暨南大学本科实验报告专用纸(附页)

霍夫曼树`Huffman-tree`

课程名称_____数据结构_____成绩评定_____
实验项目名称__霍夫曼树`Huffman-tree`__指导老师___干晓聪___
实验项目编号__16__实验项目类型__设计性__实验地点_数学系机房_
学生姓名___郭彦培___学号___2022101149___
学院_信息科学技术学院_系_数学系_专业_信息管理与信息系统_
实验时间_2024 年 6 月 13 日上午_~__2024 年 7 月 13 日中午___

1. 实验目的

实现一个霍夫曼树并提供初始化后的编解码

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

3. 程序原理

定义w_i为节点i的权值，l_i为深度，则有 Huffman tree $s.t.\min \mathrm{WPL} = \sum_{i=1}^{n} w_i l_i$

构造时循环地将权值最小的两棵树连接到新的节点即可，容易证明本贪心过程可以构造霍夫曼树。

将霍夫曼树视作只有 0，1 的字典树，并维护叶节点与原字符的对应关系即可进行编解码。

4. 程序代码

4.1. `Huffman_tree.hpp`

```
#ifndef _HUFFMAN_TREE_HPP
#define _HUFFMAN_TREE_HPP

#include <functional>
#include <string>
#include <map>
#include <vector>
#include <queue>

namespace myDS {
    class huffmanTree {
    protected:
        std::vector<int> pa;
        std::vector<int> wei;
        std::vector<std::pair<int,int>> s2code;

        bool comp(std::pair<char,int> a,std::pair<char,int> b)
        {
            return a > b;
        }

        std::size_t cap = 0;
        std::size_t MEX = 0;

        void link(int a,int b) {
            pa[a] = cap,pa[b] = cap;
            s2code[cap] = {a,b};
        }

    public:
        huffmanTree() { }

        huffmanTree(std::vector<int> _wei) {buildup(_wei);}

        void buildup(std::vector<int> _wei) {
            wei = _wei;
            MEX = wei.size()-1;
            pa.resize(MEX * 2 + 2);
            s2code.resize(MEX * 2 + 2);

std::priority_queue<std::pair<int,int>,std::vector<std::pair<int,int>>,std::greater<st
values;
            cap = MEX;
            for(int i = 1;i <= MEX;i ++) {
                values.push({wei[i],i});
            }
            while(values.size()) {
```

```
                cap ++;
                auto a = values.top();
                values.pop();
                if (values.size() == 0)
                    break;
                auto b = values.top();
                values.pop();
                link(a.second,b.second);
                values.push({a.first + b.first,cap});
            }
        }

        int getWPL() {
            int t = 0;
            for(int i = 1;i <= MEX;i ++) t += wei[i] *
(getPath(i).size());
        }

        std::vector<char> getPath(std::size_t n) {
            std::vector<char> rt;
            int t = n;
            while(pa[t]) {
                rt.push_back(s2code[pa[t]].first == t);
                t = pa[t];
            }
            std::vector<char> path;
            for(int i = 0;i < rt.size();i ++)
path.push_back(rt[rt.size()-1-i]);
            return path;
        }

        std::vector<int> getC(std::vector<char> t) {
            int ori = cap-1;
            std::vector<int> rt;
            for(auto x:t) {
                if(s2code[ori] == std::pair<int,int>()){
                    rt.push_back(ori);
                    ori = cap-1;
                }
                if(x == 0) ori = s2code[ori].second;
                else ori = s2code[ori].first;
            }
            if(s2code[ori] == std::pair<int,int>()){
                rt.push_back(ori);
                ori = cap;
            }
            return rt;
        }

    };

```

```
95      class huffmanEncoder : huffmanTree {
96      private:
97
98          std::map<char,int> wordCounter;
99          std::map<char,int> c2i;
100         std::map<int,char> i2c;
101         std::string init;
102
103     public:
104         huffmanEncoder(std::string _init) : huffmanTree(){
105             init = _init;
106             for(auto x:_init) wordCounter[x] ++;
107             pa.resize(wordCounter.size()*2+1);
108             std::vector<std::pair<char,int>> gt(1);
109             for(auto x:wordCounter) gt.push_back(x);
110             for(int i = 1;i < gt.size();i ++) c2i[gt[i].first] = i;
111             for(int i = 1;i < gt.size();i ++) i2c[i] = gt[i].first;
112             std::vector<int> wei;
113             for(auto x:gt) wei.push_back(x.second);
114             buildup(wei);
115         };
116
117         std::vector<char> encode(std::string s) {
118             std::vector<char> rt;
119             auto add = [&](std::vector<char> addit) -> void{
120                 for(int i = 0;i < addit.size();i ++)
    rt.push_back(addit[i]);
121             };
122             for(auto x:s) add(this->getPath(c2i[x]));
123             return rt;
124         }
125
126         std::string decode(std::vector<char> r) {
127             std::string rt;
128             for(auto x:getC(r)) rt.push_back(i2c[x]);
129             return rt;
130         }
131
132     };
133
134 };
135
136 #endif
```

4.2. `_PRIV_TEST.cpp`

```
1 #include <iostream>
2 #define __PRIVATE_DEBUGE
3 #include <Dev\16\Huffman_tree.hpp>
4 using namespace std;
```

```cpp

int main()
{
    string s;
    cin >> s;
     myDS::huffmanEncoder hfe(s);
    for(auto x:hfe.encode(s)) cout << (bool)x << " ";
    cout << "\n";
    for(auto x:hfe.decode(hfe.encode(s))) cout << x << " ";
    cout << "\n";
    system("pause");
}
```

5. 测试数据与运行结果

运行上述`_PRIV_TEST.cpp`测试代码中的正确性测试模块，得到以下内容：

```
aaaaaabbbcccd
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1
a a a a a a b b b c c c d
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。