허프만 코드

Date	@2024/11/06
Done	☑
Week	W 45

C++로 코드를 완성하는 스켈레톤 문제 형식

```
#pragma once

#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>

using namespace std;

namespace EnCoding
{
    // 트리 노드 구조체 정의
    struct Node {
        char ch;           // 문자
        int freq;          // 빈도수
        int char_count;    // 해당 노드를 구성하는 문자 개수
        Node* left, * right; // 왼쪽, 오른쪽 자식 노드

        Node(char ch, int freq, int char_count = 1)
            : ch(ch), freq(freq), char_count(char_count), left(nullptr), right(nullptr) {}
    };

    // 우선순위 큐에서 사용하는 비교 함수
    struct Compare {
        bool operator()(Node* left, Node* right) {
            // 빈도가 같을 경우
            if (left->freq == right->freq) {
                // 문자 개수가 적은 것을 우선
                if (left->char_count != right->char_count) {
                    return left->char_count > right->char_count;
                }
                // 빈도와 문자 개수 모두 같으면 '\0' 노드를 마지막으로 정렬
                return left->ch > right->ch;
            }
            // 빈도가 다를 경우 빈도가 작은 노드가 우선
            return left->freq > right->freq;
        }
    };

    // 허프만 트리에서 코드를 생성하는 재귀 함수
    void encode(Node* root, string str, unordered_map<char, string>& huffmanCode) {
        if (!root) return;

        // 리프 노드일 경우 문자와 해당하는 코드 문자열을 맵에 저장
        if (!root->left && !root->right) {
            huffmanCode[root->ch] = str;
        }

        // 왼쪽에 0을 추가하고 재귀 호출
        encode(root->left, str + "0", huffmanCode);
        // 오른쪽에 1을 추가하고 재귀 호출
        encode(root->right, str + "1", huffmanCode);
    }

    // 허프만 코딩 실행 함수
    void buildHuffmanTree(const string& text) {
        // 빈도 계산을 위한 맵
        unordered_map<char, int> freq;
        for (char ch : text) {
            freq[ch]++;
        }
```

```
        // 우선순위 큐를 사용하여 빈도가 낮은 문자부터 처리
        priority_queue<Node*, vector<Node*>, Compare> pq;
        for (auto pair : freq) {
            pq.push(new Node(pair.first, pair.second));
        }

        // 트리 생성 과정 디버깅 출력을 추가하여 확인
        while (pq.size() > 1) {
            // 가장 빈도가 낮은 두 노드를 추출
            Node* left = pq.top(); pq.pop();
            Node* right = pq.top(); pq.pop();

            // 디버깅: 추출한 노드 출력
            cout << "Combining nodes: " << (left->ch ? left->ch : '*') << "(" << left->freq << ") and "
                << (right->ch ? right->ch : '*') << "(" << right->freq << ")\n";

            // 두 노드를 합쳐 새로운 내부 노드를 생성 (빈도의 합을 가짐)
            int sum = left->freq + right->freq;
            int char_count = left->char_count + right->char_count; // 두 노드의 문자 개수 합산
            Node* newNode = new Node('\0', sum, char_count);
            newNode->left = left;
            newNode->right = right;

            // 새로운 노드를 우선순위 큐에 삽입
            pq.push(newNode);
        }

        // 최종 트리 루트 노드
        Node* root = pq.top();

        // 허프만 코드 저장 맵
        unordered_map<char, string> huffmanCode;
        encode(root, "", huffmanCode);

        // 허프만 코드 출력
        cout << "Huffman Codes:\n";
        cout << "문자열의 총 길이 : " << root->freq << '\n';
        for (auto pair : huffmanCode) {
            cout << pair.first << " " << pair.second << "\n";
        }
    }
    void Test() {
        string text = "APPLEBANANAMANGOJUICE";
        buildHuffmanTree(text);
    }
}
```