实验四 二叉树及其应用

4.1二叉树遍历应用

打印图形的思路就是二叉树的逆中序遍历，根据递归的层数打印对应的空格数。编号设置则是后序遍历。

运行截图：

```
AB D  CE F
   C5
         F3
      E4
A6
      D1
   B2

Process finished with exit code 0
```

代码

```
#include<iostream>

using namespace std;

struct node {
    node *lChild = nullptr, *rChild = nullptr;
    char data = 0;
    int num = 0;

    friend ostream &operator<<(ostream &o, node
*n) {
        cout << n->data << n->num;
```

```
        return o;
    }
};

void build(node *now, char ch) {
    now->data = ch;
    char chr = static_cast<char>(cin.get());
    if (chr != ' ') {
        now->lChild = new node;
        build(now->lChild, chr);
    }
    chr = static_cast<char>(cin.get());
    if (chr != ' ') {
        now->rChild = new node;
        build(now->rChild, chr);
    }
}

void print(node *now, int lay = 0) {
//    cout.put(now->data);
//    cout.put((now->num) + '0');
    if (now == nullptr) return;
    print(now->rChild, lay + 1);
    for (int i = 0; i < lay; i++)cout << "   ";
    cout << now << ' ' << endl;
    print(now->lChild, lay + 1);

}

void count(node *now) {
    static int cnt = 0;
    if (now == nullptr) return;
    count(now->lChild);
```

```
    count(now->rChild);
    now->num = ++cnt;
}

int main() {
    node *head = new node;
    build(head, static_cast<char>(cin.get()));
    count(head);
    print(head);
}
// AB D  CE F
```

4.2 二叉树遍历与栈、队列

我选择第二小题：输入某元素结点，输出从二叉树根结点到该结点之间的路径序列。

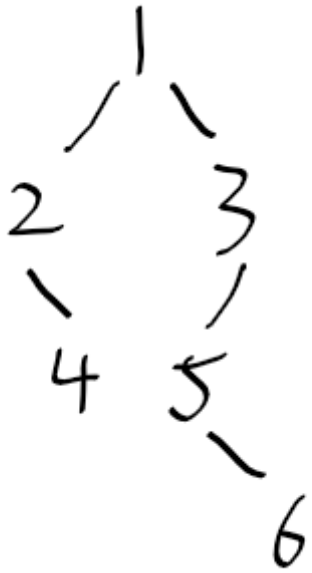
思路是使用先序遍历，找到需要找到的节点，然后再递归退层的时候将路上经过的点压入一个栈中，将栈的内容输出就得到了路径。

运行截图

```
1 2 0 4 0 0 3 5 0 6 0 0 0
5
1 3 5
```

```
1 2 0 4 0 0 3 5 0 6 0 0 0
6
1 3 5 6
Process finished with exit code 0
```

输入的数据：



代码：

```
#include<iostream>
#include <functional>

using namespace std;

template<typename T>
class myStack {
public:
    int topptr = 0;
    T *arr = nullptr;

    explicit myStack(int n) {
```

```
        arr = new T[n];
    }

    myStack() = default;

    [[nodiscard]] T top() const {
        return *(arr + topptr);
    };

    [[nodiscard]] bool empty() const {
        return topptr == 0;
    }

    void push(T x) {
        topptr++;
        *(arr + topptr) = x;
    }

    T pop() {
//        if (topptr == 0) return 0;
//        else
        return *(arr + topptr--);
    }

    friend ostream &operator<<(ostream &o,
myStack &s) {
//        for (int i = 1; i <= s.topptr; i++) {
//            o << s.arr[i] << ' ';
//        }
        while (!s.empty())
            o << s.pop() << ' ';
        return o;
    }
```

```
};

struct node {
    node *lChild = nullptr, *rChild = nullptr;
    int data = 0;

    friend ostream &operator<<(ostream &o, node
*n) {
        cout << n->data;
        return o;
    }
};


void build(node *now, int ch) {
    now->data = ch;
    int chr;
    cin >> chr;
    if (chr != 0) {
        now->lChild = new node;
        build(now->lChild, chr);
    }
    cin >> chr;
    if (chr != 0) {
        now->rChild = new node;
        build(now->rChild, chr);
    }
}

void print(node *now, int lay = 0) {
//    cout.put(now->data);
//    cout.put((now->num) + '0');
    if (now == nullptr) return;
```

```
    print(now->rChild, lay + 1);
    for (int i = 0; i < lay; i++)cout << "    ";
    cout << now << ' ' << endl;
    print(now->lChild, lay + 1);
}


int main() {
    //1 2 0 4 0 0 3 5 0 6 0 0 0
    node *head = new node;
    int x;
    cin >> x;
    build(head, x);
    cin >> x;
    auto stack = myStack<int>(100);
    function<bool(node *, int)> dfs = [&](node
*now, int x) {
        if (now == nullptr) return false;
        if (now->data == x) {
            stack.push(now->data);
            return true;
        }
        if (dfs(now->lChild, x)) {
            stack.push(now->data);
            return true;
        }
        if (dfs(now->rChild, x)) {
            stack.push(now->data);
            return true;
        }
        return false;
    };
    dfs(head, x);
```

```
//    stack.push(head->data);
    cout << stack;
}
```

4.3 二叉树与树的关系

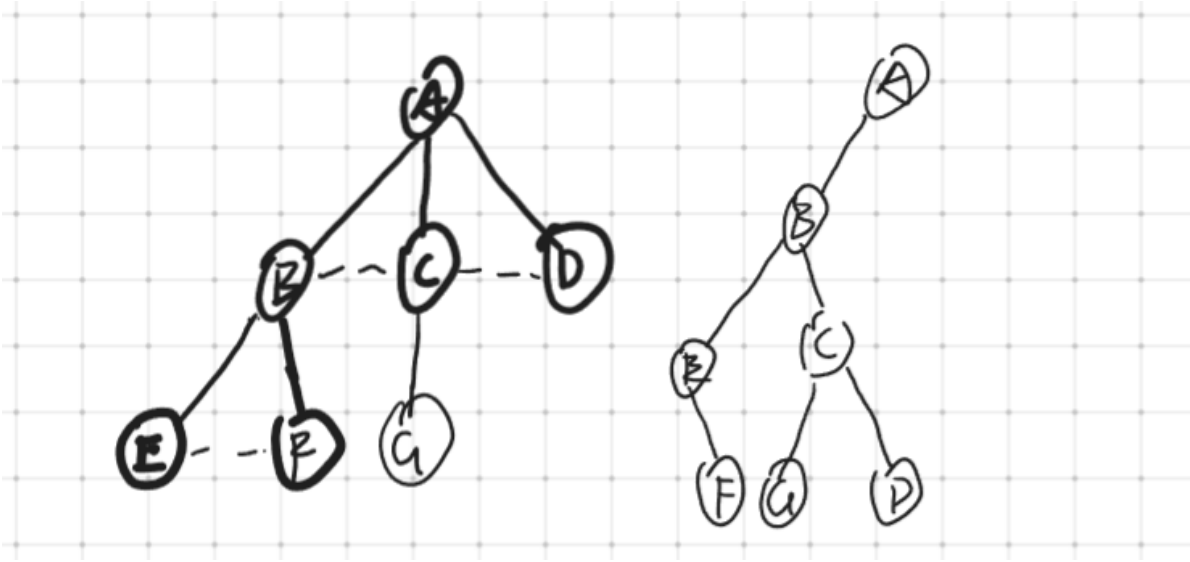
这题比较麻烦的点在于将树变成二叉树，我使用了邻接表法存储了树。然后通过广度优先搜索将树转换成了二叉树，然后再对二叉树进行前序遍历就能得到结果了。

运行截图：

```
7
A B
A C
A D
B E
B F
C G
A
    B
        E
        F
    C
        G
    D

Process finished with exit code 0
```

其中输入数据的原本形式（左）和二叉树形式（右）：



代码：

```
#include <iostream>
#include <functional>
#include <vector>
#include <queue>

using namespace std;

ostream &space(ostream &o, int x) {
    for (int j = 0; j < x; j++) {
        o << "  ";
    }
    return o;
}


int main() {
    int n;
    cin >> n;
    vector<vector<int>> tree(n + 1);
    char tmp[3];
    for (int i = 1; i < n; i++) {
```

```
        cin >> tmp;
        int x = tmp[0] - 'A' + 1;
        cin >> tmp;
        int y = tmp[0] - 'A' + 1;
        tree[x].emplace_back(y);
        tree[y].emplace_back(x);
    }
    vector<int> bin(30, -1);

    auto bfs = [&](int fst) {
        vector<int> vis(n + 1);
        std::queue<int> q;
        q.push(fst);
        vis[fst] = fst;
        bin[fst] = fst;
        while (!q.empty()) {
            int now = q.front();
            int pos = vis[now];
            q.pop();
            bool first = true;
            for (const int &i: tree[now]) {
                if (vis[i]) continue;
                if (first) pos = pos << 1,
first = false;
                else pos = (pos << 1) + 1;
                bin[pos] = i;
                vis[i] = pos;
                q.push(i);
            }
        }
    };
    bfs(1);
//
```

```
//    for (int i = 1; i <= 11; i++) cout << i
<< ' ';
//    cout << endl;
//    for (int i = 1; i <= 11; i++) cout <<
(char) (bin[i] + 'A' - 1) << ' ';
    function<void(int, int)> dfs = [&](int pos,
int sp) {
        int l = pos << 1;
        int r = l + 1;
        space(cout, sp) << (char) (bin[pos] +
'A' - 1) << endl;
        if (bin[l] != -1) dfs(l, sp + 2);
        if (bin[r] != -1) dfs(r, sp);
    };
    dfs(1, 0);
    return 0;
}
/*
7
A B
A C
A D
B E
B F
C G
*/
```

4.4 哈夫曼编码

老师的方法是使用关系表的方法存储，我决定试试用二叉链表存储。

思路：

1. **编码：**直接统计每个字符的出现次数，将次数当作权重进行排序，然后每次从集合中选择权重最小的两个节点组成一个新的节点，再把新的节点也放进集合中，为了降低时间复杂度，我使用了堆优化排序的过程。将二叉树使用扩展先序遍历写入文件中，方便后面解码的时候构建哈夫曼树。然后遍历二叉树，得到每个字符对应的编码，然后将每个字符转换为编码写入文件。
2. **解码：**先读入扩展先序比遍历的结果，构造二叉树链式结构。然后挨个读入数字，0是左子树，1是右子树，如果到了一个叶子节点就输出当前节点的字符。

运行截图

输入文件：`data.txt`

```
Hello world,
it feels good here.
```

运行编码步骤的输出：

```
Enter 'e' for encode, 'd' for decode:e
char    weight
        4
,       1
.       1
@       1
H       1
d       2
e       5
f       1
g       1
h       1
```

```
i       1
l       4
o       4
r       2
s       1
t       1
w       1
char    weight    code
        4         110
,       1         0000
.       1         10111
@       1         10110
H       1         10001
d       2         0011
e       5         111
f       1         10000
g       1         10011
h       1         10010
i       1         00011
l       4         011
o       4         010
r       2         0010
s       1         00010
t       1         10101
w       1         10100

Process finished with exit code 0
```

得到code.txt:

```
$$$$,##$s##i##$r##d##$o##l##$$$$f##H##$h##g##$$
w##t##$@##.##$##e##$$$$,##$s##i##$r##d##$o##l##
$$$$f##H##$h##g##$$w##t##$@##.##$ ##e##
10001111011011010110101000100010011001100001011
00001110101110100001111110110001011010011010010
00111101001011100101111 0111
```

运行解码步骤：

```
Enter 'e' for encode, 'd' for decode:d
Hello world,
it feels good here.
Process finished with exit code 0
```

代码：

```cpp
#include <bits/stdc++.h>

#define SP '$'
#define NU '#'
#define NL '@'
using namespace std;
using pci = pair<char, int>;

template<typename T>
ostream &operator<<(ostream &o, vector<T> v) {
    for (int &i: v) o << i;
    return o;
}

struct node {
    char data = 0;
```

```
    int weight = 0;
    node *lChild = nullptr, *rChild = nullptr;

    node() = default;

    explicit node(char a, int b) {
        this->data = a;
        this->weight = b;
    }

    explicit node(const pci &p) {
        this->data = p.first;
        this->weight = p.second;
    }

    explicit node(node *a, node *b) {
        this->data = 0;
        this->weight = a->weight + b->weight;
        this->lChild = a;
        this->rChild = b;
    }


};

struct cmp {
    bool operator()(node *a, node *b) {
        if (a->weight != b->weight)
            return a->weight > b->weight;
        else if (b->data == 0) return true;
        else if (a->data == 0) return false;
        else return a->data < b->data;
    }
```

```
};

void encode() {
    fstream fin = fstream("data.txt", ios::in);
    fstream fout = fstream("code.txt", ios::out
| ios::trunc);
    if (!fin.is_open()) {
        cout << "Cannot find data.txt";
        return;
    }
    if (!fout.is_open()) {
        cout << "Cannot open code.txt";
        return;
    }
    array<int, 128> cnt{0};
    char ch;
    while (fin.peek() != EOF) {
        ch = static_cast<char>(fin.get());
        if (ch == '\n') ch = NL;
        cnt[ch]++;
    }
    priority_queue<node *, vector<node *>, cmp>
pq;
    cout << "char    weight\n";
    for (char i = 0; i <= 120; i++) {
        if (cnt[i]) {
            cout << i << '\t' << cnt[i] <<
endl;
            pq.push(new node(i, cnt[i]));
        }
    }

    function<node *(void)> build = [&]() {
```

```
//        cout << "Begin build\n";
        while (pq.size() > 1) {
            node *tmp = pq.top();
            pq.pop();
            node *nd = new node(tmp, pq.top());
//            cout << (tmp->data == 0 ? 'x' :
tmp->data) << ' ' << (pq.top()->data == 0 ? 'x'
: pq.top()->data) << ' '
//                 << nd->weight << endl;
            pq.pop();
            pq.push(nd);
        }
        return pq.top();
    };
    node *tree = build();

    function<void(node *, int)> print = [&]
(node *now, int lay) {
        if (now == nullptr) {
            fout << NU;
            return;
        }
//        for (int i = 0; i < lay; i++)cout <<
"   ";
//        cout << (now->data == 0 ? 'x' : now-
>data) << ' ' << endl;
//        cout << (now->data == 0 ? 'x' : now-
>data);
        fout << (now->data == 0 ? SP : now-
>data);
        print(now->lChild, lay + 1);
        print(now->rChild, lay + 1);
    };
```

```
    print(tree, 0);

    map<char, vector<int>> table;
    function<void(node *, vector<int>)> go =
[&](node *now, vector<int> here) {
        if (now == nullptr) return;
        if (now->data) {
            table[now->data] = here;
        }
        here.emplace_back(0);
        go(now->lChild, here);
        *(here.end() - 1) = 1;
        go(now->rChild, here);
    };
    go(tree, vector<int>());

    cout << "char    weight    code\n";
    for (const auto &[a, x]: table) {
        cout << a << '\t' << cnt[a] << "\t  "
<< x << endl;
    }

    print(tree, 0);
    fout << endl;
    fin.seekg(0, ios::beg);
    while (fin.peek() != EOF) {
        ch = static_cast<char>(fin.get());
        ch = ch == '\n' ? NL : ch;
        fout << table[ch];
//        cout << table[ch];
    }
    fout.close();
    fin.close();
```

```
}

void decode() {
    fstream fin = fstream("code.txt", ios::in);
    if (!fin.is_open()) {
        cout << "Cannot open code.txt";
        return;
    }
    string tableStr;
    getline(fin, tableStr);
    function<void(node *)> build = [&](node
*now) {
        static int pos = 0;
        now->data = tableStr[pos];
        if (tableStr[++pos] != NU) {
            now->lChild = new node;
            build(now->lChild);
        }
        if (tableStr[++pos] != NU) {
            now->rChild = new node;
            build(now->rChild);
        }
    };
    function<void(node *, int)> print = [&]
(node *now, int lay) -> void {
        if (now == nullptr) return;
        for (int i = 0; i < lay; i++)cout << "
  ";
        cout << now->data << ' ' << endl;
        print(now->lChild, lay + 1);
        print(now->rChild, lay + 1);
    };
```

```
    node *tree = new node;
    build(tree);
//    print(tree, 0);
    char ch = static_cast<char>(fin.get());
    function<char(node *)> fd = [&](node *now)
-> char {
        if (ch - '0') {//1 -> right
            if (now->rChild == nullptr) return
now->data;
            else {
                ch = static_cast<char>
(fin.get());
                fd(now->rChild);
            }
        } else {//left
            if (now->lChild == nullptr) return
now->data;
            else {
                ch = static_cast<char>
(fin.get());
                fd(now->lChild);
            }
        }
//        return '$';
    };

    while (fin.peek() != EOF) {
        char chr = fd(tree);
        if (chr == NL) cout << '\n';
        else cout << chr;
    }
}
```

```
int main() {
    cout << "Enter 'e' for encode, 'd' for
decode:";
    string str;
    cin >> str;
    switch (str[0]) {
        case 'e':
            encode();
            break;
        case 'd':
            decode();
            break;
    }
}
```