LZW inspired De-Duplication algorithm for variable size blocks with sliding window

Justin

1 Introduction

The variable size defined in the title is a multiple of the smallest block size. So if the block size is 1 KB then the variable size blocks can be of 1, 2, 3 $\cdots$ KBs. Block size is set to 4 KB. So the block sizes will vary as a multiple of 4, ie. 4, 8, 12, 16, $\cdots$ 32 KB. An empty dictionary is used, as pre-calculating the hashes for all the blocks and adding to dictionary would be expensive on time and space. The dictionary is used only during de-duplication, and is not utilized while retrieving the original data from the de-duplicated data. A sliding block is also used in the algorithm to take into account blocks which might fall in between block boundaries. The window is advanced by an increment equal to Δd which is an integral multiple of the block size.

2 Algorithm - De-Duplication

Input: Data segment $\approx$ 200MB
Output: Index, de-duplicated data segment
begin
Initialize:
$p = false,\ size = 1,\ inc = \Delta d,\ blockPointer = 0,\ buffer = \phi$;
A: Retrieve block at *blockPointer*, append retrieved block to buffer;
Calculate hash of the buffer;
if *hash is in dictionary* **then**
set: $p = true$;
$size+ = 1$;
$blockPointer+ = inc * x$;
where x is an integral multiple of block size $\equiv\ \Delta d = blockSize/x$;
if *End of segment* **then**
Goto **D**;
else
Goto **A**;
end
else
Insert hash into dictionary;
if $p = false$ **then**
$blockPointer+ = inc * x$;
Goto **B**;
else
Generate index for previous non-unique block;
$blockPointer+ = inc * x$;
$p = false$;
$size = 0$;
$buffer = \phi$;
if *End of Segment* **then**
Goto **E**;
else
Goto **A**;
end
end
end
end

Algorithm 1: De Duplication algorithm.

B: Set $nextBlockPointer = blockPointer + (inc * x)$;
increment $blockPointer$ by inc;
set $prevBuffer = buffer$;
clear $buffer$;
F: Retrieve block at $blockPointer$;
Calculate hash;
if *hash in dictionary* **then**
    Output non-duplicate data between blocks, generate index;
    Goto **A**;
else
    Increment $blockPointer$ by inc;
    Clear buffer;
    if $blockPointer = nextBlockPointer$ **then**
        decrement $blockPointer$ by $(inc * x)$;
        retrieve block at $blockPointer$ and append to $prevBuffer$;
        Calculate hash and add to dictionary;
        Goto **C**;
    else
        Goto **F**;
    end
end
D: Generate Index;
E: End;

Algorithm 2: De Duplication algorithm. (cont.)

3 Re-duplication

1. Start
2. Read next/first entry of de-duplicated data.
3. If data block: output as it is.
4. Else if: index reference, then print blocks from the index till index+size.
5. Last entry? if yes end, else goto step 2.

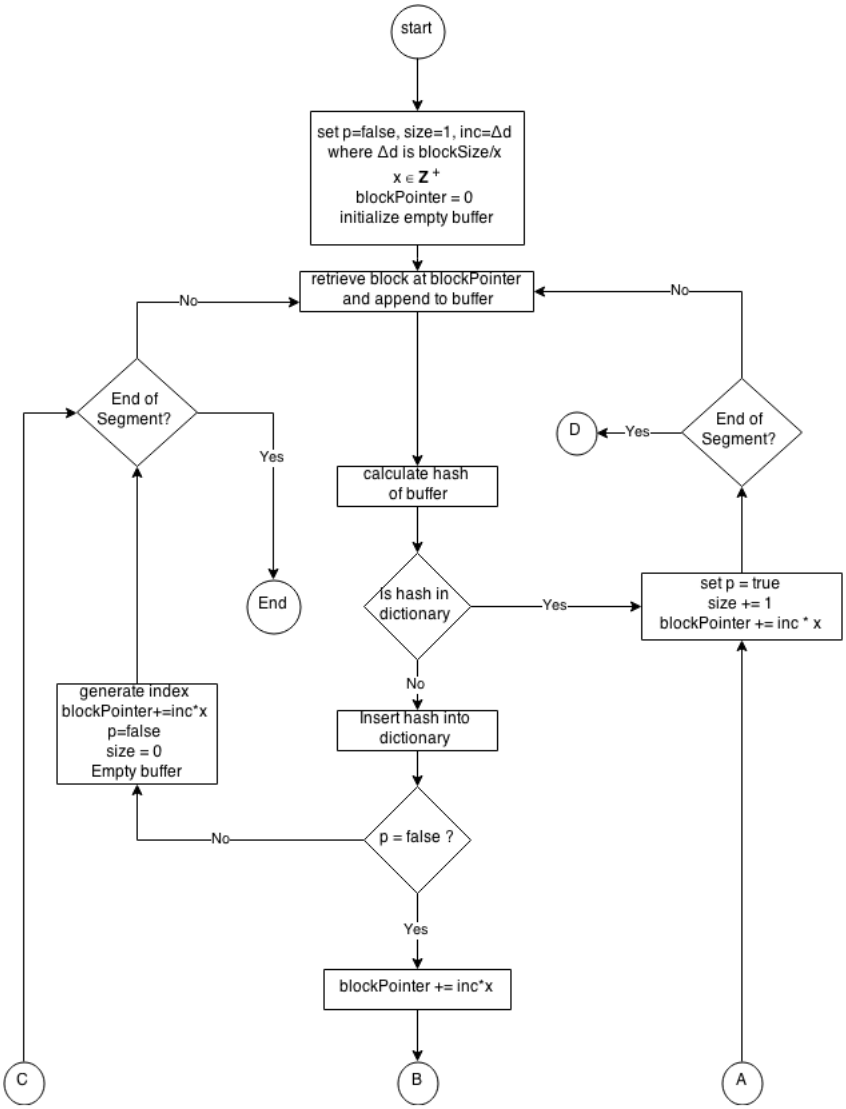

Figure 1: LZW inspired variable block de duplication

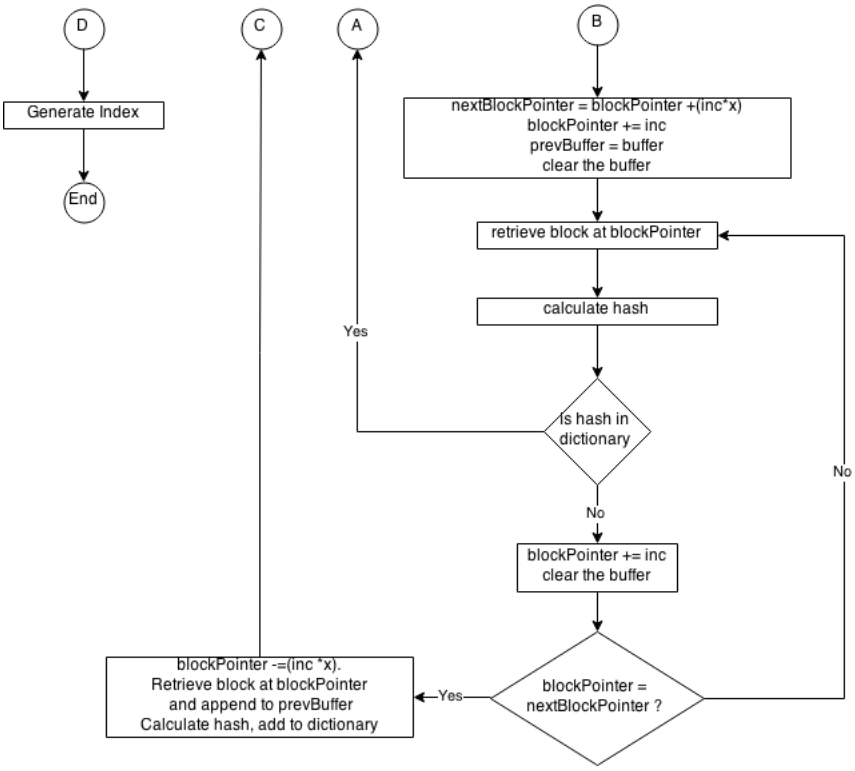

Figure 2: LZW inspired variable block de duplication (cont.)