

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).

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <sstream>
5  #include "windows.h"
6
7  using namespace std;
8
9  struct Node {
10     int index;
11     string data;
12     Node* next;
13 };
14
15 void st_Node(Node* head, int index, string data) {
16     head->index = index;
17     head->data = data;
18     head->next = NULL;
19 }
20
21 void insert_Node(Node* head, int index, string data) {
22     Node* new_Node = new Node;
23     new_Node->index = index;
24     new_Node->data = data;
25     new_Node->next = NULL;
26
27     Node* curr = head;
28     while (curr != NULL)
29     {
30         if (curr->next == NULL)
31         {
32             curr->next = new_Node;
33             return;
34         }
35         curr = curr->next;
36     }
37 }
38
39 Node* search_Node(Node* head, string data)
40 {
41     Node* curr = head;
42     while (curr != NULL)
43     {
44         if (data.compare(curr->data) == 0)
45             return curr;
46         else
47             curr = curr->next;
48     }
49     return NULL;
50 }
51
52 Node* search_Node(Node* head, int index)
53 {
54     Node* curr = head;
55     while (curr != NULL)
56     {
57         if (index == curr->index)
58             return curr;
59         else
60             curr = curr->next;
61     }
62     return NULL;
63 }
64
65 bool delete_Node(Node* head, Node* to_delete) {
66     if (to_delete == NULL)
67         return false;
68     else if (to_delete == head)
69     {
70         head = to_delete->next;
71         delete to_delete;
72         return true;
73     }
74     else {

```

```

75     Node* curr = head;
76     while (curr)
77     {
78         if (curr->next == to_delete)
79         {
80             curr->next = to_delete->next;
81             delete to_delete;
82             return true;
83         }
84         curr = curr->next;
85     }
86     return false;
87 }
88
89 vector<string> split(string str, char delimiter) {
90     vector<string> internal;
91     stringstream ss(str); // Zamiana string na stream
92     string tok;
93     while (getline(ss, tok, delimiter)) {
94         internal.push_back(tok);
95     }
96     return internal;
97 }
98
99 string LZ77(string input, int option);
100
101 int main()
102 {
103     string input, result, method_text;
104     int method, option, option2;
105
106     string intro = R"(
107     Szczepanek Piotr, Informatyka 3 rok, Algorytm LZ-77
108 )";
109
110     cout << intro;
111     main_menu:
112     string main_menu = R"(
113
114     Generujemy kompresje i dekompresje przy uzyciu metody LZ77 :
115     1- LZ-77
116
117     Wybierz 1, aby potwierdzic wybor metody
118     > )"; cout << main_menu;
119
120     cin >> method;
121
122     if (method == 1)
123         method_text = "LZ-77";
124     else
125     {
126         system("cls");
127         cout << intro;
128         goto main_menu;
129     }
130
131     method_menu:
132     system("cls");
133     cout << intro;
134
135     string main_menu_2 = R"(
136
137     Generujemy kompresje i dekompresje przy uzyciu metody )" + method_text + R"( :
138     1- Kompresja
139     2- Dekompresja
140     0- Powrot do menu
141     Wybierz jedna z powyzzszych opcji.
142     > )"; cout << main_menu_2;
143
144     cin >> option;

```

```

150     if (option == 1)
151     {
152         system("cls");
153         cout << intro;
154
155         string lz77_Compression = R"(
156
157     )" + method_text + R"( > Kompresja :)";
158         cout << lz77_Compression << endl;
159
160         cout << "\t Podaj swoj tekst : ";
161         cin.ignore();
162         getline(cin, input);
163         if (method == 1)
164             result = LZ77(input, 1);
165
166         else
167         {
168             system("cls");
169             cout << intro;
170             goto main_menu;
171         }
172
173         cout << "\n\t Wynik koncowy : " << result << endl;
174
175         back_1:
176         cout << "\n Wybierz 0 aby wrocic do menu glownego lub 1 aby powrocic do opcji wyboru metody. \n > ";
177         cin >> option2;
178
179         if (option2 == 0)
180         {
181             system("cls");
182             cout << intro;
183             goto main_menu;
184         }
185         else if (option2 == 1)
186             goto method_menu;
187         else
188             goto back_1;
189     }
190     else if (option == 2)
191     {
192         system("cls");
193         cout << intro;
194
195         string lz77_Compression = R"(
196
197     LZ-77 > Dekompresja :)";
198         cout << lz77_Compression << endl;
199         //cout << "Uwaga : Wpisz 0 dla znakow NULL";
200         cout << "\t Podaj kod : ";
201         cin.ignore();
202         getline(cin, input);
203         if (method == 1)
204             result = LZ77(input, 2);
205         else
206             main_menu;
207
208         cout << "\n\t Wynik koncowy ma postac : " << result << endl;
209
210         back_2:
211         cout << "\n Wybierz 0 aby wrocic do menu glownego lub 1 aby powrocic do opcji wyboru metody. \n > ";
212         cin >> option2;
213
214         if (option2 == 0)
215         {
216             system("cls");
217             cout << intro;
218             goto main_menu;
219         }
220         else if (option2 == 1)
221             goto method_menu;
222         else
223             goto back_2;

```

```

224     }
225 }
226 else if (option == 0)
227 {
228     system("cls");
229     cout << intro;
230     goto main_menu;
231 }
232 else
233     goto method_menu;
234
235
236 cin.get();
237 cin.ignore();
238 return 0;
239 }
240
241 string LZ77(string input, int option)
242 {
243     // Inicjalizacja zmiennych
244     string result;
245     int length, char_info_selc = 0;
246
247     if (option == 1)
248     {
249         check_char: // Wskaźnik sprawdzania długości
250         length = (int)input.length(); // Obliczanie długości łańcucha wejściowego
251         // Sprawdź, czy długość zmiennej wejściowej jest mniejsza niż 3
252         if (length <= 2)
253         {
254             cout << "Podaj co najmniej 3 znaki \n";
255             getline(cin, input);
256             goto check_char;
257         }
258         // Zadeklaruj array dla wyniku końcowego o nazwie 'result_ary'
259         int** result_ary = new int* [3];
260         for (int i = 0; i < length; ++i)
261             result_ary[i] = new int[length];
262
263         // Ustaw wartość elementów result_ary na 0, aby zapobiec poprzednim wartościom
264         for (int i = 0; i < 3; i++)
265         {
266             for (int j = 0; j < length; j++)
267                 result_ary[i][j] = 0;
268         }
269
270         // Zadeklaruj array do przechowywania każdej informacji o znaku w łańcuchu wejściowym o nazwie 'char_info'
271         int** char_info = new int* [3];
272         for (int i = 0; i < length; ++i)
273             char_info[i] = new int[length];
274         // Ustaw wartość elementów char_info na 0, aby zapobiec poprzednim wartościom
275         for (int i = 0; i < 3; i++)
276         {
277             for (int j = 0; j < length; j++)
278                 char_info[i][j] = 0;
279         }
280
281         // Ustaw pierwszy znak info na (0,0,'<first char>')
282         result_ary[0][0] = 0;
283         result_ary[1][0] = 0;
284         result_ary[2][0] = input[0];
285
286         int result_count = 1;
287
288         // Pętla do wykonania pewnych operacji na każdym znaku w łańcuchu wejściowym
289         for (int i = 1; i < length; i++)
290         {
291             // Pętla sprawdzająca, czy znak wejściowy na pozycji i jest równy dowolnemu z
292             // poprzednich znaków w wejściu i zapisuje tę informację w tablicy char_info
293             for (int j = 0; j < i; j++)
294             {
295                 // Sprawdzanie pozycji poprzedniego widoku elementu i
296                 if (input[i] == input[j])
297                 {
298                     // Ustawianie wskaźnika położenia
299                     // Ustawianie wskaźnika położenia
300                     char_info[0][char_info_selc] = i - j;
301
302                     // Zwiększenie selektora tablicy char info o 1
303                     char_info_selc++;
304                 }
305             }
306
307             // Pętla do sprawdzania długości dla każdej pozycji znaku
308             for (int j = 0; j < length; j++)
309             {
310                 // Sprawdź, czy aktualna pozycja tablicy char info nie jest równa 0
311                 if (char_info[0][j] != 0)
312                 {
313                     // punkt początkowy
314                     int start = i - char_info[0][j];
315
316                     // Ustaw licznik, aby obliczyć długość dla tej pozycji znaku
317                     int count = 1;
318
319                     // Pętla sprawdzająca długość dla tej pozycji znaku
320                     for (int k = 0; k < length; k++)
321                     {
322                         // Sprawdź następny element startu przez następny element wejścia
323                         if (input[start + count] == input[i + count])
324                             count++; // Increase count by 1
325                         else
326                         {
327                             char_info[1][j] = count; // Zapisz wartość licznika w długości - length
328
329                             // Sprawdź, czy ten znak wejściowy jest ostatnim znakiem
330                             if (i != (length - 1))
331                             {
332                                 // Zapisuje następny znak w char info
333                                 // Sprawdź, czy ta pozycja jest równa długości
334                                 if (char_info[0][j] + count == length)
335                                     char_info[2][j] = 0; // Ustaw 0 w następnym polu znaków

```

```

334         else
335             char_info[2][j] = input[char_info[0][j] + count]; // nast. znak
336     }
337     else
338         char_info[2][j] = NULL; // Ustaw NULL w następnym polu char
339     break; //koniec petli
340 }
341 }
342 }
343 }
344
345 int large = 0; // large selektor równy 0
346
347 // Pętla sprawdzająca największą długość dla każdej informacji o znaku
348 for (int k = 1; k < length; k++)
349 {
350     // sprawdź czy największy
351     if (char_info[1][large] == char_info[1][k])
352         large = k;
353     else if (char_info[1][large] < char_info[1][k])
354         large = k; // Ustaw największy
355 }
356
357 // Sprawdź, czy największa długość jest równa 0
358 if (char_info[1][large] == 0)
359     char_info[2][large] = input[i];
360 else
361 {
362     i += char_info[1][large]; // zwiększenie licznika pętli o długość największego elementu char info
363     char_info[2][large] = input[i]; //char info
364 }
365
366 //final result info
367 result_ary[0][result_count] = char_info[0][large];
368 result_ary[1][result_count] = char_info[1][large];
369 result_ary[2][result_count] = char_info[2][large];
370
371 result_count++;
372
373 // Przygotuj tablicę char info dla następnego znaku ustawiając ją na 0
374 for (int z = 0; z < 2; z++)
375 {
376     for (int j = 0; j < length; j++)
377         char_info[z][j] = 0; // każdy element w char info na 0
378 }
379
380 // Przygotuj selektor informacji o znaku dla następnego znaku, ustawiając go na 0
381 char_info_selc = 0;
382 }
383
384 // wyniki
385 for (int j = 0; j < length; j++)
386 {
387     if (result_ary[0][j] == 0 && result_ary[1][j] == 0)
388     {
389         if (result_ary[2][j] != NULL || result_ary[2][j] != 0)
390         {
391             char z = result_ary[2][j];
392             result += to_string(result_ary[0][j]) + "," + to_string(result_ary[1][j]) + "," + z + " ";
393         }
394     }
395     else
396     {
397         //char z = result_ary[2][j];
398         result += to_string(result_ary[0][j]) + "," + to_string(result_ary[1][j]) + ",0 ";
399     }
400 }
401 return result;
402 }
403
404 else if (option == 2)
405 {
406     vector<string> s_input = split(input, ' ');
407
408     for (int i = 0; i < s_input.size(); ++i)
409     {
410         vector<string> ss_input = split(s_input[i], ',');
411
412         int p = stoi(ss_input[0]),
413             l = stoi(ss_input[1]);
414         string ch;
415         if (ss_input[2][0] == '0')
416             ch = ' ';
417         else
418             ch = ss_input[2];
419
420         if (p != 0)
421         {
422             int result_len = (int)result.length();
423             for (int x = 0; x < l; x++)
424                 result += result[result_len - p + x];
425         }
426
427         if (ch[0] != '0' || ch[0] != NULL)
428             result += ch;
429     }
430
431     return result;
432 }
433 }

```

Rysunek 1 Kod programu.

```

Szczepanek Piotr, Informatyka 3 rok, Algorytm LZ-77
-----
Generujemy kompresje i dekompresje przy uzyciu metody LZ77 :
1- LZ-77

Wybierz 1, aby potwierdzic wybor metody
>

```

```

Szczepanek Piotr, Informatyka 3 rok, Algorytm LZ-77
-----
Generujemy kompresje i dekompresje przy uzyciu metody LZ-77:
1- Kompresja
2- Dekompresja
0- Powrot do menu
Wybierz jedna z powyzzszych opcji.
> █

```

```

Szczepanek Piotr, Informatyka 3 rok, Algorytm LZ-77
-----
LZ-77 > Kompresja :
      Podaj swoj tekst : Piotr

      Wynik koncowy : 0,0,P 0,0,i 0,0,o 0,0,t 0,0,r

Wybierz 0 aby wrocic do menu glownego lub 1 aby powrocic do opcji wyboru metody.
> █

```

```

Szczepanek Piotr, Informatyka 3 rok, Algorytm LZ-77
-----
LZ-77 > Dekompresja :
      Podaj kod : 0,0,P 0,0,i 0,0,o 0,0,t 0,0,r

      Wynik koncowy ma postac : Piotr

Wybierz 0 aby wrocic do menu glownego lub 1 aby powrocic do opcji wyboru metody.
>

```

*Rysunek 2 Efekt dzialania programu z Rysunku 1.*

## Zasada dzialania

W LZ77 zapamietywana 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 przysyłać wraz z komunikatem – zawartość słownika będzie na bieżąco odtwarzana przez dekodery.

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).