COP 5536 Programming Project

Varun Mankal, UFID-04827615

April 6, 2017

1 Introduction

Among three data structures, four way heap gives the best performance. So, the building of huffman tree is done using four way heap.

2 Function Prototypes

- encoder()-constructor for encoder class
  It initializes the frequency array and an arraylist to store huffman nodes.

- generate_freq_table(String filename) - generates frequency table by storing frequencies in frequency array.
- initialize()-constructs a four way cache optimized heap for the given data based on its frequencies.

- heapify(int i) - ensures that the every children of 'i' are larger that their parent 'i'.

- removemin() - returns the root node by replacing it with the last node of the heap and ensures that the min heap property is satisfied.

- insert(Huffman_Node new_node) - inserts a new node at the end of the heap and compares with its parent to make sure that the parent is smaller than its children. If any violation, then the new node swaps with the parent node. The process continues until the min heap property is satisfied.

- generate_tree() - generates huffman tree by removing two minimum nodes and inserting a new node whose frequency is sum of the frequencies of the two deleted element.

- inOrder(Huffman_Node root,String s) - recursively generates huffman codes and writes to the code table file as a key value, where key is the data and value is huffman code.

- generate_decode_tree(String code_table) - creates a decode tree by continuously generating nodes based on the codes in code table.

- generate_decode_msg(String encoded file) - generates the decoded message using decoded tree and encoded file. The entire text in encoded file is stored in string builder object. Using this object, the message is generated.

- encoding_msg() - It is used to write the encoded message in bytes to a file.

3 Structure of the program

Project contains these classes: Encoder , Four_way_Cache
and Decoder class.

Encoder generates frequency table and populates the huffman arraylist by inserting huffman nodes into it.
It passes this huffman list to the four way cache class. Then, using the four way cache class object generates the huffman tree and corresponding code table. With the code table in hand, encoder class generates encoded message and writes to encoded .bin file.
Decoder class takes code table and encoded.bin files as input, decoder object generates decode tree and decoded message.

4 Performance Analysis

Out of the three data structures, four way cache optimized heap gives better performance.

Heap	Time
Binary Heap	2.9
4-Way Heap	2
Pairing Heap	10.4

So, by looking at the table, four way gives good performance. Therefore, four way heap is used to generate huffman tree. As we know that four way heap has four children and these are pulled into the cache line all at a time by shifting 3 bits from the start location of the array list.So, to fill the first 3 spots, dummy nodes are inserted. Whereas for pairing heap, it took relatively more time than other two data structures. This is because pairing heap involves transferring of data between pointers . In addition the height of four way heap is half that of binary heap. By shifting three bits, each node can access their children with at most one cache miss. It took 10 seconds to generate encode message on thunder server using pairing heap.

5 Decoding Algorithm

For decoding, two methods are used: generate_decode_tree() and generate_decode_msg().
The former method creates a decode tree by traversing the each bit of code table. For each line in the code table, it reads each bit and traverses by creating nodes,if necessary ,till the end of the line. Then it stores the corresponding data in the leaf node.This process is repeated till end of the file.
The latter method stores the entire data of encoded bin in a string builder object by taking 8 bytes

at a time and then converting it into string and then storing in the string builder object.Once end of the file is reached, then the decoding process starts using the code table. Each character is read at a time and by traversing the code table accordingly, the corresponding data is written into the decode file.