FA HW 2

Xiaoyu Xue

2017-09-20

1 Ex 2.1

a) 51, 6, 3, 5, 14, 7, 11
b) 3, 5, 6, 7, 11, 14, 51
c) 5, 3, 11, 7, 14, 6, 51

2 Ex 2.30

```
global the tree T with vertex name 1...n; child array child[j, *]
procedure postDFS(i);
   if child[i,0] = 0 then
      print(i);
   else
      foreach int j = 1; j <= child[i, 0]; j++ do
         postDFS(child[i, j]);
         print(i);
      endfor
   endif
end_postDFS;
```

3 Ex 2.2

```
   function evaluateExpTree(vertex v);
1     if v is a leaf then
2        return(v.val)
3     else
4        return(eval(v.op, evaluateExpTree(v.left), evaluateExpTree(v.right)));
5     endif
   end_evaluateExpTree;
```

4 Ex 2.3

4.a

```
   function minVal(vertex v);
1     v.small ← v.val;
2     foreach child x of v do
3        v.small ← min(v.small, minVal(x));
4     endfor
5     return(v.small);
   end_minVal;
```

4.b

```
    function minValVertex(vertex v);
1       v.small ← v.val;
2       v.which ← v;
3       foreach child x of v do
4           w ← minValVertex(x);
5           if v.small > w.val then
6               v.small ← w.val;
7               v.which ← w;
8           endif
9       endfor
10      return(v.which)
    end_minValVertex;
```

5 Ex 2.4

5.a

```
    procedure postDFS(T);
1       foreach child v of t do
2           postDFS(v);
3       endfor
4       print(i);
    end_postDFS;
```

5.b

```
    procedure Rotate(T, X);
1       foreach child v of t do
2           Rotate(v, X);
3       endfor
4       temp ← X;
5       X ← T.val;
6       T.val ← temp;
    end_Rotate;
```

6 Ex 2.5

```
    return(eval(x, StackEval(; ; L), StackEval(; ; L)));
```

7 Ex 2.6

7.a

S is the reverse of T

7.b

Postorder DFS Print the vertex from left most without children to the root, but the preorder DFS print vertex from root to right most vertex with children.

7.c

```
    global Stack L contain reverse postorder of exp tree; Stack M for temp storage
    function iterateEval();
1     while L is not empty do
2       x ← PopFrom(L);
3       if x is a number then
4         PushInto(M, x);
5       else
6         PushInto(M, eval(x, PopFrom(M), PopFrom(M)));
7       endif
8     endwhile
9     return(PopFrom(M))
    end_iterateEval;
```

7.d

```
    global Doubly Linked List L contain reverse postorder of exp tree;
    function iterateEvalList();
1     p ← L;
2     while p ≠ Nil do
3       if p.val is a number then
4         p ← p.next;
5       else
6         p.val ← eval(x, p.prev, p.prev.prev));
7         l ← p.prev;
8         m ← p.prev.prev;
9         p.prev ← p.prev.prev.prev;
10        p.prev.next ← p;
11        l.next ← Nil;
12        m.prev ← Nil;
13      endif
14    endwhile
15    return(p.val)
    end_iterateEvalList;
```

8 Ex 2.8

```
    function countChild(T v);
1     v.numb ← 1;
2     foreach child x of v do
3       v.numb ← v.numb + countChild(x);
4     endfor
5     return(v.numb)
    end_countChild;
```

9 Ex 2.9

```
    function distSubTree(T v);
1     v.dis ← 0;
2     foreach child x of v do
3       distance ← distSubTree(x) + 1;
4       if distance > v.dis then
5         v.dis ← distance;
6       endif
7     endfor
8     return(v.dis)
    end_distSubTree;
```

10 Ex 2.10

```
    function twoDistSubTree(T v);
1     v.dis1 ← 0;
2     v.dis2 ← −∞;
3     foreach child x of v do
4       distance ← twoDistSubTree(x) + 1;
5       if distance > v.dis1 then
6         if v.dis1 > 0 then
7           v.dis2 ← v.dis1;
8         endif
9         v.dis1 ← distance;
10      elseif distance <= v.dis1 and distance > v.dis2 then
11        v.dis2 ← distance;
12      endif
13    endfor
14    return(v.dis1)
    end_twoDistSubTree;
```

11 Ex 2.19

11.a

yes

11.b

no

11.c

```
    procedure Parent(v, pv);
1       if v ≠ Nil then
2           print(v, pv);
3       endif
4       foreach child x of v do
5           Parent(x, v);
6       endfor
    end_Parent;
```

Initial Procedure Call:

```
    procedure preDFS();
1       Parent(T, Nil)
    end_preDFS;
```

11.d

```
    procedure Parent(v, pv);
1       if v ≠ Nil then
2           print(v, pv);
3       endif
4       Parent(v.left, v);
5       Parent(v.right, pv);
    end_Parent;
```

Initial Procedure Call:

```
    procedure preDFS();
1       Parent(S, Nil);
    end_preDFS;
```

11.e

```
    procedure Parent(v, pv);
1       foreach child x of v do
```

```
2          Parent(x, v);
3       endfor
4       if v ≠ Nil then
5          print(v, pv);
6       endif
    end_Parent;
```

Initial Procedure Call:

```
    procedure postDFS();
1      Parent(T, Nil);
    end_postDFS;
```

11.f

```
    procedure Parent(v, pv);
1      Parent(v.left, v);
2      Parent(v.right, pv);
3      if v ≠ Nil then
4         print(v, pv);
5      endif
    end_Parent;
```

Initial Procedure Call:

```
    procedure postDFS();
1      Parent(S, Nil);
    end_postDFS;
```

11.g

```
    procedure Parent(v);
1      foreach child x of v do
2         Parent(x, v);
3         print(x, v);
4      endfor
    end_Parent;
```

Initial Procedure Call:

```
    procedure postDFS();
1      Parent(T);
    end_postDFS;
```

12 Ex 2.20

```
    procedure Clean(L);
1      if L ≠ Nil then
2         if L.data = 0 then
```

```
 3            Clean(L.next);
 4            L.next ← Nil
 5          endif
 6          if L.next ≠ Nil and L.next.data = 0 then
 7            Clean(L.next);
 8            L.next ← L.next.next;
 9          endif
10        endif
      end_Clean;
```

Beautiful!!!

13 Ex 23

13.a

```
      procedure RecursiveSelectSort(l;; Data[1...n]);
 1      IndexofBiggest ← 1;
 2      Biggest ← Data[1];
 3      foreach TestDex ← 2 to l do
 4        if Biggest < Data[TestDex] then
 5          Biggest ← Data[TestDex];
 6          IndexOfBiggest ← TestDex;
 7        endif
 8      endfor
 9      Swap(Data[IndexofBiggest], Data[l]);
10      if l > 1 then
11        RecursiveSelectSort(l-1;; Data[1...n]);
12      endif
      end_RecursiveSelectSort;
```

13.b

```
      procedure RecursiveSelectSort2(l;; Data[1...n]);
 1      IndexofBiggest ← 1;
 2      Biggest ← Data[1];
 3      foreach TestDex ← 2 to l do
 4        if Biggest < Data[TestDex] then
 5          Biggest ← Data[TestDex];
 6          IndexOfBiggest ← TestDex;
 7        endif
 8      endfor
 9      if l > 1 then
10        Swap(Data[IndexofBiggest], Data[l]);
11        RecursiveSelectSort2(l-1;; Data[1...n]);
12        Swap(Data[l], Data[IndexofBiggest]);
13      else
```

```
14          Print(Data[1...n])
15        endif
    end_RecursiveSelectSort2;
```