生成树机制实验

学号：2022K8009929011
姓名：王泽黎

一、实验任务

基于已有代码，实现生成树运行机制，对于给定拓扑(four_node_ring.py)，计算输出相应状态下的最小生成树拓扑

自己构造一个不少于7个节点，冗余链路不少于2条的拓扑，节点和端口的命名规则可参考four_node_ring.py，使用stp程序计算输出最小生成树拓扑

二、实验流程

以four_node_ring.py拓扑为例

1. 运行four_node_ring.py拓扑，4个节点分别运行stp程序，将输出重定向到b*-output.txt文件，以b1为例：

   ```
   b1# ./stp > b1-output.txt 2>&1
   ```

2. 等待一段时间(4个节点大概30秒钟)后，执行如下命令：

   ```
   (b?/root)# pkill -SIGTERM stp
   ```

   该命令强制所有stp程序输出最终状态并退出

   - 可以在xterm或gnome-terminal中执行该命令，需要root权限

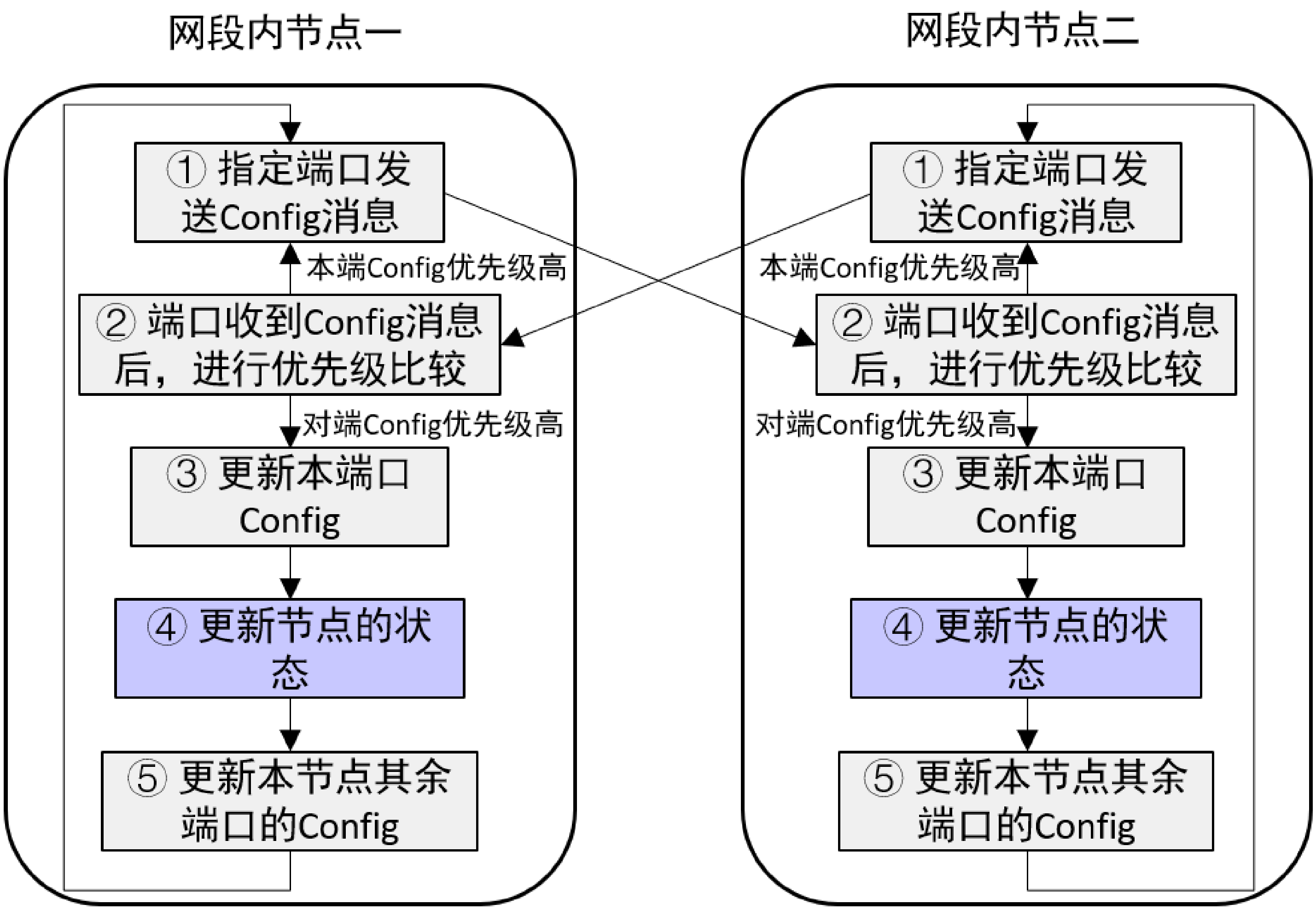
3. 执行dump_output.sh脚本，输出个4个节点的状态

   ```
   # ./dump_output.sh 4
   ```

三、实验结果与分析

（一）实现生成树运行机制

1. 总体逻辑



2. 节点主动发送 Config 消息

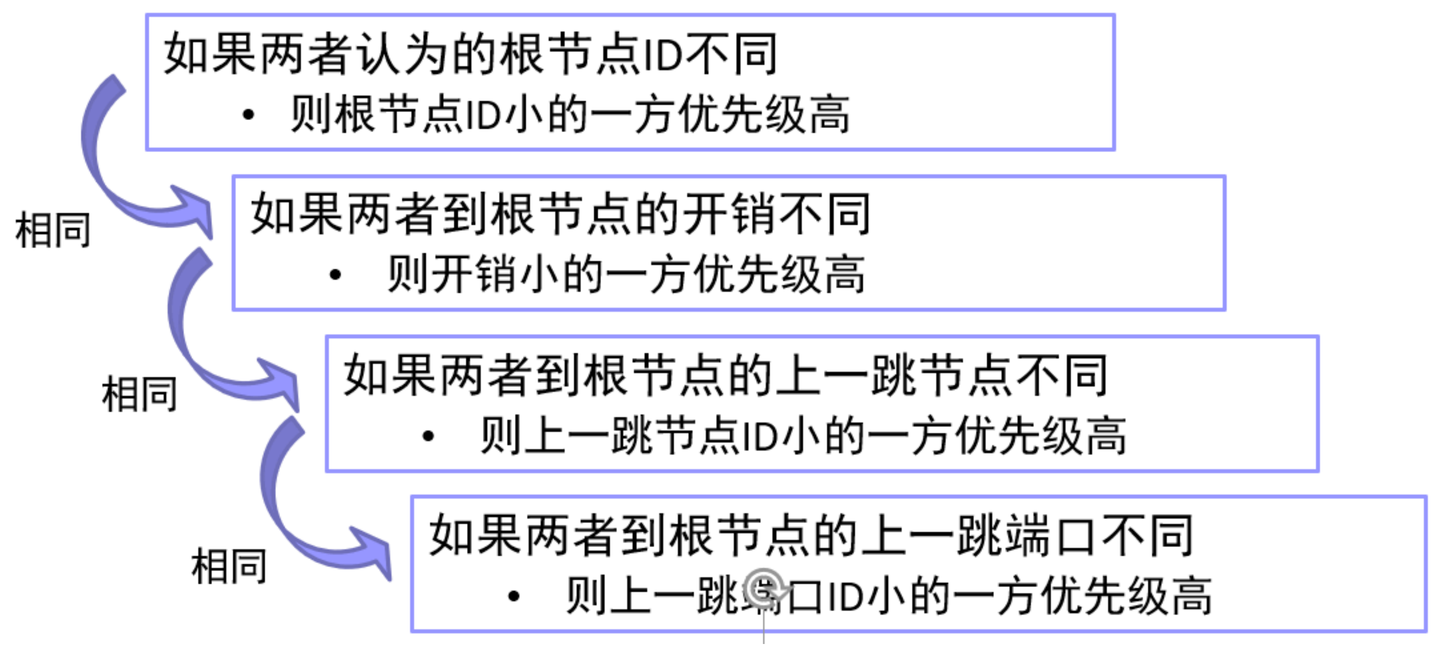
当节点认为自己是根节点时， 周期性主动发送 Config 消息，节点通过 hello 定时器周期发送 Config 消息，直到该节点不再认为自己是根节点为止。

```
//only root switch can always send config
if (is_root_before && !stp_is_root_switch(stp)){
    stp_stop_timer(&stp->hello_timer);
}

//send update config to other stps' port
stp_send_config(stp);
```

3. 处理 Config 消息

（1）Config 优先级比较

当端口收到的 Config 消息之后或节点更新状态，从所有非指定端口中选取根端口时，进行 Config 消息的优先级比较，逻辑如下：



实现代码如下：

```c
static bool config_compare(stp_port_t *p,
                u64 designed_root, u32 root_path_cost,
                u64 switch_id, u16 port_id)
{
    // 比较端口的 designated_root 和传入的 designed_root
    if(p->designated_root < designed_root){
        return true; // 如果端口的 designated_root 更小，返回 true
    }
    else if(p->designated_root > designed_root){
        return false; // 如果端口的 designated_root 更大，返回 false
    }
    else{
        // 如果 designated_root 相等，比较 designated_cost 和传入的 root_path_cost
        if(p->designated_cost < root_path_cost){
            return true; // 如果端口的 designated_cost 更小，返回 true
        }
        else if(p->designated_cost > root_path_cost){
            return false; // 如果端口的 designated_cost 更大，返回 false
        }
        else{
            // 如果 designated_cost 也相等，比较 designated_switch 和传入的 switch_id
            if(p->designated_switch < switch_id){
                return true; // 如果端口的 designated_switch 更小，返回 true
            }
            else if(p->designated_switch > switch_id){
                return false; // 如果端口的 designated_switch 更大，返回 false
            }
            else{
                // 如果 designated_switch 也相等，比较 designated_port 和传入的 port_id
                if(p->designated_port < port_id){
                    return true; // 如果端口的 designated_port 更小，返回 true
                }
                else if(p->designated_port > port_id){
                    return false; // 如果端口的 designated_port 更大，返回 false
                }
                else{
                    return false; // 如果所有字段都相等，返回 false
                }
            }
        }
    }
}
```

（2）更新端口的 Congfig

如果收到的 Config 优先级更高，就把本端口的 Config 替换为收到的 Config 消息，之后再进行后续处理。如果本地 Config 优先级更高，不做更改，若该端口是指定端口，发送 Config 消息，如下所示：

```c
if(config_superior){
    // p is the designated port, send config
    if (stp_port_is_designated(p)) {
        stp_port_send_config(p);
    }
}
else{
    //p's config is replaced by opposite config
    //p will not be designated port in this time
    p->designated_root = designed_root;
    p->designated_cost = root_path_cost;
    p->designated_switch = switch_id;
    p->designated_port = port_id;
```

（3）更新节点状态

遍历所有端口，尝试找到根端口。如果存在根端口，则该节点为非根节点，选择通过root_port 连接到根节点，更新节点认定的根、到根节点的路径开销、根端口。

```c
static stp_port_t *find_root_port(stp_t *stp)
{
    stp_port_t *root_port = NULL;
    stp_port_t *port_entry;

    for (int i = 0; i < stp->nports; i++) {
        port_entry = &stp->ports[i];

        if (!stp_port_is_designated(port_entry)) {
            if(root_port){
                if(config_compare(port_entry, root_port->designated_root, root_port->designated_cost, root_port->designated_switch, root_port->port_id)){
                    root_port = port_entry;
                }
            }
            else{
                root_port = port_entry;
            }
        }
    }

    return root_port;
}
```

```c
stp_port_t *root_port = find_root_port(stp);

if(root_port){
    stp->root_port = root_port;
    stp->designated_root = root_port->designated_root;
    stp->root_path_cost = root_port->designated_cost + root_port->path_cost;
}
else{
    stp->root_port = NULL;
    stp->designated_root = stp->switch_id;
    stp->root_path_cost = 0;
}
```

（4）更新其余端口的 Config

节点在更新自己的状态后，哪些端口的Config需要更新？

- 非指定端口 -> 非指定端口（不需要处理）
- 指定端口 -> 指定端口（需要更新信息，如下）
- 指定端口 -> 非指定端口（只有收到Config时可能，已处理）
- 非指定端口 -> 指定端口（可能，条件如下）

如果一个端口为非指定端口，且其Config较网段内其他端口优先级更高(②)，那么该端口成为指定端口：

- p->designated_switch = stp->switch_id
- p->designated_port = p->port_id

对于所有指定端口，更新其认为的根节点和路径开销：

- p->designated_root = stp->designated_root
- p->designated_cost = stp->root_path_cost

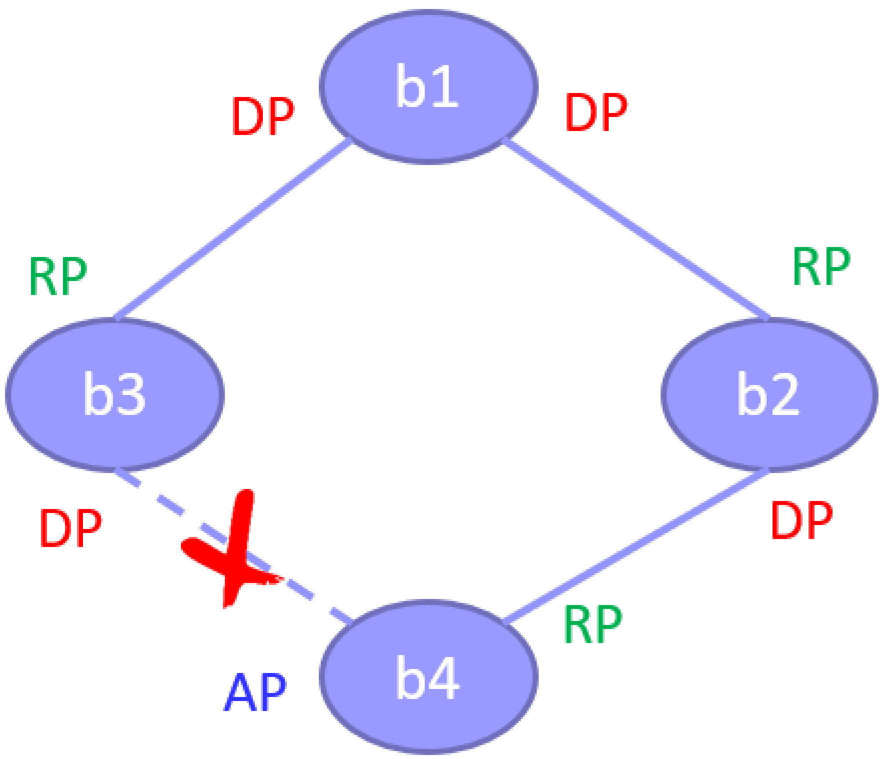
对于非指定端口：

```
for (int i = 0; i < stp->nports; i++) {
    stp_port_t* port_entry = &stp->ports[i];
    if (!stp_port_is_designated(port_entry)) {
        if(root_port){
            if(!config_compare(port_entry, stp->designated_root, stp->root_path_cost, stp->switch_id, port_entry->port_id)){
                port_entry->designated_switch = stp->switch_id;
                port_entry->designated_port = port_entry->port_id;
            }
        }
    }
}
```

对于所有指定端口：

```
for (int i = 0; i < stp->nports; i++) {
    stp_port_t* port_entry = &stp->ports[i];
    if (stp_port_is_designated(port_entry)) {
        port_entry->designated_root = stp->designated_root;
        port_entry->designated_cost = stp->root_path_cost;
    }
}
```

（二）4节点拓扑的最小生成树

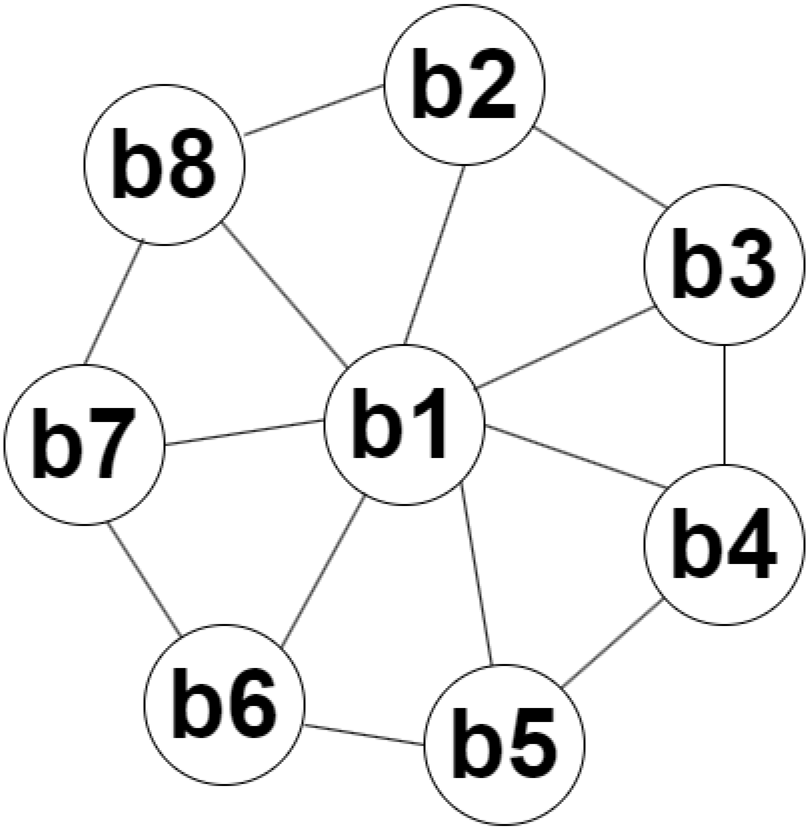


使用stp程序计算输出最小生成树拓扑，实验结果如下：

```
NODE b1 dumps:
INFO: this switch is root.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.

NODE b2 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.

NODE b3 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.

NODE b4 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.
```

可以看到 b1 节点为根节点，其两个端口都为指定端口。b4 节点的2端口为 AP 端口，不参与构建生成树拓扑其他节点和端口也都符合预期的拓扑结构。由此可以看出该生成树算法功能正确。

（三）8节点拓扑的最小生成树



使用stp程序计算输出最小生成树拓扑，实验结果如下：

```
NODE b1 dumps:
INFO: this switch is root.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 03, ->cost: 0.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 04, ->cost: 0.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 05, ->cost: 0.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 06, ->cost: 0.
INFO: port id: 07, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 07, ->cost: 0.

NODE b2 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 03, ->cost: 1.

NODE b3 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->cost: 1.

NODE b4 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 03, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 1.

NODE b5 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 04, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 1.
```

```
NODE b6 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 05, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->cost: 1.

NODE b7 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 06, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 1.

NODE b8 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 07, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 03, ->cost: 1.
```

经过分析，该实验结果满足生成树要求。可见我们的生成树算法功能是正确的。

四、实验总结

通过本次实验，我对生成树的拓扑结构和生成树机制的基本原理有了一定的了解。我学到了如何在网络中实现唯一的、基于优先级的生成树，掌握了处理Config消息的流程。