**网络实验报告 exp4**

**2020K8009929032 刘耕印**

在正文之前写一点简单的理解：在给定整个网络的图之后，最小生成树已经是一个确定的信息，这个信息可以认为是从各个结点到根结点的路径集合，分布在整个图中。最开始只有根结点知道其到根结点的路径，生成树算法就是把这些信息慢慢传递到整个图的过程。每次交换配置信息都会使信息传递的范围扩大，所以整个算法最后是收敛的。根据这种想法，理解生成树算法可能会比较顺利一点。

***生成树算法***

1. 实验内容

对于复杂的网络拓扑，其中可能存在大量的环路。为了消除冗余链路，采取图论中的生成树算法，为网络拓扑产生最小生成树，抛弃冗余网段。实验时在给定代码的基础上，实现每个结点收到配置消息后的行为，完成生成树算法的收敛。

1. 实验准备

首先根据课件，理解整个算法的大致步骤。算法初始化时每个结点均声明自己为根结点，于是每个端口都是指定端口（网段内能够发送消息的端口）。从指定端口发送初始的config消息。当某个端口收到config消息后，将其优先级与本地config的优先级相比较（config的收发主体是端口）。如果本地的优先级较高，说明本地离应当成为根结点的结点比较近，于是不需要改变本地的配置，从收到config的端口发出本地的config对对端端口进行通知，使对端端口改换其配置即可。否则，这个端口的本地配置要修改到和收到的配置相同，并对端口所在的结点进行影响。该结点首先依照各个端口存储的配置消息的优先级顺序选举根端口（结点内唯一能够从根节点方向接受消息的端口），然后更改各个端口的性质（由于结点配置的改变，有些非指定端口可能成为指定端口，而指定端口的配置也需要更新），最后把更改后的配置从所有指定端口发送出去。另外，根结点的配置是定时发送的，其余结点的配置是收到配置消息后发送的。

然后，阅读给定代码以确认实现逻辑和细节。阅读发现，main函数首先对本地结点初始化了生成树机制，将其设为根结点，并启动定时发送：

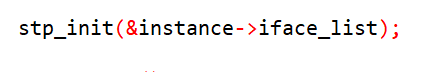


Figure 调用初始化

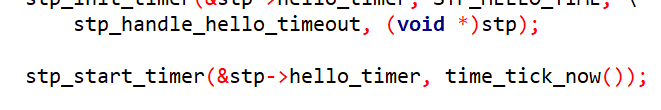


Figure 当前结点开始计时

然后，反复遍历所有端口，查看有没有收到配置消息。

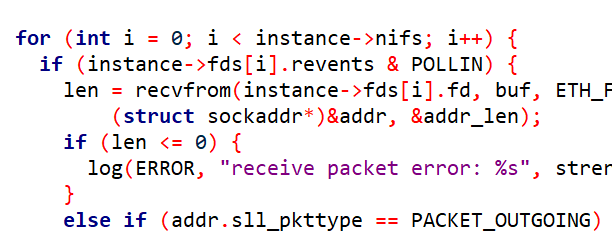


Figure ustack\_run遍历 >> handle\_packet >> stp\_port\_handle\_packet

此时，运行算法的各个结点都开始了定时发包，生成树算法开始。接下来只要正确地处理收到的包，使得生成树算法收敛即可。这项工作将由我编写函数stp\_handle\_config\_packet以完成。

最后，简单看一下数据结构和已经实现的判断方法，熟悉其使用，准备工作就结束了。

1. 实验过程

我们的代码只考虑收到配置信息后的处理即可。这部分参考课件17页，按其步骤实现即可。

首先，比较外来config与本端口存储的config优先级。优先级比较有4段，首先比较根结点id，然后比较到根结点开销，再比较上一跳交换机id，最后比较上一跳端口id。这个典型的比较在整个函数里一共使用了3次，我没有封装成函数，大概是下面这样的形式：

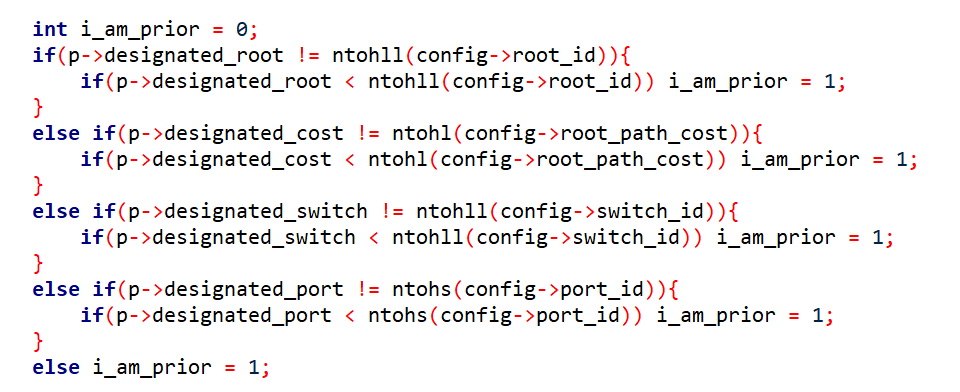


Figure 优先级比较流程

之后按照本地优先级是否较高，分别处理。

如果本地优先级比较低，需要经历5个独立的步骤来完成处理。首先，是要改变本端口的配置，与更高优先级的配置保持一致：

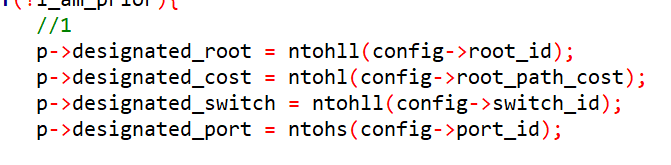


Figure 改变本端口config

这一步比较简单。然后，遍历结点的所有端口，选举优先级最高的为根端口，并让结点通过选举出的根端口连接到根结点：

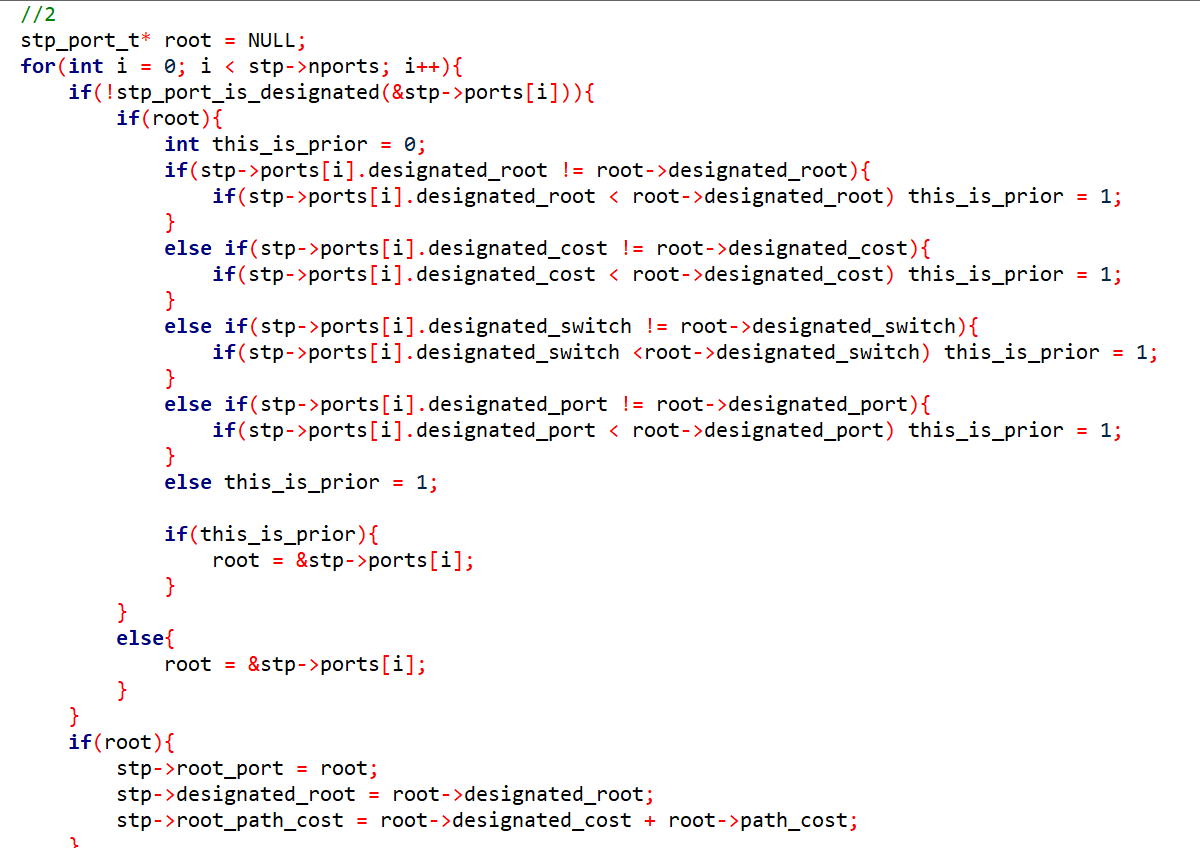


Figure 选举根端口并改变结点配置

关于这一步，我曾有疑惑为什么要遍历所有端口以选择根端口，因为我认为刚刚改变优先级的端口一定有最高的优先级，于是一定成为根端口。助教老师提醒，结点的每个端口在算法收敛之前并不一定具有同步的优先级，有些非指定端口可能具有其他端口未知的高优先级。所以一定要遍历。

接下来，由于结点状态更新，其各个端口的形态可能有变化，要遍历处理：

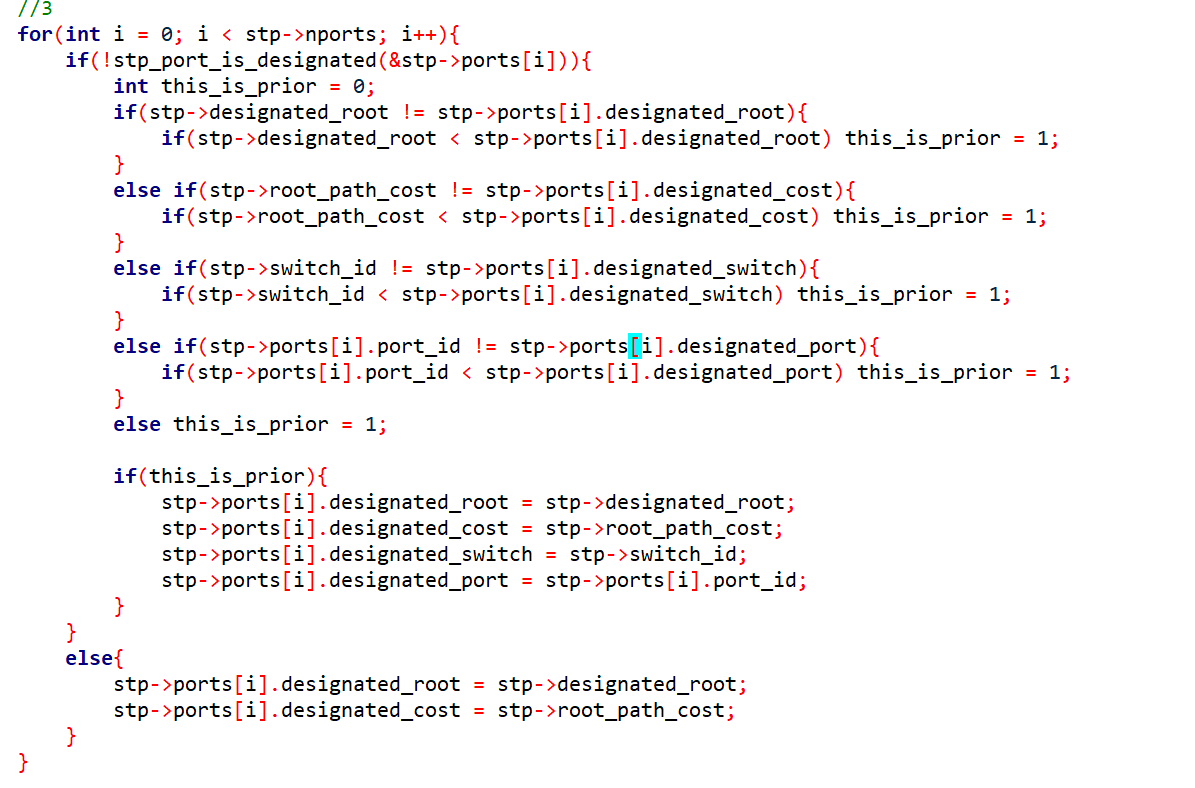


Figure 更新其余端口状态

有些非指定端口，由于结点优先级变高，可能变得比对端端口拥有更高的优先级，此时要重新比较。比较时用结点优先级代表本端口优先级，用本端口暂存的优先级代表对端端口优先级，若前者较高则本端口应当成为指定端口。同时，对于本身就是指定端口的端口，也要更新它们的优先级。

此时用函数判断本结点是否由根节点退化成了非根节点。如果发生改变，那么非根节点不能定时发送信息，要停止计时器：

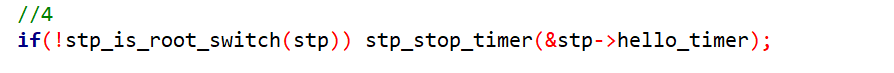


Figure 停止计时器

最后，把本结点发生更新的消息通过所有的指定端口传达给其他结点。



Figure 广播改变后配置

到此，本端口优先级较低的情况处理完毕。对端端口优先级较低的情况比较简单，只需从本端口发送现有配置即可：

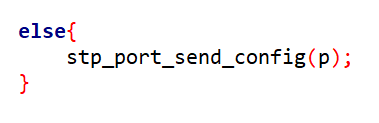


Figure 发送本端口配置

至此代码编写完毕。下面测试功能正确性，在py脚本中，创建结点后自动执行stp程序，约30秒后杀死所有stp进程，观察输出如下：

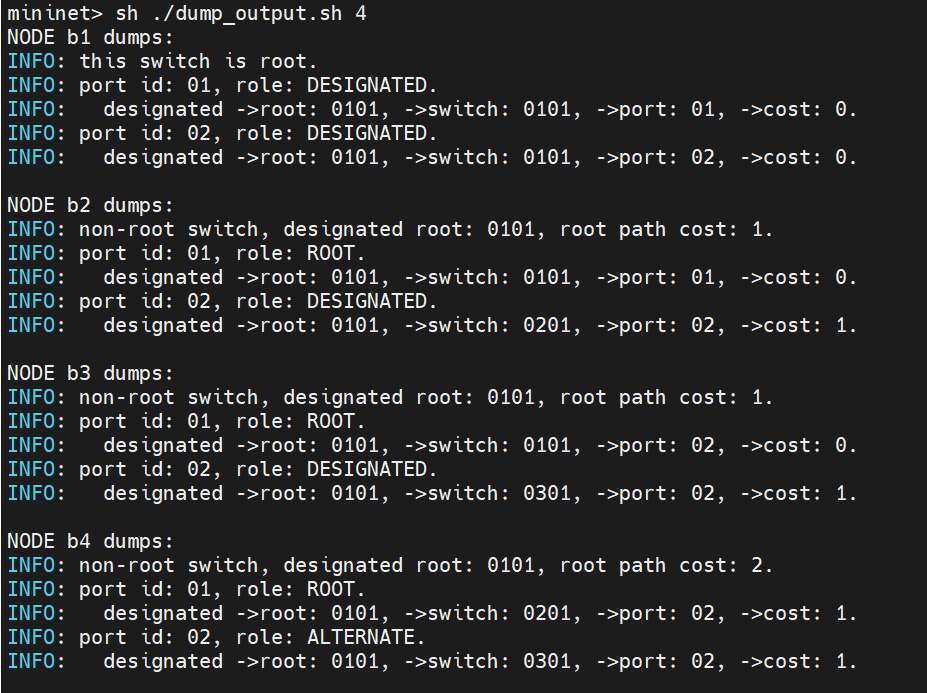


Figure 运行结果

运行后拓扑符合最小生成树结构。将代码提交到oj，成功通过。

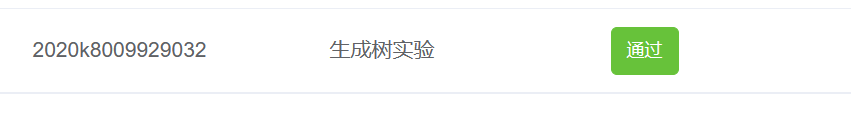


Figure 通过

接下来，修改py脚本，自建一个比较复杂的拓扑：

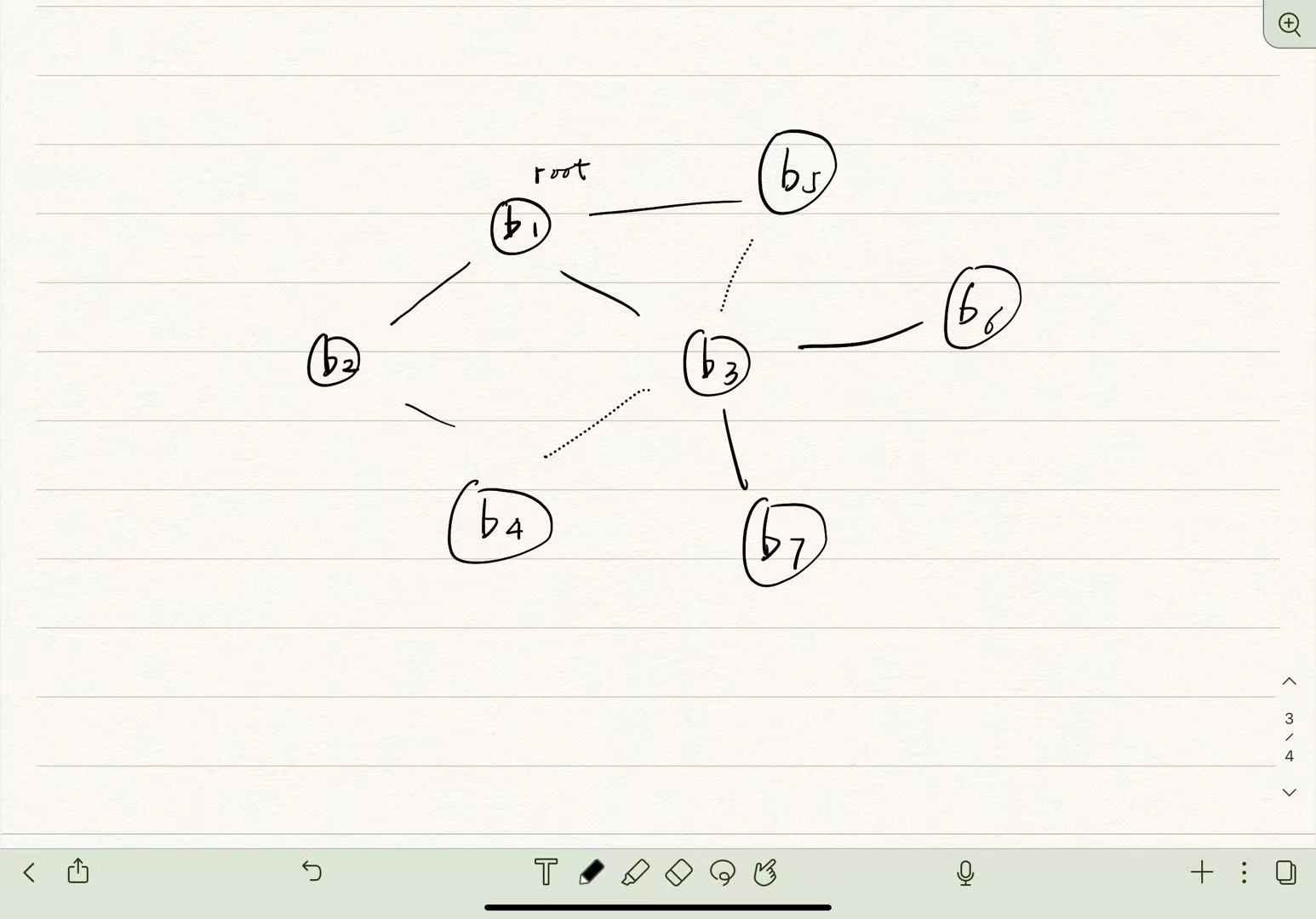


Figure 自建拓扑，其中虚线表示预计被消除的冗余链路

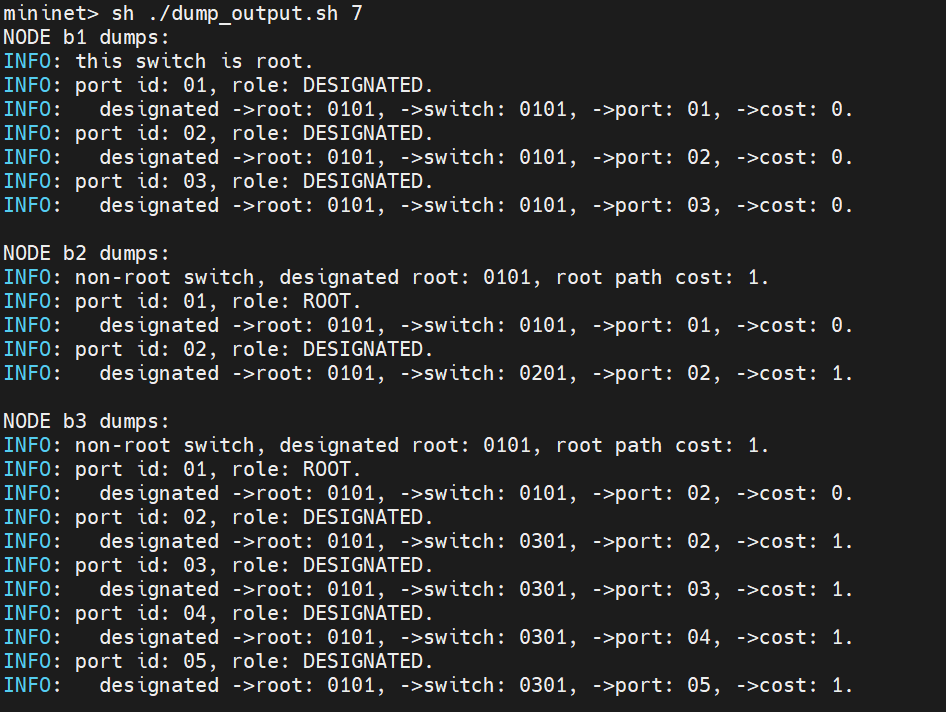
运行脚本，一段时间后观察输出：

Figure b1~b3输出

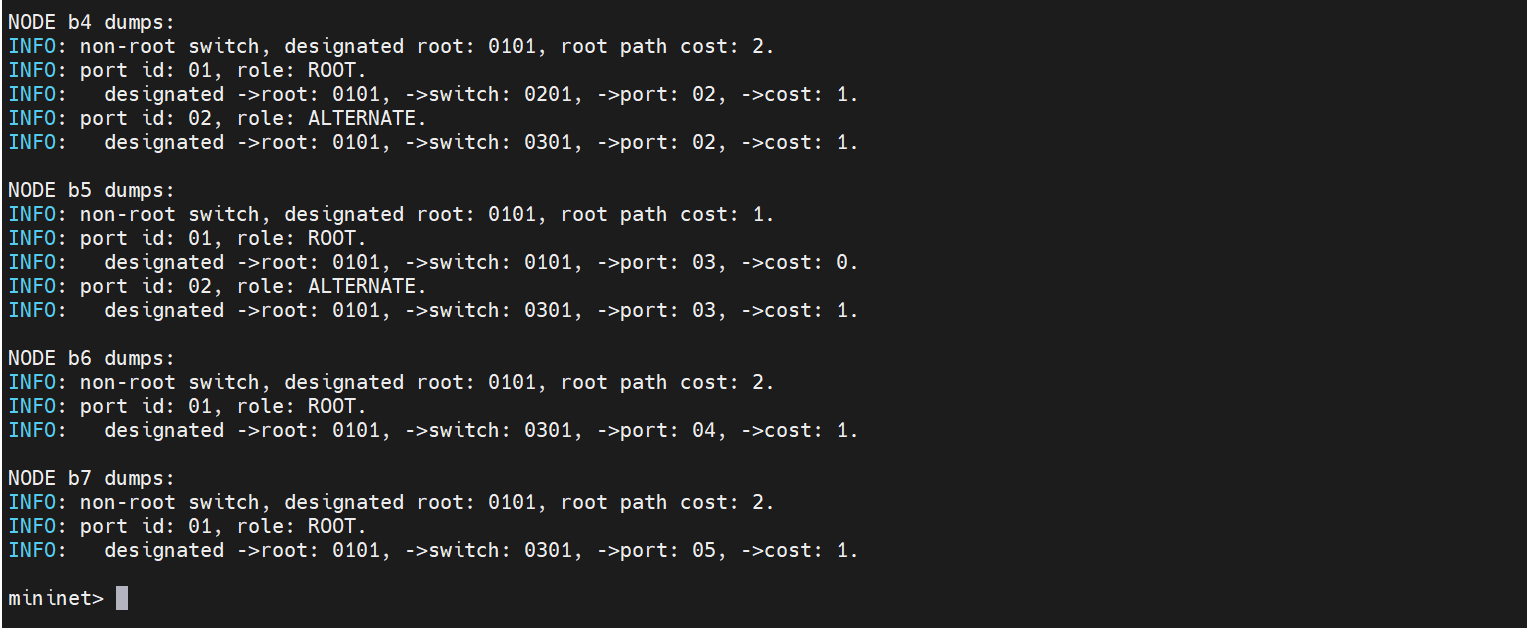


Figure b4~b7输出

输出所展现的拓扑基本符合预期。本被弃用的网段，其一方端口被置为DESIGNATED，另一方为ALTERNATE。

1. 思考

课件中提出了几道思考题，现解答如下：

1）调研说明标准生成树协议中，如何处理网络拓扑变动的情况：当节点加入时？当节点离开时？

在拓扑变化后，总的来说将重新运行stp算法，生成新拓扑结构下的最小生成树。其中有两个问题值得注意：各个结点如何获知拓扑已经发生变动？MAC表项失效问题如何处理？

稳定的生成树中，各个结点间流动着一种来自根桥的TCU(Topology Change Notification) BPDU报文，用于通知拓扑无碍。当某个结点一直收不到TCU BPDU报文时，它就知道要么是根桥失效，要么是它的上游结点失效，此时它可以声明自己是根节点，并开始向外发送TCU BPDU报文。这个报文将会向上转发直到传给根结点，此时根结点也知道了拓扑变化，于是通过拓扑变化通知机制使得拓扑变化的消息在整个网络内泛洪。每个结点获知拓扑变化后，都会加速本地的MAC表老化，使得原本的MAC关系迅速清除。在非根结点发送TCU BPDU报文传递的过程中，同时将运行stp算法，将重新收敛。

2）调研说明标准生成树协议是如何在构建生成树过程中保持网络连通的

在构建生成树期间，并不会直接将某个似乎将要变为冗余端口的端口的状态直接设为Disabled以完全禁用其功能，而是先设为Blocking，不参与数据转发但是侦听BPDU报文。如果通过侦听到的报文该端口发现自己应当被重新启用，那么它将再次参与到生成树中。需要注意的是，端口在Blocking状态并不会停留特别久。

3）实验中的生成树机制效率较低，调研说明快速生成树机制的原理

当网络拓扑发生改变时，要重新运行生成树算法，这需要比较长的时间重新收敛。快速生成树算法RSTP可以加快重新收敛的过程。在RSTP中，稳定状态下每个结点都会以2秒的间隔不断发送BPDU，而不是只有根节点发送BPDU。这样可以使得故障更快被检测到。

RSTP引入新定义：

备份端口和阻塞端口：给指定端口和根端口作备份。

合并状态：新状态包括Discarding，即不转发也不学习MAC地址；Learning，不转发但学习MAC地址；Forwarding，既转发又学习MAC地址。

引入快速收敛机制：

根端口快速切换机制：原根端口失效时，阻塞端口可在秒级时延内切换为根端口并进入转发状态。

次级BPDU立即处理：非指定端口的对端端口失效时，前者可在秒级时延内切换为指定端口，立即发送本地最优config给对端，不依赖计时器。

P/A机制：使一个指定端口尽快开始转发。两个交换机互相发送协商BPDU，经比较优先级后协商完毕，最优交换机对应端口立即进入转发状态。

以上机制能够加速算法的重新收敛。