****

项目说明文档

**离散数学课程实验报告**

**——最优2元树在通信编码中的应用**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：唐 剑 锋**

二○二○年十二月

目录

[一、实验目的 1](#_Toc60013695)

[二、实验内容 1](#_Toc60013696)

[三、实验原理 1](#_Toc60013697)

[1、霍夫曼树的创建 1](#_Toc60013698)

[四、实验过程 2](#_Toc60013699)

[1、实验代码 2](#_Toc60013700)

[2、实验截屏 7](#_Toc60013701)

[五、实验小结 7](#_Toc60013702)

[1、数据结构 7](#_Toc60013703)

[2、解题思路 7](#_Toc60013704)

[3、心得体会 8](#_Toc60013705)

## 一、实验目的

掌握求霍夫曼树的方法

## 二、实验内容

输入一组通信符号的使用频率，求各通信符号对应的前缀码。

## 三、实验原理

对于固定的叶子节点数和权值，带权路径最短的二叉树称为霍夫曼树或者最优二叉树，霍夫曼树可以用来设计二进制的前缀编码。假设有一棵二叉树，其叶子节点分别为A. B、C、D构建一棵有四个叶子节点的二叉树二叉树，使这四个字母分别处于四个叶子节点的位置。约定连接左子树与根节点的树枝为1，连接根节点与右子树的树枝为0，则在对树进行先序遍历时，记录下经过的边即可得到一个二进制序列，这个二进制序列可以作为相应叶子节点字母的霍夫曼编码，这就是霍夫曼编码的基本思路。

### 1、霍夫曼树的创建

利用二叉树进行编码译码，并且使编码具有最佳的使用效果，就要满足以下两个基本条件:

1. 任何两个不同的字符或者字符串不能有相同的编码，这就要求每个字符都有其独特的唯一的编码，即任意一个字符的编码都不能是另一个字符编码的前缀；
2. 字符串的编码要尽可能的短，以便于存储和解码。

基于以上编码要求，选择霍夫曼树作为字符编码译码的工具是合理的。霍夫曼树的构建方法如下:

1. 根据给定的n个权值构成n棵二叉树的集合F,其中每棵二叉树中只有一个带权的根节点，其左右子树均为空;
2. 在F中选取两棵根节点最小的树作为左右子树构造-颗新的二叉树，且置新的二叉树的根节点的权值为其左右子树上根节点的权值之和;
3. 在F中删除这两棵树，同时将构建的新的二叉树加入F中:
4. 重复(2)和(3) 直到F中只含一棵树为止，这棵树就是霍夫曼树。

## 四、实验过程

### 1、实验代码

根据题意，写出实验代码如下所示：

#include <algorithm>

#include <iostream>

#include <queue>

#include <sstream>

#include <string>

#include <vector>

using namespace std;

template <class T>

class HuffmanTree;

template <class T>

struct tmp2;

template <class T>

void dfs(HuffmanTree<T>\* t, vector<int>& code,

         vector<pair<T, vector<int>>>& ans);

template <class T>

class HuffmanTree {

   private:

    T \_val;

    HuffmanTree\* \_left;

    HuffmanTree\* \_right;

   public:

    friend struct tmp2<T>;

    friend void dfs<>(HuffmanTree<T>\*, vector<int>& code,

                      vector<pair<int, vector<int>>>& ans);

    HuffmanTree() : \_val(), \_left(nullptr), \_right(nullptr) {}

    HuffmanTree(T x) : \_val(x), \_left(nullptr), \_right(nullptr) {}

    HuffmanTree(HuffmanTree\* t1, HuffmanTree\* t2);

    bool operator<(const HuffmanTree& v2) const {

        return (this->\_val < v2.\_val);

    };

    bool operator>(const HuffmanTree& v2) const {

        return (this->\_val > v2.\_val);

    };

    bool operator<=(const HuffmanTree& v2) const {

        return (this->\_val <= v2.\_val);

    };

    bool operator>=(const HuffmanTree& v2) const {

        return (this->\_val >= v2.\_val);

    };

    bool operator==(const HuffmanTree& v2) const {

        return (this->\_val == v2.\_val);

    };

    ~HuffmanTree();

};

template <class T>

HuffmanTree<T>::~HuffmanTree() {

    if (this->left != nullptr) {

        delete this->left;

        this->left = nullptr;

    }

    if (this->right != nullptr) {

        delete this->right;

        this->right = nullptr;

    }

}

template <class T>

HuffmanTree<T>::HuffmanTree(HuffmanTree<T>\* t1, HuffmanTree<T>\* t2) {

    if (t1->\_val < t2->\_val) {

        this->\_left = t1;

        this->\_right = t2;

    } else {

        this->\_right = t1;

        this->\_left = t2;

    }

    this->\_val = t1->\_val + t2->\_val;

}

template <class T>

struct tmp2  //重写仿函数

{

    bool operator()(HuffmanTree<T>\* a, HuffmanTree<T>\* b) {

        return a->\_val > b->\_val;  //大顶堆

    }

};

template <class T>

void dfs(HuffmanTree<T>\* t, vector<int>& code,

         vector<pair<T, vector<int>>>& ans) {

    if (t->\_left == nullptr && t->\_right == nullptr) {

        pair<T, vector<int>> k = {t->\_val, code};

        ans.push\_back(k);

        return;

    } else {

        if (t->\_left != nullptr) {

            code.push\_back(0);

            dfs(t->\_left, code, ans);

            code.pop\_back();

        }

        if (t->\_right != nullptr) {

            code.push\_back(1);

            dfs(t->\_right, code, ans);

            code.pop\_back();

        }

        return;

    }

}

double str2num(string s) {

    stringstream ss;

    double d = 0.0;

    ss << s;

    ss >> d;

    return d;

}

bool isNum(string str) {

    for (auto&& i : str)

        if (!isdigit(i)) return false;

    return true;

}

void opening() {

    cout << "Application of Hoffman Tree in Communication Coding." << endl

         << "The input is a line of positive integers. " << endl

         << "The first number is N followed by N communication symbols. "

         << endl

         << "Now enter a line of positive integers. " << endl;

    return;

}

bool loop() {

    opening();

    priority\_queue<HuffmanTree<int>\*, vector<HuffmanTree<int>\*>, tmp2<int>>

        heap;

    string temp;

    int num = 0;

    while (true) {

        cin >> temp;

        if (!isNum(temp))

            cout << "Error: Invalid input detected. Please try again." << endl;

        else {

            num = str2num(temp);

            break;

        }

    }

    for (int i = 0; i < num; i++) {

        cin >> temp;

        if (!isNum(temp)) {

            i--;

            cout << "Error: Invalid input detected. Please try again." << endl;

        } else {

            HuffmanTree<int>\* t = new HuffmanTree<int>((int)str2num(temp));

            heap.push(t);

        }

    }

    while (heap.size() > 1) {

        auto temp = heap.top();

        heap.pop();

        auto temp2 = new HuffmanTree<int>(temp, heap.top());

        heap.pop();

        heap.push(temp2);

    }

    vector<int> code;

    vector<pair<int, vector<int>>> ans;

    dfs(heap.top(), code, ans);

    std::sort(ans.begin(), ans.end(),

              [](pair<int, vector<int>>& s1, pair<int, vector<int>>& s2) {

                  return s1.first < s2.first;

              });

    for (auto&& i : ans) {

        cout << "Weight = " << i.first << "  Code = ";

        for (auto&& j : i.second) cout << j;

        cout << endl;

    }

    cout << "Again? (y for yes, n for no)";

    char c;

    cin >> c;

    if (c == 'y')

        return true;

    else

        return false;

}

int main() {

    while (loop())

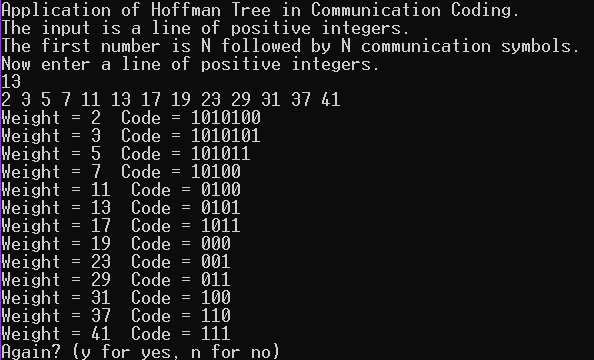
        ;

    std::cout << "Thanks for using it. See you next time! " << std::endl;

    return 0;

}

### 2、实验截屏



## 五、实验小结

### 1、数据结构

根据题意，需要建立并维护一个二叉树类。这里直接使用建立霍夫曼树类，其左右子女均为指向霍夫曼树的指针。且根据题意，每次将权重最小的两颗霍夫曼树合并生成一个大霍夫曼树，则考虑使用优先级队列，优先级为每棵树的权重。

### 2、解题思路

首先录入所有节点，并且为每一个节点建立一颗空霍夫曼树，其权值即为节点的大小。将所有节点加入最小堆。从堆中取出堆顶的两个元素（即权值最小的两棵树），较小的元素作为左子树，较大的元素作为右子树，合并成一颗大霍夫曼树并放回堆中。重复以上操作直到堆中只剩下一个元素，该元素即为最终的霍夫曼树。

### 3、心得体会

懂得了如何建立霍夫曼树，并如何从已知霍夫曼树中返回各个权值对应的编码。