

COMPUTERIZED METHODS OF DATA COMPRESSION AND ANALYSIS

BACKGROUND

[0001] The field of the present invention relates to data compression and analysis. Specifically, the invention relates to lossless compression of text and other information that can be represented by symbols.

[0002] Previous text compression techniques are usable on only one file or document at a time. Such techniques do not scale easily to enormous data sets, *i.e.*, “Big Data,” or where data is spread across many different containers. Also, previous techniques do not track the number of times a particular symbol or string of symbols appear in the uncompressed text. The number of times a particular symbol or string appears in is valuable information that is useful for improving space savings, reducing processing time, and conducting contextual analysis.

SUMMARY

[0003] An example embodiment compresses symbolic information that is organized into a plurality of documents. Each of the uncompressed documents has a plurality of symbols. A document can be a string of symbols of any length (*e.g.*, a sentence, paragraph, or text file). A symbol can be any code that constitutes or carries information (*e.g.*, a letter, number, non-alphanumeric character, syllable, or word).

[0004] The example embodiment operates on an input document by identifying two adjacent symbols in the document as a symbol pair. The example embodiment stores each unique symbol pair in a compression dictionary. The compression dictionary associates each stored symbol pair with a respective replacement symbol. The example embodiment also maintains a count of the number of times each unique symbol pair appeared in the input document. The example embodiment produces a compressed output document by replacing symbol pairs with their associated replacement symbols if the count for the symbol pair exceeds a threshold

[0005] Some embodiments are programmed to repeat the compression process by using the output document as an input document for an additional pass. Certain embodiments recursively perform additional passes until no further replacements can be made (*i.e.*, in the final pass, the output document is identical to the input document). Symbol pairs identified in recursive passes can include replacement symbols from a previous pass.

[0006] Some embodiments are programmed to be applied to multiple documents. In fact, the compression ratio improves as the size of the uncompressed data increases. The larger the data set, the more likely it is that the count associated with any given symbol pair will exceed the threshold, which results in more replacements and higher compression. Compression is optimized, in some of these embodiments, when the process recurs on all the documents until no further replacements are possible.

[0007] Some embodiments increase space savings and decrease the lookup time needed to access the compression dictionary by using, as a replacement symbol, the address of the replaced symbol pair in the compression dictionary.

[0008] Some embodiments sort the entries in the compression dictionary based on the count of the number of times a symbol pair appeared. The sorting is done to place the most frequently appearing symbol pairs in more accessible memory locations than less frequently appearing symbol pair. Sorting decreases lookup times during the replacement phase.

[0009] Some embodiments use patterns of replacement symbols and the count of the number of times certain symbol pairs appeared to analyze content.

[0010] Objects and advantages pertaining to computerized methods of data compression and analysis may become apparent upon referring to the example embodiments illustrated in the drawings and disclosed in the following written description or appended claims.

[0011] This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] Fig. 1 is a visual representation of a text standardization process applied by an embodiment.

[0013] Fig. 2 is a visual representation of a compression dictionary used the embodiment of Fig. 1.

[0014] Fig. 3 is a visual representation operation of the embodiment of Fig. 2.

[0015] Fig. 4 is a flowchart that shows the operation of the embodiment of Fig. 2.

[0016] Fig. 5 is a chart showing results achieved by an experimental embodiment.

[0017] The embodiments depicted are shown schematically. The embodiments shown are examples only and should not be construed as limiting the scope of the present disclosure or appended claims.

DETAILED DESCRIPTION

[0018] Embodiments of the data compression and analysis technique can be programmed in computer code as a set of computer instructions. When running on a computer, the computer instructions control a computer processor and cause it to perform the functions described herein. The computer instructions (e.g., computer software) can be stored in any computer memory accessible to the computer or encoded in computer hardware. Some embodiments are instantiated in special purpose computers or dedicated computer appliances. The documents on which the embodiments operate can be stored in the same or a different computer memory accessible to the computer system.

[0019] An example embodiment analyzes uncompressed text to identify adjacent pairs of symbols. The example embodiment replaces occurrences of a pair of symbols that appears more than a predefined number of times with an associated replacement symbol. Symbols can be alphanumeric characters,

discrete groups of alphanumeric characters such as syllables or words, or any code that constitutes and carries information.

[0020] In some embodiments, the original symbols in the uncompressed text are replaced with standard symbols during a preliminary phase. Fig. 1 is a representation of how an example embodiment standardizes uncompressed text. In certain circumstances, such as when there is a large variation in the lengths of the original symbols, standardization improves performance during subsequent operations.

[0021] The example embodiment of Fig. 1 identifies and counts all occurrences of unique original symbols and stores each unique original symbol. Each original symbol is stored in a standardization memory in association with a standard symbol. In Fig.1, standardization memory 140 associates an original symbol (in column 160) with its respective standard symbol (in column 150) and a count of how many times each original symbol appears in the uncompressed text (in column 170).

[0022] To identify unique original symbols, some embodiments are programmed to interpret certain non-alphanumeric characters (e.g., a space) as denoting a break between two symbols. The example embodiment interprets a space as a break between symbols yet retains the space in the standardized document. For example, document 111 is shown with spaces separating the standard symbols. However, other embodiments drop spaces and certain other non-alphanumeric characters entirely because the location of such characters can be restored or discerned from context. For example, a space between symbols can be inferred and reproduced based on the rules of the algorithm used to identify the original symbols. Likewise, the presence of certain non-alphanumeric characters can be inferred based on the grammatical rules of the language in which the text is written.

[0023] Some embodiments interpret certain non-alphanumeric characters as part of a symbol. The example embodiment of Fig. 1 identified text 105 as a symbol even though it contains an apostrophe. Other embodiments identify non-alphanumeric characters as symbols in their own right.

[0024] Some embodiments ignore capitalization while processing symbols. Such embodiments provide space savings by storing only a uniform-case version (e.g., lowercase version) of all original symbols. Such embodiments can restore the original text, including restoring capitalization, using grammatical and contextual rules regarding sentence structure and proper nouns.

[0025] Some embodiments interpret select symbols as signaling the end of a document. The example embodiment of Fig. 1 identifies the period at 107 as the separator between document 101 and document 102. Alternative embodiments separate text into documents based on carriage returns, paragraphs, cells, fields, records, rows, columns, pages, headings, tags from a markup language, information in a table of contents, or section or chapter markings. Any human-understandable or computer-readable markers or containers can be used to divide the text into documents. Other embodiments operate on documents that are separate computer files.

[0026] The example embodiment operates on multiple documents concurrently. Thus, during the standardization process, the example embodiment identifies and counts all occurrences of unique uncompressed symbols in all documents before replacing any of the original symbols in the documents.

[0027] The example embodiment replaces the original symbols of document 101 with standard symbols stored in standardization memory 140 to produce standardized document 111. Standardization memory 140 can be created during the standardization of input documents or can be pre-existing. Fig. 1 presents standardization memory 140 in a row-and-column format, but other data storage structures can be used.

[0028] The example embodiment sorts the entries in standardization memory 140 according to the counts stored in column 170. The example embodiment sorts the entries before standard symbols are assigned and stored in column 150. Sorting standardization memory 140 improves space savings because the example embodiment can assign shorter standard symbols to the symbols that appear most frequently. Sorting also allows frequently used standard symbols to be stored in highly accessible memory locations, which improves lookup times. After sorting, some embodiments may delete the counts to save space.

[0029] Fig. 1 shows the counts stored in column 170 of standardization memory 140. Alternative embodiments store the counts generated during standardization separately from the standardization memory.

[0030] A standard symbol can be an address indicating the location of the original symbol in standardization memory 140. Use of an address rather than an arbitrary symbol reduces the requisite memory space because there is no need to store extraneous symbols. Use of addresses as standard symbols also improves processing times during replacement and decompression because the example embodiment need only perform a single memory lookup rather than performing the multiple lookups that are required to translate between standard symbol and replacement symbol. When restoring the text to its original form, each standard symbol directs the example embodiment to the exact memory location of the original symbol.

[0031] Use of addresses as standard symbols, when applied after sorting, further improves lookup times because it ensures that more frequently occurring symbols are stored in more accessible memory locations than less frequently occurring symbols.

[0032] Fig. 2 is a representation of compression dictionary 200 that is used by the example embodiment to store associations between a unique symbol pair and its replacement symbol. Standardization memory 140 in Fig. 1 is shown stored separately from compression dictionary 200 in Fig. 2. Alternative embodiments can store the standardization memory as part of the compression dictionary.

[0033] Compression dictionary 200 is shown structured in rows and columns. Alternative embodiments use other data structures such as relational databases, document-oriented databases, graph databases, key-value stores, object databases, or hierarchical trees to maintain the compression dictionary (and

standardization memory if used). Some embodiments use an existing compression dictionary. Other embodiments create the compression dictionary during operation.

[0034] Fig. 3 is a representation of document 111 as it is operated on by the embodiment of Fig. 2. The example embodiment is programmed to repeat recursively until compression is complete, e.g., until no more pairs of symbols meet the threshold for replacement. The example embodiment operates on document 111 to create output document 351. Numerals 321, 331, and 341 refer to intermediate results before passes of the process carried out by the recursive embodiment; they are called "documents" here for ease of reference but can be deleted or overwritten during the next pass. Documents 112-114 are likewise processed in the same batch as document 111 to produce respective documents 352-354. Documents 322-324, 332-334, and 342-344 represent intermediate results between passes.

[0035] During each pass the example embodiment makes through the documents, the example embodiment identifies and counts the occurrence of each unique symbol pair ("the cataloging phase") and replaces symbol pairs with a count exceeding the replacement threshold ("replacement phase"). The example embodiment performs the cataloging phase on all documents before turning to the replacement phase.

[0036] Alternative embodiments identify and replace symbol pairs in one operation by processing the text of each document in sequential order. In such alternative embodiments, a pass through a document involves comparing each sequential symbol pair to entries of the compression dictionary. If the comparison results in a match, the compared symbol pair's count is incremented, and if the replacement threshold is met, the symbol pair is replaced with a replacement symbol. If the comparison does not produce a match, the symbol pair is identified as unique and added to the compression dictionary. Because such alternative embodiments do not use replacement symbols until a symbol pair's count exceeds the threshold for replacement, such embodiments schedule replacement of earlier instances of a symbol pair once the symbol pair's count exceeds the replacement threshold.

[0037] In the identification or cataloging phase of a pass, the example embodiment identifies symbol pairs in an input document by pairing each symbol with the next adjacent symbol in the document until the last symbol in the document is reached. All symbols, except first and last symbol, in the input document are therefore included in two symbol pairs: One symbol pair formed from pairing the symbol with the preceding symbol, and one symbol pair formed from pairing the symbol with the following symbol.

[0038] For example, when processing document 111, the example embodiment recognizes symbols 311 and 312 as a first pair and symbols 312 and 313 as a second pair. The example embodiment continues pairing symbols until it pairs symbol 318 and 319. The first symbol (symbol 311) and the last symbol (symbol 319) are each only paired with one other symbol (i.e., symbol 312 and symbol 318, respectively). All other symbols in the document are included in two symbol pairs.

[0039] Alternative embodiments do not compare every possible pairing of adjacent characters. Rather such embodiments skip symbol pairs having a symbol from the previously compared pair. For example, such embodiments, if operating on document 111, would recognize the pair of symbols 311 and 312, skip over the pair formed by symbols 312 and 313, and recognize the pair of symbols 313 and 314 as the next pair.

[0040] Once all pairings have been made for the symbols in documents 111-114, the example embodiment counts the number of times each unique symbol pair appeared in the documents. The example embodiment identifies and counts the unique symbol pairs by inserting every symbol pair into an array and performing a “pivot table” operation on the array to identify and count the number of occurrences of each unique symbol pair.

[0041] In each pass, the example embodiment generates a pass-related section in the compression dictionary. For example, referring to Fig. 2, columns 210 and 211 of compression dictionary 200 are the result of the cataloging phase of the first pass. Column 210 stores each unique symbol pair from the text processed in the cataloging phase of the first pass. Column 211 stores the count associated with each symbol.

[0042] Fig. 2 presents the count as being stored as a field in compression dictionary 200. A symbol pair's associated count could be stored as an attribute or metadata of the symbol pair, rather than as an entry in the compression dictionary. Alternative embodiments store counts associated with each pair separately from the compression dictionary. The example embodiment does not store a count for symbol pairs that appear only once (see entry 216 in compression dictionary 200). The presence of a symbol pair is sufficient to indicate it has appeared only once.

[0043] The example embodiment is programmed to sort the entries of compression dictionary 200 based on the counts of each symbol pair in a manner similar to that described above in connection with standardization of the symbols. In other words, after all symbol pairs are identified from documents 111-114, the example embodiment sorts entries of compression dictionary 200 (Fig. 2) before assigning replacement symbols or making any replacements to produce document 321 (Fig. 3). For the reasons similar to those described above in connection with sorting the standardization memory, the example embodiment achieves space savings and performance benefits by sorting compression dictionary 200.

[0044] A replacement symbol can be an address in the compression dictionary where the symbol pair associated with the replacement symbol is stored. In the embodiment of Fig. 2, replacement symbols are formed from a combination of the row identifier (see column 201) and a column identifier. The column identifier indicates the pass during which the symbol pair was identified (P1, P2, P3 in Fig. 2). Use of address-based replacement symbols provides similar benefits to those described in connection with standardization for similar reasons. Alternative embodiments assign unique replacement symbols that are unrelated to the address of the symbol pair.

[0045] In some cases, the replacement symbol can be just the portion of the address associated with the row identifier. For example, the replacement symbols from the first pass of the example embodiment are only the row identifier of its respective symbol pair's address. When the example embodiment decompresses a document, it is programmed to associate replacement symbols without a pass identifier as being from the first pass. In example embodiments, replacement symbols from the first pass are inserted into output documents and stored in the compression dictionary (if paired with another symbol in a sequential pass) without a pass identifier.

[0046] Alternative embodiments achieve further space savings in a similar way. For any given pass after the first pass, an embodiment can omit the pass identifier of replacement symbols that are most frequently paired with another symbol. For example, in Pass X, symbol pairs can be formed by pairing a replacement symbol from any of Passes 1, ..., N, ..., X-1 with another symbol. Suppose that, in Pass X, replacement symbols from pass N are most frequently paired with other symbols. In that case, rather than storing, in the compression dictionary, the full replacement symbol for Pass N replacement symbols, the embodiment omits the pass identifier from Pass N replacement symbols and simply stores an indication that Pass N is the most frequent pairing for Pass X. Thus, if any symbol pair stored in Pass X's section of the compression dictionary contains a replacement symbol with no pass identifier, the pass identifier is assumed to be that of Pass N. In this manner, space is saved because Pass N's pass identifier need only be stored once for all of the entries stored in the compression dictionary during Pass X.

[0047] The example embodiment of Fig. 2 only uses replacement symbols if their associated count exceeds a predetermined threshold. Every symbol pair in the document that corresponds to an entry in compression dictionary could be replaced (e.g., a threshold equal to one). However, a higher threshold allows for improved performance. If a large number of symbol pairs appear only once or only a few times, their replacement will not improve, or significantly improve, compression of the documents and will increase the storage requirements of the compression dictionary. A higher threshold can reduce processing time by eliminating the need to replace infrequently used symbol pairs.

[0048] The example embodiment improves space savings further by only retaining compression dictionary entries for symbol pairs that have a count exceeding the threshold. Because the example embodiment is programmed to perform the cataloging phase on all documents before beginning the replacement phase, symbol pairs that do not exceed the threshold will not be replaced. Thus, storing below-threshold symbol pairs increases the size of the compression dictionary without providing any without benefit. To avoid that inefficiency, the example embodiment truncates (*i.e.*, deletes portions of) the compression dictionary having entries with counts less than the threshold. For example, if the threshold for the first pass is four, the example embodiment would not retain portion 260 of Fig. 2.

[0049] Alternative embodiments retain portions of the compression dictionary with entries that have a count less than the threshold even though they will not be used with the current set of documents. If new

documents are added to the set of documents, the counts for below-threshold symbol pairs could increase to the point where they meet the replacement threshold. Some alternative embodiments store below-threshold portions of the compression dictionary in a separate archive rather than the compression dictionary. The archive can be accessed when new documents are added.

[0050] In various embodiments, the replacement symbols can be associated with a symbol pair when the symbol pair is first identified or meets the replacement threshold. The example embodiment assigns replacement symbols after both sorting and truncation so that the most frequently used replacement symbols will have the shortest length and can be placed in the most accessible memory locations. In circumstances where a replacement symbol is the address of its associated symbol pair, each replacement symbol is associated with its symbol pair by virtue of the symbol pair's position in the sort order.

[0051] Alternative embodiments can assign replacement symbols any time before the replacement phase begins. Such alternative embodiments can sort the compression dictionary based on the counts or maintain the entries in their original order. After sorting, some embodiments delete the counts to save additional space.

[0052] During the replacement stage of a pass, the example embodiment creates a compressed output document by replacing symbol pairs in an input document with replacement symbols from compression dictionary 200. Alternative embodiments create a new output document distinct from the input document by inserting either the original symbols or replacement symbols in place of a pair of the original symbols. Such alternative embodiments later delete the input document. The example embodiment creates output document 321 (see Fig. 3) by overwriting symbol pairs in input document 111 with replacement symbols where appropriate.

[0053] The compression dictionary for the example embodiment only contains entries for symbol pairs that exceed the replacement threshold because entries with counts not exceeding the threshold were not retained. Thus, the example embodiment is programmed to replace the first symbol pair that matches an entry in the compression dictionary. When processing document 111, the example embodiment identifies the pair of symbols 311 and 312 (Fig. 3) and compares it to the entries of compression dictionary 200 (Fig. 2). The pair of symbols 311 and 312 are found in compression dictionary 200 (see entry 212). Because a match is found, the pair of symbols 311 and 312 is replaced with replacement symbol 325 in output document 321. The dictionary also contains the pair of symbols 312 and 313 (see 214), and the count exceeds the threshold for replacement (see 215). However, the pair of symbols 312 and 313 is not considered for replacement, because the example embodiment already replaced symbol 312.

[0054] Embodiments that do not truncate the compression dictionary must compare symbol pairs in a document to the compression dictionary and also check the count before replacement. Alternative embodiments analyze multiple possible replacements to determine which would maximize the number of total replacements or maximize the space savings. Other embodiments determine which pairs to replace

based on minimizing the length of the replacement pairs or the lookup time required to decompress the document. Still further alternative embodiments attempt to replace the first pair of symbols and if no replacement is possible, skip over the pair that could be formed by combining the second symbol of the first pair with the next sequential symbol.

[0055] When the example embodiment cannot replace a symbol pair, it continues to consider the next symbol pair. For example, when the pair of symbols 315 and 316 are compared to the entries in the compression dictionary, no match is found. The example embodiment, therefore, moves to the next symbol pair (the pair of symbols 316 and 317). Embodiments that create a new output document rather than overwriting the symbols of the input document insert into the output document the first symbol of a symbol pair that cannot be replaced. If that last symbol of a symbol pair is the last symbol in a document, and the symbol pair does not match any of the entries in the compression dictionary, both symbols of the symbol pair must be inserted into the output document.

[0056] As described, the example embodiment is programmed to perform recursive passes until no further replacements are possible. Thus, the example embodiment begins a second pass by applying cataloging and replacement phases to document 321 to produce document 331. In the symbology of Fig. 3, symbol "R3P2" in document 331 refers to row 3 of the second pass. Symbol pairs identified in a second or subsequent pass can be two replacement symbols. See replacement symbol 335, which refers to the sequence of replacement symbol R4 (see 325) followed by replacement symbol R3 (see 326 in the first pass in document 321). Alternatively, symbol pairs identified in a later pass can be a combination of a replacement symbol and an uncompressed symbol. See replacement symbol 336, which refers to the sequence of uncompressed symbol S300 followed by replacement symbol R3. Compression dictionary 200 of Fig. 2, therefore, lists in row R3 of column 220 (P2) the sequential pair R4 followed by R3. Review of column 210 reveals that the sequence R4 followed by R3, in turn, denotes symbol S1 followed by S79 followed by S75 followed by S5. Thus, in this example, after the second pass (P2), the example embodiment has replaced the four-symbol "phrase" S1-S79-S75-S5 with the single symbol R3P2.

[0057] Sequential passes by the example embodiment result in additional columns 220 and 230 (Fig. 2) that store symbol pairs from each respective pass. In the example embodiment, each pass in which unique symbol pairs are discovered results in additional columns being added to compression dictionary 200 to store the newly identified symbol pairs and their respective counts (see columns 221 and 231). Fig. 2 presents symbols discovered in later passes in separate columns to illustrate the recursive nature of the embodiment.

[0058] The example embodiment only makes replacements from the column or section of compression dictionary 200 created during the instant pass. Storing replacement symbols in pass-specific sections reduces lookup times because the embodiment does not need to search through entries having replacement symbols that will not be used in the current pass.

[0059] Alternative embodiments store symbol pairs identified in different passes in a common column or memory area. Such embodiments can increase space savings because they use less complex replacement symbols. For example, an embodiment that stores all identified symbol pairs in the same column of the compression dictionary could use a replacement symbols that consists entirely of a row identifier. In any embodiment, close attention to the choice of symbology for the replacement symbol is capable of producing increased compression.

[0060] As shown in Fig. 3, a third pass performed on document 331 produces document 341, and a fourth pass performed on document 341 to produce document 351. If the output of fourth pass (document 351) is the same as the output of the third pass (document 341), and assuming this remains true for all other documents that are being compressed concurrently with document 111, then the example embodiment stores the last output version of each document and the compression process terminates. The example presented in Fig. 3 only shows four passes of the process applied to the example text. However, any number of passes can be performed. In some embodiments, the number of passes can be a fixed number. In other embodiments, a maximum number of passes can be set, but the process is programmed to terminate if no further replacements are made before the cap.

[0061] As more passes are performed on a set of documents, the number of symbol pairs identified and the count of each identified symbol pair tends to decrease, hence the incremental size reduction provided by each additional pass decreases. However, each additional pass will consume additional processing resources. To balance the size reduction against processing resources, certain embodiments are programmed to terminate the process if the size reduction resulting from a given pass falls below a certain limit (e.g., if a given pass produces less than a half of a percent size reduction). Other embodiments terminate the process based on a different factor, such as the number of new symbol pairs found being less than a fixed amount or the count of the most frequently occurring symbol pair falling below a certain level. For another example, an embodiment can be constructed to terminate the process if the count of the most frequently occurring symbol pair in a pass falls below a predetermined percentage of the count of the most frequently occurring symbol in the first pass.

[0062] Once the criterion for terminating the process has been met, some embodiments are programmed to replace any identified symbol pairs found in the pass that triggered the termination condition. Other embodiments terminate the process immediately upon recognizing the termination condition, without replacing symbol pairs found in that last pass. Still further embodiments terminate mid-pass if an insufficient number of symbol pairs are found after processing enough text to determine that the resultant space savings would fall below a predetermined amount. Any symbol pairs that are not replaced would not need to be stored in the compression dictionary or assigned a replacement symbol.

[0063] Compression dictionary 200 can grow to have any number of columns depending on the number of passes performed on a plurality of documents. Thus, Fig. 2 only shows a portion of compression dictionary 200 relevant to documents 310-350.

[0064] The example embodiment performs sort and truncation operations at the end of each cataloging phase. However, thresholds applied during successive passes can differ from the threshold of the first pass. The example embodiment uses lower replacement thresholds for additional passes.

[0065] Some embodiments determine ideal values for the replacement thresholds of each pass by compressing a subset of the documents in a series of trials with different pass thresholds values in each trial until the best compression is achieved.

[0066] Fig. 4 is a flowchart showing the operation of the example embodiment. Cataloging phase 400 for any pass begins with identifying all unique symbols (401). Next, the example embodiment counts the number of occurrences of each unique symbol (402). Next, the example embodiment populates the compression dictionary with the unique symbols (403). Populating the compression dictionary can include reordering the unique symbol pairs based on their counts, omitting symbol pairs with counts that do not exceed the threshold, and associating replacement symbols with each unique symbol pair. The example embodiment completes cataloging phase 400 on all documents before moving to replacement phase 450.

[0067] During replacement phase 450, the example embodiment searches the input documents for symbol pairs that match entries in the compression dictionary (451). If any replacements are possible, the example embodiment replaces symbol pairs with an associated replacement symbol (452). The example embodiment enforces the threshold at 403 by populating the compression dictionary with only symbol pairs that exceed the threshold; however other embodiments that retain symbol pairs with counts that do not exceed the threshold enforce the threshold by only replacing symbols pairs with counts exceeding the threshold. After replacements are made (452), the example embodiment operates recursively by returning to cataloging phase 400 to begin a subsequent pass on the output of the previous pass. As part of the recursion process, the example embodiment adjusts the replacement threshold (not shown) according to the settings for the subsequent pass. If no replacements are possible during a pass, the example embodiment stores the compressed output documents and terminates (499). When required, the example embodiment restores the compressed documents by reversing the replacement phase of each pass.

[0068] The example embodiment handles new documents by decompressing all compressed documents, deleting the compression dictionary, and with the new documents as part of the plurality of documents, re-applying the process (*i.e.*, 400-499) to the uncompressed documents. Compression of new documents can be scheduled for times when resource demand is expected to be low or by an offline system so that users do not experience noticeable delays when accessing compressed documents.

[0069] Alternative embodiments handle new documents by recursively applying cataloging and replacing passes to new documents without uncompressing any documents. In such embodiments, compression is

improved if the compression dictionary is not truncated because symbol pairs identified in new documents could cause the count a particular replacement symbol that did not previously meet the replacement threshold to exceed the threshold and used in the new document.

[0070] Other embodiments recursively apply only the replacement phase to the new documents. For such embodiments, the documents in an original set of documents can act as a training set.

[0071] The process works best with large volumes of data. The more data that is compressed, the more likely it is that long sequences of symbols will be repeated. Thus, higher compression ratios can be achieved as the number of symbols increases. In that way, processes described with reference to the above embodiments are well suited to “big data” applications. In fact, the disclosed embodiments can result in greater data size compared to existing techniques if applied to very small data sets, in part because of overhead associated with the compression dictionary.

[0072] Likewise, if the plurality of documents includes only documents from a particular field of knowledge, the process can produce improved compression ratios because any given field of knowledge has common terms and phrases that are often repeated in texts dedicated to the field. On the other hand, when documents from a new field of knowledge are incorporated into the plurality of documents, the process encounters a learning curve in which the achievable compression ratio is decreased. Eventually, after processing a sufficient number of documents in a new field, the system can “learn” the terms of art from the new field, and the compression ratio improves.

[0073] Fig. 5 summarizes experimental results of one embodiment. The experimental embodiment was programmed to identify each word in the input documents as a symbol. The experimental embodiment applied standardization to input documents, and both sorted and truncated the sections of the compression dictionary relating to a particular pass after the cataloging phase of that pass. The documents used in the experiment were abstracts from 15,000 randomly selected US patents (a little over one million words). Before the experimental embodiment was applied to the documents, all non-alphanumeric characters were removed, and all uppercase characters were converted to lowercase characters.

[0074] To test the premise that compression rates improve as data size increases, the experimental embodiment was applied to six different-sized groupings of documents. Table 1 details the size of each group and the achieved compression ratio. The data in Table 1 is plotted in Fig. 5 with the number of characters plotted on the x-axis (in log scale) and the compression ratio plotted on the y-axis. The dashed line in Fig. 5 represents a trend line based on a power regression of the data in Table 1.

Table 1	
Characters in Input Docs	Compression Ratio
234	104.27%
487	96.30%
5,280	77.75%
66,864	72.89%
640,621	61.96%
6,290,146	56.09%

[0075] Compression ratio for the experimental results is expressed as a percentage and indicates the size of the compressed data as a percentage of the uncompressed data. In other words, compression ratio (in percent) equals the number of compressed characters divided by the number of uncompressed characters multiplied by 100. The number of compressed characters includes the number of compressed characters in the output documents plus the number of characters in the compression dictionary.

[0076] The experimental embodiment was able to improve upon the compression ratio achievable by Byte-Pair Encoding, which is at best 60% when applied to English language texts. Shibata Y. et al. (2000) Speeding Up Pattern Matching by Text Compression. In: Bongiovanni G., Petreschi R., Gambosi G. (eds) Algorithms and Complexity. CIAC 2000. Lecture Notes in Computer Science, vol. 1767. Springer, Berlin, Heidelberg. Ultimately, the experimental embodiment achieved a compression ratio of 56.09% when applied to the full sample of approximately one million words.

[0077] The system and process can apply to various applications, for example, compression of archives such as email, news, or other library archives. The process is well suited to such applications because the large number of the symbols involved is likely to produce a significant number of repeated strings, facilitating a high level of compression. Additions to the archive can be scheduled to occur during off-peak hours and can be structured in large batches as to best balance the required processing time against the space savings achieved by compression of the added documents.

[0078] The trend line in Fig. 5 suggests that, if the system is applied to a large enough data set, the compression ratio is likely to continue to improve. The trendline crosses below 40% at around a billion characters and below 30% at around a trillion characters. Wikipedia is an example that could provide a sense of the scale at which the compression ratio is likely to be most effective. In 2015, Wikipedia's English-language version was estimated to contain approximately 23 billion characters. In November 2009 (the last month in which Wikipedia released statistics on word count across all languages), Wikipedia (all languages) had approximately six billion words and was growing at a rate of one billion words every nine months. Assuming six characters per word, Wikipedia (all languages) therefore had thirty-six billion characters in November 2009. Assuming that such a growth rate has persisted, Wikipedia (all languages) is estimated to have approximately 16 billion words or one trillion characters in March 2017. Thus, a person desiring to archive a data set on the scale of Wikipedia could benefit from compression using this process.

[0079] Applications directed to cloud-based email and messaging systems could also benefit from the process because such systems have many users. Many of the emails are likely to be replies to and messages forwarded from other users. Such messages often include all or portions of the original message, which results in very long strings of duplicated symbols that the process can sometimes replace with as little as a single replacement symbol.

[0080] Certain replacement symbols that are generated after many successive passes replace long strings of text. If two compressed documents have such higher-order replacement symbols in common, a relationship between the documents can be deduced. As described above, authors of works in a particular field tend to repeat phrases and terminology unique to the field and works. A particular author can utilize a number of common phrases, concepts, and story characters frequently discussed by that author. In such situations, two works can be identified as related based on such commonalities. The commonalities could be used to group the documents together as a collection, to identify certain documents as derivative of other documents, or to suggest works to a reader based on the reader's interest in a particular field or a particular author. Such embodiments are particularly useful to a user who is unfamiliar with a field or in situations where the actual author is difficult to identify (e.g., a ghost author or a plagiarist).

[0081] The process can be applied to analyze any data that can be codified. For example, characteristic of the human face can be codified into a set of measurements (size of each eye, the separation between the eyes, eye color, the length of the brow, etc.). Those measurements could be coded or classified, formatted, and assembled into a document, using a consistent order. In such a manner, the compression process can be applied to a set of documents, where each document contains a plurality of measurements of a particular person's facial structure. After the documents are compressed, a system could perform facial recognition by comparing the replacement symbols in the compressed documents. Because higher-order replacement symbols represent long chains of symbols, the fact that two documents both have one or more of the same higher-order replacement symbols indicates a high likelihood that the faces, represented by the measurement in the documents, are from the same person.

[0082] A match between two documents can be determined based on the total number of matching higher-order replacement symbols exceeding a threshold. Alternative embodiments of the facial recognition system can select the closest match or matches and value indicating the degree of confidence in each match.

[0083] Such a facial recognition system could be used, for example, to control access to a secure location. The secure location can be a physical place (e.g., a building) or a device (e.g., a computer system, database, or secured portion thereof). An alternative use could be to determine the identity of an unknown person. Another alternative use is merely distinguishing whether or not certain measurements represent a human face or not.

[0084] In the case of controlling access to a secure location, an example embodiment would first use the compression process, as described herein, to compress a set of documents, each representing one of a group of known and authorized persons, and to produce a compression dictionary. Later, when requesting access to the secure entity, an unknown person initiates a process by which measurements are taken of his or her face. The unknown person's face could be measured by a non-invasive method (e.g., calculating the required values from an image captured by one or more cameras). The required values are then arranged into a new document that is compressed using the same compression dictionary. The system compares the replacement symbols in the compressed version of the new document to those contained in the previously compressed documents and outputs the results (e.g., just that a match exists and access should be granted or more specifically the believed identity of the unknown person). In some embodiments, the system can have the unknown person purport to identify himself or herself by other means, such as through voice announcement or a card swipe, and check the document containing information derived from the captured image of the unknown person's face against only the document containing pre-stored information about the person that the unknown person claimed to be. In other embodiments, the system can compare the document containing the captured image of the person's face against all documents containing pre-stored information about all authorized persons.

[0085] In the facial recognition system, the symbology of facial measurements in the documents can be exact numerical measurements (e.g., pupillary separation = 64.57 mm or eye color (RGB) = 161, 202, 241). Use of exact measurements can produce a high degree of certainty in any match that is found but is also likely to result in false negatives. Slight variances in the measurement conditions or equipment calibrations over time could result in misidentification. Thus, it is advisable that the symbology uses approximate measurements or values representing ranges in which the measurements fall. The symbology of the ranges can include numerical values of the range, an average value for the range, or an arbitrary symbol that represents the range.

[0086] Each facial recognition document needs to have a sufficient number of measurements to avoid misidentification. Too few measurements results in a risk of falsely identifying a match. Another use of the facial recognition process is to investigate and improve facial recognition systems based on the process discussed herein, because by analyzing a sufficiently large set of documents representing known faces, the system can be used to determine how many measurements are required and the optimal order or groupings of measurements necessary to produce sufficient points of comparison.

[0087] Other practical applications of the process use recognized commonalities in compressed documents to analyze the content of the documents. Symbol pairs that are counted and associated with replacement symbols have an underlying meaning. Thus, thoughts, preferences, or other information about the author or subject of a document can be inferred from the symbol pairs. Data collected during the cataloging phase can provide a wealth of information of such information.

[0088] Embodiments with extended functionality could utilize the process to learn more about the content of text by noticing certain replacement symbols associated with a particular noun or verb. For example, such a system could tag or remember certain replacement symbols that contain words of interest in the symbol pair denoted by the replacement symbol. Symbol pairs that include words such as “love” or “like,” for example, can result in the associated replacement symbol being tagged in a way indicating that the symbol pair expresses a positive emotion. Symbol pairs that include words such as “hate” or “despise,” by contrast, can result in the associated replacement symbol being tagged in a way indicating that the symbol pair expresses a negative emotion. For another example, symbol pairs containing nouns could be classified according to an area of interest, e.g., pets, family members, or products.

[0089] Using such tagging, based on the proximity of or sequence in which a pair or a set of tagged replacement symbols occurs in the compressed text, the process can infer a feeling expressed by the author about a particular entity or concept. For example, if a tagged replacement symbol indicating positive emotion is found in proximity to a tagged replacement system indicating pets, or just in proximity to a specific word like “dogs,” then the system can infer that the author likes pets, or likes dogs. Tracking specific descriptive words can provide a more granular understanding of the concepts expressed by a particular author (e.g., a love of small dogs as opposed to love of dogs generally without regard to size). In such embodiments, replacement symbols assigned during later passes can inherit attributes assigned to earlier-assigned replacement symbols that make up the symbol pair associated with the later-assigned replacement symbol. For example, if a P2 replacement symbol is tagged as being a “positive emotion” indicator, and a different P2 replacement symbol is tagged as being a “pets” indicator, then in Pass 3, a P3 replacement symbol that includes both P2 replacement symbols can be tagged as indicating “loves pets.” In this manner, repeated occurrences of higher-order replacement symbols among different authors can indicate a common interest. If works of two different authors contain that same P3 replacement symbol, they may share a common love of pets.

[0090] An implementation of one example embodiment is used in connection with an email archive. Analysis of emails drafted by a particular author or a working group is used to identify the author’s or the group’s preferences, such as products, restaurants, cities, etc. that the author likes. An author that frequently combines words expressing positive feelings with words referencing a particular type of animal might be identified as potentially owning such an animal. The example embodiment then provides the associations determined from such analysis to a system for making recommendations of products or advertising to the user.

[0091] It is intended that equivalents of the disclosed example embodiments and methods shall fall within the scope of the present disclosure or appended claims. It is intended that the disclosed example embodiments and methods, and equivalents thereof, can be modified while remaining within the scope of the present disclosure or appended claims.

[0092] In the foregoing Detailed Description, various features can be grouped together in several example embodiments for the purpose of streamlining the disclosure. This method of disclosure is not to be interpreted as reflecting an intention that any claimed embodiment requires more features than are expressly recited in the corresponding claim. Rather, as the appended claims reflect, inventive subject matter may lie in less than all features of a single disclosed example embodiment. Thus, the appended claims are hereby incorporated into the Detailed Description, with each claim standing on its own as a separately disclosed embodiment. Nor is the use of the term “embodiments” intended to imply any exclusion of features discussed in connection with other embodiments. Rather, features from one embodiment or group of embodiments can be combined with features from any other embodiment or group of embodiment unless clearly inconsistent therewith. Therefore, the present disclosure shall also be construed as implicitly disclosing any embodiment having any suitable set of one or more disclosed or claimed features (*i.e.*, a set of features that are neither incompatible nor mutually exclusive) that appear in the present disclosure or the appended claims, including those combinations of features that may not be explicitly disclosed in combination in any particular embodiment. It should be further noted that the scope of the appended claims does not necessarily encompass the whole of the subject matter disclosed herein.

[0093] For purposes of the present disclosure and appended claims, the conjunction “or” is to be construed inclusively (*e.g.*, “a dog or a cat” would be interpreted as “a dog, or a cat, or both”; *e.g.*, “a dog, a cat, or a mouse” would be interpreted as “a dog, or a cat, or a mouse, or any two, or all three”), unless: (i) it is explicitly stated otherwise, *e.g.*, by use of “either...or,” “only one of,” or similar language; or (ii) two or more of the listed alternatives are mutually exclusive within the particular context, in which case “or” would encompass only those combinations involving non-mutually-exclusive alternatives. For purposes of the present disclosure and appended claims, the words “comprising,” “including,” “having,” and variants thereof, wherever they appear, shall be construed as open-ended terminology, with the same meaning as if the phrase “at least” were appended after each instance thereof.

[0094] In the appended claims, if the provisions of 35 USC § 112 ¶ 6 are desired to be invoked in an apparatus claim, then the word “means” will appear in that apparatus claim. If those provisions are desired to be invoked in a method claim, the words “a step for” will appear in that method claim. Conversely, if the words “means” or “a step for” do not appear in a claim, then the provisions of 35 USC § 112 ¶ 6 are not intended to be invoked for that claim.

[0095] If any one or more disclosures are incorporated herein by reference and such incorporated disclosures conflict in part or whole with, or differ in scope from, the present disclosure, then to the extent of the conflict, broader disclosure, or broader definition of terms, the present disclosure controls. If such incorporated disclosures conflict in part or whole with one another, then to the extent of the conflict, the later-dated disclosure controls.

[0096] The Abstract is provided as required as an aid to those searching for specific subject matter within the patent literature. However, the Abstract is not intended to imply that any elements, features, or limitations recited therein are necessarily encompassed by any particular claim.

[0097] This paragraph contains computer code (Microsoft Visual Basic for Applications 7.1 in Microsoft Excel 2016) for one implementation of the compression process wherein the symbols operated on are words:

```
Sub f_clean_up_Final_Memory_text_independent_data()
'module to clean up Final Memory (text, independent)
    'create 1st formula for lower case conversion
    Range("B4") = "=LOWER(RC[-1])"
    'detect the end of the data set at column A to downward
    a = 3
    Do Until IsEmpty(Sheets("Imem").Cells(a, 1))
    a = a + 1
    Loop
    a = a - 1
    'copy the 1st formula to following rows
    Range("B4").Copy
    Range(Cells(5, 2), Cells(a, 2)).Select
    ActiveSheet.Paste
    'copy whole formula set at column B to column A
    Range(Cells(4, 2), Cells(a, 2)).Copy
    Range("A4").PasteSpecial Paste:=xlPasteValues
    'delete contents of column B
    Range(Cells(4, 2), Cells(a, 2)).ClearContents
    'replace some symbols with "(nothing)" or "(space)"
    Range(Cells(4, 1), Cells(a, 1)).Replace What:=";", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
    Range(Cells(4, 1), Cells(a, 1)).Replace What:=":", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
    Range(Cells(4, 1), Cells(a, 1)).Replace What:=",", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
    Range(Cells(4, 1), Cells(a, 1)).Replace What:=".", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
```

Range(Cells(4, 1), Cells(a, 1)).Replace What: "~?", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "!", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "+", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "-", Replacement: " ", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "—", Replacement: " ", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "=", Replacement: " ", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "/", Replacement: " / ", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "s", Replacement: " 's", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "'s", Replacement: " 's", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "''", Replacement: "", LookAt: xlPart,
SearchOrder: xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="[", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="]", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="{", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="}", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="(", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:=")", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:=">>", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="<<", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:=">", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="<", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="@", Replacement:="@ ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="%", Replacement:=" percent", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="\$", Replacement:="\$ ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="£", Replacement:="£ ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="#", Replacement:="# ", LookAt:=xlPart,
SearchOrder:=xlByRows

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:="~*", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:="~* ", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:="~*", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:="=", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:="•", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:="• ", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
Range(Cells(4, 1), Cells(a, 1)).Replace What:=" ", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows
```

```
End Sub
```

```
Sub a_for_mem01()
```

```
' a_for_mem01
```

```
'delete previous entry at lmem sheet
```

```
Range("C4:BYB1048576").ClearContents
```

```
'delete previous entry at pro01 sheet
```

```
Sheets("pro01").Select
```

```
Cells.Delete
```

```
'delete previous entry at pro02 sheet
```

```
Sheets("pro02").Select
```

```
Cells.Delete
```

```
'delete previous entry at pro03 sheet
```

```
Sheets("pro03").Select
```

```
Cells.Delete
```

```
'go back to lmem sheet
```

```
Sheets("lmem").Select
```

```
'determine Final Memory (text, independent) length downward
```

```
a = 3
```

```
Do Until IsEmpty(Cells(a, 1))
```

```
a = a + 1
```

```
Loop
```

```

a = a - 1
'copy Final Memory to pro01 sheet
Range("A3:A" & a & "").Copy
Sheets("pro01").Select
Range("A1").Select
ActiveSheet.Paste
'split words
Selection.TextToColumns Destination:=Range("A1"), DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=True, Semicolon:=True, Comma:=True,
Tab:=True, Space:=True, Other:=True, OtherChar:= _
    "-", FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1)),
TrailingMinusNumbers:=True
'find longest row
e = 1
d = 2
Do Until IsEmpty(Cells(d, 1))
    c = 1
    Do Until IsEmpty(Cells(d, c))
        c = c + 1
    Loop
    c = c - 1
    If c > e Then
        e = c
    End If
d = d + 1
Loop
e = e 'final number of largest column number
'put header
f = 1
Do Until f = e + 1
    Cells(1, f) = "" & f & ""
f = f + 1
Loop
'insert a column at left most
Columns("A:A").Insert Shift:=xlToRight

```

```

'determine the end of the data set downward
a = 1
Do Until IsEmpty(Cells(a, 2))
a = a + 1
Loop
a = a - 1
'pivot tabling (consolidating way)
ActiveWorkbook.PivotCaches.Create(SourceType:=xlConsolidation, SourceData:= _
    Array("pro01!R1C1:R" & a & "C" & e + 1 & "", "Item1"), Version:=xlPivotTableVersion14). _
    CreatePivotTable TableDestination:="pro01!R1C" & e + 4 & "", _
    TableName:="PivotTable1", DefaultVersion:=xlPivotTableVersion14
ActiveSheet.PivotTables("PivotTable1").DataPivotField.PivotItems("Count of Value").Position = 1
With ActiveSheet.PivotTables("PivotTable1").PivotFields("Value")
    .Orientation = xlRowField
    .Position = 2
End With
ActiveSheet.PivotTables("PivotTable1").PivotFields("Page1").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Column").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Row").Orientation = xlHidden
'detect the end of the pivot table downward
i = 3
Do Until IsEmpty(Cells(i, e + 4))
i = i + 1
Loop
i = i - 1
'copy the pivot table contents to right next
Range(Cells(3, e + 4), Cells(i - 1, e + 5)).Copy
Cells(3, e + 6).Select
ActiveSheet.Paste
'sort copied table by count larger to smaller
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(4, e + 7), Cells(i - 1, e +
7)), SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(3, e + 6), Cells(i - 1, e + 7))

```

```

.Header = xlYes
.Orientation = xlTopToBottom
.SortMethod = xlPinYin
.Apply
End With

'delete (blank) if any
If Cells(i - 1, e + 6) = "(blank)" Then
j = i - 2
Else: j = i - 1
End If

'copy sorted table into lmem sheet
Range(Cells(4, e + 6), Cells(j, e + 7)).Copy
Sheets("lmem").Select
Range("E4").PasteSpecial Paste:=xlPasteValues
End Sub

Sub b_replace_word_to_address()
'module to replace words to addresses

'delete previous entry at pro01 sheet
Sheets("pro01").Select
Cells.Delete

'delete previous entry at pro02 sheet
Sheets("pro02").Select
Cells.Delete

'delete previous entry at pro03 sheet
Sheets("pro03").Select
Cells.Delete

'go back to lmem sheet
Sheets("lmem").Select

'determine Final Memory (text, independent) length downward
a = 3
Do Until IsEmpty(Cells(a, 1))
a = a + 1
Loop
a = a - 1

```

```
'copy Final Memory to lmem sheet
Range("A3:A" & a & "").Copy
Sheets("pro01").Select
Range("A1").Select
ActiveSheet.Paste
'split words
Selection.TextToColumns Destination:=Range("A1"), DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=True, Semicolon:=True, Comma:=True,
Tab:=True, Space:=True, Other:=True, OtherChar:= _
    "-", FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1)),
TrailingMinusNumbers:=True
'go back to lmem sheet
Sheets("lmem").Select
'determine the end of Memory01 data downward
b = 3
Do Until IsEmpty(Cells(b, 5))
b = b + 1
Loop
b = b - 1
'copy Memory01 data to pro02
Range(Cells(2, 5), Cells(b, 6)).Copy
Sheets("pro02").Select
Cells(2, 1).PasteSpecial Paste:=xlPasteValues
'create address column
c = 4
Do Until IsEmpty(Cells(c, 1))
    Cells(c, 3) = ("R" & c & "C5")
c = c + 1
Loop
'go to pro01 sheet
Sheets("pro01").Select
'determine the end of data set downward
d = 1
Do Until IsEmpty(Cells(d, 1))
d = d + 1
```

```

Loop
d = d - 1
'create initial length number
Sheets("pro03").Cells(1, 1) = 1
g = Sheets("pro03").Cells(1, 1)
'go to sheets pro01
Sheets("pro01").Select
'determine the max length of the data set rightward
e = 2
Do Until IsEmpty(Sheets("pro01").Cells(e, 1))
    f = 1
    Do Until IsEmpty(Sheets("pro01").Cells(e, f))
        f = f + 1
    Loop
    f = f - 1
    If f >= g Then
        Sheets("pro03").Cells(1, 1) = f
        g = Sheets("pro03").Cells(1, 1)
    End If
e = e + 1
Loop
h = g 'rightward max length value
'replace words to address numbers
k = 4
Do Until IsEmpty(Sheets("pro02").Cells(k, 1))
    'define address name
    i = Sheets("pro02").Cells(k, 3)
    'define word name
    j = Sheets("pro02").Cells(k, 1)
    'actual replacement from word to address
    Range(Cells(2, 1), Cells(d, h)).Replace What:="" & j & "", Replacement:="" & i & "",
LookAt:=xlWhole, SearchOrder:=xlByRows
    k = k + 1
Loop
'combined addresses making

```

```

p = 2
Do Until IsEmpty(Sheets("pro01").Cells(p, 1))
    'delete previous entry at pro03 sheet
    Sheets("pro03").Cells.Delete
    'make all combined address sequence
    Sheets("pro03").Cells(1, 1) = Sheets("pro01").Cells(p, 1)
    m = 2
    Do Until IsEmpty(Sheets("pro01").Cells(p, m))
        Sheets("pro03").Cells(m, 1).FormulaR1C1 = "=" & "pro03"!R" & m - 1 & "C1&" & "" & "pro01"!R" &
p & "C" & m & ""
        m = m + 1
    Loop
    'detect the end of the data set
    n = 1
    Do Until IsEmpty(Sheets("pro03").Cells(n, 1))
        n = n + 1
    Loop
    n = n - 1
    'convert the sequence from formula to value and paste into pro01
    Sheets("pro03").Cells(n, 1).Copy
    Sheets("pro01").Cells(p, h + 2).PasteSpecial Paste:=xlPasteValues
p = p + 1
Loop
'detect the end of combined address data set
q = 2
Do Until IsEmpty(Sheets("pro01").Cells(q, h + 2))
    q = q + 1
Loop
q = q - 1
'copy the combined address set to lmem sheet
Range(Cells(2, h + 2), Cells(q, h + 2)).Copy
Sheets("lmem").Select
Range("C4").Select
ActiveSheet.Paste
End Sub

```

```

Sub c_find_2_same_sequence()
'core of this program set!!!
  For x = 2 To 1000
    y = 2 * x + 3
    'delete previous entry at pro01sheet
    Sheets("pro01").Select
    Cells.Delete
    'delete previous entry at pro03 sheet
    Sheets("pro03").Select
    Cells.Delete
    'put tile "combined"
    Cells(1, 1) = "combined"
    'go back to lmem sheet
    Sheets("lmem").Select
    'detect the end of column C data end downward
    a = 3
    Do Until IsEmpty(Sheets("lmem").Cells(a, 3))
      a = a + 1
    Loop
    a = a - 1
    'copy it to pro01 sheet
    Range(Cells(3, 3), Cells(a, 3)).Copy
    Sheets("pro01").Select
    Range("A1").PasteSpecial Paste:=xlPasteValues
    'split the data set
    Selection.TextToColumns Destination:=Range("A1"), DataType:=xlDelimited, _
      TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=True, Tab:=True, _
      Semicolon:=True, Comma:=True, Space:=True, Other:=True, OtherChar:= _
      "-", FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 1), Array(4, 1)), _
      TrailingMinusNumbers:=True
    'put initial g value
    Sheets("pro03").Cells(1, 1) = 1
    g = Sheets("pro03").Cells(1, 1)
    'determine the max length of the data set rightward
    e = 2

```

```

Do Until IsEmpty(Sheets("pro01").Cells(e, 1))
    f = 1
    Do Until IsEmpty(Sheets("pro01").Cells(e, f))
        f = f + 1
    Loop
    f = f - 1
    If f >= g Then
        Sheets("pro03").Cells(1, 1) = f
        g = Sheets("pro03").Cells(1, 1)
    End If
e = e + 1
Loop
'place paired addresses
i = 2
Do Until IsEmpty(Sheets("pro01").Cells(i, 1))
    t = 1
    Do Until IsEmpty(Sheets("pro01").Cells(i, t))
        t = t + 1
    Loop
    t = t - 1
    If t > 1 Then
        h = 2
        Do Until IsEmpty(Sheets("pro01").Cells(i, h))
            Cells(i, (g + 2) + (h - 2)) = "=R" & i & "C" & h - 1 & "&"" ""&R" & i & "C" & h & ""
            h = h + 1
        Loop
    Else: Cells(i, g + 2) = "zzz"
    End If
i = i + 1
Loop
'put header
f = 1
Do Until f = g
    Cells(1, g + 1 + f) = "" & f & ""
    f = f + 1

```

```

Loop
'detect the end of the data set downward
j = 1
Do Until IsEmpty(Cells(j, g + 2))
j = j + 1
Loop
j = j - 1
'pivot tabling consolidating way
ActiveWorkbook.PivotCaches.Create(SourceType:=xlConsolidation, SourceData:= _
    Array("pro01!R1C" & g + 1 & ":R" & j & "C" & g + 1 + g - 1 & "", "Item1"), Version:=6). _
    CreatePivotTable TableDestination:="pro01!R1C" & g + 1 + g - 1 + 2 & "", TableName:= _
        "PivotTable1", DefaultVersion:=6
ActiveSheet.PivotTables("PivotTable1").DataPivotField.PivotItems("Count of Value").Position = 1
ActiveSheet.PivotTables("PivotTable1").PivotFields("Page1").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Column").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Row").Orientation = xlHidden
With ActiveSheet.PivotTables("PivotTable1").PivotFields("Value")
    .Orientation = xlRowField
    .Position = 1
End With
'delete 2 empty rows
Range(Cells(1, 2 * g + 2), Cells(2, 2 * g + 3)).Delete Shift:=xlUp
'detect the end of the pivot table
p = 1
Do Until IsEmpty(Sheets("pro01").Cells(p, 2 * g + 3))
p = p + 1
Loop
p = p - 1
If Cells(p, 2 * g + 2) = "Grand Total" Then
p = p - 1
End If
'copy the pivot table to right ward
If Cells(p, 2 * g + 2) = "zzz" Then
    Range(Cells(1, 2 * g + 2), Cells(p - 1, 2 * g + 3)).Copy
Else: Range(Cells(1, 2 * g + 2), Cells(p, 2 * g + 3)).Copy

```

```

End If
Cells(1, 2 * g + 5).Select
ActiveSheet.Paste
Selection.Columns.AutoFit
'sort by decending order
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, 2 * g + 6),
Cells(p, 2 * g + 6)), SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:= _
    xLSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(1, 2 * g + 5), Cells(p, 2 * g + 6))
    .Header = xlYes
    .MatchCase = False
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply
End With
'rename
q = 2 * g + 5
'detect the end of the data set downward
aa = 1
Do Until IsEmpty(Cells(aa, q + 1))
    aa = aa + 1
Loop
aa = aa - 1
'define frequency limit number
w = Sheets("para").Cells(x + 4, 2)
'find the beginning of less than frequency X cell
If aa = 2 Then
    If w >= 2 Then
        If Cells(aa, q + 1) <= w - 1 Then
            r = 2
        Else: r = 3
        End If
    Else: r = 3

```

```

End If
End If
If aa > 2 Then
    If w >= 2 Then
        For bb = 1 To aa
            If Cells(bb, q + 1) <= w - 1 Then
                cc = bb
                bb = aa
            Else: cc = aa + 1
            End If
        Next bb
        r = cc
    Else: r = aa + 1
    End If
End If
'If x = 2 Then
'Stop
'End If
If r > 2 Then
    'find the end of the frequency X cell
    s = r
    Do Until IsEmpty(Cells(s, q + 1))
        s = s + 1
    Loop
    s = s - 1
    'delete unneeded frequencies
    If w >= 2 Then
        Range(Cells(r, q), Cells(s + 500, q + 1)).Delete Shift:=xlUp '---- 500 is just extra buffer
    End If
    'put headers for layer ID finding table
    Cells(1, q + 3) = "1st C Position"
    Cells(1, q + 4) = "Space Position"
    Cells(1, q + 5) = "2nd C Position"
    Cells(1, q + 6) = "End Position"
    Cells(1, q + 7) = "1st layer#"

```

```

Cells(1, q + 8) = "2nd Layer#"
'enter 1st row's formula
Cells(2, q + 3) = "=Find("""C""", rc[-3], 1)"
Cells(2, q + 4) = "=find("""", rc[-4], 1)"
Cells(2, q + 5) = "=find("""C""", rc[-5], rc[-1])"
Cells(2, q + 6) = "=len(rc[-6])"
Cells(2, q + 7) = "=mid(rc[-7], rc[-4]+1, rc[-3]-(rc[-4]+1))"
Cells(2, q + 8) = "=mid(rc[-8], rc[-3]+1, rc[-2]-rc[-3])"
'detect the end of the data set
t = 1
Do Until IsEmpty(Sheets("pro01").Cells(t, q))
t = t + 1
Loop
t = t - 1
'copy the 1st row to the other rows
If t > 2 Then
    Range(Cells(2, q + 3), Cells(2, q + 8)).Copy
    Range(Cells(3, q + 3), Cells(t, q + 3)).PasteSpecial Paste:=xlAll
End If
'convert layer numbers from formula to text
Range(Cells(1, q + 7), Cells(t, q + 8)).Select
Selection.Copy
Selection.PasteSpecial Paste:=xlPasteValues
'convert layer numbers from text to value
Range(Cells(1, q + 7), Cells(t, q + 7)).Select
Selection.TextToColumns Destination:=Cells(1, q + 7), DataType:=xlFixedWidth, _
    OtherChar:="-", FieldInfo:=Array(0, 1), TrailingMinusNumbers:=True
Range(Cells(1, q + 8), Cells(t, q + 8)).Select
Selection.TextToColumns Destination:=Cells(1, q + 8), DataType:=xlFixedWidth, _
    OtherChar:="-", FieldInfo:=Array(0, 1), TrailingMinusNumbers:=True
'sort the data set by 1st layer # then 2nd layer#
If x > 2 Then
    Range(Cells(1, q), Cells(t, q + 8)).Select
    ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear

```

```

ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, q +
7), Cells(t, q + 7)), SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, q +
8), Cells(t, q + 8)), SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(1, q), Cells(t, q + 8))
    .Header = xlYes
    .MatchCase = False
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply
End With
'detect the end of the data set
aa = 1
Do Until IsEmpty(Cells(aa, q + 7))
    aa = aa + 1
Loop
aa = aa - 1
'detect the end of smaller layer number at 1st layer#
For gg = 2 To aa
    If Cells(gg, q + 7) = 2 * x + 1 Then
        dd = gg
        gg = aa
    End If
Next gg
If dd = Empty Then
    dd = 3000000000000#
End If
'detect the end of smaller layer number at 2nd layer#
For hh = 2 To aa
    If Cells(hh, q + 8) = 2 * x + 1 Then
        ee = hh
        hh = aa
    End If
Next hh

```

```

If ee = Empty Then
ee = 3000000000000#
End If
'find smaller one between dd and ee
If dd >= ee Then
    cc = ee
Else: cc = dd
End If
'delete previous layer only rows
If cc > 2 Then
    Range(Cells(2, q), Cells(cc - 1, q + 8)).Delete
End If
'detect the end of the data set
t = 1
Do Until IsEmpty(Sheets("pro01").Cells(t, q))
t = t + 1
Loop
t = t - 1
'if the data set is empty, screw up t value and force to stop this program
If t = 1 Then
t = 0
End If
'sort the data set by count of paired (frequency of pairs)
Range(Cells(1, q), Cells(t, q + 8)).Select
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, q +
1), Cells(t, q + 1)), SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(1, q), Cells(t, q + 8))
    .Header = xlYes
    .MatchCase = False
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply
End With

```

```

End If
'detect the end of the data set
t = 1
Do Until IsEmpty(Sheets("pro01").Cells(t, q))
t = t + 1
Loop
t = t - 1
'copy the sorted table into lmem sheet
Range(Cells(2, q), Cells(t, q + 1)).Copy
Sheets("lmem").Select
Cells(4, y).Select '<-----Here!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ActiveSheet.Paste
'-----module to replce 2 words cells to one-----
'delete previoue entry at pro01sheet
Sheets("pro01").Select
Cells.Delete
'go back to lmem sheet
Sheets("lmem").Select
'determine the end of the data set at Final Memory (associated) column
a = 3
Do Until IsEmpty(Sheets("lmem").Cells(a, 3))
a = a + 1
Loop
a = a - 1
'copy Final Memory (associated) to pro01 sheet
Range(Cells(3, 3), Cells(a, 3)).Copy
Sheets("pro01").Cells(1, 1).PasteSpecial Paste:=xlPasteValues
'go to pro01 sheet
Sheets("pro01").Select
'adjust column A column width
Columns("A:A").ColumnWidth = 240
'go back to lmem sheet
Sheets("lmem").Select
'determine the end of the data set at MemoryXX
b = 3

```

```

Do Until IsEmpty(Sheets("lmem").Cells(b, y)) '-----9 must be variable
b = b + 1
Loop
b = b - 1
'copy MemoryXX to pro01 sheet
Range(Cells(3, y), Cells(b, y + 1)).Copy '-----9 and 10 must be variable
Sheets("pro01").Cells(1, 3).PasteSpecial Paste:=xlPasteValues
'go to pro01 sheet
Sheets("pro01").Select
'adjust column A column width
Columns("C:D").Columns.AutoFit
'put tile
Cells(1, 5) = "MemoryXX address"
'detect the end of MemoryXX end downward
c = 1
Do Until IsEmpty(Sheets("pro01").Cells(c, 3))
c = c + 1
Loop
c = c - 1
'put MemoryXX address
d = 2
Do Until IsEmpty(Sheets("pro01").Cells(d, 3))
    Cells(d, 5) = "R" & d + 2 & "C" & y & "" '-----9 must be variable
d = d + 1
Loop
'detect the end of the data set at column A
e = 1
Do Until IsEmpty(Sheets("pro01").Cells(e, 1))
e = e + 1
Loop
e = e - 1
'go to home position
Range("A1").Select
'replce 2 words to one
f = 2

```

```

Do Until IsEmpty(Sheets("pro01").Cells(f, 5))
    g = Sheets("pro01").Cells(f, 3)
    h = Sheets("pro01").Cells(f, 5)
    Range("A2:A" & e & "").Replace What:="" & g & "", Replacement:="" & h & "",
LookAt:=xlPart, SearchOrder:=xlByRows
    f = f + 1
    Loop
    'detect the end of the data set at column A
    i = 1
    Do Until IsEmpty(Sheets("pro01").Cells(i, 1))
        i = i + 1
    Loop
    i = i - 1
    'copy column A to lmem sheets
    Range(Cells(1, 1), Cells(i, 1)).Copy
    Sheets("lmem").Select
    Range("C3").PasteSpecial Paste:=xlPasteValues
Else: x = 1000
End If
Next x
'go to lmem sheet
Sheets("lmem").Select
Range("A1").Select
End Sub
Sub d_text_to_formula()
' formulatext Macro
    For c = 1 To 1000
        'case separation Final memory cooking and Mem02< cooking
        If c = 1 Then
            d = 3
        Else: d = 2 * c + 3
        End If
        'detect the end of the data set
        a = 3
        Do Until IsEmpty(Sheets("lmem").Cells(a, d))

```

```

a = a + 1
Loop
a = a - 1
'replace space with & " "&
Range(Cells(4, d), Cells(a, d)).Replace What:=" ", Replacement:="&" "&", LookAt:=xlPart,
SearchOrder:=xlByRows
'delete previous entry at pro01 sheet
Sheets("pro01").Select
Cells.Delete
'copy lmem data to pro01 sheet
Sheets("lmem").Select
Range(Cells(3, d), Cells(a, d)).Copy
Sheets("pro01").Select
Range("A1").Select
ActiveSheet.Paste
'adjust column width
Selection.Columns.AutoFit
'put tile to column B
Range("B1") = "with ="
'detect the end of the data set downward
b = 1
Do Until IsEmpty(Sheets("pro01").Cells(b, 1))
b = b + 1
Loop
b = b - 1
If Not b = 1 Then
    'put 1st formula
    Range("B2").FormulaR1C1 = "="&RC[-1]"
    Range("B2").Copy
    Range(Cells(2, 2), Cells(b, 2)).Select
    ActiveSheet.Paste
    'select whole column B
    Columns("B:B").Select
    'convert from formula to values
    Selection.Copy

```

```

Selection.PasteSpecial Paste:=xlPasteValues
'convert from text to renewed formula
Selection.TextToColumns Destination:=Range("B1"), DataType:=xlFixedWidth, _
    OtherChar:="-", FieldInfo:=Array(0, 1), TrailingMinusNumbers:=True
'copy the renewed formula to lmem sheet
Range(Cells(2, 2), Cells(b, 2)).Copy
Sheets("lmem").Select
Cells(4, d).Select
ActiveSheet.Paste
Else: c = 1000
End If
Next c
'go back to lmem sheet
Sheets("lmem").Select
Range("A1").Select
End Sub
Sub e_formula_to_text()
'module to convert formula to text so that addresses can be seen
For b = 1 To 1000
    If b = 1 Then
        c = 3
    Else: c = 2 * b + 3
    End If
    'detect the end of the data set downward
    a = 3
    Do Until IsEmpty(Sheets("lmem").Cells(a, c))
        a = a + 1
    Loop
    a = a - 1
    If Not a = 3 Then
        'select target data set
        Range(Cells(4, c), Cells(a, c)).Select
        'change cell contents display from results to formula
        ActiveWindow.DisplayFormulas = True
        'replace "=" with "(nothing)"

```

```

        Selection.Replace What:="", Replacement="", LookAt:=xlPart, SearchOrder:=xlByRows
        'replace "&" with "(space)"
        Selection.Replace What:=""&" "&", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows
        'change back the cell contents from formula to results
        ActiveWindow.DisplayFormulas = False
    Else: b = 1000
    End If
Next b
'go back to lmem sheet
Sheets("lmem").Select
Range("A1").Select
End Sub

```

[0098] This paragraph contains computer code (Microsoft Visual Basic for Applications 7.1 in Microsoft Excel 2016) for one implementation of the compression process wherein the symbols operated on are alphanumeric character:

```

Sub f_clean_up_Final_Memory_text_independent_data()
'module to clean up Final Memory (text, independent)
    'create 1st formula for lower case conversion
    Range("B4") = "=LOWER(RC[-1])"
    'detect the end of the data set at column A to downward
    a = 3
    Do Until IsEmpty(Sheets("lmem").Cells(a, 1))
    a = a + 1
    Loop
    a = a - 1
    'copy the 1st formula to following rows
    Range("B4").Copy
    Range(Cells(5, 2), Cells(a, 2)).Select
    ActiveSheet.Paste
    'copy whole formula set at column B to column A
    Range(Cells(4, 2), Cells(a, 2)).Copy
    Range("A4").PasteSpecial Paste:=xlPasteValues
    'delete contents of column B
    Range(Cells(4, 2), Cells(a, 2)).ClearContents

```

'replace some symbols with "(nothing)" or "(space)"

Range(Cells(4, 1), Cells(a, 1)).Replace What:=";", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:=":", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="," , Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:=".", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="~?", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="!", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="+", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="-", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="—", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="-", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="/", Replacement:=" / ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="s", Replacement:=" 's", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="'s", Replacement:=" 's", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="""", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="'", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="""", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="'", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"[", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"]", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What="{", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"}", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"(", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"){", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:">>", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"<<", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:">", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"<", Replacement:"", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:@"", Replacement:@" ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:"%", Replacement:=" percent", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="\$", Replacement:="\$ ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="£", Replacement:="£ ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="#", Replacement:="# ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="~*", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="~* ", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What: "=", Replacement:="= ", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="•", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:="• ", Replacement:="", LookAt:=xlPart,
SearchOrder:=xlByRows

Range(Cells(4, 1), Cells(a, 1)).Replace What:=" ", Replacement:=" ", LookAt:=xlPart,
SearchOrder:=xlByRows

End Sub

Sub a_for_mem01()

' a_for_mem01

'delete previous entry at lmem sheet

Range("C4:BYB1048576").ClearContents

'delete previous entry at pro01 sheet

Sheets("pro01").Select

Cells.Delete

'delete previous entry at pro02 sheet

Sheets("pro02").Select

Cells.Delete

'delete previous entry at pro03 sheet

Sheets("pro03").Select

Cells.Delete

'go back to lmem sheet

Sheets("lmem").Select

'determine Final Memory (text, independent) length downward

```

a = 3
Do Until IsEmpty(Cells(a, 1))
a = a + 1
Loop
a = a - 1
'copy Final Memory to lmem sheet
Range("A3:A" & a & "").Copy
Sheets("pro01").Select
Range("A1").Select
ActiveSheet.Paste
'split alphabet
Columns("A:A").TextToColumns Destination:=Range("A1"), DataType:=xlFixedWidth, FieldInfo:= _
    Array(Array(0, 1), Array(1, 1), Array(2, 1), Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1),
Array(7, 1), Array(8, 1), Array(9, 1) _
    , Array(10, 1), Array(11, 1), Array(12, 1), Array(13, 1), Array(14, 1), Array(15, 1), Array(16, 1),
Array(17, 1), Array(18, 1), Array(19, 1) _
    , Array(20, 1), Array(21, 1), Array(22, 1), Array(23, 1), Array(24, 1), Array(25, 1), Array(26, 1),
Array(27, 1), Array(28, 1), Array(29, 1) _
    , Array(30, 1), Array(31, 1), Array(32, 1), Array(33, 1), Array(34, 1), Array(35, 1), Array(36, 1),
Array(37, 1), Array(38, 1), Array(39, 1) _
    , Array(40, 1), Array(41, 1), Array(42, 1), Array(43, 1), Array(44, 1), Array(45, 1), Array(46, 1),
Array(47, 1), Array(48, 1), Array(49, 1) _
    , Array(50, 1), Array(51, 1), Array(52, 1), Array(53, 1), Array(54, 1), Array(55, 1), Array(56, 1),
Array(57, 1), Array(58, 1), Array(59, 1), Array(60, 1)) _
    , TrailingMinusNumbers:=True
'find the end of the data set downward
b = 1
Do Until IsEmpty(Cells(b, 1))
b = b + 1
Loop
b = b - 1
'create initial length number
Sheets("pro03").Cells(1, 1) = 1
m = Sheets("pro03").Cells(1, 1)
'go to sheets pro01

```

```

Sheets("pro01").Select
'find row number which has the longest column length
e = 2
Do Until IsEmpty(Cells(e, 1))
    c = 1
    Do Until IsEmpty(Cells(e, c))
        c = c + 1
    Loop
    c = c - 1
    If c >= m Then
        Sheets("pro03").Cells(1, 1) = c
        m = Sheets("pro03").Cells(1, 1)
    End If
e = e + 1
Loop
e = m
'clear 1st row
Rows("1:1").Clear
'put header
f = 1
Do Until f = e + 1
    Cells(1, f) = "" & f & ""
    f = f + 1
Loop
'insert a column at left most
Columns("A:A").Insert Shift:=xlToRight
'determine the end of the data set downward
a = 1
Do Until IsEmpty(Cells(a, 2))
    a = a + 1
Loop
a = a - 1
'pivot tabling (consolidating way)
ActiveWorkbook.PivotCaches.Create(SourceType:=xlConsolidation, SourceData:= _
    Array("pro01!R1C1:R" & a & "C" & e + 1 & "", "Item1"), Version:=xlPivotTableVersion14). _

```

```

        CreatePivotTable TableDestination:="pro01"!R1C" & e + 4 & "", _
        TableName:="PivotTable1", DefaultVersion:=xlPivotTableVersion14
ActiveSheet.PivotTables("PivotTable1").DataPivotField.PivotItems("Count of Value").Position = 1
With ActiveSheet.PivotTables("PivotTable1").PivotFields("Value")
    .Orientation = xlRowField
    .Position = 2
End With
ActiveSheet.PivotTables("PivotTable1").PivotFields("Page1").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Column").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Row").Orientation = xlHidden
'detect the end of the pivot table downward
i = 3
Do Until IsEmpty(Cells(i, e + 4))
    i = i + 1
Loop
i = i - 1
'not to use (blank)
If Cells(i - 1, e + 4) = "(blank)" Then
    j = i - 2
    Else: j = i - 1
End If
'copy the pivot table contents to right next
Range(Cells(3, e + 4), Cells(j, e + 5)).Copy
Cells(3, e + 6).Select
ActiveSheet.Paste
'sort copied table by count larger to smaller
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(4, e + 7), Cells(j, e + 7)),
SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(3, e + 6), Cells(j, e + 7))
    .Header = xlYes
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply

```

```

End With
'copy sorted table into lmem sheet
Range(Cells(4, e + 6), Cells(j, e + 7)).Copy
Sheets("lmem").Select
Range("E4").PasteSpecial Paste:=xlPasteValues
End Sub
Sub b_replace_word_to_address()
'module to replace words to addresses
'delete previous entry at pro01 sheet
Sheets("pro01").Select
Cells.Delete
'delete previous entry at pro02 sheet
Sheets("pro02").Select
Cells.Delete
'delete previous entry at pro03 sheet
Sheets("pro03").Select
Cells.Delete
'go back to lmem sheet
Sheets("lmem").Select
'determine Final Memory (text, independent) length downward
a = 3
Do Until IsEmpty(Cells(a, 1))
a = a + 1
Loop
a = a - 1
'copy Final Memory to lmem sheet
Range("A3:A" & a & "").Copy
Sheets("pro01").Select
Range("A1").Select
ActiveSheet.Paste
'split alphabet
Columns("A:A").TextToColumns Destination:=Range("A1"), DataType:=xlFixedWidth, _
    OtherChar:="-", FieldInfo:=Array(Array(0, 1), Array(1, 1), Array(2, 1), Array(3, 1), Array(4, 1),
Array(5, 1), Array(6, 1), Array(7, 1), Array(8, 1), Array(9, 1) _

```

```

, Array(10, 1), Array(11, 1), Array(12, 1), Array(13, 1), Array(14, 1), Array(15, 1), Array(16, 1),
Array(17, 1), Array(18, 1), Array(19, 1) _
, Array(20, 1), Array(21, 1), Array(22, 1), Array(23, 1), Array(24, 1), Array(25, 1), Array(26, 1),
Array(27, 1), Array(28, 1), Array(29, 1) _
, Array(30, 1), Array(31, 1), Array(32, 1), Array(33, 1), Array(34, 1), Array(35, 1), Array(36, 1),
Array(37, 1), Array(38, 1), Array(39, 1) _
, Array(40, 1), Array(41, 1), Array(42, 1), Array(43, 1), Array(44, 1), Array(45, 1), Array(46, 1),
Array(47, 1), Array(48, 1), Array(49, 1) _
, Array(50, 1), Array(51, 1), Array(52, 1), Array(53, 1), Array(54, 1), Array(55, 1), Array(56, 1),
Array(57, 1), Array(58, 1), Array(59, 1), Array(60, 1)) _
, TrailingMinusNumbers:=True
'go back to lmem sheet
Sheets("lmem").Select
'determine the end of Memory01 data downward
b = 3
Do Until IsEmpty(Cells(b, 5))
b = b + 1
Loop
b = b - 1
'copy Memory01 data to pro02
Range(Cells(2, 5), Cells(b, 6)).Copy
Sheets("pro02").Select
Cells(2, 1).PasteSpecial Paste:=xlPasteValues
'create address column
c = 4
Do Until IsEmpty(Cells(c, 1))
Cells(c, 3) = ("R" & c & "C5")
c = c + 1
Loop
'go to pro01 sheet
Sheets("pro01").Select
'determine the end of data set downward
d = 1
Do Until IsEmpty(Cells(d, 1))
d = d + 1

```

```

Loop
d = d - 1
'create initial length number
Sheets("pro03").Cells(1, 1) = 1
g = Sheets("pro03").Cells(1, 1)
'go to sheets pro01
Sheets("pro01").Select
'determine the max length of the data set rightward
e = 2
Do Until IsEmpty(Sheets("pro01").Cells(e, 1))
    f = 1
    Do Until IsEmpty(Sheets("pro01").Cells(e, f))
        f = f + 1
    Loop
    f = f - 1
    If f >= g Then
        Sheets("pro03").Cells(1, 1) = f
        g = Sheets("pro03").Cells(1, 1)
    End If
e = e + 1
Loop
h = g 'rightward max length value
'replace words to address numbers
k = 4
Do Until IsEmpty(Sheets("pro02").Cells(k, 1))
    'define address name
    i = Sheets("pro02").Cells(k, 3)
    'define word name
    j = Sheets("pro02").Cells(k, 1)
    'actual replacement from word to address
    Range(Cells(1, 1), Cells(d, h)).Replace What:="" & j & "", Replacement:="" & i & "",
LookAt:=xlWhole, SearchOrder:=xlByRows
    k = k + 1
Loop
'combined addresses making

```

```

p = 2
Do Until IsEmpty(Sheets("pro01").Cells(p, 1))
    'delete previous entry at pro03 sheet
    Sheets("pro03").Cells.Delete
    'make all combined address sequence
    Sheets("pro03").Cells(1, 1) = Sheets("pro01").Cells(p, 1)
    m = 2
    Do Until IsEmpty(Sheets("pro01").Cells(p, m))
        Sheets("pro03").Cells(m, 1).FormulaR1C1 = "=" & "pro03"!R" & m - 1 & "C1&" & "" & "pro01"!R" &
p & "C" & m & ""
        m = m + 1
    Loop
    'detect the end of the data set
    n = 1
    Do Until IsEmpty(Sheets("pro03").Cells(n, 1))
        n = n + 1
    Loop
    n = n - 1
    'convert the sequence from formula to value and paste into pro01
    Sheets("pro03").Cells(n, 1).Copy
    Sheets("pro01").Cells(p, h + 2).PasteSpecial Paste:=xlPasteValues
    p = p + 1
Loop
'detect the end of combined address data set
q = 2
Do Until IsEmpty(Sheets("pro01").Cells(q, h + 2))
    q = q + 1
Loop
q = q - 1
'copy the combined address set to lmem sheet
Range(Cells(2, h + 2), Cells(q, h + 2)).Copy
Sheets("lmem").Select
Range("C4").Select
ActiveSheet.Paste
End Sub

```

```

Sub c_find_2_same_sequence()
'core of this program set!!!
  For x = 2 To 1000
    y = 2 * x + 3
    'delete previous entry at pro01sheet
    Sheets("pro01").Select
    Cells.Delete
    'delete previous entry at pro03 sheet
    Sheets("pro03").Select
    Cells.Delete
    'put tile "combined"
    Cells(1, 1) = "combined"
    'go back to lmem sheet
    Sheets("lmem").Select
    'detect the end of column C data end downward
    a = 3
    Do Until IsEmpty(Sheets("lmem").Cells(a, 3))
      a = a + 1
    Loop
    a = a - 1
    'copy it to pro01 sheet
    Range(Cells(3, 3), Cells(a, 3)).Copy
    Sheets("pro01").Select
    Range("A1").PasteSpecial Paste:=xlPasteValues
    'split the data set
    Selection.TextToColumns Destination:=Range("A1"), DataType:=xlDelimited, _
      TextQualifier:=xlDoubleQuote, ConsecutiveDelimiter:=True, Tab:=True, _
      Semicolon:=True, Comma:=True, Space:=True, Other:=True, OtherChar:= _
      "-", FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 1), Array(4, 1)), _
      TrailingMinusNumbers:=True
    'put initial g value
    Sheets("pro03").Cells(1, 1) = 1
    g = Sheets("pro03").Cells(1, 1)
    'determine the max length of the data set rightward
    e = 2
  
```

```

Do Until IsEmpty(Sheets("pro01").Cells(e, 1))
    f = 1
    Do Until IsEmpty(Sheets("pro01").Cells(e, f))
        f = f + 1
    Loop
    f = f - 1
    If f >= g Then
        Sheets("pro03").Cells(1, 1) = f
        g = Sheets("pro03").Cells(1, 1)
    End If
e = e + 1
Loop
'place paired addresses
i = 2
Do Until IsEmpty(Sheets("pro01").Cells(i, 1))
    t = 1
    Do Until IsEmpty(Sheets("pro01").Cells(i, t))
        t = t + 1
    Loop
    t = t - 1
    If t > 1 Then
        h = 2
        Do Until IsEmpty(Sheets("pro01").Cells(i, h))
            Cells(i, (g + 2) + (h - 2)) = "=R" & i & "C" & h - 1 & "&"" ""&R" & i & "C" & h & ""
            h = h + 1
        Loop
    Else: Cells(i, g + 2) = "zzz"
    End If
i = i + 1
Loop
'put header
f = 1
Do Until f = g
    Cells(1, g + 1 + f) = "" & f & ""
    f = f + 1

```

```

Loop
'detect the end of the data set downward
j = 1
Do Until IsEmpty(Cells(j, g + 2))
j = j + 1
Loop
j = j - 1
'pivot tabling consolidating way
ActiveWorkbook.PivotCaches.Create(SourceType:=xlConsolidation, SourceData:= _
    Array("pro01!R1C" & g + 1 & ":R" & j & "C" & g + 1 + g - 1 & "", "Item1"),
Version:=xlPivotTableVersion14). _
    CreatePivotTable TableDestination:="pro01!R1C" & g + 1 + g - 1 + 2 & "", TableName:= _
        "PivotTable1", DefaultVersion:=xlPivotTableVersion14
ActiveSheet.PivotTables("PivotTable1").DataPivotField.PivotItems("Count of Value").Position = 1
ActiveSheet.PivotTables("PivotTable1").PivotFields("Page1").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Column").Orientation = xlHidden
ActiveSheet.PivotTables("PivotTable1").PivotFields("Row").Orientation = xlHidden
With ActiveSheet.PivotTables("PivotTable1").PivotFields("Value")
    .Orientation = xlRowField
    .Position = 1
End With
'delete 2 empty rows
Range(Cells(1, 2 * g + 2), Cells(2, 2 * g + 3)).Delete Shift:=xlUp
'detect the end of the pivot table
p = 1
Do Until IsEmpty(Sheets("pro01").Cells(p, 2 * g + 3))
p = p + 1
Loop
p = p - 1
If Cells(p, 2 * g + 2) = "Grand Total" Then
    p = p - 1
Else: p = p
End If
'copy the pivot table to right ward
If Not Cells(p, 2 * g + 2) = "zzz" Then

```

```

        Range(Cells(1, 2 * g + 2), Cells(p, 2 * g + 3)).Copy
    Else: Range(Cells(1, 2 * g + 2), Cells(p - 1, 2 * g + 3)).Copy
    End If
    Cells(1, 2 * g + 5).Select
    ActiveSheet.Paste
    Selection.Columns.AutoFit
    'sort by decending order
    ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear
    ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, 2 * g + 6),
Cells(p, 2 * g + 6)), SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:= _
        xlSortNormal
    With ActiveWorkbook.Worksheets("pro01").Sort
        .SetRange Range(Cells(1, 2 * g + 5), Cells(p, 2 * g + 6))
        .Header = xlYes
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
    'rename
    q = 2 * g + 5
    'detect the end of the data set downward
    aa = 1
    Do Until IsEmpty(Cells(aa, q + 1))
    aa = aa + 1
    Loop
    aa = aa - 1
    'define frequency limit number
    w = Sheets("para").Cells(x + 4, 2)
    'find the beginning of less than frequency X cell
    If aa = 2 Then
        If w >= 2 Then
            If Cells(aa, q + 1) <= w - 1 Then
                r = 2
            Else: r = 3
        End If
    End If

```

```

        End If
    Else: r = 3
    End If
End If
If aa > 2 Then
    If w >= 2 Then
        For bb = 1 To aa
            If Cells(bb, q + 1) <= w - 1 Then
                cc = bb
                bb = aa
            Else: cc = aa + 1
            End If
        Next bb
        r = cc
    Else: r = aa + 1
    End If
End If
If r > 2 Then
    'find the end of the frequency X cell
    s = r
    Do Until IsEmpty(Cells(s, q + 1))
        s = s + 1
    Loop
    s = s - 1
    'delete unneeded frequencies
    If w >= 2 Then
        Range(Cells(r, q), Cells(s + 500000, q + 1)).Delete Shift:=xlUp '---- 50000 is just
extra buffer
    End If
    'put headers for layer ID finding table
    Cells(1, q + 3) = "1st C Position"
    Cells(1, q + 4) = "Space Position"
    Cells(1, q + 5) = "2nd C Position"
    Cells(1, q + 6) = "End Position"
    Cells(1, q + 7) = "1st layer#"

```

```

Cells(1, q + 8) = "2nd Layer#"
'enter 1st row's formula
Cells(2, q + 3) = "=Find("""C""", rc[-3], 1)"
Cells(2, q + 4) = "=find("""", rc[-4], 1)"
Cells(2, q + 5) = "=find("""C""", rc[-5], rc[-1])"
Cells(2, q + 6) = "=len(rc[-6])"
Cells(2, q + 7) = "=mid(rc[-7], rc[-4]+1, rc[-3]-(rc[-4]+1))"
Cells(2, q + 8) = "=mid(rc[-8], rc[-3]+1, rc[-2]-rc[-3])"
'detect the end of the data set
t = 1
Do Until IsEmpty(Sheets("pro01").Cells(t, q))
t = t + 1
Loop
t = t - 1
'copy the 1st row to the other rows
If t > 2 Then
    Range(Cells(2, q + 3), Cells(2, q + 8)).Copy
    Range(Cells(3, q + 3), Cells(t, q + 3)).PasteSpecial Paste:=xlAll
End If
'convert layer numbers from formula to text
Range(Cells(1, q + 7), Cells(t, q + 8)).Select
Selection.Copy
Selection.PasteSpecial Paste:=xlPasteValues
'convert layer numbers from text to value
Range(Cells(1, q + 7), Cells(t, q + 7)).Select
Selection.TextToColumns Destination:=Cells(1, q + 7), DataType:=xlFixedWidth, _
    OtherChar:="-", FieldInfo:=Array(0, 1), TrailingMinusNumbers:=True
Range(Cells(1, q + 8), Cells(t, q + 8)).Select
Selection.TextToColumns Destination:=Cells(1, q + 8), DataType:=xlFixedWidth, _
    OtherChar:="-", FieldInfo:=Array(0, 1), TrailingMinusNumbers:=True
'sort the data set by 1st layer # then 2nd layer#
If x > 2 Then
    Range(Cells(1, q), Cells(t, q + 8)).Select
    ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear

```

```

ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, q +
7), Cells(t, q + 7)), SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, q +
8), Cells(t, q + 8)), SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(1, q), Cells(t, q + 8))
    .Header = xlYes
    .MatchCase = False
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply
End With
'detect the end of smaller layer number at 1st layer#
aa = 1
Do Until Cells(aa, q + 7) = 2 * x + 1
    aa = aa + 1
    If aa = t Then
        Cells(aa, q + 7) = 2 * x + 1
    End If
Loop
aa = aa - 1
'detect the end of smaller layer number at 2nd layer#
bb = 1
Do Until Cells(bb, q + 8) = 2 * x + 1
    bb = bb + 1
    If bb = t Then
        Cells(bb, q + 8) = 2 * x + 1
    End If
Loop
bb = bb - 1
find smaller one between aa and bb
If aa >= bb Then
    cc = bb
Else: cc = aa
End If

```

```
'delete previous layer only rows
If Not cc = 1 Then
    Range(Cells(2, q), Cells(cc, q + 8)).Delete
End If
'detect the end of the data set
t = 1
Do Until IsEmpty(Sheets("pro01").Cells(t, q))
    t = t + 1
Loop
t = t - 1
'sort the data set by count of paired (frequency of pairs)
Range(Cells(1, q), Cells(t, q + 8)).Select
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Clear
ActiveWorkbook.Worksheets("pro01").Sort.SortFields.Add Key:=Range(Cells(2, q +
1), Cells(t, q + 1)), SortOn:=xlSortOnValues, Order:=xlDescending, DataOption:=xlSortNormal
With ActiveWorkbook.Worksheets("pro01").Sort
    .SetRange Range(Cells(1, q), Cells(t, q + 8))
    .Header = xlYes
    .MatchCase = False
    .Orientation = xlTopToBottom
    .SortMethod = xlPinYin
    .Apply
End With
End If
'<<<<<<<<<<<<<<<-----
'detect the end of the data set
t = 1
Do Until IsEmpty(Sheets("pro01").Cells(t, q))
    t = t + 1
Loop
t = t - 1
'copy the sorted table into lmem sheet
Range(Cells(2, q), Cells(t, q + 1)).Copy
Sheets("lmem").Select
Cells(4, y).Select '<-----Here!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```

ActiveSheet.Paste
'-----module to replce 2 words cells to one-----
'delete previoue entry at pro01sheet
Worksheets("pro01").Select
Cells.Delete
'go back to lmem sheet
Worksheets("lmem").Select
'determine the end of the data set at Final Memory (associated) column
a = 3
Do Until IsEmpty(Worksheets("lmem").Cells(a, 3))
a = a + 1
Loop
a = a - 1
'copy Final Memory (associated) to pro01 sheet
Range(Cells(3, 3), Cells(a, 3)).Copy
Worksheets("pro01").Cells(1, 1).PasteSpecial Paste:=xlPasteValues
'go to pro01 sheet
Worksheets("pro01").Select
'adjust column A column width
Columns("A:A").Columns.AutoFit
'go back to lmem sheet
Worksheets("lmem").Select
'determine the end of the data set at MemoryXX
b = 3
Do Until IsEmpty(Worksheets("lmem").Cells(b, y)) '-----9 must be variable
b = b + 1
Loop
b = b - 1
'copy MemoryXX to pro01 sheet
Range(Cells(3, y), Cells(b, y + 1)).Copy '-----9 and 10 must be variable
Worksheets("pro01").Cells(1, 3).PasteSpecial Paste:=xlPasteValues
'go to pro01 sheet
Worksheets("pro01").Select
'adjust column A column width
Columns("C:D").Columns.AutoFit

```

```

'put tile
Cells(1, 5) = "MemoryXX address"
'detect the end of MemoryXX end downward
c = 1
Do Until IsEmpty(Sheets("pro01").Cells(c, 3))
c = c + 1
Loop
c = c - 1
'put MemoryXX address
d = 2
Do Until IsEmpty(Sheets("pro01").Cells(d, 3))
    Cells(d, 5) = "R" & d + 2 & "C" & y & ""    '<-----9 must be variable
d = d + 1
Loop
'detect the end of the data set at column A
e = 1
Do Until IsEmpty(Sheets("pro01").Cells(e, 1))
e = e + 1
Loop
e = e - 1
'replce 2 words to one
f = 2
Do Until IsEmpty(Sheets("pro01").Cells(f, 5))
    g = Sheets("pro01").Cells(f, 3)
    h = Sheets("pro01").Cells(f, 5)
    Range("A2:A" & e & "").Replace What:="" & g & "", Replacement:="" & h & "",
LookAt:=xlPart, SearchOrder:=xlByRows
f = f + 1
Loop
'detect the end of the data set at column A
i = 1
Do Until IsEmpty(Sheets("pro01").Cells(i, 1))
i = i + 1
Loop
i = i - 1

```

```

        'copy column A to lmem sheets
        Range(Cells(1, 1), Cells(i, 1)).Copy
        Sheets("lmem").Select
        Range("C3").PasteSpecial Paste:=xlPasteValues
    Else: x = 1000
End If
Next x
'go to lmem sheet
Sheets("lmem").Select
Range("A1").Select
End Sub
Sub d_text_to_formula()
' formulatext Macro
For c = 1 To 1000
    'case separation Final memory cooking and Mem02< cooking
    If c = 1 Then
        d = 3
    Else: d = 2 * c + 3
    End If
    'detect the end of the data set
    a = 3
    Do Until IsEmpty(Sheets("lmem").Cells(a, d))
        a = a + 1
    Loop
    a = a - 1
    'replace space with "&" "&"
    Range(Cells(4, d), Cells(a, d)).Replace What:=" ", Replacement:="&" "&", LookAt:=xlPart,
SearchOrder:=xlByRows
    'delete previous entry at pro01 sheet
    Sheets("pro01").Select
    Cells.Delete
    'copy lmem data to pro01 sheet
    Sheets("lmem").Select
    Range(Cells(3, d), Cells(a, d)).Copy
    Sheets("pro01").Select

```

```

Range("A1").Select
ActiveSheet.Paste
'adjust column width
Selection.Columns.AutoFit
'put tile to column B
Range("B1") = "with ="
'detect the end of the data set downward
b = 1
Do Until IsEmpty(Sheets("pro01").Cells(b, 1))
b = b + 1
Loop
b = b - 1
If Not b = 1 Then
    'put 1st formula
    Range("B2").FormulaR1C1 = "="&"="&RC[-1]"
    Range("B2").Copy
    Range(Cells(2, 2), Cells(b, 2)).Select
    ActiveSheet.Paste
    'select whole column B
    Columns("B:B").Select
    'convert from formula to values
    Selection.Copy
    Selection.PasteSpecial Paste:=xlPasteValues
    'convert from text to renewed formula
    Selection.TextToColumns Destination:=Range("B1"), DataType:=xlFixedWidth, _
        OtherChar:="-", FieldInfo:=Array(0, 1), TrailingMinusNumbers:=True
    'copy the renewed formula to lmem sheet
    Range(Cells(2, 2), Cells(b, 2)).Copy
    Sheets("lmem").Select
    Cells(4, d).Select
    ActiveSheet.Paste
Else: c = 1000
End If
Next c
'go back to lmem sheet

```

```

        Sheets("Imem").Select
        Range("A1").Select
    End Sub
    Sub e_formula_to_text()
        'module to convert formula to text so that addresses can be seen
        For b = 1 To 1000
            If b = 1 Then
                c = 3
            Else: c = 2 * b + 3
            End If
            'detect the end of the data set downward
            a = 3
            Do Until IsEmpty(Sheets("Imem").Cells(a, c))
                a = a + 1
            Loop
            a = a - 1
            If Not a = 3 Then
                'select target data set
                Range(Cells(4, c), Cells(a, c)).Select
                'change cell contents display from results to formula
                ActiveWindow.DisplayFormulas = True
                'replce "=" with "(nothing)"
                Selection.Replace What:="=", Replacement:="", LookAt:=xlPart, SearchOrder:=xlByRows
                'replace "&" "&" with "(space)"
                Selection.Replace What:="&" "&", Replacement:=" ", LookAt:=xlPart,
                SearchOrder:=xlByRows
                'change back the cell contents from formula to results
                ActiveWindow.DisplayFormulas = False
            Else: b = 1000
            End If
        Next b
        'go back to Imem sheet
        Sheets("Imem").Select
        Range("A1").Select
    End Sub

```

CLAIMS

What is claimed is:

Claim 1 A computerized method of compressing symbolic information organized into a plurality of documents, each document having a plurality of symbols, the method comprising:

with a first document of the plurality of documents as an input document, automatically with a computer:

- (a) identifying a plurality of symbol pairs, each symbol pair consisting of two sequential symbols in the input document;
- (b) for each unique symbol pair of the plurality of symbol pairs, updating a count identifying the number of appearances of the unique symbol pair; and
- (c) producing a compressed document by causing the compressed document to include, at each position associated with one of the plurality of symbol pairs from the input document, a replacement symbol associated by a compression dictionary with the unique symbol pair matching the one of the plurality of symbol pairs, if the count for the unique symbol pair exceeds a threshold.

Claim 2 The method of claim 1 wherein the threshold is two.

Claim 3 The method of claim 1 further comprising sorting the compression dictionary in order of the count for the respective unique symbol pairs.

Claim 4 The method of claim 3 further comprising removing from the compression dictionary all unique symbol pairs having a count that does not exceed the threshold and making an association in the compression dictionary between each unique symbol pair having a count that exceeds the threshold and a replacement symbol.

Claim 5 The method of claim 1 further comprising performing an additional pass by repeating parts (a), (b), and (c) using as the input document the compressed document.

Claim 6 The method of claim 5 further comprising performing additional passes, until the compressed document created by one of the passes is unaltered from the compressed document created by the immediately previous pass.

Claim 7 The method of claim 1 wherein each symbol from the first document indicates a word.

Claim 8 The method of claim 7 further comprising, before part (a), creating the first document by replacing each word of a text document with a corresponding one of the plurality of symbols.

Claim 9 The method of claim 1 wherein each symbol in the first document indicates an alphanumeric character.

Claim 10 The method of claim 1 wherein each replacement symbol comprises an address of an entry in the compression dictionary, which entry stores the unique symbol pair associated with replacement symbol.

Claim 11 The method of claim 1 further comprising performing parts (a), (b), and (c) on each of the other documents of the plurality of documents, wherein the respective counts for the symbol pairs identifies the number of previous appearances of that symbol pair in any of the plurality of documents.

Claim 12 The method of claim 11 wherein parts (a) and (b) are performed on each of the plurality of documents before part (c) is performed on any of the plurality of documents.

Claim 13 The method of claim 12 further comprising, before part (c) is performed on any of the plurality of documents, making an association in the compression dictionary between each unique symbol pair and a replacement symbol if and only if the unique symbol pair has a count that exceeds the threshold.

Claim 14 The method of claim 11 further comprising sorting the compression dictionary in order of the count for the respective unique symbol pairs.

Claim 15 The method of claim 14 further comprising removing from the compression dictionary all unique symbol pairs having a count that does not exceed the threshold and making an association in the compression dictionary between each unique symbol pair having a count that exceeds the threshold and a replacement symbol.

Claim 16 The method of claim 11 further comprising repeating parts (a), (b), and (c) on each of the other documents to make at least one additional pass by repeating parts (a), (b), and (c), using as the input document the compressed document previously produced by the process, until the compressed document created by one of the passes is unaltered from the compressed document created by the immediately previous pass.

Claim 17 The method of claim 16 further comprising assigning the first document at least one of a plurality of attributes based on the frequency with which a replacement symbol associated with a unique symbol pair appears in an output document associated with the first document.

Claim 18 The method of claim 17 further comprising suggesting a related document based the at least one attribute of the first document and the at least one attribute of the related document.

Claim 19 A computer system having at least one processor capable of compressing symbolic information organized into a plurality of documents, each document having a plurality of symbols, the computer system programmed to:

with a first document of the plurality of documents as an input document, automatically with the processor:

- (a) identify a plurality of symbol pairs, each symbol pair consisting of two sequential symbols in the input document;
- (b) for each unique symbol pair, update a count identifying the number of appearances of the symbol pair; and
- (c) produce a compressed document by causing the compressed document to include, at each position associated with one of the plurality of symbol pairs from the input

document, a replacement symbol associated by a compression dictionary with the unique symbol pair matching the one of the plurality of symbol pairs, if the count for the unique symbol pair exceeds a threshold.

Claim 20 A computer-readable storage medium that is not a transitory propagating signal storing a set of computer instructions for compressing symbolic information organized into a plurality of documents, each document having a plurality of symbols, wherein the set of computer instructions, when executed on the computer, causes the computer, automatically:

with a first document of the plurality of documents as an input document, automatically:

- (a) to identify a plurality of symbol pairs, each symbol pair consisting of two sequential symbols in the input document;
- (b) for each unique symbol pair, to update a count identifying the number of appearances of the symbol pair; and
- (c) to produce a compressed document by causing the compressed document to include, at each position associated with one of the plurality of symbol pairs from the input document, a replacement symbol associated by a compression dictionary with the unique symbol pair matching the one of the plurality of symbol pairs, if the count for the unique symbol pair exceeds a threshold.