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-242Игнатенко 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**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, Графика

Автоматически созданное описание**

**Сжатый файл**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

**Декомпрессия**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Выходной файл**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

**Листинг программы**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336 | #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  // windowSize = Size of dictionary  // bufferSize = Size of lookahead buffer  // Important: windowSize < 255 & windowSize > bufferSize!  #define windowSize 60  #define bufferSize 40  #define arraySize bufferSize + windowSize  **typedef** **enum** { false, true } **bool**;  // ============================================================================  // This method searches for a match from str[] in window[] of strLen length.  // Returns the position of the match starting from the beginning of window[],  // or -1 if no match is found.  // Is invoked during every iteration of the compression algorithm.  **int** **findMatch**(**unsigned** **char** window[], **unsigned** **char** str[], **int** strLen) {  **int** j, k, pos = -**1**;  **for** (**int** i = **0**; i <= windowSize - strLen; i++) {  pos = k = i;  **for** (j = **0**; j < strLen; j++) {  **if** (str[j] == window[k])  k++;  **else**  **break**;  }  **if** (j == strLen)  **return** pos;  }  **return** -**1**;  }  // ============================================================================  // This method contains the logic of the compression algorithm.  // Is invoked when "-c" option is specified in launch command, followed by file name.  **int** **compress**(**char**\* inputPath) {  **FILE** \*fileInput;  **FILE** \*fileOutput;  **bool** last = false;  **int** inputLength = **0**;  **int** outputLength = **0**;  **int** endOffset = **0**;  **int** pos = -**1**;  **int** i, size, shift, c\_in;  **size\_t** bytesRead = (**size\_t**) -**1**;  **unsigned** **char** c;  **unsigned** **char** array[arraySize];  **unsigned** **char** window[windowSize];  **unsigned** **char** buffer[bufferSize];  **unsigned** **char** loadBuffer[bufferSize];  **unsigned** **char** str[bufferSize];  // Open I/O files  **char** path[**30**] = "input/";  strcat(path, inputPath);  fileInput = fopen(path, "rb");  fileOutput = fopen("output/output.lz77", "wb");  // If unable to open file, return alert  **if** (!fileInput) {  fprintf(stderr, "Unable to open fileInput %s", inputPath);  **return** **0**;  }  // Get fileInput length  fseek(fileInput, **0**, SEEK\_END);  inputLength = ftell(fileInput);  fseek(fileInput, **0**, SEEK\_SET);  fprintf(stdout, "Input file size: %d bytes", inputLength);  // If file is empty, return alert  **if** (inputLength == **0**)  **return** **3**;  // If file length is smaller than arraySize, not worth processing  **if** (inputLength < arraySize)  **return** **2**;  // Load array with initial bytes  fread(array, **1**, arraySize, fileInput);  // Write the first bytes to output file  fwrite(array, **1**, windowSize, fileOutput);  // LZ77 logic beginning  **while** (true) {  **if** ((c\_in = fgetc(fileInput)) == EOF)  last = true;  **else**  c = (**unsigned** **char**) c\_in;  // Load window (dictionary)  **for** (**int** k = **0**; k < windowSize; k++)  window[k] = array[k];  // Load buffer (lookahead)  **for** (**int** k = windowSize, j = **0**; k < arraySize; k++, j++) {  buffer[j] = array[k];  str[j] = array[k];  }  // Search for longest match in window  **if** (endOffset != **0**) {  size = bufferSize - endOffset;  **if** (endOffset == bufferSize)  **break**;  }  **else** {  size = bufferSize;  }  pos = -**1**;  **for** (i = size; i > **0**; i--) {  pos = findMatch(window, str, i);  **if** (pos != -**1**)  **break**;  }  // No match found  // Write only one byte instead of two  // 255 -> offset = 0, match = 0  **if** (pos == -**1**) {  fputc(**255**, fileOutput);  fputc(buffer[**0**], fileOutput);  shift = **1**;  }  // Found match  // offset = windowSize - position of match  // i = number of match bytes  // endOffset = number of bytes in lookahead buffer not to be considered (EOF)  **else** {  fputc(windowSize - pos, fileOutput);  fputc(i, fileOutput);  **if** (i == bufferSize) {  shift = bufferSize + **1**;  **if** (!last)  fputc(c, fileOutput);  **else**  endOffset = **1**;  }  **else** {  **if** (i + endOffset != bufferSize)  fputc(buffer[i], fileOutput);  **else**  **break**;  shift = i + **1**;  }  }  // Shift buffers  **for** (**int** j = **0**; j < arraySize - shift; j++)  array[j] = array[j + shift];  **if** (!last)  array[arraySize - shift] = c;  **if** (shift == **1** && last)  endOffset++;  // If (shift != 1) -> read more bytes from file  **if** (shift != **1**) {  // Load loadBuffer with new bytes  bytesRead = fread(loadBuffer, **1**, (**size\_t**) shift - **1**, fileInput);  // Load array with new bytes  // Shift bytes in array, then splitted into window[] and buffer[] during next iteration  **for** (**int** k = **0**, l = arraySize - shift + **1**; k < shift - **1**; k++, l++)  array[l] = loadBuffer[k];  **if** (last) {  endOffset += shift;  **continue**;  }  **if** (bytesRead < shift - **1**)  endOffset = shift - **1** - bytesRead;  }  }  // Get fileOutput length  fseek(fileOutput, **0**, SEEK\_END);  outputLength = ftell(fileOutput);  fseek(fileOutput, **0**, SEEK\_SET);  fprintf(stdout, "**\n**Output file size: %d bytes**\n**", outputLength);  // Close I/O files  fclose(fileInput);  fclose(fileOutput);  **return** **1**;  }  // ============================================================================  // This method contains the logic of the inverse algorithm, used to decompress.  // Is invoked when "-d" option is specified in launch command.  **int** **decompress**() {  **FILE** \*fileInput;  **FILE** \*fileOutput;  **int** shift, offset, match, c\_in;  **bool** done = false;  **unsigned** **char** c;  **unsigned** **char** window[windowSize];  **unsigned** **char** writeBuffer[windowSize];  **unsigned** **char** readBuffer[**2**];  // Open I/O files  fileInput = fopen("output/output.lz77", "rb");  fileOutput = fopen("output/file", "wb");  **if** (!fileInput) {  fprintf(stderr, "Unable to open fileInput %s", "output.lz77");  **return** **0**;  }  // Load array with initial bytes and write to file  fread(window, **1**, windowSize, fileInput);  fwrite(window, **1**, windowSize, fileOutput);  // Inverse algorithm beginning  **while** (true) {  // Read file by couples/triads to reconstruct original file  **size\_t** bytesRead = fread(readBuffer, **1**, **2**, fileInput);  **if** (bytesRead >= **2**) {  offset = (**int**) readBuffer[**0**];  match = (**int**) readBuffer[**1**];  // If first byte of readBuffer is 255 -> offset = 0, match = 0  **if** (offset == **255**) {  offset = **0**;  c = (**unsigned** **char**) match;  match = **0**;  shift = match + **1**;  }  **else** {  shift = match + **1**;  c\_in = fgetc(fileInput);  **if** (c\_in == EOF)  done = true;  **else**  c = (**unsigned** **char**) c\_in;  }  // Load and write occurrence to file  **for** (**int** i = **0**, j = windowSize - offset; i < match; i++, j++)  writeBuffer[i] = window[j];  fwrite(writeBuffer, **1**, (**size\_t**) match, fileOutput);  **if** (!done)  fputc(c, fileOutput);  // Shift window  **for** (**int** i = **0**; i < windowSize - shift; i++)  window[i] = window[i + shift];  **for** (**int** i = **0**, j = windowSize - shift; i < match; i++, j++)  window[j] = writeBuffer[i];  window[windowSize - **1**] = c;  }  **else** {  **break**;  }  }  // Close I/O files  fclose(fileInput);  fclose(fileOutput);  **return** **1**;  }  // ============================================================================  // This method is the controller, reads user inputs.  // Is invoked on program launch.  **int** **main**(**int** argc, **char**\* argv[]) {  **clock\_t** begin = clock();  **if** (argc < **2**) {  printf("Needs 2 arguments: [-c|-d] [file\_path]");  } **else** {  // Start decompression  **if** (strcmp(argv[**1**], "-d") == **0**) {  **int** result = decompress();  **if** (result == **0**) {  fprintf(stderr, "**\n**Decompression FAIL");  } **else** **if** (result == **1**) {  printf("**\n**Decompression OK");  }  }  // Start compression  **else** **if** (strcmp(argv[**1**], "-c") == **0**) {  **int** result = compress(argv[**2**]);  **if** (result == **0**) {  fprintf(stderr, "**\n**Compression FAIL**\n**");  } **else** **if** (result == **1**) {  printf("**\n**Compression OK");  } **else** **if** (result == **2**) {  fprintf(stderr, "**\n**File too small**\n**");  } **else** **if** (result == **3**) {  fprintf(stderr, "**\n**File is EMPTY**\n**");  }  } **else** {  printf("Invalid arguments");  }  }  // Print execution time  **clock\_t** end = clock();  printf("**\n\n**Execution time: ");  printf("%f", ((**double**) (end - begin) / CLOCKS\_PER\_SEC));  printf(" [seconds]**\n**");  **return** **0**;  } |