 4
103 103 103 103

$$\begin{array}{r} \text{xor} \\ \hline 103 & 103 & 103 & 103 & \text{st-2} & -4-7 \\ 4 & 4 & 4 & 4 & 2; & 2; 6 \end{array}$$

8. z. koeff. stopien 1 indeks 5 z. karta = 410 ✓

101; 107; 109; 107; 1; 5; 0; 7; 4

(V)

9. z. koeff. st. 3 inol. 10 62 z. karta = 89

110; 110; 110; 110; 3; 10; 6; 2; 1; 6; 4

Pokrywa się z 6. symb koeff. od 10

$$\begin{array}{r} 106 & 106 & 106 & 106 \\ \text{xor} \quad 110 & 110 & 110 & 110 \\ \hline 4; & 4; & 4; & 4; & \text{st-1} & -10 \\ & & & & 2 & 6; 2; \end{array}$$

10. z koeff. st. 2 inol. 3; 4;

7; 7; 7; 7; 3; 3; 4; 7; 3; z. koeff. = 37

Pokrywa się z 2. z. koeff. inol. 4

7 7 4 4

$$\begin{array}{r} \text{xor} \\ \hline 100 & 100 & 100 & 100 \\ 99 & 99 & 99 & 99 & ; 1 : 3 \end{array}$$

(V)

11. 1. krok stopień 6 ind. 7; 2; 3; 6; 4; 8
pokrywa się z symbolem kodu, co symbol
zakodowany nr. 3.

12. 12; 12; 12; 12; 6; 7; 2; 3; 6; 4; 8

Pokrywa się z symbolem kodu, co symbol
zakodowany nr. 3.

12. symbol kodu stopień 2 indeks 3; 10

9; 9; 9; 9; 2; 10; 3; 1; 5;

Pokrywa się z kodem, zakodowanym

in 3 i in 10 w jive anane

13. symbol kodu stopień 2 indeks 9; 3; 3; 7

in 3 i in 7 w znane

14. symbol kodu stopień 1 indeks 4

w 7 jest znany

15. symbol kodu stopień 3 indeks 10; 5; 2

-11- znane

16. 2. krok stopień 1 indeks 3

a robi

1

robi = 415 V

17.

102; 102; 102; 102; 1; 6; 5; 1; 4

17¹. 2. krok st 1 ind. 10

18¹. krok st 1 ins 3

19¹. krok st 2 ins 6; 4

20¹. krok st 2 ins 8

-15-

21+1 2. Kodn rt 2 ind 3; 1;
 22 2. Kodn rt 3 ind 6; 1; 7
 23+1 2. Kodn rt 3 ind 9 2 1 ^{z. hantm = 438V}
~~106 106 106 106~~ [~]
~~97 97 97 97~~
~~x 08~~ ~~105 105 105 105~~ , t-2 - 9-1
~~18; 98; 98; 98; 1; 2~~ ^①

4.3 Dowód poprawnego działania operacji XOR symboli wejściowych.

Załączam xor1.txt, xordane2.txt, examplexor.txt

dane wejściowe:

.

```
podaj dlugosc pakietu
10
pakiet nr 1. Litwo, Ojc
pakiet nr 2. yzyno moja
pakiet nr 3. ! ty jeste
pakiet nr 4. s jak zdro
pakiet nr 5. wie;
Ile
pakiet nr 6. cie trzeba
pakiet nr 7. cenic, te
pakiet nr 8. n tylko si
pakiet nr 9. e dowie,
pakiet nr 10. Kto cie st
pakiet nr 11. racil. Dzi
pakiet nr 12. s pieknosc
pakiet nr 13. twa w cal
pakiet nr 14. ej ozdobie
pakiet nr 15.
Widze i
pakiet nr 16. opisuje, b
pakiet nr 17. o tesknie
pakiet nr 18. po tobie.
podaj tolerancje na bledy,
```

pierwsze dwa znaki każdego z symboli wejściowych
tworzących pierwszy symbol kodu
oraz operacja xor wykonana na nich

18, p 112	o 111
14, e 101	j 106
1, L 76	i 105
5, w 119	i 105
8, n 110	spacja 32
7, spacja 32	c 99
9, e 101	spacja 32
xor: =5	=102

pierwszy symbol kodu odczytany przez program odczytujący
zakodowane dane(wyświetla wartości znaków)

♣ 5	Jak widać, stworzony po części symbol kodu
f 102	według instrukcji zgadza się z tym stworzonym
d 100	według tej samej, losowo wygenerowanej
/ 47	instrukcji przez koder danych LT. Działanie
♣ 5	operacji xor na dynamicznych tablicach
A 65	znakowych zostało opracowane w osobnym
I 73	projektie i nie było implementowane „na
8	wiarę” w koderze danych nie znając
B 66	poprawności wykonywanych operacji na
32	symbolach wejściowych.
7	
↑ 18	
↓ 14	
☺ 1	
♣ 5	
8	
7	
	9
☺ 2	
7	
♣ 5	

// W plikuxor1.txt pokazuję jeszcze jeden ciekawy sposób na sprawdzanie poprawności operacji XOR, polegający na szukaniu zer w zakodowanych danych

4.4 Dowód poprawnego zapisu zakodowanych danych.

(załączam pliki exampledane, odczytdane oraz praca dane .txt do pełnego podglądu)

praca programu (dwa pierwsze symbole kodu)

```
Stopień symbolu kodu: 1
Sasiad symbolu kodu: 1
wartosci tablicy zawierajacej sume kontrolna: 8 ; 7 ; 3 ;
dlugosc 3( zksorowany pakiet + indeksy ) 6
dlugosc 4 ( suma kontrolna w charach ) 3
wypisuje wynikowy pakiet 1. D 68 2. a 97 3. n 110 4. e 101 5. ☺ 1 6. ☺ 1 7. 8 8. 7 9. ♥ 3
=====
Stopień symbolu kodu: 1
Sasiad symbolu kodu: 4
wartosci tablicy zawierajacej sume kontrolna: 7 ; 1 ; 4 ;
dlugosc 3( zksorowany pakiet + indeksy ) 6
dlugosc 4 ( suma kontrolna w charach ) 3
wypisuje wynikowy pakiet 1. k 107 2. l 108 3. a 97 4. d 100 5. ☺ 1 6. ♦ 4 7. 7 8. ☺ 1 9. ♦ 4
```

Odczyt liter oraz wartosci ASCII (program odczytujacy zakodowany plik):

```
Hello world!
D 68
a 97
n 110
e 101
☺ 1
☺ 1
8
7
♥ 3
; 59
k 107
l 108
a 97
d 100
☺ 1
♦ 4
7
☺ 1
♦ 4
; 59

// odczytane dane z pliku "exampledane.txt" (zakodowane dane) zgadzają się z „pracadane.txt” (pracą, którą dokumentował koder), pokazuje to program wyświetlający znaki zakodowanego pliku (kod znajduje się w punkcie 10.)
```

5. Wnioski:

Wszystkie cele założone w punkcie drugim sprawozdania zostały zrealizowane pomyślnie. W całym sprawozdaniu brakuje jednak dowodów i badań formalnych, które można przeprowadzić jedynie posiadając działającą aplikację dekodera. Praca oprogramowania kodującego i dekodującego kody LT może być testowana wieloma wzorami wspomianymi w pracy Michael'a Luby'ego „LT codes” niestety jednak, doprowadzenie pracy do tego miejsca nie powiodło się. Powiodło się za to na tyle formalne i wykonalne, na ile to możliwe udowodnienie działania kodera danych, testując jego działanie, oraz analizując szczegółowo zapisane symbole kodu.

Sukcesem również okazał się sposób zapisu informacji zakodowanych, zauważę, że niewiele grup projektowych zdecydowało się na tworzenie pliku z zakodowanymi danymi(niezależnie od tematu projektu), co jest według mnie niezbędne do wykonania go w stu procentach.

Praca wykonana przy tworzeniu efektywnego oraz uniwersalnego kodera danych LT daje bardzo dużo doświadczenia z zakresu dynamicznego alokowania pamięci oraz zarządzania nią, z racji na język w którym pisana była aplikacja. Przy tworzeniu wymogiem była również wiedza odnośnie używania oraz zachowania się wskaźników do pamięci w praktyce. Wnioskując, tworzenie tego typu aplikacji w Języku C++ jest bardzo czasochłonne i powolne, lecz wartościowe w kontekście aspektów wymienionych wyżej. Element, który został wykonany nie efektywnie, lecz nauczył mnie bardzo wiele, to brak użycia struktur danych, typu array vector itd. z biblioteki STL języka C++, który mógł znacznie skrócić kod oraz ilość czasu poświęconą na pracę z programem. Kod kodera danych chociaż nie jest idealnie ułożony, został napisany w poprawny sposób, dbając o nie występowanie błędów w zakodowanym pliku tekstowym. Gdybym miał okazję pracować po raz drugi z podobnymi projektami, to użyłbym aplikacji automatycznej kontroli wersji oprogramowania(np. git), zamiast zarządzać nimi ręcznie, tworząc przy każdym kolejnym dniu pracy kopię dotychczas już wykonanego projektu oprogramowania z nową nazwą.

6. Źródła:

- [1] rand() srand(): https://www.youtube.com/watch?v=-NA_B_vJP14 // podstawowe losowanie(indeksy)
- [2] funkcja memcpy(): <<https://en.cppreference.com/w/cpp/string/byte/memcpy> //>
- [3] dist(), mt19937
https://en.cppreference.com/w/cpp/numeric/random/uniform_real_distribution ,
<https://www.youtube.com/watch?v=oW6iuFbwPDg> // losowanie z przedziału (a;b)
- [4] <https://www.youtube.com/watch?v=DfziDXHYoik> //idea custom distribution
- [5] Język C++. Szkoła programowania. Wydanie VI Stephen Prata // idea dynamicznie lokowanej pamięci, tablic
- [6] https://youtu.be/C4qi_oJoUrE?si=D7Abj7HORCRnRYRg //wprowadzenie od It
- [7] memset() <https://www.geeksforgeeks.org/memset-in-cpp/>
- [8] <https://www.youtube.com/watch?v=RiP4BgxO3es> //inspiracja do formatu sprawozdania z kodu
- [9] https://www.researchgate.net/publication/221498536_LT_codes //LT codes pdf

7.Kod kodera

*używać pliku tekowego test.txt jako dane wejściowe, nie deklarować długości symboli wejściowych, co do których nie jest się pewnym(żeby nie przekroczyć ograniczeń programowych, maksymalnej liczby symboli wejściowych-255. Ustawienia kodera domyślnie generują 3 razy więcej symboli kodu niż symboli wejściowych, aby to zmienić, trzeba edytować pętle for w

funkcji main(). Nie deklarować prawdopodobieństwa nie powodzenia większego niż 1;). po skończeniu kodowania uruchomić program wklejony na końcu sprawozdania, po programie kodera, aby wyświetlić to, co zostało zakodowane w pliku wynikowym "example.txt".

Kod kompilował kompilator GNU GCC compiler*

```
int main()
{
    int pakietlenght;
    cout<<"podaj dlugosc pakietu"<<endl;
    cin>>pakietlenght;
    wczytajpakiety(pakietlenght);
    cout<<"podaj tolerancje na bledy, wartosci nizsze powoduj
    cin>> D;
    cout<<"podaj stała C wieksza od zera, najlepiej rowna 1:
    cin>>c;cout<<endl;
    tablicasoliton(liczbaPakietow);
    tablica_robust_soliton(liczbaPakietow, calculateR(liczbaPa
    float znakkk;
    cin>>znakkk;
    ofstream plik("example.txt", std::ios::out | std::ios::bi
    for(int i=0; i<(liczbaPakietow*3);i++){
        cout<<"=====
        degree(pakietlenght, (time(0)+i));
        combine(pakietlenght);

#include <iostream>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <algorithm>
#include <cmath>
#include <iomanip>
#include <random>
using namespace std;

//2 liczniki do testowania dzialania kodowania, tj licznik ile pakietow zakodowanych wygenerowano
//oraz ile zakodowanych pakietow stopnia 1 wygenerowano, przydatne przy sprawdzaniu czy rozklad
//prawdopodobienstwa wygenerowany na poczatku programu zgadza sie z faktycznym losowaniem stopni

int licznikstopnia1=0;
int licznikcalosciowy=0;
double R; // uzywane przy rozkladzie robust-soliton
float c;// stała do obliczania R
```

```

double D; //niezawodnosc

double rozklad[256];

const int MAX_K = 255;

double suma=0;

int kontrol;// zmienna, reszta z dzielenia liczby znakow w pliku tekstowym przez dlugosc pakietu, moze
byc przydatna przy zakomunikowaniu jak zapisac ostatni symbol wejsciowy do tablicy znakow ktora jest
dluzsza niz ten symbol

char** inputs; // tabela 2D przechowujaca symbole wejsciowe

int liczbaPakietow;

unsigned char* degré;//tabela char przechowujaca stopien oraz indeksy "sasiadow" symbolu kodu

unsigned char* encodedpakiet;

unsigned char* temporary;//pakiet sluzacy do podmieniania wartosci encoded packiet

unsigned char* sumach;// zmienna przechowujaca sume kontrolna np 234 w postaci tablicy znakow o
wartosciach [2][3][4]

int licznika;// !! ta zmienna zrobic jako lokalna przewidywana dlugosc pakietu po dodaniu czegos

int dlugosc(ifstream& c, int div) {// funkcja sprawdzajaca ile pakietow zostanie wygenerowanych o
dlugosci div

char znak;

int liczbaznakow = 0;

while (c.get(znak)) {

liczbaznakow++;

c.clear(); // usuwanie flagi strumienia, np. o koncu pliku tekstopiego

c.seekg(0); // Resetowanie wskaznika na poczatek

kontrol=liczbaznakow % div;

return (liczbaznakow + div - 1) / div; // Zaokraglenie w góre

}

void wczytajpakiety(int pakietIn){

ifstream file("test.txt", ios::in | ios::binary);

if (!file) {cout << "Nie udalo sie otworzyc pliku!";}

liczbaPakietow = dlugosc(file, pakietIn);// Obliczanie liczby pakietów za pomoca predefiniowanej funkcji

```

```

inputs = new char*[liczbaPakietow];// Alokacja pamieci na tablice 2D char
for (int i = 0; i < liczbaPakietow; i++) {
    inputs[i] = new char[pakietIn];
    memset(inputs[i], 0, pakietIn);
}

for(int j=0; j<liczbaPakietow;j++){//odczytywanie pakietow o dlugosci pakietIn z pliku test
    file.read(inputs[j], pakietIn);
    cout<<"pakiet nr "<<(j+1)<<. " ;
    for( int d=0; d<pakietIn; d++)
        cout<<inputs[j][d];
    cout<<endl;
}
file.close();
}

void xorowanie(int a, int pakietlenght){
    for(int l=0; l<pakietlenght;l++){
        encodedpakiet[l]^=inputs[a-1][l];
    }cout<<endl;
}

double calculateR(int inp ){cout<<"R: "<<(c*log((inp/D))*sqrt(inp))<<endl; return(c*log((inp/D))*sqrt(inp));
}

void tablicasoliton(int inp) {
    double rho[MAX_K + 1] = {0.0};
    rho[1] = 1.0 / inp;
    cout<<"pr kom 1: "<<rho[1]<<endl;
    for (int i = 2; i <= inp; ++i) {
        rho[i] = 1.0 / (i * (i - 1));
    }
    rozklad[1] = rho[1];
    for (int i = 2; i <= inp; ++i) {
        rozklad[i] = rho[i];}
}

```

```

}

void tablica_robust_soliton(int inp, double R){

    double rho[MAX_K + 1] = {0.0};

    rho[1] = R/(inp);

    if(1<=((inp/R)-1)) {

        for(int i=2;i<((inp/R)-1);i++){

            if((i==(inp/R))&&(i!=1))

                rho[i]=(R*log(R/D));

            else

                rho[i] = R/(i*inp);

        }

    }

    for (int i = 1; i <= inp; ++i) {

        rozklad[i]=rozklad[i]+rho[i];

        cout<<"komorka robust-soliton "<<rozklad[i]<<endl;

        suma=suma+rozklad[i];

    }

    cout<<"suma robust soliton: "<<suma<<endl;

    for (int i = 1; i <= inp; ++i) {

        rozklad[i]=rozklad[i]/suma;

        cout<<"pr komorki robust-soliton:"<<i<<. " <<rozklad[i]<<endl;

    }

    for(int i=2;i<=inp;i++){

        rozklad[i]=rozklad[i]+rozklad[i-1];

        cout<<"koncowa tablica od indeksu 2: "<<rozklad[i]<<endl;

    }

}

int randrobustsoliton(int inp,unsigned int a) {

    static mt19937 gen(a);
}

```

```

uniform_real_distribution<> dist(0.0, 1.0);

double r = dist(gen); // Losowa liczba z przedziału [0, 1)

for (int i = 1; i <= inp; ++i) {

    if (r < rozklad[i]) {

        return i;

    }

}

return 0;

}

void degree(int pakietlenght, int czas) {

    bool aktywacja = true;

    srand(czas); // Ziarno generatora losowego

    int table = randrobustsoliton(liczbaPakietow,time(0)); // Liczba sąsiadów

    if(table==1)

        licznikstopnia1+=1;

        licznikcalosciowy+=1;

        degré = new unsigned char[table+1];

        degré[0] = table; // Stopień zapisany na początku tablicy

        cout << "Stopień symbolu kodu: " << (int)degré[0] << endl;

        encodedpakiet = new unsigned char[pakietlenght]; // Rezerwacja pamięci dla pakietu wynikowego

        int i = 1; // Zmienna iterująca

        while (i <= table) {

            int temp = (rand() % liczbaPakietow) + 1;

            // Sprawdzanie powtórek

            bool powtorka = false;

            for (int j = 1; j <= i; j++) {

                if (degré[j] == temp) {

                    powtorka = true;

                    break;

                }

            }

            if (powtorka)

                degré[i] = temp;

            else

                degré[i] = rand();

            i++;

        }

        delete [] degré;

        degré = NULL;

    }

}

```

```

    }

}

if (!powtorka) {

    degré[i] = temp;

    cout << "Sasiad symbolu kodu: " << (int)degré[i]<< endl ;

    if (aktywacja) {

        memcpy(encodedpakiet, inputs[temp - 1], pakietlenght);

        aktywacja = false;

    } else {

        xorowanie(temp, pakietlenght);

    }

    i++; // Tylko gdy brak powtórek zwiększamy licznik

}

}

void combine(int len1){ //funkcja laczaca symbol kodu i indeksy

    int len2 = ((int)degré[0] +1);

    temporary = new unsigned char[len1 + len2];

    licznika=len1+len2;

    for(int l=0; l<len1;l++){

        temporary[l]=encodedpakiet[l];

    }

    for(int l=len1; l<len1+len2;l++){

        temporary[l]=degré[l-len1];

    }

}

void combine2(int len2,int len1){ //funkcja laczaca symbol kodu i indeksy

    delete[] encodedpakiet;

    len1 += ((int)degré[0] +1);

}

```

```

cout<<"dlugosc 3( zksorowany pakiet + indeksy ) "<<len1<<endl;
cout<<"dlugosc 4 ( suma kontrolna w charach ) "<<len2<<endl;
encodedpakiet = new unsigned char[len1 + len2];
licznika=len1+len2;
for(int l=0; l<len1;l++){
    encodedpakiet[l]=temporary[l];
}
for(int l=len1; l<len1+len2;l++){
    encodedpakiet[l]=sumach[l-len1];
}
}

void inttochar(int suma, int pakietIn){// funkcja konwertujaca sume kontrolna, np. 3245 na tablice char :
['3' '2' '4' '5']

int i=1;
int counterofch=1;//dlugosc sumy kontrolnej
do{
    i=i*10;
    if(i<suma)
        counterofch+=1;
}while(i<suma);

sumach = new unsigned char[counterofch];
cout<<"wartosci tablicy zawierajacej sume kontrolna: ";
for(int i=0;i<counterofch;i++){

    sumach[i]=suma%10;
    cout<<(int)sumach[i]<<" ";
    suma=suma/10;
}
cout<<endl;
combine2(counterofch, pakietIn);
}

void calcsum(int pakietIn){// funkcja obliczajaca sume kontrolna symbolu kodu jako int

```

```

int suma1=0;// suma kontrolna zakodowanego symbolu

for(int i=0; i<pakietln+degre[0]+1;i++){
    suma1+=temporary[i];
}

inttochar(suma1,pakietln);

}

void deletemem(){

    delete[] degré;//tabela char przechowujaca stopien oraz indeksy "sasiadow" symbolu kodu
    delete[] encodedpakiet;
    delete[] temporary;//pakiet sluzacy do podmieniania wartosci encoded packiet
    delete[] sumach;

}

void deleteinputs(){

    for (int k = 0; k < liczbaPakietow; k++) {
        delete[] inputs[k];
    }
    delete[] inputs;
}

int main()

{
    int pakietlenght;

    cout<<"podaj dlugosc pakietu"<<endl;
    cin>>pakietlenght;
    wczytajpakiety(pakietlenght);

    cout<<"podaj prawdopodobienstwo nie powodzenia sie dekodowania w procesie LT"<<endl;cout<<
    Gdy wypisze tablice prawdopodobienstw, nacisnij znak z klawiatury a pozniej enter aby
    kontynuowac"<<endl;

    cin>> D;

    cout<<"podaj stala C wieksza od zera, najlepiej rowna 1: ";
    cin>>c;cout<<endl;
}

```

```

tablicasoliton(liczbaPakietow);

tablica_robust_soliton(liczbaPakietow,calculateR(liczbaPakietow));

float znakkk;

cin>>znakkk;

ofstream plik("example.txt", std::ios::out | std::ios::binary| std::ios::trunc ); // w pliku example.txt
znadowac sie beda symbole kodu

for(int i=0; i<(liczbaPakietow*5);i++){                                //////////////TU USTAWIAMY ILE SYMBOLI
KODU CHCEMY WYGENEROWAC

    cout<<"======"<<endl;

    degree(pakietlenght,(time(0)+i));

    combine(pakietlenght);

    calcsum(pakietlenght);

    cout<<"wypisuje wynikowy pakiet ";

    for(int i=0; i<liczniaka; i++){//!!!!!!!!

        plik<<encodedpakiet[i];

        cout<<(i+1)<<". "<<encodedpakiet[i]<< " <<(int)encodedpakiet[i]<< " ;

    }cout<<endl;plik<<".";

    deletemem();}

    deleteinputs();

    plik.close();

    cout<<"liczba jedynek wynosi: "<<licznikstopnia1<<endl;

    cout<<"liczba operacji wynosi: "<<licznikcalosciowy<<endl;

    return 0;
}

```

8. Kod programu odczytującego (podgląd zakodowanych danych)

Uwagi: program odczytuje znaki z pliku example.txt jako zmienne char(a nie unsigned char)-znaki , których wartości liczbowe są większe niż 127 są wyświetlane jako liczby ujemne liczbowo, nie zmienia to jednak poprawności danych zakodowanych.(kompilator odmawiał użycia unsigned charów w tym przypadku, z jakiejś racji...).

Przy zmianach w kodowaniu plików tekstowych, np. z utf8 na utf16LE i z powrotem na utf8 następuje utrata danych wejściowych. Najlepiej tego nie zmieniać.

PROGRAM WYSWIETLAJĄCY ZNAKI I ICH NUMERY Z example.txt:

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <algorithm>

using namespace std;

int main()
{
    ifstream c("example.txt", ios::in | ios::binary);
    if (!c) {cout << "Nie udalo sie otworzyc pliku!";}
    cout << "Hello world!" << endl;
    int liczbaznakow=0;
    char znak=0;
    while (c.get(znak)) {
        cout<<zak<<" "<<(int)zak<<endl;
        liczbaznakow++;
    }
    cout<< liczbaznakow<<endl;
    return 0;
}
```