数据结构实验报告

姓名：陈卫星
学号：PB21051184
院系：信息科学技术学院

问题描述

实验题目

- 二叉树及其应用

实验要求

- 以二叉链表作为二叉树的存储结构，实现通过层序遍历创建二叉树、层序遍历二叉树、二叉树后序线索化及其遍历、中缀表达式求值

测试数据

- ```
  A,B,C,Empty,D,E,F,G,Empty,Empty,H,Empty,Empty,Empty,Empty,Empty,Empty
  1+2*(3/2+4)
  ```

设计过程

定义存储结构

- 为提高程序泛用性，定义节点数据类型 `ndType`：
- ```
  class ndType{
      string s;
      double val;
  };
  ```
- 对于一般的树， `s` 储存其节点名称；对于表达式树， `s` 储存运算符，且当 `s=="0"` 时 `val` 存储运算数。
- 随后定义节点类型 `Tree`

- ```
  class Tree{
      ndType val;
  public:
      Tree *lson,*rson;
      bool ltag,rtag;
  }
  ```

层序遍历创建二叉树

- 层序遍历即为广度优先搜索，使用队列模拟其建树过程。
- 对于每个出队的节点，其子节点为紧接着读入的两个节点。若某个子节点为空 `"Empty"` 则不再入队，直至清空队列元素。
- ```
  void Read_I(Tree& T){
      cout<<"Start Read"<<endl;
      Queue<Tree*>q;
      if(!T.ReadStr())ERROR("The Tree is empty!");
      q.push(&T);
      ndType val;
      Tree* cur;
      while(!q.empty()){
          cur=q.front();q.pop();
          if(val.ReadStr())
              q.push(cur->lson=new Tree(val));
          if(val.ReadStr())
              q.push(cur->rson=new Tree(val));
      }
      cout<<"Done Read\n"<<endl;
  }
  ```

层序遍历二叉树

- 直接用队列模拟广搜即可
- ```
  void LevelOrderTraversal(const Tree& T){
      cout<<"Start LevelOrderTraversal"<<endl;
      Queue<const Tree*>q;
      q.push(&T);
      const Tree *empNode=new Tree(ndType("Empty")),*cur;
      while(!q.empty()){
          cur=q.front();q.pop();
          if(cur!=&T)cout<<',';
          cout<<*cur;
          if(!cur->empty()){
  ```
  ```

```
            q.push(cur->lson?cur->lson:empNode);
            q.push(cur->rson?cur->rson:empNode);
        }
    }
    cout<<"\nDone LevelOrderTraversal\n"<<endl;
}
```

二叉树后序线索化

- 对于前驱，若某节点无子节点则需寻找祖先路径上最近的有左子节点的祖先。
- 对于后继，若某节点有右兄弟则需寻找右兄弟子树中最左侧叶节点。
- 虽然在考虑上述情况后，处理直接处理各节点前驱后继需遍历的节点总数至多为2ν，总时间复杂度仍为$O(\nu)$，但需额外编写两个较复杂的函数，且分类情况较多。
- 于是考虑先得到后序遍历，然后为后序遍历中相邻的两个节点处理前驱、后继关系的方法
- ```
  void Tree::PostOrderTraversal(Queue<Tree*>& q){//后序遍历
      if(lson)lson->PostOrderTraversal(q);
      if(rson)rson->PostOrderTraversal(q);
      q.push(this);
  }
  void Tree::LinkLeft(Tree* pre){
      if(!lson&&pre){
          ltag=1;
          lson=pre;
      }
  }
  void Tree::LinkRight(Tree* nxt){
      if(!rson&&nxt){
          rtag=1;
          rson=nxt;
      }
  }
  void PostThread(Tree& T){
      cout<<"Start PostThread"<<endl;
      Queue<Tree*>q;
      T.PostOrderTraversal(q);
      Tree *pre=NULL,*cur;
      while(!q.empty()){
          cur=q.front();q.pop();
          if(pre)pre->LinkRight(cur);
          cur->LinkLeft(pre);
          pre=cur;
      }
      cout<<"Done PostThread\n"<<endl;
  ```
  ```

```
}
```

后序线索二叉树遍历

- 若按后序遍历从前往后找，则对于每个节点的后继有以下情况：
    1. 若为根，则无后继
    2. 若 `rtag==1`，则后继为 `rson`
    3. 否则：
        1. 若有右兄弟节点，则后继为右兄弟子树内最左侧的叶节点
        2. 若无右兄弟节点，则后继为父节点
- 若按后序遍历从后往前找，则对于每个节点的前驱有以下情况：
    1. 若 `ltag==1`，则前驱为 `lson`
    2. 否则：
        1. 若无左子节点，则无前驱
        2. 若 `rtag==0` 且有右子节点，则前驱为 `rson`
        3. 否则，前驱为 `lson`
- 比较两种方案，第二种不需要额外的函数
- ```
  void PostThreadTraversal(Tree T){
      Tree* cur=&T;
      cout<<"Start PostThreadTraversal"<<endl;
      Stack<Tree*>stk;
      do{
          stk.push(cur);
          if(cur->ltag)cur=cur->lson;
          else if(!cur->lson)cur=NULL;
          else if(!cur->rtag&&cur->rson)cur=cur->rson;
          else cur=cur->lson;
      }while(cur);
      while(!stk.empty()){
          if(cur)cout<<',';
          cur=stk.top();stk.pop();
          cout<<*cur;
      }
      cout<<"\nDone PostThreadTraversal\n"<<endl;
  }
  ```
  ```

中缀表达式求值

- 首先通过栈匹配每一对括号，`jump[i]`记录括号包含的表达式长度
- ```
  void Expression(Tree& T){
      cout<<"Start Read Expression"<<endl;
      string s;
      cin>>s;
      int len=s.length();
      int* jump=new int[len];
      Stack<int>stk(len);
      for(int i=0;i<len;i++){
          jump[i]=1;
          if(s[i]=='(')stk.push(i);
          else if(s[i]==')'){
              if(stk.empty())ERROR("Unmatched brackets!");
              int lid=stk.top();stk.pop();
              jump[lid]=i-lid+1;
          }
      }
      if(!stk.empty())ERROR("Unmatched brackets!");
      T.Divide(s,jump);
      cout<<"Done Calc Expression\n"<<endl;
  }
  ```
- 随后构造表达式树，每次找到运算优先级最低的运算符作为当前子树的根，并分割表达式为左右子树
- 待左右子树均运算完后合并
- ```
  class Superior_{
      const char operators[7]="+-*/()";
  public:
      bool operator ()(char x,char y)const{
          int a,b;
          for(a=0;a<6&&operators[a]!=x;a++);
          for(b=0;b<6&&operators[b]!=y;b++);
          return (a>>1)<(b>>1);
      }
      int operator [](char x)const{
          int a;
          for(a=0;a<6&&operators[a]!=x;a++);
          return a;
      }
  }Superior;
  ```

```
ndType ndType::operator +(const ndType& _)const{
    return ndType("+",val+_.val);
}
ndType ndType::operator -(const ndType& _)const{
    return ndType("-",val-_.val);
}
ndType ndType::operator *(const ndType& _)const{
    return ndType("*",val*_.val);
}
ndType ndType::operator /(const ndType& _)const{
    if(iszero(_.val))ERROR("Divided by zero!");
    return ndType("/",val/_.val);
}
void Tree::Divide(string s,int* jump){
    int mid=0;
    for(int i=0;i<(int)s.length();i+=jump[i])
        if(!Superior(s[mid],s[i]))mid=i;
    char op=s[mid];
    double v;
    istringstream is;
    switch(op){
        case '+':case '-':case '*':case '/':
            (lson=new Tree)->Divide(s.substr(0,mid),jump);
            (rson=new Tree)-
>Divide(s.substr(mid+1),jump+mid+1);
            switch(op){
                case '+':val=lson->val+rson->val;break;
                case '-':val=lson->val-rson->val;break;
                case '*':val=lson->val*rson->val;break;
                case '/':val=lson->val/rson->val;break;
            }
            break;
        case
'(':Divide(s.substr(1,s.length()-2),jump+1);break;
        case '0':case '1':case '2':case '3':case '4':case
'5':case '6':case '7':case '8':case '9':case '.':
            is=istringstream(s);
            if(is>>v)val=ndType(v);
            else ERROR("Unknow character!");
            break;
        default:
            ERROR("Unknow character!");break;
    }
}
```

调试分析

算法时空复杂度分析：

- 由于本程序中的栈和队列均为动态长度数组，且二叉树通过结构体指针连接，故空间复杂度为$O(\nu)$
- 由于实现处理了括号的匹配，所以构造表达式树时每个字符至多被经过两次，时间复杂度同其他函数，为$O(\nu)$

调试结果

- 测试数据为：
  - ```
    A,B,C,Empty,D,E,F,G,Empty,Empty,H,Empty,Empty,Empty,Empty,
    Empty,Empty
    1+2*(3/2+4)
    ```
- 使用键盘输入测试数据后，运行结果为：
  - 

```
274         jump[i]=1;
275         if(s[i]=='(')stk.push(i);
276         else if(s[i]==')'){
277             if(stk.empty())ERROR("Unmatched brackets!");
278             int lid=stk.top();stk.pop();
279             jump[lid]=i-lid+1;
280         }
281     }
282     if(!stk.empty())ERROR("Unmatched brackets!");
283     T.Divide(s,jump);
284     cout<<"Done Calc Expression"<<endl;
285 }
286 int main(){
287     Tree T;
288     Read_I(T);
289     LevelOrderTraversal(T);
290     PostThread(T);
291     PostThreadTraversal(T);
292     Tree exp;
293     Expression(exp);
294     LevelOrderTraversal(exp);
295     exp.PrintVal();
296     return 0;
297 }
298 /*
299 A,B,C,Empty,D,E,F,G,Empty,Empty,H,Empty,Empty,Empty,Empty,Empty,Empty
300 1+2*(3/2+4)
301 */
```

```
C:\Windows\system32\cmd.exe
Start Read
A, B, C, Empty, D, E, F, G, Empty, Empty, H, Empty, Empty, Empty, Empty, Empty, Empty
Done Read

Start LevelOrderTraversal
A, B, C, Empty, D, E, F, G, Empty, Empty, H, Empty, Empty, Empty, Empty, Empty, Empty
Done LevelOrderTraversal

Start PostThread
Done PostThread

Start PostThreadTraversal
G, D, B, H, E, F, C, A
Done PostThreadTraversal

Start Read Expression
1+2*(3/2+4)
Done Calc Expression

Start LevelOrderTraversal
+, 1, *, Empty, Empty, 2, +, Empty, Empty, /, 4, 3, 2, Empty, Empty, Empty, Empty, Empty, Empty
Done LevelOrderTraversal

val=12
请按任意键继续. . .
```
```

附录

原程序文件清单：

```
bits/stdc++.h
二叉树.cpp
```