

HUFFMAN ENCODER



Instructor: Sir Muhammad Faizan

Group:

Mustafa Nouman – 01-134242-087

Kamran Ahmed – 01-134242-055

Muhammad Hamid – 01-134251-051

Huffman Encoder Using Binary Trees

The Concept:

Huffman Encoding is a technique to compress text files by replacing the bits of the most frequent occurring characters with lower bit count. For example, in the string "aaaa bb c ddddd", d occurs five times. So, if we replace its bits with lower bit count, it should help in reducing the size of the file.

Our Implementation:

This project contains two header files and two source files:

- hfm.h
- hfm.cpp
- binary_tree.h
- binary_tree.cpp

Binary_tree.h and binary_tree.cpp

The binary_tree.h header file contains the following:

- *TreeNode Structure*: A struct for representing node in a binary tree. It contains int frequency, char character, and *TreeNode** left and *TreeNode** right pointers.
- *BinaryTree Class*: A binary tree implementation. Contains these private data members and functions:
 - *TreeNode* root*
 - *map<char, string> Traversal(TreeNode* &root, map<char, string> mp string s):*
 - This function is called inside of *GetCodeByTraversal* function. It basically calculates the 0s and 1s taken in reaching a character. It then stores 0s and 1s in the form of a bit string against the character using a map. It does this for all the characters,

basically traversing using recursion till each leafnode is reached and character's bit string calculated. Then it returns the final map.

- *string decodeII(string s, TreeNode* &node, string &bitstream, int &counter):*
Decodes the bit stream using recursion

Contains these public functions:

- *BinaryTree():* The constructor for binary tree
- *void CreateNode(char character, int frequency):* Creates root node of binary tree (only implemented for root node initialization)
- *int returnFrequency():* Returns frequency stored in the root node
- *TreeNode GetRoot:* Returns the pointer to the root node of the tree
- *map<char, string> GetCodeByTraversal():* A wrapper over the traversal function. Returns the map that is returned from traversal function.
- *string GetStringDecoded(string &bitstream):* Calls the decodeII function.

hfm.h and hfm.cpp

Hfm.h header file contains the following class:

HuffmanEncoder:

This class contains the following private data members and functions:

- *vector<BinaryTree> BTlist:* A vector for storing binary trees
- *char* CharacterList:* A list of unique characters in the text
- *map<char, string> BinaryCodes:* Map for storing bits string against each character
- *string Text:* The text extracted from file
- *string BitStream:* The stream of bits
- *string DecodedText:* The decoded text
- *void readFile(const string &filename):* Reads the file and stores the text in the "Text" data member.
- *void CreateLists(string str):* Extracts unique characters and their frequencies using map and for loop. stores unique characters in "CharacterList". Creates binary trees,

initializes their roots and stores them in the "BTlist" by looping. Sorts the binary tree list "BTlist" in the end.

- *void MergeBinaryTrees()*: Loops over the binary trees list, extracts two minimum frequency trees using GetMinimumTree function. Create a new tree and set it's root frequency equal to the sum of extracted trees' frequencies. Attach the extracted trees as sub trees of the new tree and push that new tree back in the list. Keeps on repeating till only one tree is left.
- *BinaryTree GetMinimumTree()*: Returns the minimum tree in the list
- *void SortList()*: Sorts the "BTlist" by frequencies.
- *void WriteAsBitStream()*: Writes one long bitstream by placing each character's bit code instead of the character itself. Stores this bitstream in the "BitStream" data member.
- *void WriteBytesToFile()*: Tries writing the bit stream to a .bin file by opening the file object in ios::binary and ios::out. But this implementation is not correct and is under development.

Public functions are the following:

- *void encode(string filename)*: Basically a wrapper over the private functions. Takes the filename and implements above mentioned functionality over the file's text.
- *void decode()*: Calls the GetStringDecoded function of the tree inside "BTlist".

Conclusion

Our project shows how Huffman Encoding can compress data by assigning shorter bits to frequently occurring characters. Using binary trees, frequency analysis, recursive traversal, and decoding, the implementations demonstrates workflow of Huffman encoder and decoder.

Zip file link:

https://drive.google.com/file/d/1gVpN8em4cwrHaUVy8oyex3J4TtCevrIT/view?usp=drive_link