Akademia Nauk Stosowanych w Nowym Sączu Teoretyczne i technologiczne podstawy multimediów – laboratorium			
Temat: Algorytm LZ77			**L2**
Nazwisko i imię: Szczepanek Piotr		**Ocena sprawozdania**	**Zaliczenie:**
Data wykonania ćwiczenia: 15.11.2022	**Grupa: L2**		

LZ77 (Lempel-Ziv 77) – metoda strumieniowej słownikowej kompresji danych.

Metoda wykorzystuje fakt iż w danych przeznaczonych do zakodowania powtarzają się pewne ciągi bajtów/znaków (zależy jaką formą danych wejściowych się posługujemy).

lz77

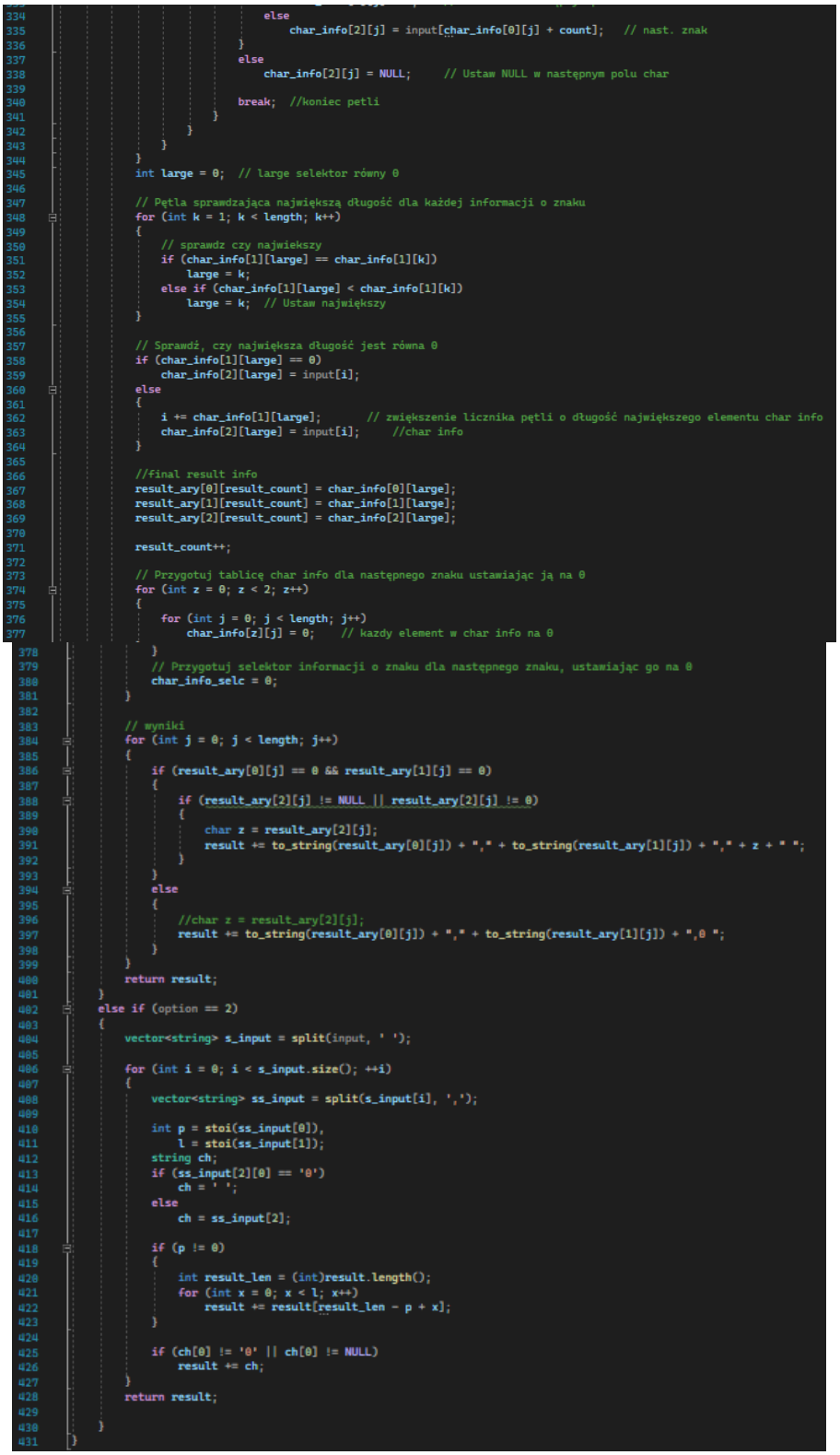
```cpp
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include "windows.h"

using namespace std;

struct Node {
    int index;
    string data;
    Node* next;
};

void st_Node(Node* head, int index, string data) {
    head->index = index;
    head->data = data;
    head->next = NULL;
}

void insert_Node(Node* head, int index, string data) {
    Node* new_Node = new Node;
    new_Node->index = index;
    new_Node->data = data;
    new_Node->next = NULL;

    Node* curr = head;
    while (curr != NULL)
    {
        if (curr->next == NULL)
        {
            curr->next = new_Node;
            return;
        }
        curr = curr->next;
    }
}
```

```cpp
Node* search_Node(Node* head, string data)
{
    Node* curr = head;
    while (curr != NULL)
    {
        if (data.compare(curr->data) == 0)
            return curr;
        else
            curr = curr->next;
    }
    return NULL;
}

Node* search_Node(Node* head, int index)
{
    Node* curr = head;
    while (curr != NULL)
    {
        if (index == curr->index)
            return curr;
        else
            curr = curr->next;
    }
    return NULL;
}

bool delete_Node(Node* head, Node* to_delete) {
    if (to_delete == NULL)
        return false;
    else if (to_delete == head)
    {
        head = to_delete->next;
        delete to_delete;
        return true;
    }
    else {
```

```
        Node* curr = head;
        while (curr)
        {
            if (curr->next == to_delete)
            {
                curr->next = to_delete->next;
                delete to_delete;
                return true;
            }
            curr = curr->next;
        }
        return false;
    }
}

vector <string> split(string str, char delimiter) {
    vector<string> internal;
    stringstream ss(str); // Zamiana string na stream
    string tok;

    while (getline(ss, tok, delimiter)) {
        internal.push_back(tok);
    }

    return internal;
}

 string LZ77(string input, int option);


int main()
{
    string input, result, method_text;
    int method, option, option2;

    string intro = R"(
    Szczepanek Piotr, Informatyka 3 rok, Algorytm LZ-77
)";
    cout << intro;
main_menu:
    string main_menu = R"(
--------------------------------------------------------------------------------
Generujemy kompresje i dekompresje przy uzyciu metody LZ77 :
  1- LZ-77

Wybierz 1, aby potwierdzic wybor metody
 > )"; cout << main_menu;

    cin >> method;

    if (method == 1)
        method_text = "LZ-77";

    else
    {
        system("cls");
        cout << intro;
        goto main_menu;
    }

method_menu:
    system("cls");
    cout << intro;

    string main_menu_2 = R"(
--------------------------------------------------------------------------------
Generujemy kompresje i dekompresje przy uzyciu metody )" + method_text + R"(:
  1- Kompresja
  2- Dekompresja
  0- Powrot do menu
Wybierz jedna z powyzszych opcji.
 > )"; cout << main_menu_2;

    cin >> option;
```

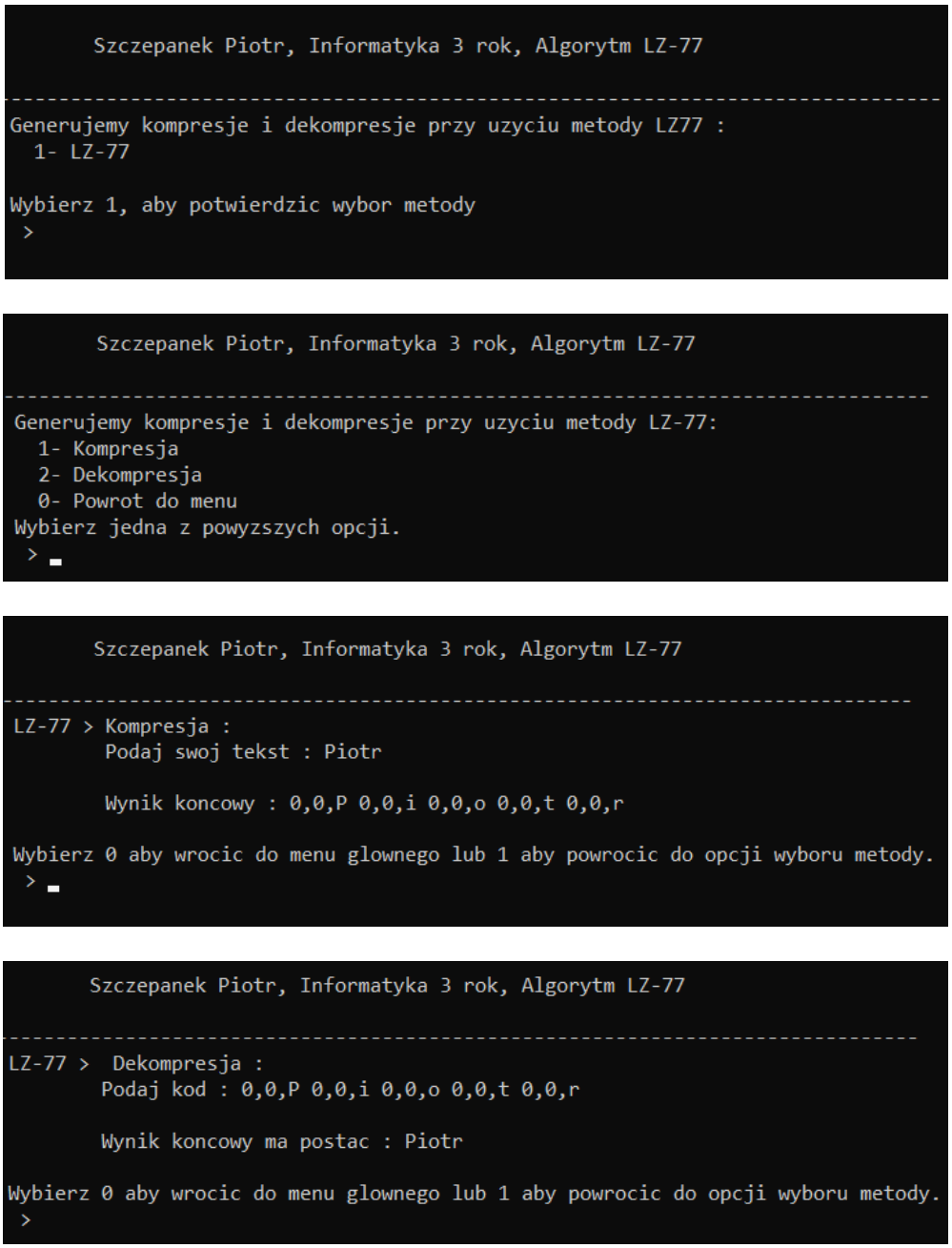
```
    if (option == 1)
    {
        system("cls");
        cout << intro;

        string lz77_Compression = R"(
--------------------------------------------------------------------------
)" + method_text + R"( > Kompresja :)";
        cout << lz77_Compression << endl;

        cout << "\t Podaj swoj tekst : ";
        cin.ignore();
        getline(cin, input);
        if (method == 1)
            result = LZ77(input, 1);

        else
        {
            system("cls");
            cout << intro;
            goto main_menu;
        }

        cout << "\n\t Wynik koncowy : " << result << endl;

    back_1:
        cout << "\n Wybierz 0 aby wrocic do menu glownego lub 1 aby powrocic do opcji wyboru metody. \n  > ";
        cin >> option2;

        if (option2 == 0)
        {
            system("cls");
            cout << intro;
            goto main_menu;
        }
        else if (option2 == 1)
            goto method_menu;
        else
            goto back_1;
    }
    else if (option == 2)
    {
        system("cls");
        cout << intro;

        string lz77_Compression = R"(
--------------------------------------------------------------------------
LZ-77 >  Dekompresja :)";
        cout << lz77_Compression << endl;
        //cout << "Uwaga : Wpisz 0 dla znakow NULL";
        cout << "\t Podaj kod : ";
        cin.ignore();
        getline(cin, input);
        if (method == 1)
            result = LZ77(input, 2);
        else
            main_menu;

        cout << "\n\t Wynik koncowy ma postac : " << result << endl;

    back_2:
        cout << "\n Wybierz 0 aby wrocic do menu glownego lub 1 aby powrocic do opcji wyboru metody. \n  > ";
        cin >> option2;

        if (option2 == 0)
        {
            system("cls");
            cout << intro;
            goto main_menu;
        }
        else if (option2 == 1)
            goto method_menu;
        else
            goto back_2;
```

```
        }
        else if (option == 0)
        {
            system("cls");
            cout << intro;
            goto main_menu;
        }
        else
            goto method_menu;


        cin.get();
        cin.ignore();
        return 0;
    }

    string LZ77(string input, int option)
    {
        // Inicjalizacja zmiennych
        string result;
        int length, char_info_selc = 0;

        if (option == 1)
        {
        check_char:     // Wskaźnik sprawdzania długości
            length = (int)input.length();   // Obliczanie długości łańcucha wejściowego
            // Sprawdź, czy długość zmiennej wejściowej jest mniejsza niż 3
            if (length <= 2)
            {
                cout << "Podaj co najmniej 3 znaki \n";
                getline(cin, input);
                goto check_char;
            }
            // Zadeklaruj arry dla wyniku końcowego o nazwie 'result_ary'
            int** result_ary = new int* [3];
            for (int i = 0; i < length; ++i)
                result_ary[i] = new int[length];
        // Ustawić wartość elementów result_ary na 0, aby zapobiec poprzednim wartościom
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < length; j++)
                result_ary[i][j] = 0;
        }

        // Zadeklaruj arry do przechowywania każdej informacji o znaku w łańcuchu wejściowym o nazwie 'char_info'
        int** char_info = new int* [3];
        for (int i = 0; i < length; ++i)
            char_info[i] = new int[length];
        // Ustaw wartość elementów char_info na 0, aby zapobiec poprzednim wartościom
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < length; j++)
                char_info[i][j] = 0;
        }

        // Ustaw pierwszy znak info na (0,0,'<first char>')
        result_ary[0][0] = 0;
        result_ary[1][0] = 0;
        result_ary[2][0] = input[0];

        int result_count = 1;

        // Pętla do wykonywania pewnych operacji na każdym znaku w łańcuchu wejściowym
        for (int i = 1; i < length; i++)
        {
            // Pętla sprawdzająca, czy znak wejściowy na pozycji i jest równy dowolnemu z
            // poprzednich znaków w wejściu i zapisuje tę informację w tablicy char_info
            for (int j = 0; j < i; j++)
            {
                // Sprawdzanie pozycji poprzedniego widoku elementu i
                if (input[i] == input[j])
                {
                    // Ustawianie wskaźnika położenia
                    char_info[0][char_info_selc] = i - j;

                    // Zwiększenie selektora tablicy char info o 1
                    char_info_selc++;
                }
            }

            // Pętla do sprawdzania długości dla każdej pozycji znaku
            for (int j = 0; j < length; j++)
            {
                // Sprawdź, czy aktualna pozycja tablicy char info nie jest równa 0
                if (char_info[0][j] != 0)
                {
                    // punkt poczatkowy
                    int start = i - char_info[0][j];

                    // Ustaw licznik, aby obliczyć długość dla tej pozycji znaku
                    int count = 1;

                    // Pętla sprawdzająca długość dla tej pozycji znaku
                    for (int k = 0; k < length; k++)
                    {
                        // Sprawdź następny element startu przez następny element wejścia
                        if (input[start + count] == input[i + count])
                            count++;    // Increase count by 1
                        else
                        {
                            char_info[1][j] = count;    // Zapisz wartość licznika w długości - length

                            // Sprawdź, czy ten znak wejściowy jest ostatnim znakiem
                            if (i != (length - 1))
                            {
                                // Zapisuje następny znak w char info
                                // Sprawdź, czy ta pozycja jest równa długości
                                if (char_info[0][j] + count == length)
                                    char_info[2][j] = 0;    // Ustaw 0 w następnym polu znaków
```



```
                    else
                        char_info[2][j] = input[char_info[0][j] + count];   // nast. znak
                }
                else
                    char_info[2][j] = NULL;     // Ustaw NULL w następnym polu char

                break;  //koniec petli
            }
        }
    }
}
int large = 0;  // large selektor równy 0

// Pętla sprawdzająca największą długość dla każdej informacji o znaku
for (int k = 1; k < length; k++)
{
    // sprawdz czy najwiekszy
    if (char_info[1][large] == char_info[1][k])
        large = k;
    else if (char_info[1][large] < char_info[1][k])
        large = k;  // Ustaw największy
}

// Sprawdź, czy największa długość jest równa 0
if (char_info[1][large] == 0)
    char_info[2][large] = input[i];
else
{
    i += char_info[1][large];       // zwiększenie licznika pętli o długość największego elementu char info
    char_info[2][large] = input[i];     //char info
}

//final result info
result_ary[0][result_count] = char_info[0][large];
result_ary[1][result_count] = char_info[1][large];
result_ary[2][result_count] = char_info[2][large];

result_count++;

// Przygotuj tablicę char info dla następnego znaku ustawiając ją na 0
for (int z = 0; z < 2; z++)
{
    for (int j = 0; j < length; j++)
        char_info[z][j] = 0;    // kazdy element w char info na 0
}
// Przygotuj selektor informacji o znaku dla następnego znaku, ustawiając go na 0
char_info_selc = 0;
}

// wyniki
for (int j = 0; j < length; j++)
{
    if (result_ary[0][j] == 0 && result_ary[1][j] == 0)
    {
        if (result_ary[2][j] != NULL || result_ary[2][j] != 0)
        {
            char z = result_ary[2][j];
            result += to_string(result_ary[0][j]) + "," + to_string(result_ary[1][j]) + "," + z + " ";
        }
    }
    else
    {
        //char z = result_ary[2][j];
        result += to_string(result_ary[0][j]) + "," + to_string(result_ary[1][j]) + ",0 ";
    }
}
return result;
}
else if (option == 2)
{
    vector<string> s_input = split(input, ' ');

    for (int i = 0; i < s_input.size(); ++i)
    {
        vector<string> ss_input = split(s_input[i], ',');

        int p = stoi(ss_input[0]),
            l = stoi(ss_input[1]);
        string ch;
        if (ss_input[2][0] == '0')
            ch = " ";
        else
            ch = ss_input[2];

        if (p != 0)
        {
            int result_len = (int)result.length();
            for (int x = 0; x < l; x++)
                result += result[result_len - p + x];
        }

        if (ch[0] != '0' || ch[0] != NULL)
            result += ch;
    }
    return result;

}
}
```

Rysunek 1 Kod programu.



Rysunek 2 Efekt działania programu z Rysunku 1.

Zasada działania

W LZ77 zapamiętywana jest w słowniku pewna liczba ostatnio kodowanych danych – przeciętnie kilka do kilkudziesięciu kilobajtów. Jeśli jakiś ciąg powtórzy się, to zostanie zastąpiony przez liczby określające jego pozycję w słowniku oraz długość ciągu; do zapamiętania tych dwóch liczb trzeba przeznaczyć zazwyczaj o wiele mniej bitów niż do zapamiętania zastępowanego ciągu.

Metoda LZ77 zakłada, że ciągi powtarzają się w miarę często, tzn. na tyle często, żeby wcześniejsze wystąpienia można było zlokalizować w słowniku – ciągi powtarzające się zbyt rzadko nie są brane pod uwagę.

Bardzo dużą zaletą kodowania LZ77 jest to, że słownika nie trzeba zapamiętywać i przesyłać wraz z komunikatem – zawartość słownika będzie na bieżąco odtwarzana przez dekoder.

Algorytm kompresji jest bardziej złożony i trudniejszy w realizacji niż algorytm dekompresji. W metodzie LZ77 można wpływać na prędkość kompresji oraz zapotrzebowania pamięciowe, regulując parametry kodera (rozmiar słownika i bufora kodowania).