how U doin'?

$$🐳^{🐳} = \int_{🎃}^{🎅} 🐑 \, d🍀$$

Tree*

10170437 Mark Taylor

June 26, 2020

Contents

1 Problems 2

2 Code 2
- 2.1 Tree.h . . . 2
- 2.2 Tree_test.cpp . . . 5
- 2.3 SeqStack.h . . . 6
- 2.4 Stack.h . . . 8

3 Output 8

4 Appendix 9

1 Problems

1. 编写算法:输入树的边(双亲-孩子有序对)建立对应树的孩子-兄弟链表存储结构;
2. 编写算法:求树的深度;
3. 编写算法:输出从树根到所有叶子结点的路径;
4. 编写算法:统计树中叶子结点的个数。

2 Code

2.1 Tree.h

*This article was typeset by Mark Taylor using the LaTeX document processing system.

```cpp
// tree header
#ifndef TREE_H
#define TREE_H
#include "SeqStack.h"
#include <cassert>
#include <iostream>

template<typename T>
struct TreeNode {
        T _data;
        TreeNode<T>* _first_child, * _next_sibling;
        TreeNode(const T value = {}, TreeNode<T>* fc = nullptr, TreeNode<T>* ns = nullptr)
                : _data(value), _first_child(fc), _next_sibling(ns) {}
};

template<typename T>
class Tree {
public:
        Tree() { _root = nullptr; }
        TreeNode<T>* create_tree(TreeNode<T>* t);

        TreeNode<T>* get_root()const { return _root; }
        bool empty()const { return _root == nullptr; }
        size_t depth(TreeNode<T>* t)const;
        size_t count_node(TreeNode<T>* t)const;
        size_t count_leaves(TreeNode<T>* t)const;

        // We do NOT set traverses as const member functions
        // given that we may need to modify those nodes.
        // see also <https://isocpp.org/wiki/faq/pointers-to-members>
        void preorder(TreeNode<T>* t, void (*visit)(TreeNode<T>*) = visit);
        void postorder(TreeNode<T>* t, void (*visit)(TreeNode<T>*) = visit);

        // print path from root node to leaf
        void print_path(TreeNode<T>* t)const;

private:
        TreeNode<T>* _root;
        static void visit(TreeNode<T>* t);
};

// create tree by level
template<>        // specialization for char type
TreeNode<char>* Tree<char>::create_tree(TreeNode<char>* t)
{
        // to be added..
        return nullptr;
}

template<typename T>
size_t Tree<T>::depth(TreeNode<T>* t) const
{
        if (t == nullptr) return 0;
        size_t fc_depth = depth(t->_first_child) + 1;
        size_t ns_depth = depth(t->_next_sibling);

        return fc_depth > ns_depth ? fc_depth : ns_depth;
}

template<typename T>
size_t Tree<T>::count_node(TreeNode<T>* t) const
```

```
{
        if (t == nullptr) return 0;
        size_t count = 1;
        count += count_node(t->_first_child);
        count += count_node(t->_next_sibling);

        return count;
}

template<typename T>
size_t Tree<T>::count_leaves(TreeNode<T>* t) const
{
        static size_t count = 0;
        if (t != nullptr) {
                if (t->_first_child == nullptr)
                        ++count;
                else
                        count_leaves(t->_first_child);

                count_leaves(t->_next_sibling);
        }
        return count;
}

template<typename T>
void Tree<T>::preorder(TreeNode<T>* t, void(*visit)(TreeNode<T>*))
{
        if (t != nullptr) {
                visit(t);
                preorder(t->_first_child, visit);
                preorder(t->_next_sibling, visit);
        }
}

template<typename T>
void Tree<T>::postorder(TreeNode<T>* t, void(*visit)(TreeNode<T>* ))
{
        if (t != nullptr) {
                postorder(t->_first_child, visit);
                visit(t);
                postorder(t->_next_sibling, visit);
        }
}

template<typename T>
void Tree<T>::print_path(TreeNode<T>* t) const
{
        static SeqStack<T> s;
        while (t != nullptr) {
                s.push(t->_data);
                if (t->_first_child == nullptr) {
                        s.bottom_up_traverse();
                        std::cout << '\n';
                }
                else
                        print_path(t->_first_child);
                s.pop();
                t = t->_next_sibling;
        }
}

```

```
123 // operation for node p
124 template<typename T>
125 inline void Tree<T>::visit(TreeNode<T>* p)
126 {
127         assert(p != nullptr);
128         std::cout << p->_data;
129 }
130
131 #endif // !TREE_H
```

Listing 1: Tree header

2.2 Tree_test.cpp

```
1  #include "Tree.h"
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8          Tree<char> tree;
9          auto root = tree.get_root();
10         //tree.create_tree(root);        // to be added
11         //          A
12         //        / | \
13         //       B  C  D
14         //       |    / \
15         //       E    F  G
16         cout <<
17                 "            A\n"
18                 "          / | \\     \n"
19                 "         B  C  D     \n"
20                 "         |    / \\   \n"
21                 "         E   F   G   \n";
22
23
24         root = new TreeNode<char>('A');
25         root->_first_child = new TreeNode<char>('B');
26         root->_first_child->_first_child=new TreeNode<char>('E');
27
28         root->_first_child->_next_sibling = new TreeNode<char>('C');
29         root->_first_child->_next_sibling->_next_sibling = new TreeNode<char>('D');
30
31         root->_first_child->_next_sibling->_next_sibling
32                 ->_first_child = new TreeNode<char>('F');
33
34         root->_first_child->_next_sibling->_next_sibling
35                 ->_first_child->_next_sibling = new TreeNode<char>('G');
36
37         cout << "All paths from root to leaves are as follows:\n";
38         tree.print_path(root);
39         cout << "number of tree nodes: " << tree.count_node(root) << '\n';
40         cout << "number of tree leaves: " << tree.count_leaves(root) << '\n';
41         cout << "depth:  " << tree.depth(root) << '\n';
42         cout << "preorder: "; tree.preorder(root); cout << '\n';
43         cout << "postorder: "; tree.postorder(root);
```

```
44
45          cout << "\n\nPress any key to leave...\n";
46          char wait;
47          cin >> noskipws >> wait;
48
49          return 0;
50  }
```

Listing 2: Tree test

2.3 SeqStack.h

```
1   // sequential stack header
2   #pragma once
3   #ifndef SEQSTACK_H
4   #define SEQSTACK_H
5
6   #include "Stack.h"
7   #include <iostream>
8   #include <cassert>
9
10  const int defaultStackSize = 30;
11  const int stackIncreament = 20;                 // let it double its capacity
12
13  template<typename T>
14  class SeqStack : public Stack<T> {
15  public:
16          SeqStack(int sz = defaultStackSize);
17          SeqStack(const SeqStack<T>& stack);
18          SeqStack<T>& operator=(const SeqStack<T>& stack);
19          ~SeqStack() { delete[] elem; }
20          void push(const T& x);
21          void pop();
22          void pop(T& x);
23          T& top()const;
24          void clear() { _top = -1; }
25          inline bool isEmpty()const { return _top == -1; }
26          bool isFull()const { return _top == maxSize - 1; }
27          int max_size()const { return maxSize; }
28          inline int size()const { return _top + 1; }
29          void bottom_up_traverse()const;
30
31  private:
32          T* elem;                                        // pointer to stack array
33          int _top;                                        // index of stack top: [0, size), when stack's
            ↪  empty, _top=-1
34          int maxSize;                                  // maximum volume of the stack
35          void overflowProcess();
36  };
37
38  template<typename T>
39  SeqStack<T>::SeqStack(int sz) {        // construct an empty stack with a size of sz
40          _top = -1;
41          maxSize = sz;
42          elem = new T[maxSize];
43          assert(elem != nullptr);
44  }
```

```
45
46  template<typename T>
47  SeqStack<T>::SeqStack(const SeqStack<T>& stack) { // copy constructor
48          _top = stack._top;
49          maxSize = stack.maxSize;
50          elem = new T[maxSize];
51          assert(elem != nullptr);
52          for (int i = 0; i <= _top; ++i) elem[i] = stack.elem[i];
53  }
54
55  template<typename T>
56  SeqStack<T>& SeqStack<T>::operator=(const SeqStack<T>& stack) { // copy assignment
57          if (&stack == this)         return *this;
58          delete[] elem;
59          _top = stack._top;
60          maxSize = stack.maxSize;
61          elem = new T[maxSize];
62          assert(elem != nullptr);
63          for (int i = 0; i <= _top; ++i) elem[i] = stack.elem[i];
64          return *this;
65  }
66
67  template<typename T>
68  void SeqStack<T>::push(const T& x){ // add a new element at the top of the stack
69          if (isFull()) overflowProcess();
70          elem[++_top] = x;
71  }
72
73  template<typename T>
74  void SeqStack<T>::pop() { // pop out the top element of the stack
75          assert(!isEmpty());
76          --_top;
77  }
78
79  template<typename T>
80  void SeqStack<T>::pop(T& x) { // pop out the top element of the stack and assign it to x
81          assert(!isEmpty());
82          x = elem[_top--];
83  }
84
85  template<typename T>
86  T& SeqStack<T>::top() const { // get the top element
87          assert(!isEmpty());
88          return elem[_top];
89  }
90
91  template<typename T>
92  void SeqStack<T>::bottom_up_traverse() const
93  {
94          for (int i = 0; i <= _top; ++i) {
95                  std::cout << elem[i] << '\t';
96          }
97  }
98
99  template<typename T>
100 void SeqStack<T>::overflowProcess() { // private member function, expanding the stack's size
101         T* newArray = new T[maxSize += stackIncreament];
102         assert(newArray != nullptr);
103         for (int i = 0; i <= _top; ++i) newArray[i] = elem[i];
104         delete elem;
105         elem = newArray;
```

```
106 }
107
108 #endif // !SEQSTACK_H
```

Listing 3: Sequential stack header

2.4 Stack.h

```
1  // stack header, abstract base class for interfaces
2  #pragma once
3  #ifndef STACK_H
4  #define STACK_H
5
6  template<typename T>
7  class Stack {
8  public:
9          virtual void push(const T& x) = 0;
10         virtual void pop() = 0;
11         virtual void pop(T& x) = 0;
12         virtual T& top()const = 0;
13         virtual void clear() = 0;
14         virtual bool isEmpty()const = 0;
15         virtual int size()const = 0;
16 };
17
18 #endif // !STACK_H
```

Listing 4: Stack header

3 Output

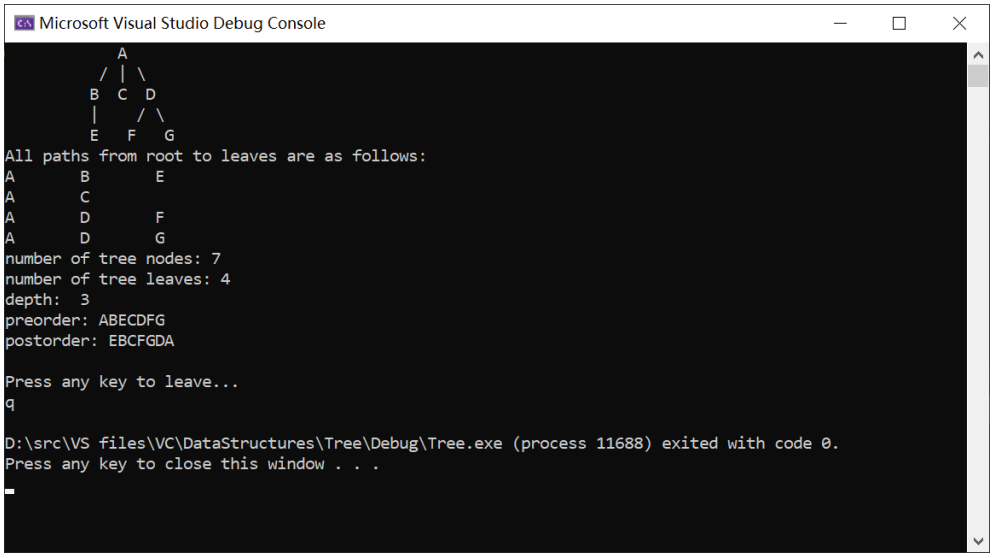

Figure 1: Tree test

4 Appendix

Hello from the Beatles.😃