【第三十一周】哈弗曼编码（Halfman-Coding）与二叉字典树

ascii

```cpp
/*************************************************************************
  > File Name: 1.ascii.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

int main() {
    printf("%c\n", 'a');
    printf("%d\n", 'a'); // 十进制
    printf("%X\n", 'a');
    printf("%c\n", 'b');
    printf("%d\n", 'b'); // 十进制
    printf("%X\n", 'b');
    printf("%c\n", 'c');
    printf("%d\n", 'c'); // 十进制
    printf("%X\n", 'c');
    return 0;
}
```

1167. 连接棒材的最低费用

为了装修新房，你需要加工一些长度为正整数的棒材 。棒材以数组 `sticks` 的形式给出，其中 `sticks[i]` 是第 `i` 根棒材的长度。

如果要将长度分别为 `x` 和 `y` 的两根棒材连接在一起，你需要支付 `x + y` 的费用。 由于施工需要，你必须将所有棒材连接成一根。

返回你把所有棒材 `sticks` 连成一根所需要的最低费用。注意你可以任意选择棒材连接的顺序。

示例：

```
输入：sticks = [2,4,3]
输出：14
解释：从 sticks = [2,4,3] 开始。
1. 连接 2 和 3 ，费用为 2 + 3 = 5 。现在 sticks = [5,4]
2. 连接 5 和 4 ，费用为 5 + 4 = 9 。现在 sticks = [9]
所有棒材已经连成一根，总费用 5 + 9 = 14
```

```cpp
class Solution {
public:
    int connectSticks(vector<int>& sticks) {
        multiset<int> s;
        for (auto x : sticks) s.insert(x);
        int ans = 0;
        while (s.size() - 1) {
            int a = *(s.begin()); s.erase(s.begin());
            int b = *(s.begin()); s.erase(s.begin());
            s.insert(a + b);
            ans += a + b;
        }
        return ans;
    }
};
```

676. 实现一个魔法字典

设计一个使用单词列表进行初始化的数据结构，单词列表中的单词 **互不相同** 。 如果给出一个单词，请判定能否只将这个单词中**一个**字母换成另一个字母，使得所形成的新单词存在于你构建的字典中。

实现 `MagicDictionary` 类：

- `MagicDictionary()` 初始化对象
- `void buildDict(String[] dictionary)` 使用字符串数组 `dictionary` 设定该数据结构，`dictionary` 中的字符串互不相同
- `bool search(String searchWord)` 给定一个字符串 `searchWord` ，判定能否只将字符串中 **一个** 字母换成另一个字母，使得所形成的新字符串能够与字典中的任一字符串匹配。如果可以，返回 `true` ；否则，返回 `false` 。

示例：

```
输入
["MagicDictionary", "buildDict", "search", "search", "search", "search"]
[[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]
输出
[null, null, false, true, false, false]

解释
MagicDictionary magicDictionary = new MagicDictionary();
magicDictionary.buildDict(["hello", "leetcode"]);
magicDictionary.search("hello"); // 返回 False
magicDictionary.search("hhllo"); // 将第二个 'h' 替换为 'e' 可以匹配 "hello" ，所以返回 True
magicDictionary.search("hell"); // 返回 False
magicDictionary.search("leetcoded"); // 返回 False
```

```cpp
class Node {
public :
    Node() {
        flag = false;
        for (int i = 0; i < 26; i++) next[i] = nullptr;
    }
    bool flag;
    Node *next[26];
};

class Trie {
public :
    void insert(string &s) {
        Node *p = &root;
        for (auto x : s) {
            int ind = x - 'a';
            if (p->next[ind] == nullptr) p->next[ind] = new Node();
            p = p->next[ind];
        }
        p->flag = true;
        return ;
    }
    bool __search(string &s, int pos, Node *p, int n) {
        if (pos == s.size()) return p->flag && n == 0;
        int ind = s[pos] - 'a';
        if (p->next[ind] && __search(s, pos + 1, p->next[ind], n)) return true;
        if (n) {
            for (int i = 0; i < 26; i++) {
                if (i == ind || p->next[i] == nullptr) continue;
                if (__search(s, pos + 1, p->next[i], n - 1)) return true;
            }
        }
        return false;
```

```
    }
    bool search(string &s, int n) {
        return __search(s, 0, &root, n);
    }

private:
    Node root;

};

class MagicDictionary {
public:
    MagicDictionary() {}
    Trie tree;
    void buildDict(vector<string> dictionary) {
        for (auto s : dictionary) tree.insert(s);
        return ;
    }

    bool search(string searchWord) {
        return tree.search(searchWord, 1);
    }
};

/**
 * Your MagicDictionary object will be instantiated and called as such:
 * MagicDictionary* obj = new MagicDictionary();
 * obj->buildDict(dictionary);
 * bool param_2 = obj->search(searchWord);
 */
```

255. 验证前序遍历序列二叉搜索树

给定一个整数数组，你需要验证它是否是一个二叉搜索树正确的先序遍历序列。

你可以假定该序列中的数都是不相同的。

参考以下这颗二叉搜索树：

```
     5
    / \
   2   6
  / \
 1   3
```

示例：

```
输入:[5,2,6,1,3]
输出:false
```

```cpp
class Solution {
public:
    int pre_ind;
    bool judge(vector<int> &preorder, int l, int r) {
        if (r - l < 1) return true;
        int ind = l + 1;
        while (ind < r && preorder[ind] < preorder[l]) ind += 1;
        if (!judge(preorder, l + 1, ind)) return false;
        if (pre_ind != -1 && preorder[l] < preorder[pre_ind]) return false;
        pre_ind = l;
        if (!judge(preorder, ind, r)) return false;
        return true;
    }
    bool verifyPreorder(vector<int>& preorder) {
        pre_ind = -1;
        return judge(preorder, 0, preorder.size());
    }
};
```

面试题 17.17. 多次搜索

给定一个较长字符串`big`和一个包含较短字符串的数组`smalls`，设计一个方法，根据`smalls`中的每一个较短字符串，对`big`进行搜索。输出`smalls`中的字符串在`big`里出现的所有位置`positions`，其中`positions[i]`为`smalls[i]`出现的所有位置。

示例：

```
输入：
big = "mississippi"
smalls = ["is","ppi","hi","sis","i","ssippi"]
输出： [[1,4],[8],[],[3],[1,4,7,10],[5]]
```

```cpp
class Solution {
public:
    vector<int> sunday(string &s, string &t) {
        vector<int> ret;
        if (t.size() == 0) return ret;
        int last_ind[128] = {0};
        for (int i = 0; t[i]; i++) last_ind[t[i]] = i + 1;
        int lens = s.size(), lent = t.size(), i = 0;
        while (i + lent <= lens) {
            bool flag = true;
            for (int j = 0; j < lent; j++) {
                if (s[i + j] == t[j]) continue;
                flag = false;
                break;
            }
            if (flag) {
```

```
                ret.push_back(i);
                i += 1;
            } else {
                i += lent - last_ind[s[i + lent]] + 1;
            }
        }
        return ret;
    }
    vector<vector<int>> multiSearch(string big, vector<string>& smalls) {
        vector<vector<int>> ret;
        for (auto s : smalls) {
            ret.push_back(sunday(big, s));
        }
        return ret;
    }
};
```

32. 最长有效括号

给你一个只包含 `'('` 和 `')'` 的字符串，找出最长有效（格式正确且连续）括号子串的长度。

示例：

```
输入：s = "(()"
输出：2
解释：最长有效括号子串是 "()"
```

```
class Solution {
public:
    int longestValidParentheses(string s) {
        if (s == "") return 0;
        int ans = 0, __dp[s.size() + 5], *dp = __dp + 3;
        memset(__dp, 0, sizeof(__dp));
        dp[0] = 0;
        for (int i = 1; i < s.size(); i++) {
            dp[i] = 0;
            if (s[i] == '(') continue;
            if (s[i - 1] == '(') dp[i] = dp[i - 2] + 2;
            else {
                int j = i - dp[i - 1] - 1;
                if (j < 0 || s[j] == ')') continue;
                dp[i] = dp[i - 1] + 2 + dp[j - 1];
            }
            ans = max(ans, dp[i]);
        }
        return ans;
    }
};
```

76. 最小覆盖子串

给你一个字符串 `s` 、一个字符串 `t` 。返回 `s` 中涵盖 `t` 所有字符的最小子串。如果 `s` 中不存在涵盖 `t` 所有字符的子串，则返回空字符串 `""` 。

注意：

- 对于 `t` 中重复字符，我们寻找的子字符串中该字符数量必须不少于 `t` 中该字符数量。
- 如果 `s` 中存在这样的子串，我们保证它是唯一的答案。

示例 1：

```
输入：s = "ADOBECODEBANC", t = "ABC"
输出："BANC"

```

示例：

```
输入：s = "a", t = "a"
输出："a"
```

```cpp
class Solution {
public:
    string minWindow(string s, string t) {
        int cnt = 0, cnts[128] = {0};
        for (auto x : t) {
            cnts[x] -= 1;
            if (cnts[x] == -1) cnt += 1;
        }
        int ans_len = s.size() + 1, l = 0, r = 0;
        string ans = "";
        while (r <= s.size()) {
            if (cnt) {
                if (r == s.size()) break;
                cnts[s[r]] += 1;
                if (cnts[s[r]] == 0) cnt -= 1;
                r += 1;
            } else {
                cnts[s[l]] -= 1;
                if (cnts[s[l]] == -1) cnt += 1;
                l += 1;
            }
            if (cnt == 0 && r - l < ans_len) {
                ans_len = r - l;
                ans = s.substr(l, r - l);
            }
        }
        return ans;
    }
```

```
};
```

468. 验证IP地址

编写一个函数来验证输入的字符串是否是有效的 IPv4 或 IPv6 地址。

- 如果是有效的 IPv4 地址，返回 `"IPv4"` ；
- 如果是有效的 IPv6 地址，返回 `"IPv6"` ；
- 如果不是上述类型的 IP 地址，返回 `"Neither"` 。

IPv4 地址由十进制数和点来表示，每个地址包含 4 个十进制数，其范围为 0 - 255， 用(".")分割。比如， `172.16.254.1`；

同时，IPv4 地址内的数不会以 0 开头。比如，地址 `172.16.254.01` 是不合法的。

IPv6 地址由 8 组 16 进制的数字来表示，每组表示 16 比特。这些组数字通过 (":")分割。比如， `2001:0db8:85a3:0000:0000:8a2e:0370:7334` 是一个有效的地址。而且，我们可以加入一些以 0 开头的数字，字母可以使用大写，也可以是小写。所以， `2001:db8:85a3:0:0:8A2E:0370:7334` 也是一个有效的 IPv6 address地址 (即，忽略 0 开头，忽略大小写)。

然而，我们不能因为某个组的值为 0，而使用一个空的组，以至于出现 (::) 的情况。 比如， `2001:0db8:85a3::8A2E:0370:7334` 是无效的 IPv6 地址。

同时，在 IPv6 地址中，多余的 0 也是不被允许的。比如， `02001:0db8:85a3:0000:0000:8a2e:0370:7334` 是无效的。

示例：

```
输入: IP = "172.16.254.1"
输出: "IPv4"
解释: 有效的 IPv4 地址，返回 "IPv4"
```

```cpp
class Solution {
public:
    bool valid_ipv4(string &s, int l, int r) {
        if (r - l <= 0 || r - l > 3) return false;
        if (r - l > 1 && s[l] == '0') return false;
        int num = 0;
        for (int i = l; i < r; i++) {
            if (s[i] < '0' || s[i] > '9') return false;
            num = num * 10 + s[i] - '0';
        }
        return num < 256;
    }

    bool valid_ipv6(string &s, int l, int r) {
        if (r - l <= 0 || r - l > 4) return false;
        for (int i = l; i < r; i++) {
            if (s[i] <= '9' && s[i] >= '0') continue;
            if (s[i] <= 'F' && s[i] >= 'A') continue;
```

```
            if (s[i] <= 'f' && s[i] >= 'a') continue;
            return false;
        }
        return true;
    }

    string validIPAddress(string queryIP) {
        int cnt = 0, flag = 0, pre_ind = 0;
        for (int i = 0; i <= queryIP.size(); i++) {
            // ipv4
            if (queryIP[i] == '.' || (flag == 1 && queryIP[i] == 0)) {
                if (flag == 0) flag = 1;
                if (flag != 1 || !valid_ipv4(queryIP, pre_ind, i)) {
                    flag = -1;
                    break;
                }
                cnt += 1;
                pre_ind = i + 1;
            }
            // ipv6
            if (queryIP[i] == ':' || (flag == 2 && queryIP[i] == 0)) {
                if (flag == 0) flag = 2;
                if (flag != 2 || !valid_ipv6(queryIP, pre_ind, i)) {
                    flag = -1;
                    break;
                }
                cnt += 1;
                pre_ind = i + 1;
            }
        }
        if (flag == 1 && cnt == 4) return "IPv4";
        if (flag == 2 && cnt == 8) return "IPv6";
        return "Neither";
    }
};
```

89. 格雷编码

格雷编码是一个二进制数字系统，在该系统中，两个连续的数值仅有一个位数的差异。

给定一个代表编码总位数的非负整数 *n*，打印其格雷编码序列。即使有多个不同答案，你也只需要返回其中一种。

格雷编码序列必须以 0 开头。

示例：

```
输入: 2
输出:[0,1,3,2]解释:
00 - 0
01 - 1
```

```
11 - 3
10 - 2

对于给定的n，其格雷编码序列并不唯一。
例如，[0,2,3,1] 也是一个有效的格雷编码序列。

00 - 0
10 - 2
11 - 3
01 - 1
```

```cpp
class Solution {
public:
    vector<int> grayCode(int n) {
        vector<int> ret(1 << n);
        if (n == 0) {
            ret[0] = 0;
            return ret;
        }
        vector<int> code_n_1 = grayCode(n - 1);
        int len_n_1 = code_n_1.size();
        for (int i = 0; i < len_n_1; i++) {
            ret[i] = code_n_1[i] << 1;
            ret[2 * len_n_1 - i - 1] = code_n_1[i] << 1 | 1;
        }
        return ret;
    }
};
```

haffman

```cpp
/*************************************************************************
  > File Name: 10.haffman.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
```

```cpp
using namespace std;

struct node {
    node(int freq = 0, node *lchild = nullptr, node *rchild = nullptr)
    : freq(freq), ch(0), lchild(lchild), rchild(rchild) {}
    node(int freq = 0, char ch = 0)
    : freq(freq), ch(ch), lchild(nullptr), rchild(nullptr) {}
    char ch;
    int freq;
    node *lchild, *rchild;
};

struct CMP {
    bool operator()(const node *a, const node *b) const {
        return a->freq < b->freq;
    }
};

void extract_code(node *root, string prefix, vector<string> &code) {
    if (root->ch != 0) {
        code[root->ch] = prefix;
        return ;
    }
    extract_code(root->lchild, prefix + '0', code);
    extract_code(root->rchild, prefix + '1', code);
    return ;
}

int main() {
    int n, freq, freq_arr[128];
    char ch;
    cin >> n;
    multiset<node *, CMP> s;
    for (int i = 0; i < n; i++) {
        cin >> ch >> freq;
        freq_arr[ch] = freq;
        s.insert(new node(freq, ch));
    }
    while (s.size() > 1) {
        node *a = *(s.begin()); s.erase(s.begin());
        node *b = *(s.begin()); s.erase(s.begin());
        s.insert(new node(a->freq + b->freq, a, b));
    }
    node *root = *(s.begin());
    vector<string> code(128);
    extract_code(root, "", code);
    for (int i = 0; i < 128; i++) {
        if (code[i] == "") continue;
        printf("%c(%5d) : %s\n", i, freq_arr[i], code[i].c_str());
```

```
    }
    return 0;
}
```

开课吧