**Huffman’s Compression Algorithm**



**BTech/II Year CSE/IV Semester**

**19CSE212/Data Structures and Algorithm**

**Case Study Report**

|  |  |
| --- | --- |
| **Rollno** | **Name** |
| **CB.EN.U4CSE21204** | Anish Nandhan |
| **CB.EN.U4CSE21226** | Harjith L R S |

**Introduction -**

Hybrid data structure is the concept of using two or more data structures combined to attain a desired algorithm or output. These structures leverage the strengths of different data structures to provide efficient operations for specific use cases.

Significance of hybrid data structures in solving complex problems -

* Performance Optimization: By combining different data structures, hybrid data structures can optimize specific operations for improved performance.
* Space Efficiency: Hybrid data structures can leverage the space efficiency of certain data structures while still providing desired functionalities. This can be crucial in scenarios with limited memory resources or when dealing with large-scale data.
* Scalability: Hybrid data structures can often provide better scalability compared to individual data structures. They can handle large datasets and accommodate growing requirements without sacrificing performance.

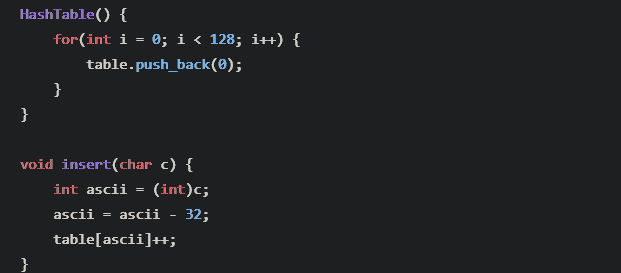
**Problem Statement -**

Design and implement an efficient lossless compression algorithm for text files. The program should take a text file as input and generate a compressed version of the file. The compressed file should be smaller in size, making efficient use of available storage space. The program should also be capable of decompressing the compressed file while being lossless.

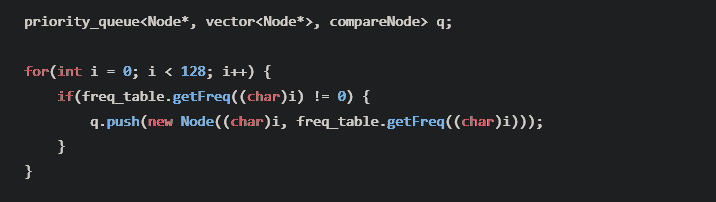
**Hybrid data structure implementation -**

The problem has been solved using hybrid data structure comprising of priority queue and hash tables.

* Hash table -



* Priority Queue -

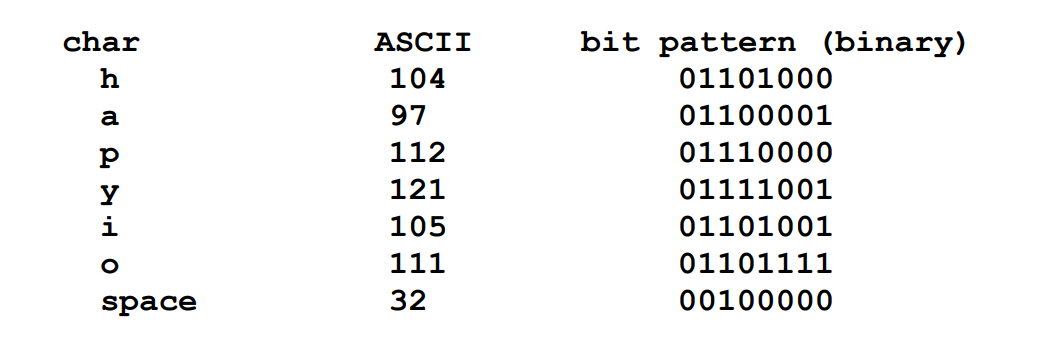


* Huffman tree

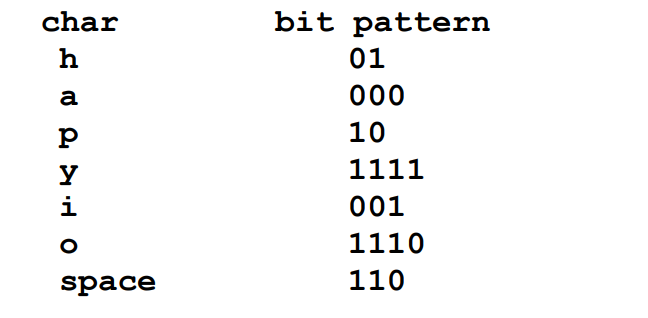
**Concept -**

The basic concept of Huffman encoding is to optimize the number bits used to store the contents of the file. It comes under lossless compression where the exact same file will be reproduced after decoding the compressed file.

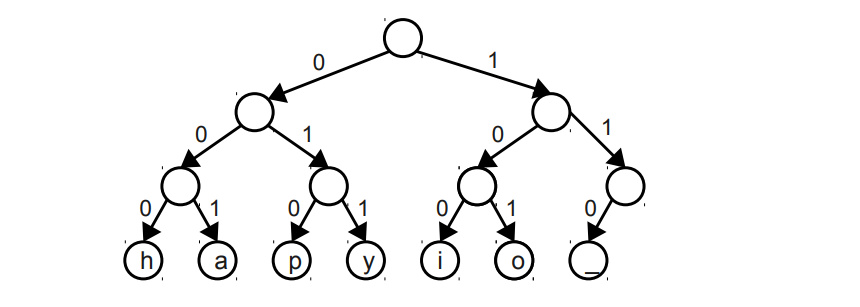
* Characters have unique code by default having 8 bits to represent them.



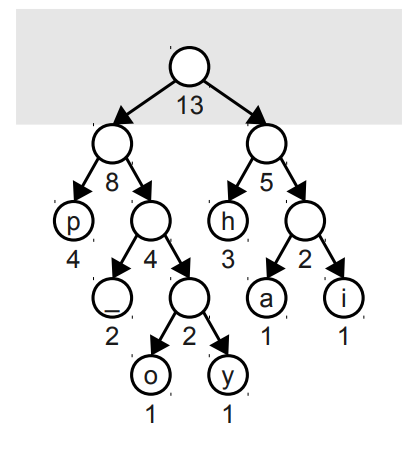
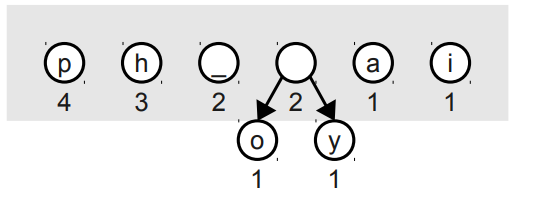
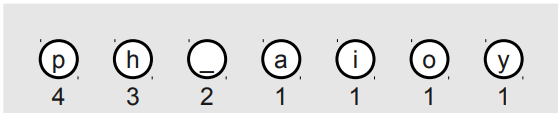
* These can be reduced by creating a new set of bits depending upon the file. These need less space to be them.



* These Huffman codes have to be used for encoding the original string. These are stored in a specialized tree called Huffman tree, based on the principle where two different characters do not have same prefix.



* The codes are stored in a priority queue based on the number occurrence of a character. First two elements are popped which have minimum frequency. Then they are added up and new node with null value is created and pushed into the same queue. Until only one root element is present.

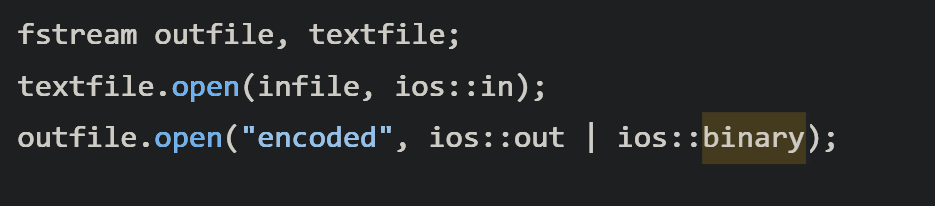


**Implementation -**

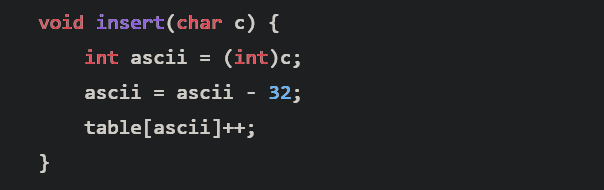
The solution to this problem statement is to use Huffman’s compression algorithm, implemented by using hybrid data structure making up of priority queue and hash tables.

**Algorithm -**

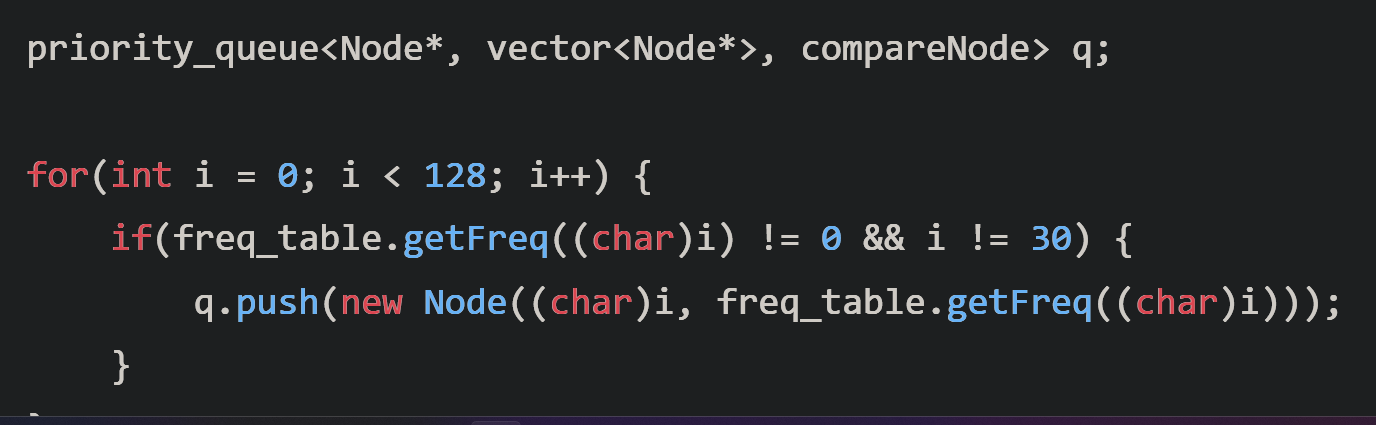
Step 1 – Input a text file for applying Huffman’s compression algorithm



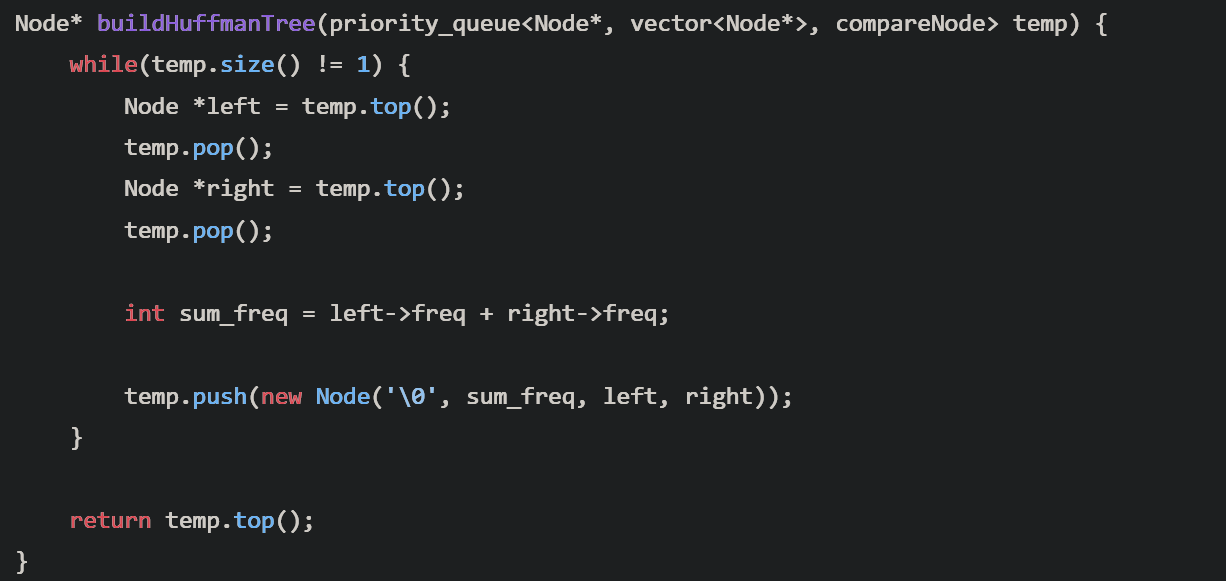
Step 2 – Counting frequency of each character using hash function to convert ascii values to index of an array. These frequencies will be stored in a vector table.



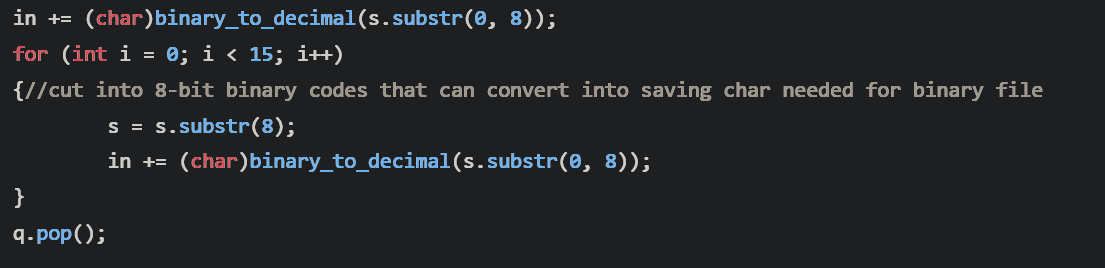
Step 3 – Using these frequencies push them into a priority queue to make a Huffman’s tree used for encoding.



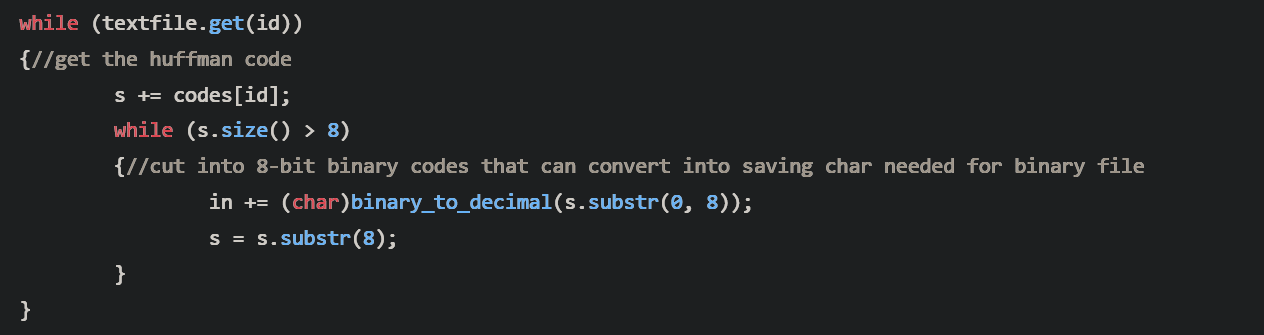
Step 4 – Create Huffman’s tree using these frequencies by popping the two least frequent elements from priority queue and pushing a new node having frequency of sum of those two. Repeat this process until only one element is present int data structure.



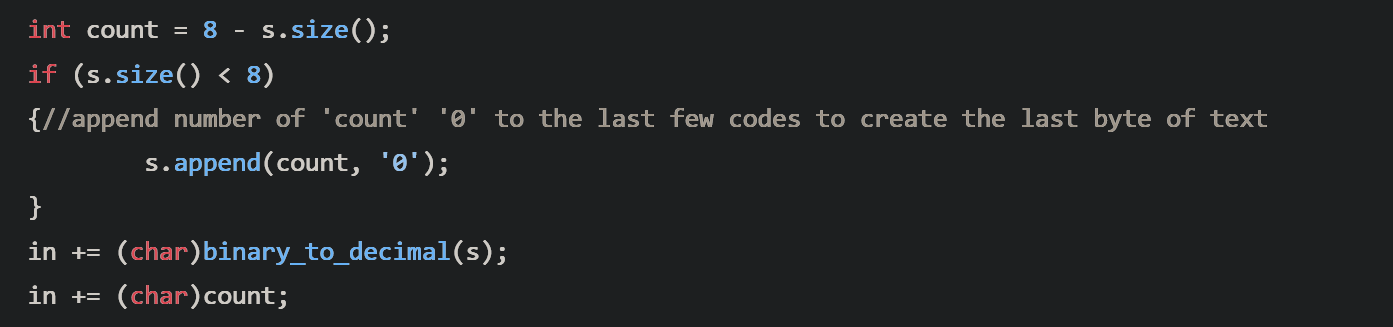
Step 5 – Huffman tree’s root node is available. Converting each character to binary equivalent using the Huffman’s tree.



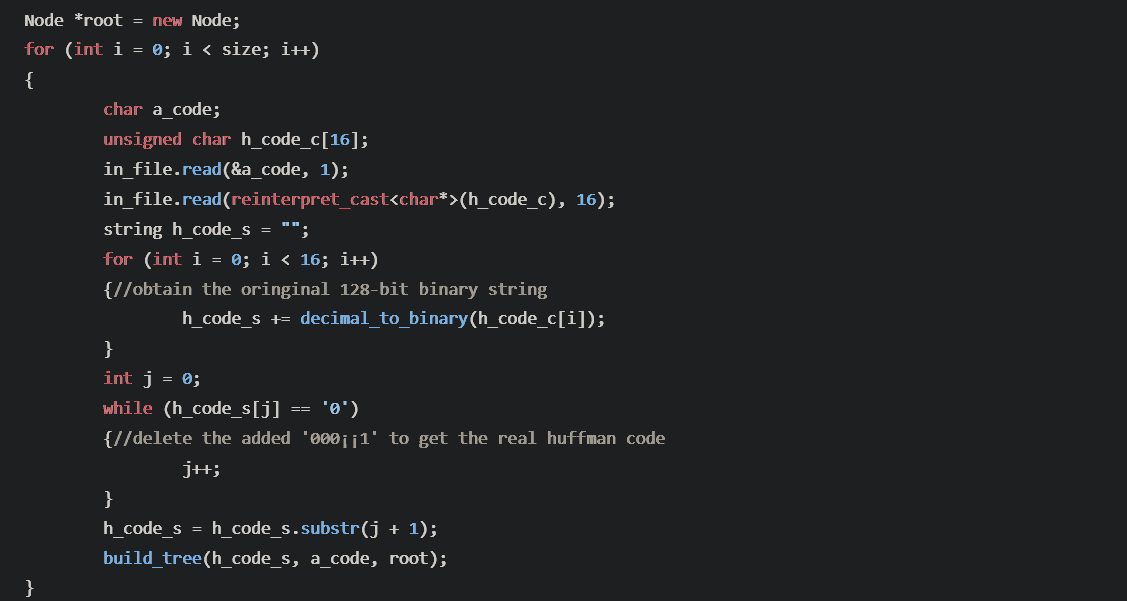
Step 6 – Writing it into the compressed output file



Step 7 – Adding buffer 0s as OS may add garbage values if the size of binary value is not divisible by 8.



Step 8 – Decompressing the compressed file. Using the huffman tree encoded in compressed file. Rebuild the Huffman tree using this encoded information.



Step 9 – With this tree recreated decode the variable length encoded binary message and writing it into a new file.

Step 10 – Output the decompressed file.

**Practical Application -**

**The practical applications of Huffman text compression include:**

* File Compression: Huffman coding is commonly used in file compression algorithms such as ZIP and GZIP. By applying Huffman coding to the text content within a file, it can significantly reduce its size, making it easier to store, transfer, and archive.
* Data Transmission: Huffman coding is widely employed in data transmission protocols, such as those used in telecommunications and network communications. When transmitting text-based data over networks or communication channels, the compressed data can be transmitted faster due to its reduced size.
* Text Messaging and Chat Applications: Text compression can be beneficial in text messaging and chat applications, where character limits or bandwidth limitations exist.

**Performance Analysis:**

* The first part of the process counting the frequencies of the characters is O(n), since we iterate through all the characters in the file.
* The process of creating a priority queue is O(nlogn):
  + Inserting an element in a priority queue is O(logn)
  + We insert n elements
  + So time complexity = O(n) . O(logn) = O(nlogn)
* Building the Huffman tree is again O(klogk), where k is the number of unique symbols:
  + Deleting from a priority queue(implemented as a binary heap) is O(logk)
  + And we delete k and insert k symbols.

So, the time complexity for the Huffman coding algorithm = O(n) + O(nlogn) = O(nlogn).

**Space Complexity:**

Space complexity of the code -

* O(k) - where k is the number of unique symbols

**Presentation:**

https://tome.app/uiwgerurouddfgfuadfjdgdfofwdjf/compress-to-impress-the-art-of-huffman-text-compression-clj2i327t0qlspj3bzyp5m27z

**GitHub repo:**

<https://github.com/AnishNandhan/huffman-text-compression>

**References:**

* <https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1176/assnFiles/assign6/huffman-encoding-supplement.pdf>
* <https://www.youtube.com/watch?v=JsTptu56GM8&t=270s&pp=ygUQaHVmZm1hbiBlbmNvZGluZw%3D%3D>
* <https://www.youtube.com/watch?v=B3y0RsVCyrw&t=782s&pp=ygUQaHVmZm1hbiBlbmNvZGluZw%3D%3D>
* <https://edu.anarcho-copy.org/Algorithm/grokking-algorithms-illustrated-programmers-curious.pdf>