电子科技大学信息与软件工程学院

实 验 报 告

学　　号 2018091602012

姓　　名 杨贺然

（实验）　课程名称 程序设计与算法基础 II

理论教师 郝宗波

实验教师 郝宗波

电子科技大学教务处制表

电 子 科 技 大 学

实 验 报 告

学生姓名：杨贺然 **学 号：**2018091602012 **指导教师：**郝宗波

实验地点：清水河基础实验大楼 A508 **实验时间：**2019.6.14

一、实验室名称：学校实验中心软件实验室

二、实验项目名称：非线性结构及相关算法的设计与应用

第 1 部分：二叉树的链式存储、序列化和反序列化

第 2 部分：公园景点间的最短路径查询程序的设计

三、实验学时：8 学时

四、实验原理：

二叉树是由结点指针将多个结点关联起来的抽象数据结构，是存在于内存中的，不能进行持久化。如果需要将一颗二叉树的结构持久化保存在磁盘文件中，需要将其转换为字符串并保存到文件中。所谓序列化是对二叉树进行先序遍历产生一个字符序列，与一般的先序遍历不一样，需要记录空结点，用 # 字符表示。

反序列化就是通过先序序列化的结果串 str 构建对应的二叉树，其过程是用 i 从头扫描 str；采用先序方法，当 i 越界时返回 NULL；否则当遇到#字符时返回 NULL，当遇到其它字符时，创建一个结点，可以采用递归的方法构造该二叉树；也可以采用非递归方法构造该二叉树。

公园内所有景点与景点之间的道路构成的集合可以抽象为一张图，可以采用

邻接矩阵存储图，空间复杂度 $O(n^2)$。对于查询带权图两点间的最短路径，可以使用 Floyd 算法，以 $O(n^3)$ 的时间复杂度计算出来。

五、实验目的：

通过本实验练习，掌握需要包括:磁盘文件的读写方法；掌握二叉树用递归与非递归方式序列化，反序列化方法，掌握利用递归与非递归方式求出二叉树的先序，中序，后序遍历序列，掌握释放树形结构空间的方法。掌握图的存储方式和求最短路径的算法。

六、实验内容：

（一）、第一部分:

1. 采用二叉链式存储创建二叉树 B1；
2. 采用先序序列化显示输出序列，并存储到文件中；
3. 从文件中读出序列，并反序列化的递归方法构造二叉树 B2；
4. 从文件中读出序列，并反序列化的非递归方法构造二叉树 B3；
5. 使用非递归方法输出二叉树中序遍历序列；
6. 使用非递归方法输出二叉树后序遍历序列；
7. 销毁释放二叉树 B1，B2，B3。

（二）、第二部分:

1. 设计数据结构与界面，输入直接相邻的两个旅游景点的名字以及它们之间的距离，并将每对直接相连的景点间的距离存到磁盘文件中；
2. 设计算法，实现计算给定的两个旅游景点间的最短路径；
3. 对公园的所有旅游景点，设计算法实现计算所有的景点对之间的最短路径，并将最短路径上的各旅游景点及每段路径长度写入磁盘文件 AllPath.dat 中；
4. 编写程序从文件 AllPath.dat 中读出所有旅游景点间的最短路径信息，存入

内存链表中管理；请运用所学的数据结构知识，设计内存链表的数据结构，实现用户输入任意两个旅游景点，能快速地从内存链表查询出两景点间的最短路径。

七、实验器材（设备、元器件）：

PC 机一台，装有 C 语言集成开发环境。

八、数据结构与程序：

（一）、第一部分

1. 定义二叉树结构

记录一个二叉树中节点，需要知道这个节点的编号和左右儿子的地址，结构如下：

```
typedef struct BinaryTree{
    char p;
    struct BinaryTree *lch, *rch;
}BTree;
```

2. 定义栈结构

为了非递归实现反序列化一棵二叉树与中序，后序遍历一棵二叉树，需要用到栈来模拟系统的递归操作，结构如下：

```
typedef struct Seqstack{
    BTree* a;
    struct Seqstack *las;
}stack;
```

3. 实现栈的基本操作

栈的基本操作有：入栈，询问栈是否为空，出栈，查询栈顶元素，实现如下：

```
BTree* top(){return sta -> a;}

int empty()
{
    if (sta == NULL) return 1;
    return 0;
}

void push(BTree* x)
{
    stack* ncon = (stack*)malloc(sizeof(stack));
    ncon -> a = x;
    ncon -> las = sta;
    sta = ncon;
    return ;
}

void pop()
{
    if (sta == NULL) return ;
    stack* tmp = sta;
    sta = sta -> las;
    free(tmp);
    return ;
}
```

4. **实现用递归方法建立一棵二叉树**

使用递归方法，可以输入二叉树一点的左右儿子，用先序方式建立一棵二叉树，实现如下：

```
void buildBinaryTree(BTree* now)
{
    printf("Pls input the left child of %c: ", now -> p);
    scanf("%s", str);
    if (str[0] == '#')
        now -> lch = NULL;
    else
    {
        now -> lch = (BTree*)malloc(sizeof(BTree));
        now -> lch -> p = str[0];
        buildBinaryTree(now -> lch);
    }
    printf("Pls input the right child of %c: ", now -> p);
    scanf("%s", str);
    if (str[0] == '#')
        now -> rch = NULL;
    else
    {
        now -> rch = (BTree*)malloc(sizeof(BTree));
        now -> rch -> p = str[0];
        buildBinaryTree(now -> rch);
    }
    return ;
}
```

在递归建树之前还需要输入根节点进行准备工作，实现如下：

```
void createBinaryTree()
{
    printf("Pls input the root: ");
    B1 = (BTree *)malloc(sizeof(BTree));
    scanf("%s", str);
    B1 -> p = str[0];
    buildBinaryTree(B1);
    puts("Create successfully.");
    return ;
}
```

5. 实现序列化一棵二叉树

对于序列化一棵二叉树，可以使用先序遍历，当遍历到一个空节点时输出 # 后退出，否则输出此节点的编号，先递归左子树，再递归右子树即可。实现如下：

```
void preOrder(BTree* now)
{
    if (now == NULL)
    {
        if (first)
        {
            printf("#");
            fprintf(fp, "#");
            first = 0;
        }
        else
        {
            printf(",#");
            fprintf(fp, ",#");
        }
        return ;
    }
    if (first)
    {
        printf("%c", now -> p);
        fprintf(fp, "%c", now -> p);
        first = 0;
    }
    else
    {
        printf(",%c", now -> p);
        fprintf(fp, ",%c", now -> p);
    }
    preOrder(now -> lch);
    preOrder(now -> rch);
    return ;
}
```

6. 实现递归反序列化一棵二叉树

根据序列化二叉树的方式，我们按照先序的方式补充二叉树的信息即可完成反序列化的操作。实现如下：

```
void buildTreeFromFile(BTree* now)
{
    if (pt >= n) return ;
    if (str[pt] == '#')
    {
        now -> lch = NULL;
        pt += 2;
    }
    else
    {
        now -> lch = (BTree*)malloc(sizeof(BTree));
        now -> lch -> p = str[pt];
        pt += 2;
        buildTreeFromFile(now -> lch);
    }
    if (str[pt] == '#')
    {
        now -> rch = NULL;
        pt += 2;
    }
    else
    {
        now -> rch = (BTree*)malloc(sizeof(BTree));
        now -> rch -> p = str[pt];
        pt += 2;
        buildTreeFromFile(now -> rch);
    }
    return ;
}
```

7. 实现非递归反序列化一棵二叉树

根据先序遍历的性质，仿照递归方式反序列化，可以实现非递归方式反序列化。先建立这个节点，然后依次建立它的左儿子与右儿子节点即可。利用标记表示此时建立完左子树还是右子树。实现如下：

```
void buildTree(BTree * root)
{
    BTree * now = root;
    int f = 1;
    push(root);
    while (pt < n)
    {
        while (str[pt] != '#' && pt < n)
        {
            if (!empty())
            {
                now = top();
                if (f)
                {
                    now -> lch = (BTree*)malloc(sizeof(BTree));
                    now -> lch -> p = str[pt];
                    now = now -> lch;
                    now -> lch = NULL;
                    now -> rch = NULL;
                }
                else
                {
                    now -> rch = (BTree*)malloc(sizeof(BTree));
                    now -> rch -> p = str[pt];
                    pop();
                    now = now -> rch;
                    now -> lch = NULL;
                    now -> rch = NULL;
                    f = 1;
                }
                push(now);
            }
            pt += 2;
        }
        pt += 2;
        if (!f) pop();
        else f = 0;
    }
    return ;
}
```

8. 实现对二叉树的中序，后序遍历

对于中序遍历，可以先递归到一个节点无左儿子，将其输出后改变指针为其右儿子，重复上述操作。实现如下：

```
void inOrderTravel(BTree * now)
{
    while (!empty() || now != NULL)
    {
        while (now)
        {
            push(now);
            now = now -> lch;
        }
        if (!empty())
        {
            now = top();
            pop();
            if (first)
            {
                printf("%c", now -> p);
                first = 0;
            }
            else printf(",%c", now -> p);
            now = now -> rch;
        }
    }
    putchar('\n');
    return ;
}
```

对于输出后续遍历,仅将输出此节点编号的时间调整到后面即可。实现如下:

```
BTree* p;
do {
    while (now)
    {
        push(now);
        now = now -> lch;
    }
    p = NULL;
    while (!empty())
    {
        now = top();
        if (now -> rch == p)
        {
            if (first)
            {
                printf("%c", now -> p);
                first = 0;
            }
            else printf(",%c", now -> p);
            p = now;
            pop();
        }
        else
        {
            now = now -> rch;
            break;
        }
    }
} while (!empty());
```

9. 销毁二叉树

按照后序遍历的方式销毁即可，实现如下：

```
void destructTree(BTree* now)
{
    if (now == NULL) return ;
    destructTree(now -> lch);
    destructTree(now -> rch);
    free(now);
    return ;
}
```

（二）、第二部分

1. **设计图与最短路径的存储方式**

用邻接矩阵存储图，可以用一个二维数组直接进行存储，$a[i][j]$ 存储的即为 i 到 j 的路径长度。

对于存储最短路径，观察到存储最短路径的矩阵应为一个对称矩阵，所以采用压缩的方式进行存储，对于一条最短路径，存储它的长度和经过的点。实现如下：

```
typedef struct Record {
    int len;
    int path[MAXN];
}Rec;
```

2. **设计将景点名称转化为编号的方式**

将景点名称转化为编号，利用顺序查找可以在 $O(nl)$ 的时间复杂度下完成，在景点较多的情况下处理时间较长。程序中使用 Trie 树可以在 $O(l)$ 的时间复杂度下完成。Trie 树结构，插入和查询方式如下：

```
typedef struct TrieTree {
    int id;
    struct TrieTree* nxt[26];
}Trie;
```

```
int add(char S[])
{
    int i, j;
    Trie* now = tree;
    for (i = 0; S[i]; i++)
    {
        if (now -> nxt[S[i] - 'a'] == NULL)
        {
            now -> nxt[S[i] - 'a'] = (Trie*)malloc(sizeof(Trie));
            if (now -> nxt[S[i] - 'a'] == NULL)
                return 0;
            for (j = 0; j < 26; j++) now -> nxt[S[i] - 'a'] -> nxt[j] = NULL;
            now -> nxt[S[i] - 'a'] -> id = 0;
        }
        now = now -> nxt[S[i] - 'a'];
    }
    now -> id = n;
    return 1;
}
```

```
int query(char S[])
{
    int i;
    Trie* now = tree;
    for (i = 0; S[i]; i++)
    {
        if (now -> nxt[S[i] - 'a'] == NULL) return 0;
        now = now -> nxt[S[i] - 'a'];
    }
    return now -> id;
}
```

3. 读入景点信息，生成邻接矩阵

用户通过标准输入，输入两景点间距离。经过 Trie 树上查找与添加后获得两景点的编号，然后直接加入邻接矩阵中。实现如下：

```
void getMap()
{
    fp = fopen("dis.dat", "w");
    memset(dis, 0x3f, sizeof dis);
    int i, u, v, w;
    puts("Pls input the two spots' name (no space, lower case) and the distance (integer).");
    puts("Press Ctrl-Z to end.");
    while (1)
    {
        if (scanf("%s %s %d", from, to, &w) == -1) break;
        fprintf(fp, "%s %s %d\n", from, to, w);
        u = query(from);
        if (!u)
        {
            u = ++n;
            strcpy(name[n], from);
            if (!add(from))
            {
                puts("No memory to allocate.");
                exit(0);
            }
        }
```

```
        v = query(to);
        if (!v)
        {
            v = ++n;
            strcpy(name[n], to);
            if (!add(to))
            {
                puts("No memory to allocate.");
                exit(0);
            }
        }
        dis[u][v] = dis[v][u] = min(dis[u][v], w);
    }
    for (i = 1; i <= n; i++) dis[i][i] = 0;
    puts("Map builds successfully");
    fclose(fp);
    return ;
}
```

4. 实现求任意两点间的最短路径算法

这里利用 Floyd 算法求出任意两点间最短路径。实现如下:

```
void Floyd()
{
    int i, j, k;
    for (i = 1; i <= n; i++)
        for (j = 1; j <=n; j++)
            if (i != j && dis[i][j] != 0x3f3f3f3f)
                path[i][j] = i;
    for (k = 1; k <= n; k++)
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                if (dis[i][j] > dis[i][k] + dis[k][j])
                {
                    dis[i][j] = dis[i][k] + dis[k][j];
                    path[i][j] = path[k][j];
                }
    puts("The shortest path calculate finished.");
    return ;
}
```

利用 Floyd 算法中的 dis 数组与 path 数组，可以求出任意两点之间的最短路径，并将其输出到文件中。实现如下：

```
void getPath()
{
    fp = fopen("AllPath.dat", "w");
    int i, j, k;
    Rec tmp;
    for (i = 1; i <= n; i++)
        for (j = i + 1; j <= n; j++)
        {
            tmp.path[0] = 0;
            fprintf(fp, "%d", dis[i][j]);
            k = j;
            while (k != i)
            {
                tmp.path[++tmp.path[0]] = k;
                k = path[i][k];
            }
            fprintf(fp, " %d %d", tmp.path[0] + 1, i);
            for (k = tmp.path[0]; k; k--)
                fprintf(fp, " %d", tmp.path[k]);
            fputc('\n', fp);
        }
    puts("Paths are saved to AllPath.dat.");
    fclose(fp);
    return ;
}
```

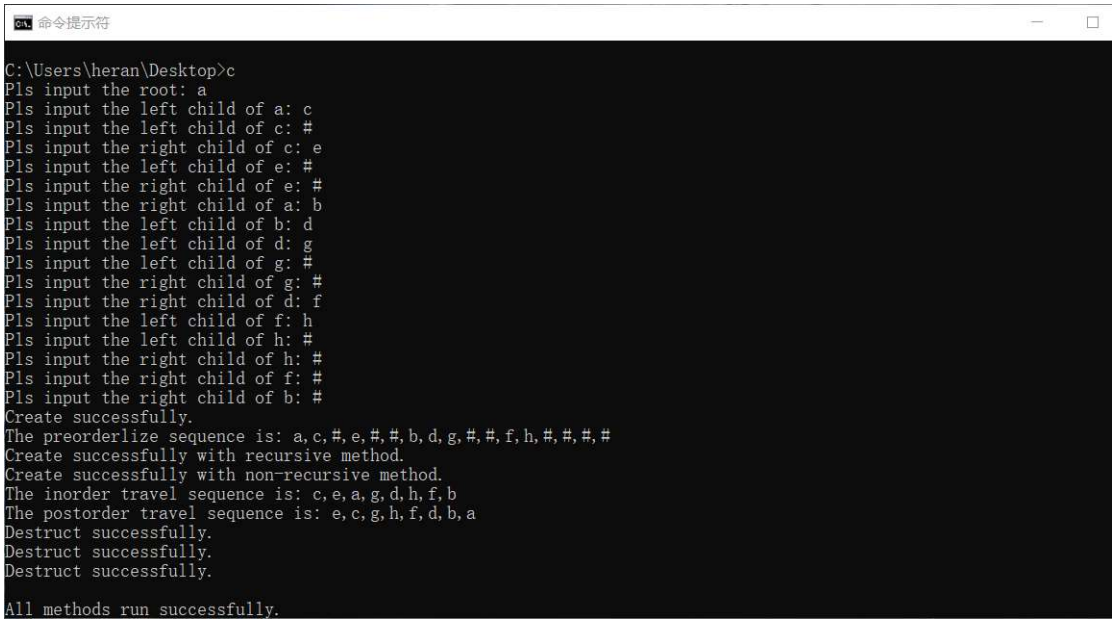
5. **从文件中读入最短路径信息，并处理用户询问**

因为采用压缩存储，我们需要得到真实的存储位置，可以预处理出一个数组保存这些位置。然后在 Trie 树上找到询问的两点的编号，得到这两点最短路径的存储位置，可以快速输出，但是内存消耗量较高。实现如下：

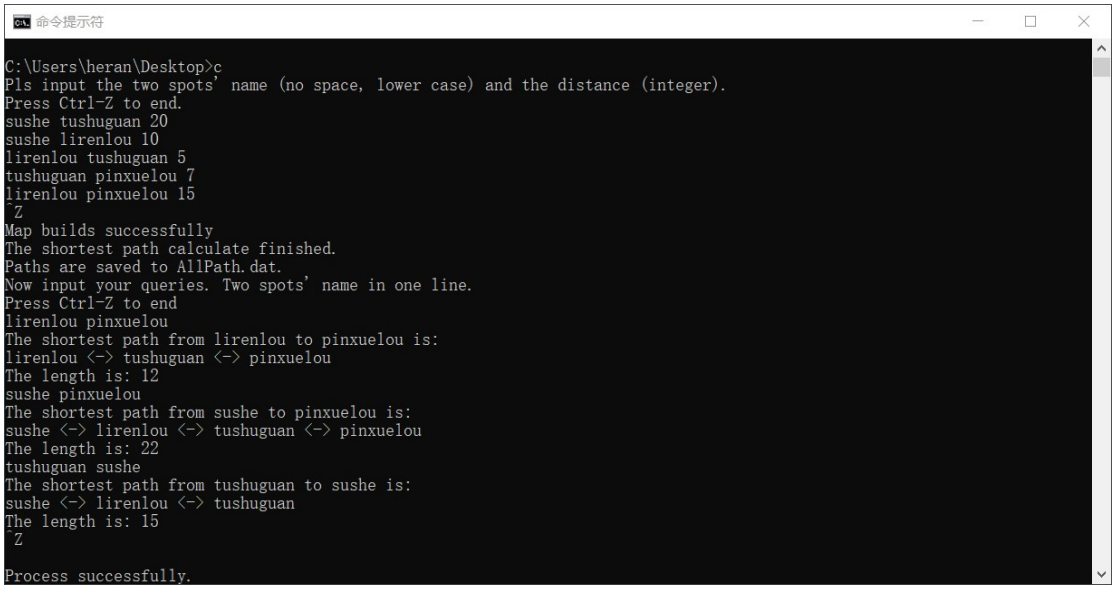
```
void processQuery()
{
    fp = fopen("AllPath.dat", "r");
    int i, j, k, u, v;
    for (i = 1; i <= n; i++)
        for (j = i + 1; j <= n; j++)
        {
            fscanf(fp, "%d %d", &rec[u].len, &rec[u].path[0]);
            for (k = 1; k <= rec[u].path[0]; k++)
                fscanf(fp, "%d", &rec[u].path[k]);
            pt[i][j] = u++;
        }
    fclose(fp);
    puts("Now input your queries. Two spots' name in one line.");
    puts("Press Ctrl-Z to end");
    while (1)
    {
        if (scanf("%s %s", from, to) != 2) break;
        u = query(from);
        if (!u)
        {
            puts("The departure isn't appeared in the map!");
            continue;
        }
        v = query(to);
        if (!v)
        {
            puts("The destination isn't appeared in the map!");
            continue;
        }
        if (u > v)
        {
            i = u; u = v; v = i;
        }
        else if (u == v)
        {
            puts("The two spots are same.");
            continue;
        }
        u = pt[u][v];
        printf("The shortest path from %s to %s is:\n", from, to);
        printf("%s", name[rec[u].path[1]]);
        for (i = 2; i <= rec[u].path[0]; i++)
            printf(" <-> %s", name[rec[u].path[i]]);
        printf("\nThe length is: %d\n", rec[u].len);
    }
    puts("\nProcess successfully.");
    return ;
}
```

九、程序运行结果：

（一）、第一部分



（二）第二部分



十、实验结论：

对于二叉树的序列化，可以用递归实现，对于反序列化的非递归实现，用栈

操作即可。对于求最短路径，可以用 Floyd 算法求解。

十一、总结及心得体会：

对于字符串编号存储，还可以使用哈希的方式，如果使用优秀的哈希函数与碰撞处理方式，可以达到更优的运行时间。存储全部最短路径还可以采用链表存储优化内存空间。