PROJECT REPORT

COP5536: ADVANCED DATA STRUCTURES

By,

Hareesh Nutalapati
UFID – 68011198

CLASSES AND FUNCTION PROTOTYPES

Project is implemented in java. We have the following classes in the program:
1. MaxPriorityQueue
2. CacheFourWayHeap
3. HauffmanTree
4. encoder
5. Store
6. decoder
7. Iterator

The following classes are initially used to determine the fastest data structure to generate Hauffman Tree.

1. BinaryHeap
2. PairingHeap
3. CacheFourWayHeap

MaxPriorityQueue is an abstract class. BinaryHeap, PairingHeap and CacheFourWayHeap extends MaxPriorityQueue.

Methods in MaxPriorityQueue:

The following are abstract methods for which implementation is given various classes.

1. public abstract void heapify(Freq_table, ft)
2. public abstract void insert(HuffmanTree r)
3. public abstract HuffmanTree pop()
4. public abstract void printHeap()
5. public abstract int getSize();
6. public abstract Boolean isNotEmpty()

Methods in CacheFourWayHeap:
1. Public void heapify(Freq_Table ft):

This method takes input as frequency table, traverse it along it's size and inserts into the Huffman tree.

2. Public void insert(HuffmanTree r):

This method takes HuffmanTree reference as an input and adds into the heap. It also calls bubbleUp method internally.

3. Private void bubbleUp(int index):

This method takes index as an input and loops based on the condition that index is less than parent index then it swaps the current index with parent index iteratively.

4. Private void bubbleDown(int index):

This method takes index as an input and loops, swap the current index with leftchild index if the current index is greater than the left child index.

5. Private void swap(int i, int j):

This method is used for index based swapping.

6. Private int childIndex(int I, int c):

This method returns the child index based on input.

7. Private int parentIndex(int i):

This method returns the parent index based on input.

8.Private boolean hasParent(int i):

This method returns true if it has a parent.

9. Private boolean hasChild(int i, int c):

This method returns true if it has a child.

10. Public boolean isNotEmpty():

This method checks if heap is empty or not.

Methods in HauffmanTree class:

1. Public void init(Freq_Table ft, int priorityQueue):

This method takes frequency table and integer variable where the integer value of 1 denotes CachefourwayHeap. Then it calls heapify method passing the frequency table as input then insert is called which internally merges two huffman trees.

2. Public HashMap<Integer, String> getSymbolTable()

This method is called in the encode method. It traverses through the huffman tree and appends ‘0’ to the left subtree and ‘1’ to the right subtree. It computes and returns the symbol table.

3. Private HuffmanTree merge(HuffmanTree ht1, HuffmanTree ht2)

This method takes two generate a new huffman tree by merging the new node to the existing huffman tree.

Methods in Encoder class :

1. Public void encode():

The Huffman tree built by 4-way heap is used for assigning codes to the tree. After assigning the codes, the encoded bits are written into a binary file byte by byte. For a distinct input message the corresponding encoded message is also written into ‘code_table.txt’ file.

2. public static void main(String args[]):

This method reads the input file and calls the encode method.

Methods in Store class :

The following are the getter and setter methods which are used for implementing various funtionalities.

1. public int getNumb()
2. public int getFreq()
3. public void setNumb()
4 public void setFreq()

Methods in decoder class :

1. public void decode()

This method uses the binary tree to decode the encoded.bin using code table.

2. public static void main(String args[])

This method takes two input files namely encoded.bin and code_table.txt and calls the decode method.

Methods in Iterator class:

1.boolean hasNext():

It checks bit index with 8 and index with array length and returns boolean value appropriately.

2. boolean getNextBit():

If bit index is 8, it increments the index and sets the bit index as 0.

Structure of the program

Encoder

- First input file which needs to be encoded is read.
- Now, a hash map is generated between elements in the file and their respective frequencies.
- Using the map which contains elements and the corresponding frequencies, Huffman tree is built.
- We generate a code table and write it to code_table.txt.
- Using the code table, we traverse the input file and convert it into bytes and write it to encoded.bin.

Decoder

- The decoder takes encoded.bin and code_table.txt as arguments.
- By using the code table a decoder tree is built.
- Next, it converts the binary encoded file to string.
- We iterate through the string and traverse through the decoder tree to get the elements.
- Finally, we write elements to the file decoded.txt.

Performance Analysis Results and Explanation

Huffman tree is built by using three priority queue data structures Binary heap, 4-way Cache Optimized Heap and Pairing Heap. The runtime is measured by using the sample_input_large.txt file provided. This running time is measured for 10 iterations and the average time is taken for these three priority queues. The reading time from an input file is ignored. The running time is measured for the implementation of respective priority queue data structure and construction of Huffman tree without assigning the codes to the tree.

Conclusions

- It was observed that Cache optimized 4-way heap has better running time performance of 2.7677 sec.
- Next, Binary Heap has a running time of 2.8947 sec.
- Pairing Heap running time comes out to be 3.91667 sec.

Binary heap is relatively slower than 4-way heap. Because for insertion in 4-way heap the number of levels moved up will be halved. Remove min operation do 4 compares per level which is 2 for binary heap. But the number of levels is half.

For 4-way heap to utilize the cache optimization, the array is shifted by 3 positions so that all the siblings are in the same cache line and it has approximately $\log_4 n$ cache misses for average remove min operation. Initially all the siblings are in 2 cache lines which has $\log_2 n$ cache misses for average min operation.

	Binary Heap	**4-way Cache Optimized Heap**	**Pairing Heap**
Runtime Measured	2.8947 sec	2.7677 sec	3.91667 sec

Running Time for three priority queues.

Based on the above conclusions, 4-way heap proved to be the fastest and efficient way of generating Huffman Tree. Using this 4-way heap data structure Huffman Encoder and Decoder has been implemented.

<u>Decoding Algorithm Used and It’s complexity</u>

- The decoder is constructed using the binary tree. Code Table is taken as the input**.**
- The codes are from the code table are read and we check whether the bit is ‘0’ or ‘1’.
- A new left node is created and initialized to -1 if the bit is ‘0’ and left child of root is empty.
- A new right node is created and initialized to -1 if the bit is ‘1’ and right child of root is empty.
- If we are reading the last bit in the code of a element then we create a left or right node corresponding to bit ‘0’ or ‘1’ initialized with the value of the element corresponding to that code.
- While decoding, we traverse the decoder tree to find the leaf node which contains the data.

The complexity of the algorithm is of order **O(nm),** where ‘n’ is the number of encoded messages and ‘m’ is the length of largest encoded message.