Data Compression and Its Application in Web Data Transmission

XIAOLING XIA and DI BAI

ABSTRACT

Aiming at the problem of data transmission in distributed framework of web project, a compression algorithm combining Huffman encoding and Run length encoding is used to compress data in this paper, thus reducing the amount of data and improving the speed of data transmission. The integrity of data can be guaranteed since lossless compression is used. Experimental results show that the algorithm is effective in compressing data, effectively reducing the size of data storage and speeding up the transmission of background data, so it has very good application value.

KEYWORDS
Huffman encoding; Run length encoding; data transmission; data compression.

INTRODUCTION

Data compression, as its name implies, is a method of compressing the target data by some kind of algorithm, thus reducing the storage space and improving the efficiency of data transmission. It is mainly divided into lossy compression and lossless compression algorithms. Lossy compression is widely used in video and audio file compression. The lossless compression algorithm is mainly to compress some data and eliminate the repeated data, which makes the data storage capacity smaller. With the development of information technology, the amount of data generated is increasing, and a large amount of redundant data has appeared, which has brought great pressure to the transmission of data. It is an important topic about how to transfer and store data faster, more and better without damaging data integrity and correctness in the field of today's research.
DATA COMPRESSION ALGORITHM

Run Length Encoding.

Run Length Encoding, referred to as RLE algorithm, is a lossless data compression algorithm. The principle is simple: By counting the number of repetitions of consecutive bytes, the string is represented by identifier, frequency and repeating byte units, thus shortening the length of the total byte string and achieving the purpose of compression. The encoding of the repeating segment requires 3 bits. When the number of repetitions count is less than 3, the encoding will lengthen the original string. When count>=3, encoding reduces the length of the original string. A reduced length can be obtained:

\[
\text{length} = \begin{cases} 
0 & \text{if } 0 < \text{count} < 3 \\
\text{count} - 3 & \text{if } \text{count} \geq 3
\end{cases}
\]  

(1)

Where count is the length of the repeating string. It can be concluded that the compression ratio is:

\[
\text{g} = \begin{cases} 
0 & \text{if } 0 < \text{count} < 3 \\
\frac{\text{length}}{\text{count}} & \text{if } \text{count} \geq 3
\end{cases}
\]  

(2)

We can draw the conclusion by formula (1) and (2) that when count>=3, the greater the count, the higher the compression ratio. That is to say, the compression algorithm has good compression effect for strings with high repetition rate of substring.

Huffman Coding.

Huffman encoding widely used in data compression, is a lossless compression coding. The algorithm calculates the frequency of each element in the original data, and then generates an optimal binary string to represent the unit according to its frequency. Huffman encoding is mainly achieved by the Huffman tree, and the main construction idea of the tree is bottom-up. Figure 1 shows the structure of a constructed Huffman tree whose leaf nodes are data units and their frequencies:

![Huffman tree](image-url)
If $P(x)$ is used to represent the frequency of $x$ in the original data, $D(x)$ indicates the depth of the leaf node where $x$ is located in the Huffman tree, that is, $D(x)$ is the corresponding encoding length of $x$. Then, the length of the string after the Huffman encoding of the original data is:

$$length = \sum_{i=1}^{n} [D(i) * P(i)]$$  \hspace{1cm} (3)$$

Where $n$ represents the number of data units in the original data.

**DATA COMPRESSION**

**Problem description.**

In the enterprise web project development, the structure of a project is complex. So developers will integrate distributed frameworks in the project to separate front and rear ends, enabling both front and rear ends to be developed simultaneously and to improve development efficiency. However, the main problem in this way is the transmission of background data to the front end. When the amount of data transmitted exceeds a certain size, the transmission rate is very slow, and the integrity of the data also cannot be guaranteed.

On a certain platform, it is necessary to statistics the search records of all kinds of works by each user. It is unusually slow for the data interaction in this platform because of the data of the search record is too large, and it takes a long time for the administrator to see the records on the interface. The information of its search records are shown in Table 1.

**Data compression process**

As you can see in Table 1, a large amount of information is repeated in each column. As the data becomes larger, the repetition rate increases. So the more space can be compressed. Under the premise that data is ordered, each column is compressed separately and encoded into a string to corresponding these records. Data compression algorithms are mainly divided into 5 steps in this paper.

Step 1: Data preprocessing. This part mainly includes statisticcing the frequency of each data unit and the discretization of data. First, find out all the different data units

<table>
<thead>
<tr>
<th>User name</th>
<th>Works name</th>
<th>Works area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>Thermometer</td>
<td>Physics</td>
</tr>
<tr>
<td>Jack</td>
<td>Pedometer</td>
<td>Life</td>
</tr>
<tr>
<td>Rose</td>
<td>Fire Extinguisher</td>
<td>Life</td>
</tr>
<tr>
<td>Rose</td>
<td>Thermometer</td>
<td>Physics</td>
</tr>
<tr>
<td>Jack</td>
<td>Aerocraft</td>
<td>science</td>
</tr>
<tr>
<td>Rose</td>
<td>Pedometer</td>
<td>Life</td>
</tr>
<tr>
<td>Tom</td>
<td>Fire Extinguisher</td>
<td>Life</td>
</tr>
<tr>
<td>Jeff</td>
<td>Aerocraft</td>
<td>science</td>
</tr>
</tbody>
</table>
that appear in the data information, and start numbering them from 1 in the order of access. After traversing the data information, each data unit is converted into the corresponding number and merged into a set of T (n). Then, an array Num[n] is created to count the number of times each data cell appears, where n represents the number of different units in the data information.

Step 2: Construction of Huffman tree. Create an entity class called Tree whose member variables include: content (the content of the data unit), the count (the corresponding weight of the node, the frequency), the leftChild (left child node), and the rightChild (right child node). The frequency statistics array Num[n] and the encoding rules of each data unit obtained from step 1 are converted into corresponding Tree objects as leaf nodes and stored in an object list called treeList. Combined with the construction principle of Huffman tree in the last section, the algorithm design process is as follows:

Input: an object Tree list treeList;
Output: Huffman tree;
(1) The treeList object in count in accordance with the order from small to large;
(2) Take out the smallest two trees in treeList, Ti, Tj;
(3) Take Ti and Tj as the left and right child nodes of the new tree Tk. The small weight is the left child node, the big one is the right child node, and the weight of the Tk is the sum of the weights of Ti and Tj;
(4) Add Tk to the list treeList, and removes Ti and Tj from the list;
(5) Repeat (1)-(4) until the list treeList has a capacity of 1. The rest of the tree is Huffman tree;

Step 3: Huffman encoding of data. Getting the Huffman encoding corresponding to each data unit through Huffman tree, and then the original data information can be converted to the corresponding encoding string. The recursive algorithm is used to traverse the Huffman tree to record the path of access. When searching the leaf node, its access path is the Huffman encoding corresponding to the data unit stored by the leaf node. Then iterate through the data, obtain the corresponding encoding, and integrate into string. HuffmanCoding (x) is used to represent the encoding of data unit x, so the implementation is as follows:

```java
void getHuffmanCoding(coding,huffmanTree){
    if (huffmanTree is Leaf node){
        huffmanCoding(huffmanTree.content)←coding;
        return ;
    }
    getHuffmanCoding(coding+"0",huffmanTree.leftChild);
    getHuffmanCoding(coding+"1",huffmanTree.rightChild);
}
```

Step four: Using RLE to encode the string generated by Huffman. The original data is encoded by Huffman encoding into a string containing only "0" and "1", and then there is a lot of "0" or "1" repeated substrings in that string. Thus, the string can be further compressed by RLE, while maintaining the string order. Repeat identifier is unnecessary for this encoding since the number of repetitions required for encoding must be greater than 1. In order to reduce the complexity of the program, the number of repeats is represented by only one character. And its construction rules: 3-9 are represented by corresponding characters, 10-35 by character A-Z, and 36-61 by...
character A-Z. So the range of repetitions is between [3, 61]. With D(x) to represent the corresponding characters of digital x, its algorithm design process is as follows:

Input: Huffman encoded string huffmanString
Output: the string rleString after RLE

1. Set the repetition number count and the initial value of the repeating unit ch:
   Count = 1; ch = huffmanString[0];
2. Loop statistics string huffmanString;
3. Determine the size of count, and if count>=61, encode the repeating string directly: rleString = rleString + D(count) + ch;
   And set the value of count to 0;
4. Determine whether the ch is equal to the current loop string. If equal, repeat the number count plus 1, jump to Step1 directly. If not, proceed to the next step;
5. Determine the size of count. If count>2, the repeating string is encoded. Otherwise, no encoding is added directly to the string rleString after the string is repeated:
   For, J, from 0, to, count, step, 1
   RleString - rleString + ch;
6. Repeat (1)-(5) until the loop ends. Then execute the Step5 again, and the rleString is the final encoded string after the end of the unknown repeating string.

Step five: Data encapsulation. Because there will be more data compression on multiple columns, then there will be a number of compressed strings, and Huffman decoding is inseparable from the Huffman tree. Therefore, you need to match these strings with the Huffman tree, which is encapsulated into a list, returned to the front end, and then decoded by the front end.

RESULT COMPARISON

As you can see from table 2, when the amount of data is relatively small, the data compressed by the compression algorithm is faster than the uncompressed one, but the compression rate of the uncompressed data is completed within 1s. When the amount of data increases to 140 thousands, the uncompressed data lead to data transmission failure, while the compressed data can still be transmitted, and the transmission rate is kept within 1s.

<table>
<thead>
<tr>
<th>Data quantity Data type</th>
<th>Data before compression</th>
<th>Compressed data</th>
<th>compression ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size (byte)</td>
<td>transmission speed(ms)</td>
<td>Size (byte)</td>
</tr>
<tr>
<td>20000</td>
<td>917727</td>
<td>120</td>
<td>217458</td>
</tr>
<tr>
<td>50000</td>
<td>2771534</td>
<td>251</td>
<td>505472</td>
</tr>
<tr>
<td>80000</td>
<td>4531244</td>
<td>371</td>
<td>786420</td>
</tr>
<tr>
<td>110000</td>
<td>6076319</td>
<td>562</td>
<td>1048250</td>
</tr>
<tr>
<td>140000</td>
<td>8676691</td>
<td>--</td>
<td>1359221</td>
</tr>
<tr>
<td>170000</td>
<td>10619344</td>
<td>--</td>
<td>1517259</td>
</tr>
<tr>
<td>200000</td>
<td>12037500</td>
<td>--</td>
<td>1818465</td>
</tr>
</tbody>
</table>

TABLE 2. COMPARISON OF TRANSMISSION RESULTS BEFORE AND AFTER DATA COMPRESSION.
SUMMARY

In this paper, through the analysis of principle of run length encoding and Huffman encoding of these two algorithms, to generate large amounts of data in actual project data compression, so as to improve the efficiency of data transmission, greatly reducing the consumption of IO server. At the same time, in order to compress and transfer data simply and quickly, the data is discretized and encapsulated before the data compression.

REFERENCES