Thời gian còn lại 0:07:48

Câu hỏi **4**

Không hoàn thành

Chấm điểm của 2,00

Huffman compression is an information encryption algorithm used to compress data based on optimizing the encoding of characters in the original string by constructing a binary code representing each character in it. The algorithm aims to build a binary encoding representing each character so that the most frequent characters will have a short binary code representing it, and vice versa.

Implement steps:

1. Building a table represent the frequency of occurrence of characters in a string
2. Building a encoding binary tree from table at step 1
3. Traversing the tree to generate a character-to-binary mapping table
4. Encoding input string and return the result

Example: string s = “ABRACADABRA”

1. **Building a table represent the frequency of occurrence of characters in a string**

Character	Frequency
A	5
B	2
C	1
D	1
R	2

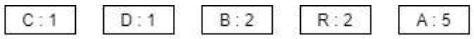
2. **Building a encoding binary tree from table at step 1**

   How to do:

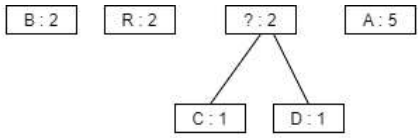
   Extract the 2 nodes x, y with the **lowest frequency** in the queue, respectively, and replace them with a new node z representing node mixing. The frequency of z is calculated as the sum of the frequencies of node x and node y. The node with the lower frequency will be the left child and the node with the higher frequency will be the right child. After n - 1 merge, there is only 1 node left in the queue and that is the root node of the encryption tree.

   Demonstration:

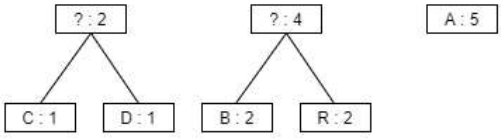
2.1



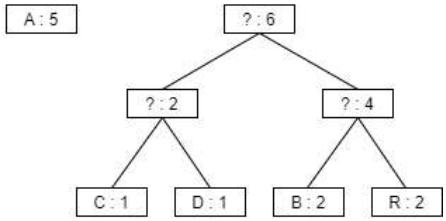
2.2



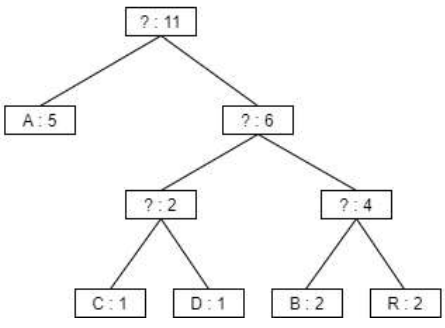
2.3



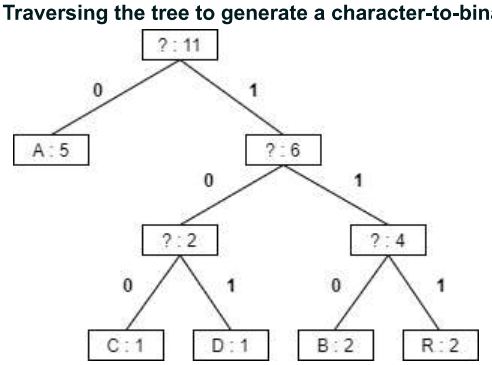
2.4



2.5



3. **Traversing the tree to generate a character-to-binary mapping table**



Character	Encoded string
A	0
B	110
C	100
D	101
R	111

4. **Encoding input string and return the result**

   Encoded string: 01101110100010101101110

Class **BTNode** is used to store a node in binary tree, described on the following:

```
class Node
{
private:
    int freq;
    char character;
    Node *pLeft, *pRight;
    friend class Comparison;
    friend class HuffmanCode;

public:
    Node(){};
    Node(int freq, char character) : freq(freq), character(character), pLeft(NULL), pRight(NULL){};
      Node(int freq, char character, Node *pLeft, Node *pRight) : freq(freq), character(character),
pLeft(pLeft), pRight(pRight){};
};
```

Where `character` is the value of node (char), `freq` is the frequency of character (integer), `pLeft` and `pRight` are the pointers to the left node and right node of it, respectively.

Class **Comparison** is utility class for constructing Encoding binary tree, described on the following:

```
class Comparison
{
public:
    bool operator()(Node *a, Node *b)
    {
        return a->freq > b->freq;
    }
};
```

Class **HuffmanCode** is used to implement encoding process from Huffman algorithm, described on the following:

```
class HuffmanCode
{
private:
    map<char, int> freqTable;
    map<char, string< hashTable;

public:
    Node *constructTree();
    void encodeCharacter(Node *root, string prefix);

    void constructFreqTable(string s)
    {
        for (int i = 0; i < s.size(); i++)
        {
            freqTable[s[i]]++;
        }
    }

    string encode(string s)
    {
        string result = "";
        for (int i = 0; i < s.size(); i++)
        {
            result += hashTable[s[i]];
        }
        return result;
    }

    void printFreqTable()
    {
        for (auto const &i : freqTable)
        {
            cout << i.first << " - " << i.second << endl;
        }
    }

    void printHashTable()
    {
        for (auto const &i : hashTable)
        {
            cout << i.first << " - " << i.second << endl;
        }
    }

    void deleteTree(Node *root)
    {
        if (!root)
        {
            return;
        }
        deleteTree(root->pLeft);
        deleteTree(root->pRight);
        delete root;
    }
};
```

Where `freqTable` is the character frequency statistic table, `hashTable` is the binary-mapping table.

Request: Implement function:

```
// Function for step 2
Node *HuffmanCode::constructTree()
// Function for step 3
void HuffmanCode::encodeCharacter(Node *root, string prefix)
```

Input: A string in plaintext

Output: A string in ciphertext

Note: In this exercise, the libraries `iostream`, `stack`, `queue`, `map`, `vector`, `list algorithm`, `string` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`string s = "ABRACADABRA";`
`HuffmanCode myCode;`
`myCode.constructFreqTable(s);`
`Node *tree = myCode.constructTree();`
`myCode.encodeCharacter(tree, "");`
`string encodedString = myCode.encode(s);`
`cout << "Encoded string: " << encodedString;`
`myCode.deleteTree(tree);`
	`Encoded string: 01101110100010101101110`

Answer: (penalty regime: 0, 0, 0, 10, 20, ... %)

Reset answer

```
Node *HuffmanCode::constructTree()
{
    // STUDENT'S ANSWER
}

void HuffmanCode::encodeCharacter(Node *root, string prefix)
{
    // STUDENT'S ANSWER
}
```

Precheck Kiểm tra

◀ Testing

Chuyển tới...

Link google meet Final test (11/12/2021) ►

Địa chỉ: Nhà A1- 268 Lý Thường Kiệt, Phường 14, Quận 10, Tp.HCM.
Email: elearning@hcmut.edu.vn