Home work Documentation

Huffman compression/Huffman coding

Huffman compression is an algorithm that is commonly used for lossless data compression. This algorithm was developed by David. A. Huffman in 1952.

The idea of the algorithm is as follows: knowing the probabilities(weights) of symbols appearing in a message, we can describe the procedure for constructing codes consisting of some number of bits. Symbols that have the highest frequency of appearance are more likely to be assigned shorter codes. Huffman codes are prefixed (that is, no codeword is a prefix of another), which allows them to be uniquely decoded.

Huffman algorithm at the input receives a table of the frequency of occurrence of characters in the message. Further, based on this table, a Huffman coding tree is built.

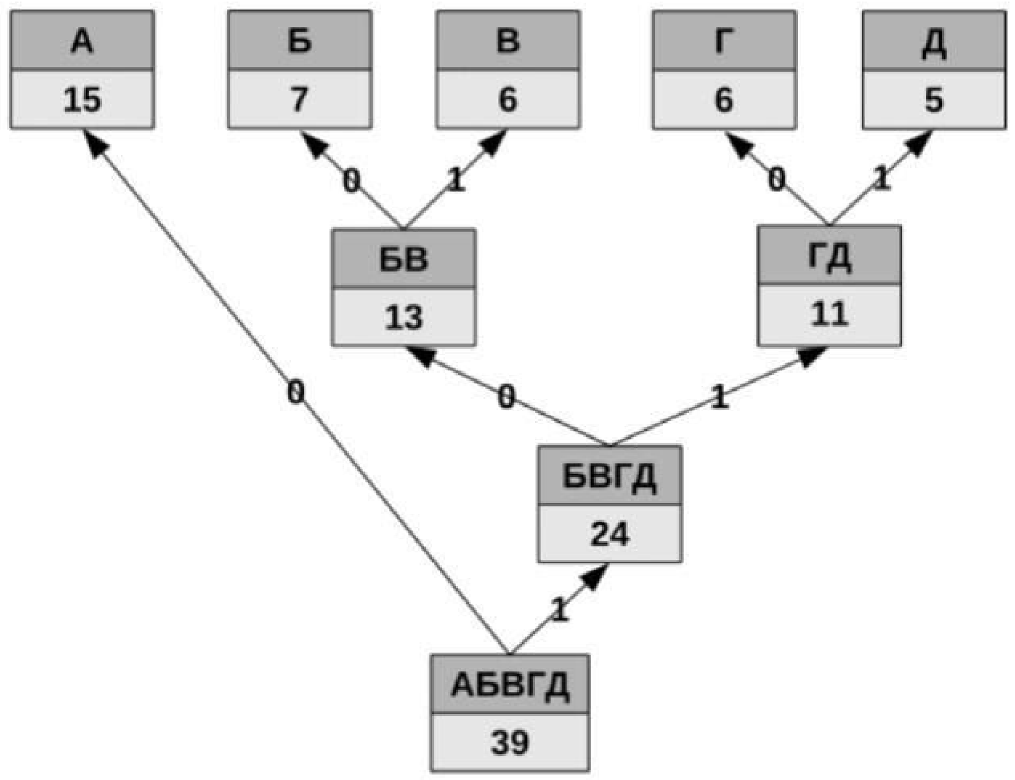
1. Create a leaf node for each symbol.
2. Two free tree nodes with the least weights are selected.
3. Their parents are created with a weight equal to their total weight.
4. The parent is added to the free list.
5. The steps starting from the second are repeated until only one free node remains in the free list. Which will be the root of our tree.

Suppose we have the following frequency table:

INPUT:

Symbol	А	Б	В	Г	Д
Frequency	15	7	6	6	5

Here is the visualization of how Huffman tree is constructed



OUTPUT:

Symbol	А	Б	В	Г	Д
Code	0	100	101	110	111

Huffman decoding works as follows:

1. We start from the root and do the following until a leaf is found.
2. If the current bit is 0, we move to the left node of the tree.
3. If the bit is 1, we move to the right node of the tree.
4. If during traversal, we encounter a leaf node, we print the character of that particular leaf node and then again continue the iteration of the encoded data starting from step 1.

List of libraries

- map.h
- map2.h
- huffman_encode.h
- huffman_decode.h

List of used structures and functions for **map.h**

- struct map_in
- struct map
- map_pair()
- create_map()
- map_set
- map_show()
- map_size()
- map_value()
- map_key()

List of used structures and functions for **huffman_encode.h**

- struct node
- struct nodes
- createNode()
- createNodes()
- swap()
- heapify()
- nodes_size_one()
- min_element()
- insert_node()
- buildtree()
- printArr()
- isLeaf()
- create_tree()
- buildHuffmanTree()
- printCodes()

List of used structures and functions for **map2.h**

- struct map_inn
- struct mapp
- map_pair2()
- create_map2()
- map_set2()

- map_show2()
- map_size2()
- map_value2()
- map_key2()

List of used structures and functions for **huffman_decode.h**

- decode_file()
- encoded_string()
- encoded_code()

List of functions for **hw.c**

- user_input()
- read_file()
- write_file()
- write_code_file()

Description of used libraries, structures and libraries

map.h - was used in order to count the number of occurrences of the symbols

- **struct map_in** is a single node of the linked list. Fields of type char and int.
- **struct map** is a list of nodes linked with each other. Fields of type map_in
- **map_in map_pair(char key)** returns node of type map_in of the list by allocating parameter key to the node
- **map *create_map()** returns a linked list of maps
- **void map_set(map *, char)** sets a single node of map to the list of maps with new key as parameter char
- **void map_show(map*)** prints the keys and values of the list of maps
- **int map_size(map*)**returns the size of the map
- **int *map_value(map *)** returns an array of the values of map
- **char* map_key(map *)** returns char array of the keys of the map

huffman_encode.h - is the group of function that is used to implement huffman compression

- **struct node** is a is used in order to save values of symbols and their frequencies at one node
- **struct nodes** is to store the list of single node

- **node* create_node(char, int)** allocates 1st parameter to the node's char field, 2nd parameter to the node's int field
- **nodes* create_nodes(int)** creates a tree, as parameter it takes a volume of the tree;
- **void swap(node **a, node**b)**swaps the node a with the node b
- **void heapify(nodes **arr, int id)** is the function to build binary tree
- **int nodes_size_one (nodes **arr)** is used to check if node is not empty and has 1 element
- **node* min_element(node**arr)** is to get minimum value node in the tree
- **void insert_node(nodes**arr, node* elem)** is used to insert elem in to the tree arr as it is requires huffman code
- **void build_tree(nodes**arr**)builds the tree
- **void print_arr(int arr[], int n)**prints an array of size n
- **int is_leaf(node*)**check if it's the end of the tree
- **nodes* create_tree(char [], int [], int size)** creates a tree with given parameters and size as 3rd parameter
- **nodes* build_huffman_tree(char [], int[], size)** function that builds exact huffman tree
- **void print_codes(nodes* ,int [], int)**prints codes of the huffman tree

map2.h- is used in order to store symbol and the code of that symbol

- **struct map_inn** is a single node of the linked list. Fields of type char and int.
- **struct mapp** is a list of nodes linked with each other. Fields of type map_inn
- **map_inn map_pair2(char key, char* value)** returns node of type map_inn of the list by allocating parameters key and value to the node
- **mapp *create_map()** returns a linked list of maps
- **void map_set2(mapp *, char, char*)** sets a single node of map to the list of maps with new key and value as parameter char and char*
- **void map_show2(mapp*)** prints the keys and values of the list of maps
- **int map_size2(mapp*)**returns the size of the map
- **char **map_value2(mapp *)** returns an array of the values of map
- **char* map_key(map *)** returns char array of the keys of the map

huffman_decode.h - is the group of function that is used to implement huffman decompression

- **char* decode_file(node* root, char *s)** decodes given tree with the given string of codes

    - **void encoded_string(node * , mapp*, char[], int)** sets symbols and their corresponding codes to the map
    - **void encoded_code(char*encodedString, char*text, char*key, char** value, mapp* mp)** makes a single string of codes 1st parameter is a pointer to the buffer string that will contain the encoded string, 2nd parameter is a actual text, 3rd and 4th are the keys and values of the map, where the symbols and their codes are saved, 5th is to know mapp's size.

- **user_input()** is a function that interacts with the user
- **read_file()** reads data from the file that is needed to be compressed
- **write_file()** writes data that was compressed and already decompressed
- **write_ code_file()** writes codes of the compressed data and its symbols with their codes