Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет телекоммуникаций
и информатики»

Факультет ИВТ

Кафедра вычислительных систем

Курсовая работа

на тему «СЖАТИЕ ДАННЫХ»
ВАРИАНТ 4.3 Алгоритм Лемпела – Зива (Lempel – Ziv) LZ77

Выполнил:
студент гр. ИС-242
Игнатенко Г.Д.

Проверил:
старший преподаватель Кафедры ВС
Фульман В.О.

Новосибирск, 2023

Тема курсовой работы
Сжатие данных. Алгоритм Лемпела – Зива LZ77

Задание на курсовую работу
Реализовать программу lz77compress сжатия текстовых файлов на английском языке алгоритмом Зива-Лемпела. Сжатие осуществляется с аргументом командной строки -c (compress), а распаковка – с аргументом -d (decompress). Опция -o указывает имя выходного файла

$ lz77compress -c -o file.lz77 file.txt # сжатие file.txt в file.lz77
$ lz77compress -d -o file1.txt file.lz77 # распаковка file.lz77 в file1.txt

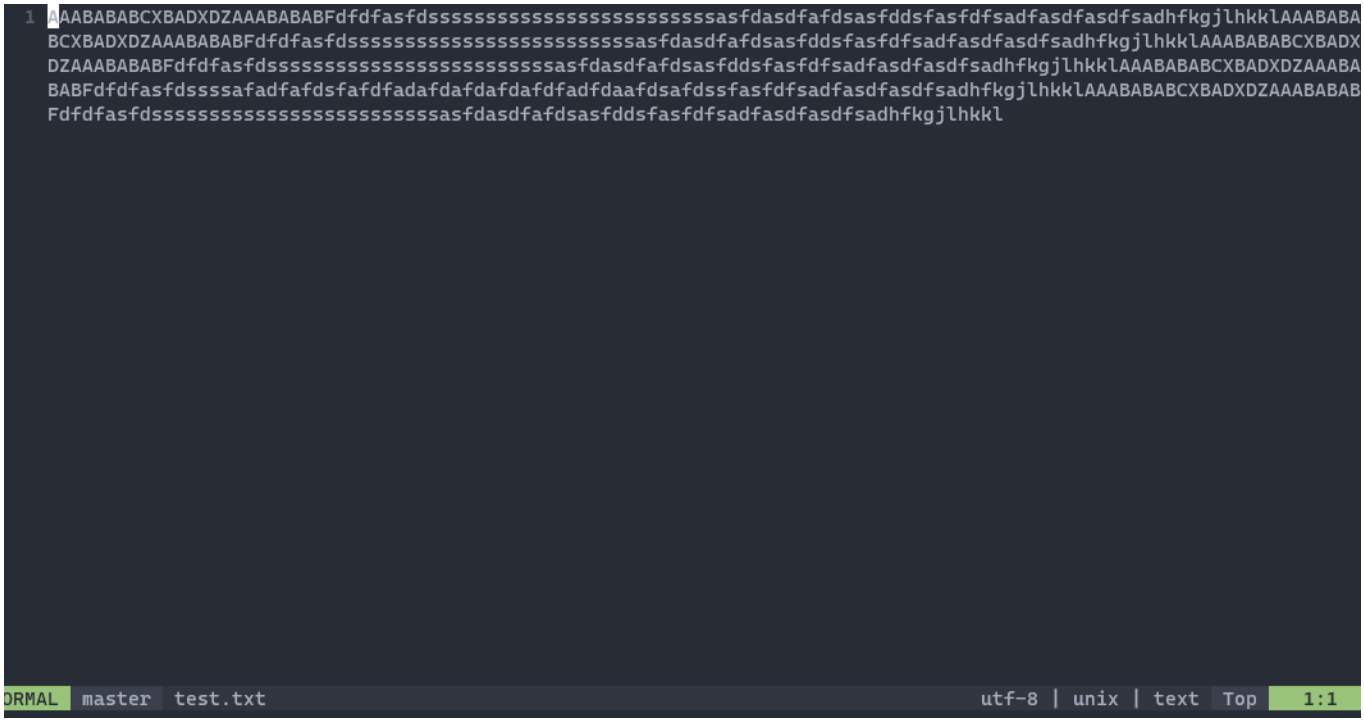
Критерии оценки
Оценка «хорошо»: реализован алгоритм сжатия, для записи кодов в файл используются структуры данных.
Оценка «отлично»: можно задать любой размер словаря и буфера, для формирования файлового элемента используется битовый массив (как описано в общей информации к разделу 4).

Указание к выполнению задания
LZ77 использует скользящее по сообщению окно. Метод кодирования согласно принципу скользящего окна учитывает уже ранее встречавшуюся информацию, то есть информацию, которая уже известна для кодировщика и декодировщика (второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на ее первое вхождение). Окно состоит из двух частей – словаря (большая часть) и буфера. Первая, большая по размеру, включает уже просмотренную часть сообщения. Вторая, меньшая по размеру, содержит еще незакодированные символы входного потока. Алгоритм пытается найти в словаре фрагмент, совпадающий с содержимым буфера. Алгоритм LZ77 выдает коды, состоящие из трех элементов: • смещение подстроки, совпадающей с началом содержимого буфера, относительно начала словаря; • длина этой подстроки; • первый символ буфера, следующий за подстрокой. В конце итерации алгоритм сдвигает окно на длину равную длине подстроки, обнаруженной в словаре.

Анализ задачи
1. Запуск программы производится с двумя аргументами: режим работы и название файла. В директорий input берется нужный файл.
2. Анализ аргументов на ошибки
3. Открытие файла, анализ входных данных, поиск совпадений.
4. Сжатие данных
5. Вывод данных в файл в папку output

Тестовые данные

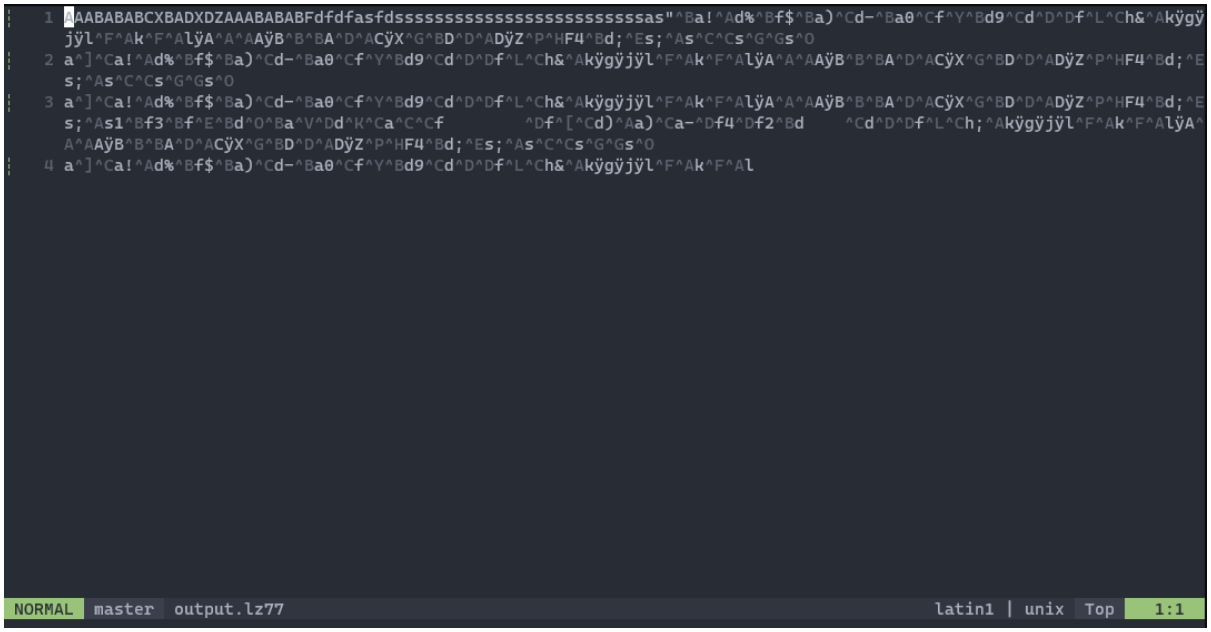
Входной файл



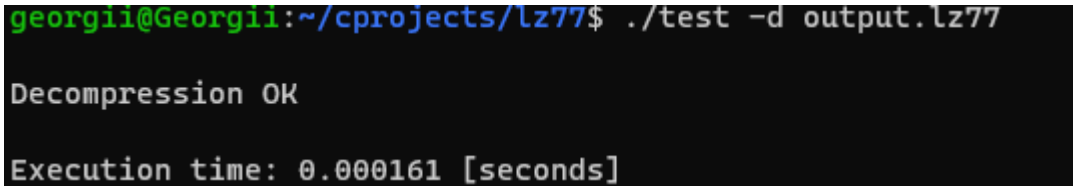

```
georgii@Georgii:~/cprojects/lz77$ ./test -c test.txt
Input file size: 539 bytes
Output file size: 485 bytes

Compression OK

Execution time: 0.001102 [seconds]
```

```
georgii@Georgii:~/cprojects/lz77$ cd output/
georgii@Georgii:~/cprojects/lz77/output$ ls
output.lz77
```

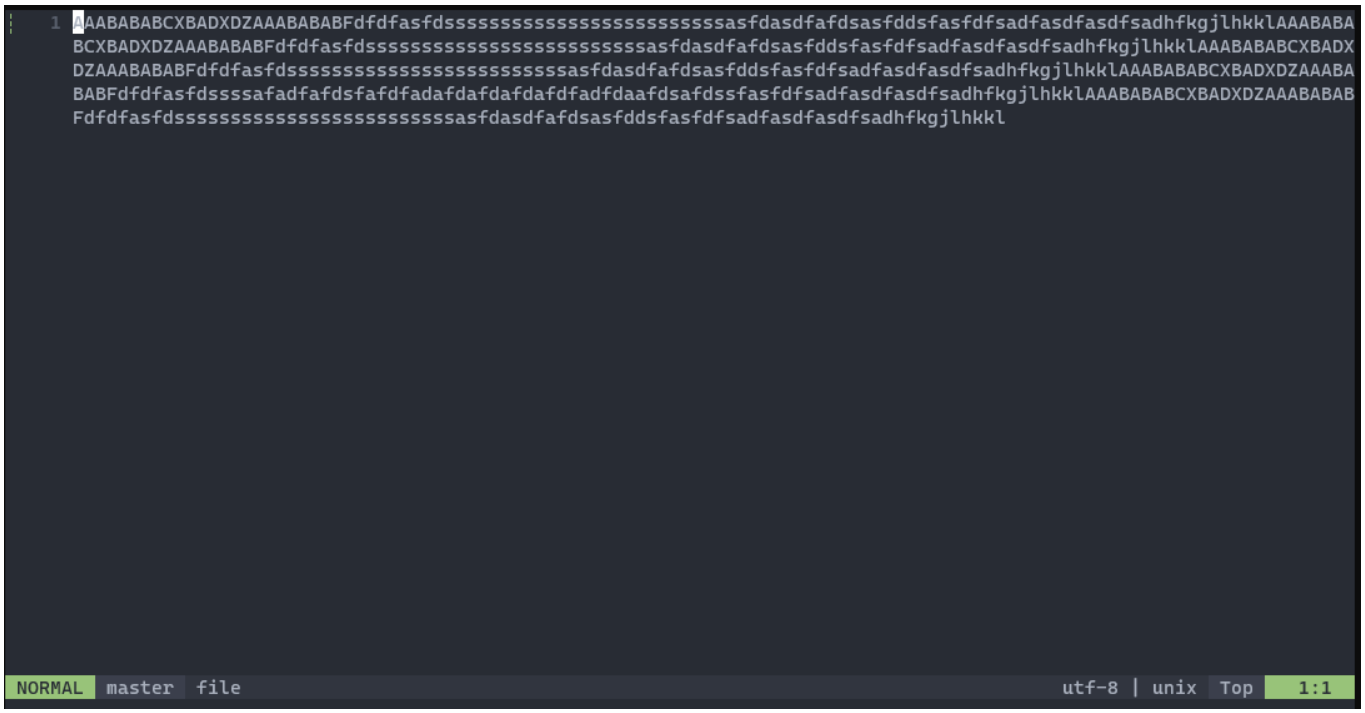
Сжатый файл



Декомпрессия



Выходной файл



Листинг программы

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <time.h>
4   #include <string.h>
5
6   // windowSize = Size of dictionary
7   // bufferSize = Size of lookahead buffer
8   // Important: windowSize < 255 & windowSize > bufferSize!
9   #define windowSize 60
10  #define bufferSize 40
11  #define arraySize bufferSize + windowSize
12
13  typedef enum { false, true } bool;
14
15  //
16  ============================================================================
17
18  // This method searches for a match from str[] in window[] of strLen length.
19  // Returns the position of the match starting from the beginning of window[],
20  // or -1 if no match is found.
21  // Is invoked during every iteration of the compression algorithm.
22  int findMatch(unsigned char window[], unsigned char str[], int strLen) {
23      int j, k, pos = -1;
24
25      for (int i = 0; i <= windowSize - strLen; i++) {
26          pos = k = i;
27
28          for (j = 0; j < strLen; j++) {
29              if (str[j] == window[k])
30                  k++;
31              else
32                  break;
33          }
34          if (j == strLen)
35              return pos;
36      }
37
38      return -1;
39  }
40
41  //
42  ============================================================================
43
44  // This method contains the logic of the compression algorithm.
45  // Is invoked when "-c" option is specified in launch command, followed by file name.
46  int compress(char* inputPath) {
47      FILE *fileInput;
48      FILE *fileOutput;
49      bool last = false;
50      int inputLength = 0;
51      int outputLength = 0;
52      int endOffset = 0;
```

```
53      int pos = -1;
54      int i, size, shift, c_in;
55      size_t bytesRead = (size_t) -1;
56      unsigned char c;
57      unsigned char array[arraySize];
58      unsigned char window[windowSize];
59      unsigned char buffer[bufferSize];
60      unsigned char loadBuffer[bufferSize];
61      unsigned char str[bufferSize];
62
63      // Open I/O files
64      char path[30] = "input/";
65      strcat(path, inputPath);
66      fileInput = fopen(path, "rb");
67      fileOutput = fopen("output/output.lz77", "wb");
68
69      // If unable to open file, return alert
70      if (!fileInput) {
71          fprintf(stderr, "Unable to open fileInput %s", inputPath);
72          return 0;
73      }
74
75      // Get fileInput length
76      fseek(fileInput, 0, SEEK_END);
77      inputLength = ftell(fileInput);
78      fseek(fileInput, 0, SEEK_SET);
79
80      fprintf(stdout, "Input file size: %d bytes", inputLength);
81
82      // If file is empty, return alert
83      if (inputLength == 0)
84          return 3;
85
86      // If file length is smaller than arraySize, not worth processing
87      if (inputLength < arraySize)
88          return 2;
89
90      // Load array with initial bytes
91      fread(array, 1, arraySize, fileInput);
92
93      // Write the first bytes to output file
94      fwrite(array, 1, windowSize, fileOutput);
95
96      // LZ77 logic beginning
97      while (true) {
98          if ((c_in = fgetc(fileInput)) == EOF)
99              last = true;
100         else
101             c = (unsigned char) c_in;
102
103         // Load window (dictionary)
104         for (int k = 0; k < windowSize; k++)
105             window[k] = array[k];
106
107         // Load buffer (lookahead)
```

```
108         for (int k = windowSize, j = 0; k < arraySize; k++, j++) {
109             buffer[j] = array[k];
110             str[j] = array[k];
111         }
112
113         // Search for longest match in window
114         if (endOffset != 0) {
115             size = bufferSize - endOffset;
116             if (endOffset == bufferSize)
117                 break;
118         }
119         else {
120             size = bufferSize;
121         }
122
123         pos = -1;
124         for (i = size; i > 0; i--) {
125             pos = findMatch(window, str, i);
126             if (pos != -1)
127                 break;
128         }
129
130         // No match found
131         // Write only one byte instead of two
132         // 255 -> offset = 0, match = 0
133         if (pos == -1) {
134             fputc(255, fileOutput);
135             fputc(buffer[0], fileOutput);
136             shift = 1;
137         }
138             // Found match
139             // offset = windowSize - position of match
140             // i = number of match bytes
141             // endOffset = number of bytes in lookahead buffer not to be considered (EOF)
142         else {
143             fputc(windowSize - pos, fileOutput);
144             fputc(i, fileOutput);
145             if (i == bufferSize) {
146                 shift = bufferSize + 1;
147                 if (!last)
148                     fputc(c, fileOutput);
149                 else
150                     endOffset = 1;
151             }
152             else {
153                 if (i + endOffset != bufferSize)
154                     fputc(buffer[i], fileOutput);
155                 else
156                     break;
157                 shift = i + 1;
158             }
159         }
160
161         // Shift buffers
162         for (int j = 0; j < arraySize - shift; j++)
```

```
163             array[j] = array[j + shift];
164         if (!last)
165             array[arraySize - shift] = c;
166 
167         if (shift == 1 && last)
168             endOffset++;
169 
170         // If (shift != 1) -> read more bytes from file
171         if (shift != 1) {
172             // Load loadBuffer with new bytes
173             bytesRead = fread(loadBuffer, 1, (size_t) shift - 1, fileInput);
174 
175             // Load array with new bytes
176             // Shift bytes in array, then splitted into window[] and buffer[] during next iteration
177             for (int k = 0, l = arraySize - shift + 1; k < shift - 1; k++, l++)
178                 array[l] = loadBuffer[k];
179 
180             if (last) {
181                 endOffset += shift;
182                 continue;
183             }
184 
185             if (bytesRead < shift - 1)
186                 endOffset = shift - 1 - bytesRead;
187         }
188     }
189 
190     // Get fileOutput length
191     fseek(fileOutput, 0, SEEK_END);
192     outputLength = ftell(fileOutput);
193     fseek(fileOutput, 0, SEEK_SET);
194 
195     fprintf(stdout, "\nOutput file size: %d bytes\n", outputLength);
196 
197     // Close I/O files
198     fclose(fileInput);
199     fclose(fileOutput);
200 
201     return 1;
202 }
203 
204 //
205 ================================================================================
206 
207 // This method contains the logic of the inverse algorithm, used to decompress.
208 // Is invoked when "-d" option is specified in launch command.
209 int decompress() {
210     FILE *fileInput;
211     FILE *fileOutput;
212     int shift, offset, match, c_in;
213     bool done = false;
214     unsigned char c;
215     unsigned char window[windowSize];
216     unsigned char writeBuffer[windowSize];
217     unsigned char readBuffer[2];
```

```
218
219     // Open I/O files
220     fileInput = fopen("output/output.lz77", "rb");
221     fileOutput = fopen("output/file", "wb");
222
223     if (!fileInput) {
224         fprintf(stderr, "Unable to open fileInput %s", "output.lz77");
225         return 0;
226     }
227
228     // Load array with initial bytes and write to file
229     fread(window, 1, windowSize, fileInput);
230     fwrite(window, 1, windowSize, fileOutput);
231
232     // Inverse algorithm beginning
233     while (true) {
234         // Read file by couples/triads to reconstruct original file
235         size_t bytesRead = fread(readBuffer, 1, 2, fileInput);
236
237         if (bytesRead >= 2) {
238             offset = (int) readBuffer[0];
239             match = (int) readBuffer[1];
240
241             // If first byte of readBuffer is 255 -> offset = 0, match = 0
242             if (offset == 255) {
243                 offset = 0;
244                 c = (unsigned char) match;
245                 match = 0;
246                 shift = match + 1;
247             }
248             else {
249                 shift = match + 1;
250                 c_in = fgetc(fileInput);
251                 if (c_in == EOF)
252                     done = true;
253                 else
254                     c = (unsigned char) c_in;
255             }
256
257             // Load and write occurrence to file
258             for (int i = 0, j = windowSize - offset; i < match; i++, j++)
259                 writeBuffer[i] = window[j];
260             fwrite(writeBuffer, 1, (size_t) match, fileOutput);
261
262             if (!done)
263                 fputc(c, fileOutput);
264
265             // Shift window
266             for (int i = 0; i < windowSize - shift; i++)
267                 window[i] = window[i + shift];
268
269             for (int i = 0, j = windowSize - shift; i < match; i++, j++)
270                 window[j] = writeBuffer[i];
271             window[windowSize - 1] = c;
272         }
```

```
273         else {
274             break;
275         }
276     }
277 
278     // Close I/O files
279     fclose(fileInput);
280     fclose(fileOutput);
281 
282     return 1;
283 }
284 
285 //
286 ================================================================================
287 
288 // This method is the controller, reads user inputs.
289 // Is invoked on program launch.
290 int main(int argc, char* argv[]) {
291     clock_t begin = clock();
292 
293     if (argc < 2) {
294         printf("Needs 2 arguments: [-c|-d] [file_path]");
295     } else {
296         // Start decompression
297         if (strcmp(argv[1], "-d") == 0) {
298             int result = decompress();
299             if (result == 0) {
300                 fprintf(stderr, "\nDecompression FAIL");
301             } else if (result == 1) {
302                 printf("\nDecompression OK");
303             }
304         }
305             // Start compression
306         else if (strcmp(argv[1], "-c") == 0) {
307             int result = compress(argv[2]);
308             if (result == 0) {
309                 fprintf(stderr, "\nCompression FAIL\n");
310             } else if (result == 1) {
311                 printf("\nCompression OK");
312             } else if (result == 2) {
313                 fprintf(stderr, "\nFile too small\n");
314             } else if (result == 3) {
315                 fprintf(stderr, "\nFile is EMPTY\n");
316             }
317         } else {
318             printf("Invalid arguments");
319         }
320     }
321 
322     // Print execution time
323     clock_t end = clock();
324     printf("\n\nExecution time: ");
325     printf("%f", ((double) (end - begin) / CLOCKS_PER_SEC));
326     printf(" [seconds]\n");
327 
```

```
328 return 0;
329 }
330
331
332
333
334
335
336
```