There are the following JAVA classes implementing the multi-frontal direct solver algorithm for finite difference problem:

- *A.java*
- *A1.java*
- *A2.java*
- *AN.java*
- *Aroot.java*
- *BS.java*
- *E2.java*
- *Eroot.java*
- *Executor.java*
- *Main.java*
- *P1.java*
- *P2.java*
- *P3.java*
- *Production.java*
- *Vertex.java*

Let us focus on the classes *P1.java, P2.java, P3.java*. The classes implement three graph grammar productions generating the binary elimination tree for the multi-frontal solver.

They start with generation of the root node with two son nodes, as presented in Figure 1 The following production allows for further derivation of the tree nodes, as presented in Figure 2 The following production completes the derivation, by generating the leaf nodes, see Figure 3

Please implement productions *P1,P2* and P3.

Hint:

Production *P1* looks like this:

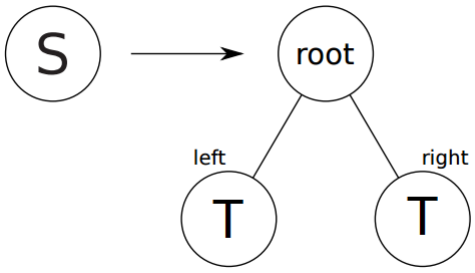

Figure 1: Productions *P1* generating the root of the elimination tree.

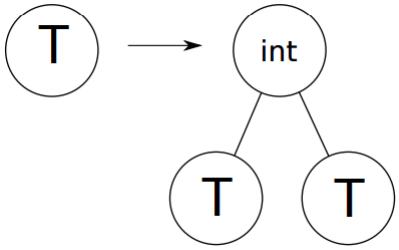

Figure 2: Productions *P2* generating the new internal node with two son nodes.

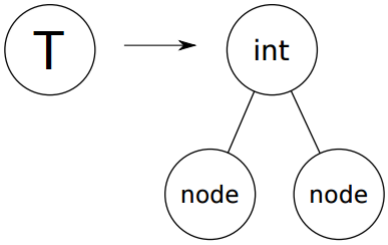

Figure 3: Productions *P3* generating the new internal node with two son nodes.

```
1   class P1 extends Production {
2     P1(Vertex Vert,CyclicBarrier Barrier){
3       super(Vert,Barrier);
4     }
5   Vertex apply(Vertex S) {
6     System.out.println(''p1'');
7     Vertex T1 = new Vertex(null,null,S,''T'');
8     Vertex T2 = new Vertex(null,null,S,''T'');
9     S.set_left(T1);
10    S.set_right(T2);
11    S.set_label(''root'');
12    return S;
13    }
14  }
```

The *Executor* class constructs the elimination tree by applying the graph grammar productions, starting from the root, ending in the leaves.

```
1     import java.util.concurrent.BrokenBarrierException;
2     import java.util.concurrent.CyclicBarrier;
3   class Executor extends Thread {
4     public synchronized void run() {
5       barrier barrier = new barrier(this);
6       Vertex S = new Vertex(null,null,null,''S'');
7       try {
8       //[(P1)]
9       CyclicBarrier barrier = new CyclicBarrier(1);
10      P1 p1 = new P1(S,barrier);
11      p1.start();
12      barrier.await();
13      //[(P2)_1]
14      barrier = new CyclicBarrier(1);
15      P2 p2a = new P2(p1.m_vertex.m_left,barrier);
16      p2a.start();
17      barrier.await();
18      //[(P2)_1(P2)_2]
19      barrier = new CyclicBarrier(1);
20      P2 p2b = new P2(p1.m_vertex.m_right,barrier);
21      p2b.start();
```

```
22      barrier.await();
23      //[(P3)_1]
24      barrier = new CyclicBarrier(1);
25      P3 p3a = new P3(p2a.m_vertex.m_left,barrier);
26      p3a.start();
27      barrier.await();
28      //[(P3)_2]
29      barrier = new CyclicBarrier(1);
30      P3 p3b = new P3(p2a.m_vertex.m_right,barrier);
31      p3b.start();
32      barrier.await();
33      //[(P3)_3]
34      barrier = new CyclicBarrier(1);
35      P3 p3c = new P3(p2b.m_vertex.m_left,barrier);
36      p3c.start();
37      barrier.await();
38      //[(P3)_4]
39      barrier = new CyclicBarrier(1);
40      P3 p3d = new P3(p2b.m_vertex.m_right,barrier);
41      p3d.start();
42      barrier.await();
43      //done
44      System.out.println(''done'');
45      GraphDrawer drawer = new GraphDrawer();
46      drawer.draw(p3a);
47      } catch (InterruptedException | BrokenBarrierException e) {
48        e.printStackTrace(); }
49    }
50  }
```

Please modify the class to generate the elimination tree presented in the Figure 4.

Please plot the tree on the paper, by checking the GraphDrawer report.

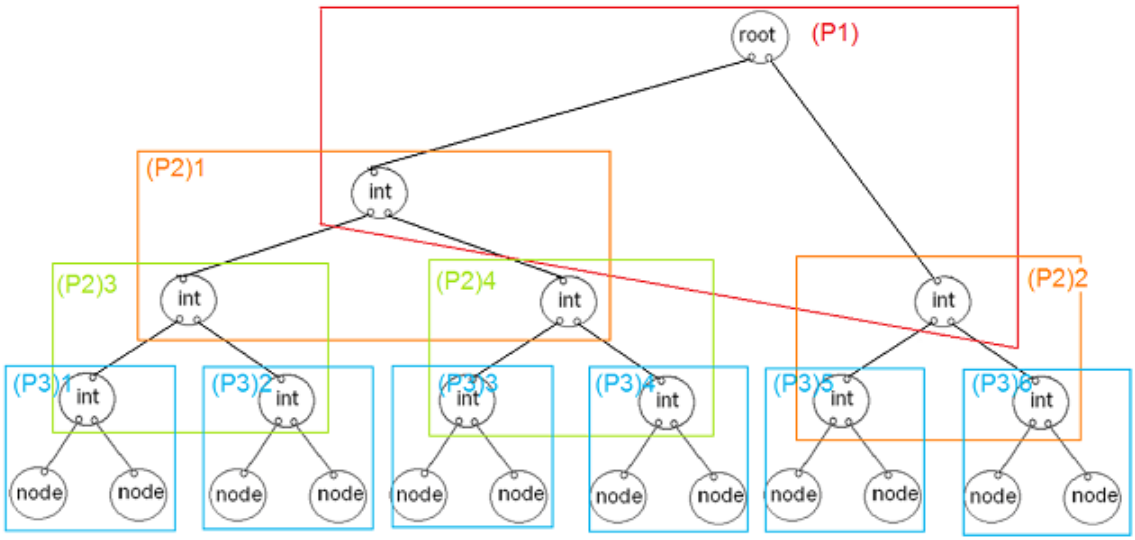

Figure 4: Elimination tree representing a mesh with 6 elements.