数据结构 ： 用先序遍历建立二叉树 实验报告

毛子恒 李臻 张梓靖

2020 年 11 月 20 日

小组成员

班级：2019211309	姓名：毛子恒	学号：2019211397	分工：代码 文档
班级：2019211310	姓名：李臻	学号：2019211458	分工：测试 文档
班级：2019211308	姓名：张梓靖	学号：2019211379	分工：可视化 文档

目录

1 需求分析

1.1 题目描述

按照先序遍历输入一个二叉树，建立二叉树，输出该二叉树的各种表示形式。

1.2 输入描述

程序从标准输入中读入数据。
输入一行字符串 str（$n > 0, m > 0$），表示二叉树按先序遍历所得的序列输出。
其中 * 代表前一个非 * 字符对应的结点左子树或右子树为空。

1.3 输出描述

程序向标准输出中输出结果。
输出 n 行，每行 3 个字符，用空格分隔，表示二叉树的结点，结点的左子树和右子树。
其中 * 代表对应的结点左子树或右子树为空，** 代表对应的结点为叶子结点。

1.4 样例输入输出

1.4.1 样例输入输出 1

【输入】

```
2 2
3 0
0 7
2 2
4 6
0 8
```

【输出】

```
7 6
0 15
12 18
0 56
```

1.4.2 样例输入输出 2

【输入】

```
8 8
282 0 708 0 449 0 0 39
685 79 0 0 385 0 638 0
745 0 244 0 658 795 0 0
923 985 0 0 0 0 152 0
602 167 0 0 0 143 0 0
568 233 35 0 0 0 0 0
0 0 284 0 310 23 0 625
```

```
542 0 303 293 286 0 387 510
8 8
578 781 779 0 0 0 0 0
0 531 0 940 106 464 0 735
346 141 287 273 0 746 804 665
917 0 0 0 0 0 409 788
0 0 176 784 0 383 13 879
927 881 899 146 118 0 0 709
42 84 0 254 0 0 90 0
552 0 0 955 0 0 0 0
```

【输出】

```
860 781 1487 0 449 0 0 39
685 610 0 940 491 464 638 735
1091 141 531 273 658 1541 804 665
1840 985 0 0 0 0 561 788
602 167 176 784 0 526 13 879
1495 1114 934 146 118 0 0 709
42 84 284 254 310 23 90 625
1094 0 303 1248 286 0 387 510
429492 320070 501898 582545 0 700135 575069 865491
422726 630526 601375 538152 8374 184111 62425 396480
1251999 1316644 1480896 698554 93810 434038 204730 1304297
539878 1256666 719017 964508 104410 457040 13680 723975
480517 684822 597515 177858 34576 77488 0 224132
340414 572266 452517 228575 24698 134222 28140 194530
464585 60307 156745 920805 2714 330594 232366 477657
984569 498533 559515 892291 0 335576 401997 683773
```

1.4.3 样例输入输出 3

【输入】

```
2 3
1 6 0
3 0 7
3 1
5
0
8
```

【输出】

```
Cannot add matrix A and B, An != Bn, Am != Bm.
5
71
```

1.4.4 样例输入输出 4

【输入】

```
4 2
2 5
0 3
2 0
7 0
1 3
1 0 6
```

【输出】

```
Cannot add matrix A and B, An != Bn, Am != Bm.
Cannot multiply matrix A and B, Am != Bn.
```

1.5 程序功能

程序通过先序遍历序列生成二叉树，再将二叉树的每个结点及其左右子树依照先序遍历的顺序打印出来。

2 概要设计

2.1 问题解决的思路

使用三元组表存储稀疏矩阵，并且设计三元组表形式的矩阵加法和乘法算法、一维数组存储的矩阵和三元组表存储的矩阵之间的转化算法。

2.2 矩阵的定义

```
1  //数据对象
2  typedef struct node
3  {
4     char data;
5     struct node *lc, *rc;
6  } Node;
7
8  /*
9   * 操作：生成二叉树
10  * 前件：point指向一棵空树
11  * 后件：point指向一棵树
12  */
13  Node *buildTree();
14
15 /*
16  * 操作：把二叉树安装先序遍历的顺序输出
17  * 前件：point指向一棵树
18  * 后件：先序输出point指向的树的每一个结点和其左右子树
19  */
20 void preorderTraverse(Node *point);
21
```

```
/*
 * 操作：释放二叉树空间
 * 前件：point指向一棵树
 * 后件：point指向一棵空树
 */
void destroyTree(Node *point);
```

2.3 主程序的流程

1. 输入
2. 生成二叉树
3. 依照先序遍历的顺序，输出结点的左右子树
4. 继续遍历直到结点的子树为 *
5. 释放空间

3 详细设计

3.1 二叉树的实现

二叉树的设计中基本操作的伪代码算法如下：

```
// 生成二叉树
Node *buildTree()
{
    定义字符 ch <- getchar()
    if (ch为'*')
        返回空指针
    定义结点 *point 并分配内存
    point->data <- ch
    point->lc <- buildTree()
    point->rc <- buildTree()
    返回 point
}

// 先序遍历二叉树并输出
void preorderTraverse(Node *point)
{
    printf("%c ", point->data)
    if (point->lc不为空)
        printf("%c ", point->lc->data)
    else printf("*")
    if (point->rc不为空)
        printf("%c ", point->rc->data)
    else printf("*")
    puts("") //换行
    if (point->lc不为空)
```

```
        preorderTraverse(point->lc) //继续遍历左子树
    if (point->rc不为空)
        preorderTraverse(point->rc) //继续遍历左子树
}

// 释放树空间
void destroyTree(Node *point)
{
    if (point->lc不为空)
        destroyTree(point->lc)
    if (point->rc不为空)
        destroyTree(point->rc)
    释放point
}
```

3.2 函数的调用关系图

函数调用关系图如图 1。

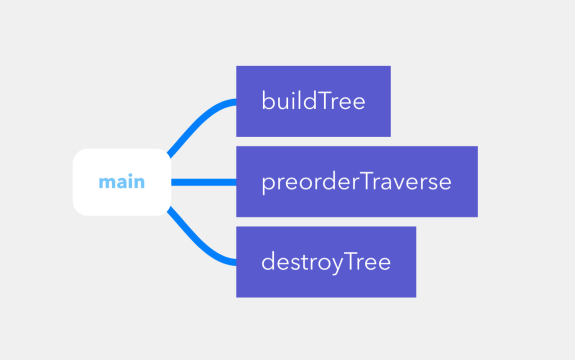

图 1: 函数调用关系图

4 调试分析报告

4.1 调试过程中遇到的问题和思考

程序在一些小细节中出现了问题，比如 val 数组需要多分配一个空间，矩阵相乘后 c 矩阵的列数等于 b 矩阵的列数。

初次实现矩阵相加算法时设计有误，使得两个矩阵的某个对应位置只要有一个为零，就会忽略这个位置，这个问题通过 if 判断修复。

在随机测试中发现数据规模较大时结果很容易超出 int 类型的范围，故矩阵元素采用 long long 类型。

4.2 设计实现的回顾讨论

由于二维数组的内存分配、函数传参较为复杂，所以输入、输出时使用一维数组存储的矩阵。

期望矩阵的规模不超过 10^3，所以 $MATRIXINCREASESIZE$ 常量的值设置为 10^5。

4.3 算法复杂度分析

initMatrix, expandMatrix, destroyMatrix 函数的复杂度为 $O(1)$

array2Matrix, matrix2Array, addMatrix 函数的复杂度为 $O(n^2)$，mulMatrix 函数的复杂度为 $O(n^3)$，视矩阵的稀疏程度，算法的时间复杂度会有常数级别的优化。在本报告的最后部分有讨论。

主函数的时间复杂度为 $O(n^2)$，整体时间复杂度为 $O(n^3)$。

整体空间复杂度为 $O(n^2)$。

4.4 改进设想的经验和体会

4.4.1 改进 1

可以在输入/输出时简化数组存储的矩阵和三元组表存储的矩阵之间转化的过程，会有常数级别的优化。

5 用户使用说明

使用 gcc 编译生成可执行文件。

```
gcc -o main -std=c11 main.c matrix.c
```

执行可执行文件：

```
./main
```

在 Windows cmd 下：

```
main
```

之后通过标准输入输入数据，输入格式参考 1.2 节的输入描述，结果通过标准输出返回。如果输入合法并且程序正常运行结束，主函数返回值为 0。

6 测试结果

测试环节分为三个步骤。

6.1 测试第一部分

对 1.4 节给出的样例进行测试。

6.2 测试第二部分

测试边界条件。

【输入】

```
1 0
1 1
2
```

【输出】

```
Please check your input.
```

【输入】

```
4 3
7 0 6
0 0 5
1 0 0
0 0 2
3 3
5 7 0
0 3 0
1 3 5
```

【输出】

```
Cannot add matrix A and B, An != Bn.
41 67 30
5 15 25
5 7 0
2 6 10
```

【输入】

```
3 3
7 0 6
1 0 4
1 0 0
3 4
5 7 0 1
0 3 0 0
1 3 5 0
```

【输出】

```
Cannot add matrix A and B, Am != Bm.
41 67 30 7
9 19 20 1
5 7 0 1
```

【输入】

```
3 4
7 0 6 4
1 0 4 0
1 0 0 6
3 4
5 7 0 1
0 3 0 0
1 3 5 0
```

【输出】

```
12 7 6 5
1 3 4 0
```

```
2 3 5 6
Cannot multiply matrix A and B, Am != Bn.
```

6.3 测试第三部分

将原解法与二维数组实现的矩阵加法和乘法 (testing/test.c) 比对。

测试在 macOS Catalina 10.15.6 下进行。

在 $n <= 10$，$n <= 100$，$n <= 1000$ 的范围下分别随机生成 1000 组测试数据，分别传入 main 和 test，并且比对两程序的输出。

3000 组数据中两程序的输出均相同。

数据生成程序 (testing/data.cpp) 如下：

```cpp
#include <bits/stdc++.h>
using namespace std;

const int SIZE = 1e3, PERCENT = 50;

int main()
{
    srand(time(0));
    int n = rand() % SIZE + 1;
    printf("%d %d\n", n, n);
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
            printf("%d ", rand() % 100 > PERCENT ? 0 : rand() % 1000 + 1);
        puts("");
    }
    printf("%d %d\n", n, n);
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= n; ++j)
            printf("%d ", rand() % 100 > PERCENT ? 0 : rand() % 1000 + 1);
        puts("");
    }
    return 0;
}
```

比对脚本 (testing/chk.sh) 如下：

```sh
for i in {1..1000}
do
    sleep 1
    ./data >in.in
    ./main <in.in >out.out
    ./test <in.in >out1.out
    if ! diff out.out out1.out
    then
        break
```

```
    fi
    echo "Correct"
done
```

7 可视化

随机生成若干组数据，由三元组表实现和二维数组实现分别计算，比对运行时间，并且使用 JavaScript 将结果可视化。

比对脚本 (testing/timecount.py) 如下：

```
import os
import json
import time
a = []
b = []
for i in range(100):
    os.system("./data >in.in")
    starttime = time.time()
    os.system("./main <in.in >out.out")
    endtime = time.time()
    a.append(endtime-starttime)
    starttime = time.time()
    os.system("./test <in.in >out1.out")
    endtime = time.time()
    b.append(endtime-starttime)
    print(i)

json_str = json.dumps([a, b])
with open("80%result.json", mode="w") as file:
    file.write(json_str)
```

数据规模如下：

$n = 1000$，20% 的元素是 0、50% 的元素是 0、80% 的元素是 0 的数据各 100 组，共 300 组数据。

7.0.1 实现细节

利用已有的时间数据，使用 Highcharts 库绘制二维柱状图，比较两种算法的运行时间差距。

7.0.2 用户使用说明

使用现代浏览器打开 Chart/index.html，即可看到柱状图，将鼠标指针放到某个柱上可以看到对应的值。

7.0.3 示例

见图 3。

图 2: 可视化示例

7.0.4 结论

可以看出在规模较大的数据中，相比二维数组的实现，三元组表的实现在绝大多数情况下都有明显的性能优势，并且当矩阵越稀疏时性能优势越明显。