A greedy algorithm for orderer-preserved compressing

Peisheng Wang

December 9, 2008

Abstract

Order-Preserving compression has brought advantage to sorting algorithm, especially for string-oriented sorting algorithms and word-RAM algorithms for keys of bounded length. Minimazation procedure that to seperate weight set of symbols into two almost equal size subset is crucial for constructing weight-balanced alphabet tree to generate compressing schema.

We propose a greedy algorithm with $O(n)$ time complexity but $O(1)$ space compexity for minimization procedure, which has lower space compexity than the algorithm in [1]. And it also provides the same compressed ratio as [1].

A improved greedy algorithm which have better compressed ratio under some cases is also presented.

Experimentation shows that, our new compressor has better compressed ratio and twice faster in compressing and the same speed in decompressing.

Contents

1 Introduction

Order-Preserving compression has brought advantage to sorting algorithm, especially for string-oriented sorting algorithms and word-RAM algorithms for keys of bounded length. There is a promising viewpoint that Order-Preserving compression can also be used to provide linear scan for dynamic hash table.

In [1], a linear-approximation algorithm for creating compression dictionary is introduced, which is also near-optimized. and it is admirable to improve it.

In section 2, schedule of this project is presented.

In section 3, minimization algorithm in [1] is introduced.

In section 4, we propose a greedy algorithm.

In section 5, we improve the greedy algorithm and in section 6 experimentation for comparision with previous version is presented. And in section 7, Conclusion is presented.

2 Previous algorithm

To construct a optimal alphabetic tree is core for order-prevering compressing.

definition of optimal alphabetic tree

Given a sequence of n positive weights w_1, w_2, w_n, find a binary tree in which all weights appear in the leaves such that:

- The weights on the leaves occur in order when traversing the tree from left to right. Such a tree is called an alphabetic tree.
- The $\sum_{1 \le i \le} W_i l_i$ is minimized, where l_i is the depth (distance from root) of the ith leave from left. If so, this is an optimal alphabetic tree.

If we drop the first condition, the problem becomes the well-known problem of building Huffman trees, which is known to have the same complexity as sorting.

In [1], a linear-Approximation Algorithm algorithm is introduced, i.e. weight-balanced tree. And procedure **minimize** is to seperate the wegith set into two almost equal weight size subset and make left and right tree respectively and it is crucial to generate compressing schema.

In the following, we will introduce the MINIMIZE algorithm in [1].

- Procedure1 minimize(i, j)

  1. Denote $w_0, w_1, ..., w_n$ as the weight of the n ordered symbols.
  2. if (i == j) return a tree with one node containing w_i
  3. Find k such that $M = |(w_i + w_{i+1} + ... + w_k) - (w_{k+1} + ... + w_j)|$ is minimum
  4. return k

  Procedure2 is improved version of procedure1 and has time complexity with $O(n)$ and $O(n)$ space complexity.

- Procedure2 minimize(i, j)

Table 1: Comparion between procedure2 and greedy algorithm

algorithm	time compleixy	space complexity
previous	O(n)	O(n)
greedy	O(1)	O(n)

1. Denote $w_0, w_1, .., w_n$ as the weight of the n ordered symbols.
2. Let $sum(i, k) = w_i + w_1 + ... + w_k$, where $i \leq k \leq j$.
3. if $(i == j)$ return a tree with one node containing w_i
4. Find k such that $M = |sum(i, j) - 2sum(i, k) + sumi, i|$ is minimum
5. return k

3 gready algorithm

We propos a greedy algorithm, which has $O(n)$ time complexity and $O(1)$ space complexity.

minimize(i, j)

1. Let $left = i$, $right = j$, $sum_{left} = w_i$, $sum_{right} = w_j$.
2. if $(i + 1 == j)$ return i
3. if $sum_{left} < sum_{right}$
4. $sum_{left} + = w_{left}$; $left + +$;
5. else if $sum_{left} > sum_{right}$
6. $sum_{right} + = w_{right}$; $right - -$;
7. else
8. $sum_{left} + = w_{left]}$; $left + +$;
9. $sum_{right} + = w_{right}$; $right - -$;
10. return left

3.1 improved greedy algorithm

But there is a problem, when $sum(i, k - 1) == sum(k + 1, j)$,Procedure1 will always return $k-1$, but sometimes return k can lead to higher compressed ratio than $k - 1$. For example, in figure expamle, procedure 1, 2 doesn't lead to best encoding schema.

The reason for the difference is that when we put the kth element on the left part or right part, it will cause left sub tree or right sub tree to be re-balanced and re-balanced degree are different.

In the alphabet tree, the depth of the elements on the left and on the right of the kth element will be firstly affected, moslty will increase 1. For simplicity, we

Table 2: example

symbols	number
a	1
b	1
c	1
d	1
e	3

Table 3: procedure1,2

symbols	r
a	00
b	010
c	011
d	10
e	11
total bist	16

Table 4: put k th element on the left/improve greedy algorithm

symbols	bits
a	000
b	001
c	010
d	011
e	1
total bits	15

Table 5: comparion to previous version

version	ratio	compressing	decompressing
previous	64.61%	4.43s	0.66s
new	62.67%	2.01s	0.67s

use the two elements to detemine where should put the kth element. Al least, from [1] it is still weight-balanced and will not descend. in term of compressed ratio. And most of the time the improved greedy algorithm perform the same as previous algorithm.

minimize(i, j)

1. Let $left = i$, $right = j$, $sum_{left} = w_i$, $sum_{right} = w_j$.
2. if $(i + 1 == j)$ return i
3. while $(i <= j)$
4. if $(sum_{left} < sum_{right})$
5. $sum_{left} + = w_{left}$; $left + +$;
6. else if $(sum_{left} > sum_{right})$
7. $sum_{right} + = w_{right}$; $right - -$;
8. else
9. if$(w_{left-1} < w_{right+1}\ )$
10. $sum_{left} + = w_{left}$; $left + +$;
11. else
12. $sum_{right} + = w_{right}$; $right - -$
13. return left;

4 design

On contrast to previous version, when compressing a file, we store bit pattern instead of alphabet tree in the file header. please see UML configure.

5 Experimentation and performance

We use this program to compress a file which has file size about 11.2M and contains 1462877 words. We calcuate the compressed ratio(compressed file size /original file size), compressing elapsed, and decompressing time.

Our new compressor has better compressed ratio and twice faster in compressing and the same performance in decompressing.

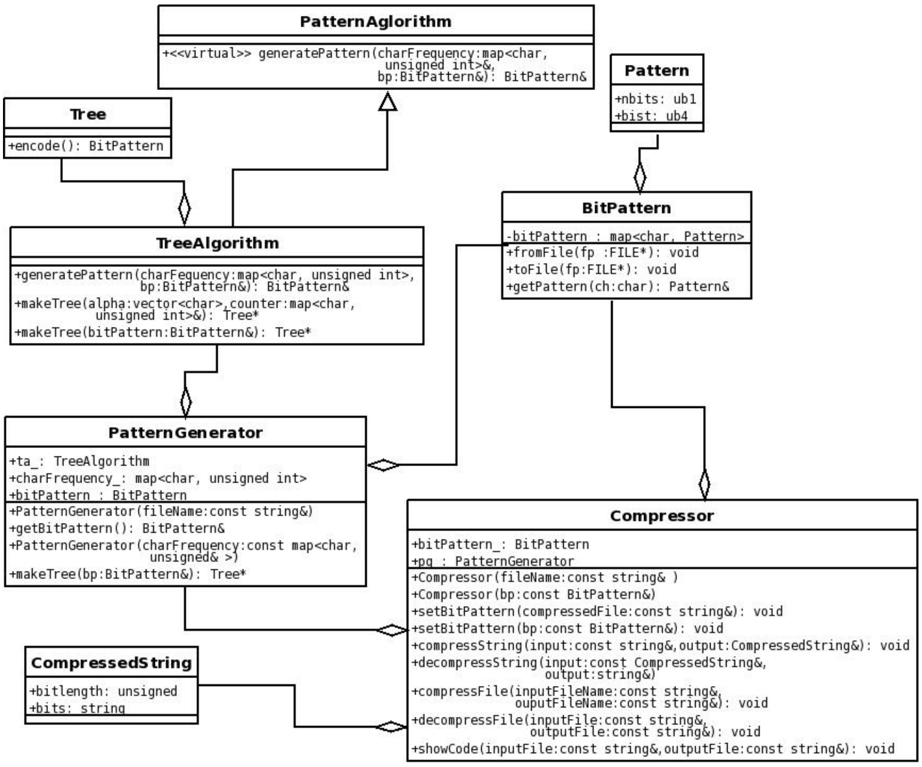

Figure 1: UML of ordered-preserving compressor

6 Conclusion

Order-Preserving compression has brought advantage to sorting algorithm, especially for string-oriented sorting algorithms and word-RAM algorithms for keys of bounded length. Minimazation procedure that to seperate weight set of symbols into two almost equal size subset is crucial for constructing weight-balanced alphabet tree to generate compressing schema.

We propose a greedy algorithm with $O(n)$ time complexity but $O(1)$ space compexity for minimization procedure, which has lower space compexity than the algorithm in [1]. And it also provides the same compressed ratio as [1].

and a improved greedy algorithm which have better compressed ratio under some cases is also presented.

Experimentation shows that, our new compressor has better compressed ratio and twice faster in compressing and the same speed in decompressing.

References

[1] M. M. ALEJANDRO LOPEZ-ORTIZ. Fast string sorting using order-preserving compression. *ACM Journal of Experimental Algorithmics*, 10(1.4):112, 2005.

A Schedule

A.1 Stage1

Totol duration : 2 weeks

Milestone

1. Preparation
   - Duration: 1 week
   - People in charge: Peisheng Wang
   - Description Read previous source and related paper.
2. improvement and experimentation
   - Duration: 1 week
   - People in charge: Peisheng Wang
   - Description
     - Redesign and provide better APIs. Make it more structured and scalable.
     - Propose and implement a greedy algorithm for minimization, which also has O(N) time complexity but better space complexity O(1). Experimentation shows that it has the same compressed ratio and speed as the algorithm in the paper [1].
     - Improve the greedy algorithm, which can have better compressed ratio under some cases than the algorithm in the paper [1].
     - documentation,including TR.

A.2 Stage2

MileStone	Finish Date	In Charge	Description
Maintenance	2008-12-05	Peisheng Wang	Reorganize the structure of ylib, code tunning, improve TR

B How to use fast-string sorting?

B.1 example

For usage of these codes, please see example in src directory:

- compresee.cc, to compress a file.
- decompress.cc, to decompress a compressed file.
- showcode.cc, to show the bit pattern for each keyword in input file.
- test-keys.cc, to test compressing and decompressed string.
- test-sort.cc, to test sorting on compressed strings.

Index