# Praca pisemna

# Algorytm LZW – Ćwiczenia nr 4

**Opracowane przez**

Mikołaj Kmiećkowiak, nr albumu 59153, grupa ćwiczeniowa nr 1, student zaoczny na uczelni Vistula

//klasa modelująca słownik inicjacyjny

class MK\_59153\_InitialDictionary

{

//deklaracja pola prywatnego

string MK\_59153\_\_source;

//właściwość ustawiająca nam wartość powyższego pola prywatnego (publiczna bo musi być do niej

jakiś dostęp)

public string MK\_59153\_SetSource

{

//setter, bo tylko ją ustawiamy

set { MK\_59153\_\_source = value; }

}

//metoda zwracająca słownik wstępny

public List<string> MK\_59153\_MakeInitialDicrionary()

{

//utworzenie zmiennej źródło i przypisanie jej wartości naszego źródła oraz zadeklarowanie

następnego znaku

string MK\_59153\_source = MK\_59153\_\_source;

string MK\_59153\_nextChar;

//zainicjowanie listy, która będzie zwracana po wykonaniu metody

List<string> MK\_59153\_initialDictionary = new List<string>();

//pętla, która będzie się powtarzała tak długo, jak źródło nie jest puste

//bierze pierwszy znak, dodaje go do słownika, a następnie z całego źródła usuwa dany znak

while(MK\_59153\_source.Length != 0)

{

MK\_59153\_nextChar = MK\_59153\_source.Substring(0,1);

MK\_59153\_initialDictionary.Add(MK\_59153\_nextChar);

MK\_59153\_source = MK\_59153\_source.Replace(MK\_59153\_nextChar, "");

}

//zwraca wstępny słownik

return MK\_59153\_initialDictionary;

}

}

Klasę do inicjowania wstępnego słownika napisałem inspirując się tym co Pan Piotr zademonstrował na zajęciach. Zmieniłem jednak pętlę w taki sposób, żeby nie usuwała każdego znaku z kodu źródłowego pojedynczo, tylko żeby usuwała wszystkie wystąpienia danego znaku w całym kodzie źródłowym. Pozwoliło mi to na pozbycie się if statementu i zredukowanie ilości wykonywanych poleceń.

W klasie znajduje się deklaracja pola prywatnego, właściwość przypisująca mu wartość i metoda, która wykorzystuje wcześniej wspomnianą zmienną. Metoda zwraca słownik wstępny w postaci listy stringów.

//klasa wykonująca kompresję

class MK\_59153\_Compression

{

//deklaracja pól prywatnych

string MK\_59153\_\_source;

List<string> MK\_59153\_\_initialDictionary;

//właściwości ustawiające nam wartości powyższych pól prywatnych (publiczne, bo musi być do nich

jakiś dostęp)

public string MK\_59153\_SetSource

{

//setter, bo tylko ją ustawiamy

set { MK\_59153\_\_source = value; }

}

public List<string> MK\_59153\_SetInitialDictionary

{

//setter, bo tylko ją ustawiamy

set { MK\_59153\_\_initialDictionary = value; }

}

//metoda zwracająca kod kompresji

public List<int> MK\_59153\_GetCompressedList()

{

//utworzenie zmiennych

string MK\_59153\_multipleCharacters = "";

string MK\_59153\_moreByOneCharacters;

string MK\_59153\_nextCharacter;

string MK\_59153\_source = MK\_59153\_\_source;

//utworzenie słownika i przypisanie do niego słownika wstępnego

//byłbym wdzięczny za wyjaśnienie na zajęciach dlaczego

"List<string> dictionary = \_initialDictionary;" powoduje, że obie zmienne się wiążą i

//przy zmienieniu dictionary zmienia się również \_initialDictionary

// z góry dziękuję

List<string> MK\_59153\_dictionary = MK\_59153\_\_initialDictionary.ToList();

//utworzenie listy kodu kompresji

List<int> MK\_59153\_compressedList = new List<int>();

//pętla, która będzie się wykonywała dopóki nie wyczerpiemy całego źródła

while (MK\_59153\_source.Length > 0)

{

//przypisanie wartości następnego znaku

MK\_59153\_nextCharacter = MK\_59153\_source.Substring(0, 1);

//multipleCharacters jest to zmienna której idex znajdujący się w słowniku będzie

przypisywana do kod kompresji

//moreByOneCharacters jest to multipleCharacters ale o jeden znak więcej

(+ nextCharacter) i są to znaki dopisywane do słownika

MK\_59153\_moreByOneCharacters = MK\_59153\_multipleCharacters + MK\_59153\_nextCharacter;

//jeżeli w słowniku znajduje się większa ilość znaków to zwiększymy zwiększymy sprawdzane

znaki o kolejny znak i sprawdzimy ponownie

if (MK\_59153\_dictionary.Contains(MK\_59153\_moreByOneCharacters))

{

MK\_59153\_multipleCharacters = MK\_59153\_moreByOneCharacters;

}

//jeżeli natomiast w słowniku nie znajduje się ta większa ilość znaków to dodamy go do

słownika a

//do kodu kompresji dopiszemy tą mniejszą grupę znaków

//dodatkowo zamienimy tą mniejszą grupę znaków na pojedynczy znak który nie został

dopisany do słownika żeby cała pętla powtórzyła się pomyślnie

else

{

MK\_59153\_compressedList.Add(MK\_59153\_dictionary.IndexOf(MK\_59153\_multipleCharacters));

MK\_59153\_dictionary.Add(MK\_59153\_moreByOneCharacters);

MK\_59153\_multipleCharacters = MK\_59153\_nextCharacter;

}

//skracanie źródła żeby przy kolejnym powtórzeniu pętli nextCharacter był faktycznie

następnym znakiem

MK\_59153\_source = MK\_59153\_source.Remove(0, 1);

}

// dopisanie ostatniego znaku z ciągu

MK\_59153\_compressedList.Add(MK\_59153\_dictionary.IndexOf(MK\_59153\_multipleCharacters));

//zwraca kod kompresji

return MK\_59153\_compressedList;

}

}

W klasie kompresji standardowo znajduje się deklaracja pól prywatnych, właściwości przypisujące im wartości i metoda, która wykorzystuje wcześniej wspomniane zmienne. Ta metoda zwraca listę intów. Metodę napisałem sugerując się jedną z odpowiedzi na <https://stackoverflow.com/questions/8713850/lzw-data-compression> [Zdjęcie 1 str. 9].

Graphical user interface, text, application, email

Description automatically generated

*Zdjęcie 1 – część metody którą się sugerowałem*

*Źródło – https://stackoverflow.com/questions/8713850/lzw-data-compression*

W mojej metodzie wykorzystuję 4 zmienne: źródło, następny znak źródła, grupa znaków (zmienna której indeks w słowniku będzie dodawana do kodu kompresji) oraz grupa znaków, tylko że o dodatkowym, następnym znaku (zmienna która będzie dodawana do słownika).

Metoda działa w bardzo prosty sposób. Na początku bierze pierwszy znak ze źródła, sumuje ten znak razem z grupą znaków i przypisuje wartość do grupy znaków z dodatkowym znakiem. Sprawdza, czy grupa znaków z dodatkowym znakiem znajduje się w słowniku i na tej podstawie decyduje czy zaczerpnąć ze słownika kolejny znak ze źródła czy dodać odpowiednio kod do kodu kompresji i dodatkowe słowo w słowniku.

Uzyskany kod kompresji: 0 1 2 0 0 2 1 10 6 1 3 4 14 3 9 2 3 13 9 17 8 5 23 17 14 15 3 14 1 24 7 0 20 17 5 19 30 18 4 9 35 7 10 11 43 45 13 28 6 2 6 31 8 8 19 19 56 39 7 12 30 20 21 12 35 10 56 38 25 34 35 53 75 5 13 62 79 11 57 32 67 20 4 32 30 10 40 27 87 39 16 39 32 25 36 57 28 25 58 50 19

class MK\_59153\_Decompression

{

//deklaracja pól prywatnych

List<string> MK\_59153\_\_initialDictionary;

List<int> MK\_59153\_\_compressedList;

//właściwości ustawiające nam wartości powyższych pól prywatnych (publiczne, bo musi być do nich jakiś dostęp)

public List<int> MK\_59153\_SetCompressedList

{

//setter, bo tylko ją ustawiamy

set { MK\_59153\_\_compressedList = value; }

}

public List<string> MK\_59153\_SetInitialDictionary

{

//setter, bo tylko ją ustawiamy

set { MK\_59153\_\_initialDictionary = value; }

}

//metoda zwracająca ciąg zdekompresowany

public string MK\_59153\_GetDecompressedString()

{

//utworzenie zmiennych

string MK\_59153\_decompressedString = "";

int MK\_59153\_i = 0;

//utworzenie list i przypisanie im wartości

List<int> MK\_59153\_compressedList = MK\_59153\_\_compressedList;

List<string> MK\_59153\_dictionary = MK\_59153\_\_initialDictionary.ToList();

Console.ForegroundColor = ConsoleColor.Red;

//pętala wykonująca się dla każdego kodu w kodzie kompresji

while (MK\_59153\_i != MK\_59153\_compressedList.Count)

{

//dodawanie do zdekompresowanego stringa odpowiednich znaków

MK\_59153\_decompressedString = MK\_59153\_decompressedString +

MK\_59153\_dictionary[MK\_59153\_compressedList[MK\_59153\_i]];

//obsługiwanie wyjątku, w którym algorytm odwołuje się do nieistniejącego

jeszcze indeksu w słowniku

try

{

//dodawanie do słownika nowych elementów

MK\_59153\_dictionary.Add**(**MK\_59153\_dictionary**[**MK\_59153\_compressedList[MK\_59153\_i]**]** +

MK\_59153\_dictionary**[**MK\_59153\_compressedList[MK\_59153\_i + 1]**]**.Substring(0, 1)**)**;

}

catch (ArgumentOutOfRangeException MK\_59153\_ex)

{

//po przechwyceniu błędu w konsoli pokazuje się informacja, na której

pozycji skompresowanej listy

//algorytm odwołał się do jeszcze nieistniejącego indeksu

Console.WriteLine(">Argument out of range exception occurred at " +

MK\_59153\_i + " index of compressed list");

//dodawanie do słownika nowych elementów, gdy algorytm odwołuje się do

nieistniejącego jeszcze indeksu w słowniku

MK\_59153\_dictionary.Add**(**MK\_59153\_dictionary**[**MK\_59153\_compressedList[MK\_59153\_i]**]** +

MK\_59153\_dictionary**[**MK\_59153\_compressedList[MK\_59153\_i]**]**.Substring(0, 1)**)**;

}

//iteracja

MK\_59153\_i++;

}

Console.ResetColor();

//zwracanie ciągu zdekompresowanego

return MK\_59153\_decompressedString;

}

}

W klasie dekompresji standardowo znajduje się deklaracja pól prywatnych, właściwości przypisujące im wartości i metoda, która wykorzystuje wcześniej wspomniane zmienne. Ta metoda zwraca stringa zdekompresowanego ciągu. Metodę napisałem w pełni samodzielnie. Do jej działania wykorzystuję 2 zmienne (string jako wynik dekompresji oraz iterator) i 2 listy (słownik wstępny oraz kod kompresji). Dekompresja również nie jest skomplikowana. Używam pętli która przechodzi przez każdy indeks kodu kompresji. W pętli dodaję rozkodowany znak do wyniku dekompresji, a następnie dopisuję do słownika nowe słowo, które uzyskuję dzięki połączeniu dopiero co odczytanego znaku i pierwszej litery następnego odczytanego znaku. Oczywiście znajdują się przypadki, w których ten następny odczytany znak jeszcze nie znajduje się w słowniku (w takich przypadkach oznacza to, że właśnie go tworzymy), ale to nie stanowi żadnego problemu, bo w takim przypadku wiemy co będzie jego pierwszą literą (pierwszą literą będzie pierwsza litera słowa, przy pomocy której go tworzymy). Z tego powodu wykorzystuję try catch, gdzie obsługuję to zdarzenie.

class MK\_59153\_Program

{

//deklaracja kodu Źródłowego jako zmienna statyczna

static string MK\_59153\_kodŹródłowy;

static void Main(string[] args)

{

//interfejs użytkownika

Console.WriteLine("Wpisz własny ciąg znaków do kompresji i dekompresji i naciśnij ENTER");

Console.WriteLine("Lub nic nie wpisuj i naciśnij ENTER (domyślny ciąg zostanie

wykorzystany)");

Console.WriteLine();

Console.ForegroundColor = ConsoleColor.White;

string MK\_59153\_wiadomość = "Podany ciąg: ";

Console.Write(MK\_59153\_wiadomość);

Console.BackgroundColor = ConsoleColor.Gray;

Console.ForegroundColor = ConsoleColor.Black;

MK\_59153\_kodŹródłowy = Console.ReadLine();

//gdy użytkownik nie poda żadnego ciągu, domyślny ciąg zostanie wykorzystany

if (MK\_59153\_kodŹródłowy == "")

{

MK\_59153\_kodŹródłowy = "DBCDDCBBBCDBADBADACBCABACBACDCBCACDACADDBAAADBDCBDDDABACBCCAAACBCDBCBDADDBBBBCCBDDDBBAADDCDCCDADBDCDCCACADCDCAADDCDBAAABBACCDBDABBDCDBCCBCADDDDACCCCCBCBADDCDDCDBBCDCCBDCDBDABDBBDAABBAACACABDACAAADACAABDBCAABADCCADDBCACACBAACA";

Console.SetCursorPosition(MK\_59153\_wiadomość.Length, 3);

Console.WriteLine(MK\_59153\_kodŹródłowy);

}

Console.ResetColor();

//powołanie do życia obiektów

MK\_59153\_InitialDictionary MK\_59153\_initialDictionary = new MK\_59153\_InitialDictionary();

MK\_59153\_Compression MK\_59153\_compression = new MK\_59153\_Compression();

MK\_59153\_Decompression MK\_59153\_decompression = new MK\_59153\_Decompression();

//wywoływanie właściwości ustawiającej i przypisanie jemu wartości

MK\_59153\_initialDictionary.MK\_59153\_SetSource = MK\_59153\_kodŹródłowy;

//utworzenie listy i przypisanie do niej wartości używając metody

List<string> MK\_59153\_initialDictionaryList =

MK\_59153\_initialDictionary.MK\_59153\_MakeInitialDicrionary();

//wywoływanie właściwości ustawiających i przypisanie im wartości

MK\_59153\_compression.MK\_59153\_SetSource = MK\_59153\_kodŹródłowy;

MK\_59153\_compression.MK\_59153\_SetInitialDictionary = MK\_59153\_initialDictionaryList;

//utworzenie listy i przypisanie do niej wartości używając metody

List<int> MK\_59153\_compressedList = MK\_59153\_compression.MK\_59153\_GetCompressedList();

Console.WriteLine();

Console.WriteLine("Kod kompresji:");

//wypisanie do konsoli całego kodu kompresji

foreach (int MK\_59153\_numbers in MK\_59153\_compressedList)

{

Console.Write("\x1B[4m" + MK\_59153\_numbers + "\x1B[0m" + " ");

}

Console.WriteLine();

Console.WriteLine();

//wywoływanie właściwości ustawiających i przypisanie im wartości

MK\_59153\_decompression.MK\_59153\_SetInitialDictionary = MK\_59153\_initialDictionaryList;

MK\_59153\_decompression.MK\_59153\_SetCompressedList = MK\_59153\_compressedList;

//utworzenie stringa i przypisanie do niej wartości używając metody

string MK\_59153\_decompressedString = MK\_59153\_decompression.MK\_59153\_GetDecompressedString();

Console.WriteLine();

Console.WriteLine("Ciąg po dekompresji:");

//wyświetlenie ciągu po dekompresji

Console.WriteLine(MK\_59153\_decompressedString);

Console.WriteLine();

//sprawdzenie, czy ciąg po dekompresji jest identyczny jak ten przed kompresją

//i wyświetlenie użytkownikowi odpowiedniej wiadomości

if (MK\_59153\_decompressedString == MK\_59153\_kodŹródłowy)

{

Console.ForegroundColor = ConsoleColor.Black;

Console.BackgroundColor = ConsoleColor.DarkGreen;

Console.WriteLine("Sukces! Ciąg przed kompresją jest identyczny jak po jego

dekompresji!");

Console.ResetColor();

}

else

{

Console.BackgroundColor = ConsoleColor.Red;

Console.WriteLine("Coś poszło nie tak :/");

Console.ResetColor();

}

Console.ReadLine();

}

}

W metodzie main zrobiłem według mnie dość czytelny interfejs użytkownika. Dałem mu możliwość wyboru, czy chce umieścić własny ciąg, czy też woli wybrać domyślny. Gdy użytkownik będzie wpisywał swój ciąg, znaki które stawia, lub ciąg który wklei będą się wyraźnie wyróżniały od reszty konsoli. Po naciśnięciu ENTER aplikacja utworzy słownik wstępny, dokona kompresji (kod kompresji zostanie wyświetlony) oraz dekompresji (również zostanie wyświetlany). Całość jest podsumowana komentarzem, czy wszystko się powiodło czy też nie.

Text

Description automatically generated

*Zdjęcie 2 – konsola po wykonaniu działania*

*Źródło – opracowanie własne*