生成树机制实验

实验内容

- 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程序计算输出生成树拓扑

实验步骤及结果

代码完善

```
//比较本端口和发送端口的config优先级，如果本端口优先级更高返回0,否则返回1
int recv_has_higher_pirority(stp_port_t * p, struct stp_config *config) {
    if (p->designated_root != ntohll(config->root_id)) {
        if(p->designated_root < ntohll(config->root_id)) {
            return 0;
        } else {
            return 1;
        }
    } else if (p->designated_cost != ntohl(config->root_path_cost)) {
        if (p->designated_cost < ntohl(config->root_path_cost)) {
            return 0;
        } else {
            return 1;
        }
    } else if (p->designated_switch != ntohll(config->switch_id)) {
        if (p->designated_switch < ntohll(config->switch_id)) {
            return 0;
        } else {
            return 1;
        }
    } else if (p->designated_port != ntohs(config->port_id)) {
        if (p->designated_port < ntohs(config->port_id)) {
            return 0;
        } else {
            return 1;
        }
    } else {
        return 1;
    }
}

int compare_ports_pirority(stp_port_t * p1, stp_port_t * p2) {
    if (p1->designated_root != p2->designated_root) {
        if(p1->designated_root < p2->designated_root) {
            return 0;
        } else {
            return 1;
        }
    } else if (p1->designated_cost != p2->designated_cost) {
        if (p1->designated_cost < p2->designated_cost) {
```

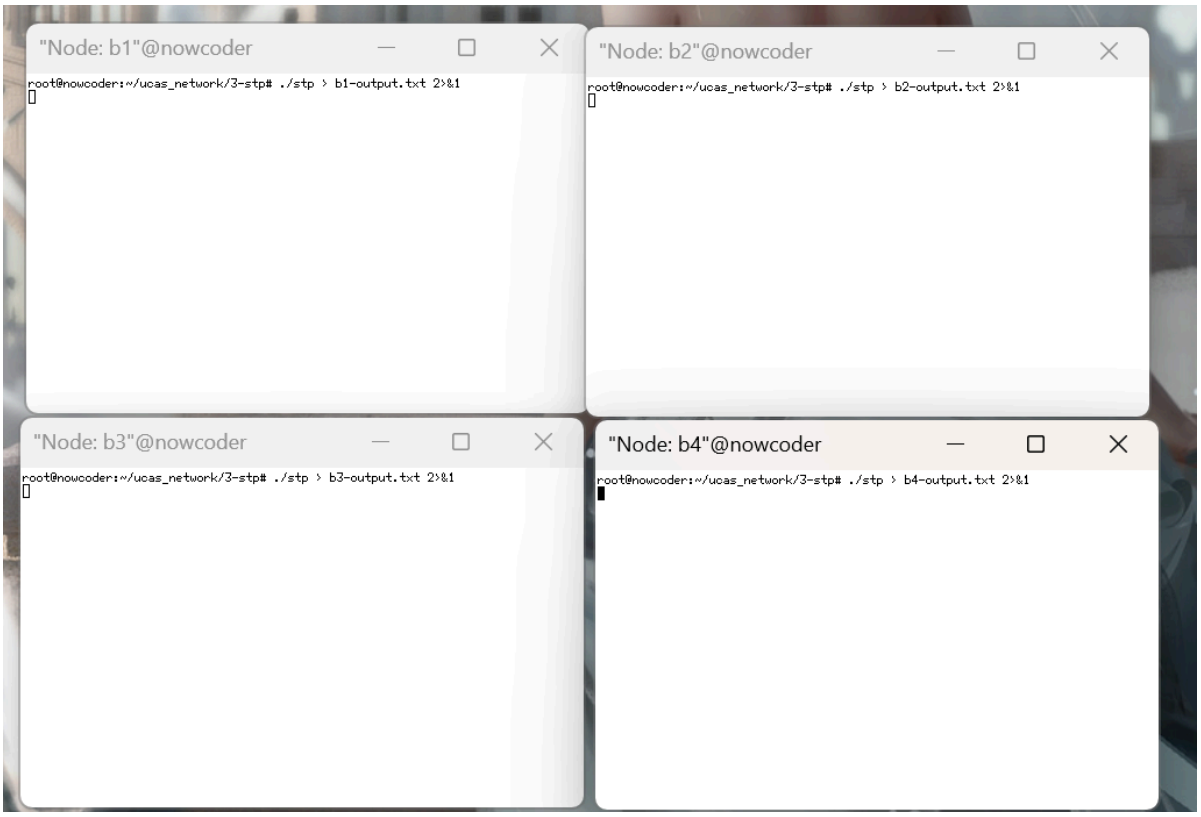
```
            return 0;
        } else {
            return 1;
        }
    } else if (p1->designated_switch != p2->designated_switch) {
        if (p1->designated_switch < p2->designated_switch) {
            return 0;
        } else {
            return 1;
        }
    } else if (p1->designated_port != p2->designated_port) {
        if (p1->designated_port < p2->designated_port) {
            return 0;
        } else {
            return 1;
        }
    } else {
        return 1;
    }
}
//更新本端口的config信息
void update_port_config(stp_port_t *p, struct stp_config *config) {
    p->designated_root = ntohll(config->root_id);
    p->designated_switch = ntohll(config->switch_id);
    p->designated_port = ntohs(config->port_id);
    p->designated_cost = ntohl(config->root_path_cost);
}
//更新根节点
void update_root(stp_t *stp, struct stp_config *config) {
    stp_port_t * non_designated_ports[STP_MAX_PORTS];
    int num_non_ports = 0;
    for (int i =0 ; i < stp->nports; i++) {
        if(!stp_port_is_designated(&stp->ports[i])) {
            non_designated_ports[num_non_ports] = &stp->ports[i];
            num_non_ports ++;
        }
    }
    int judge = 0;
    if (num_non_ports == 1) {
        stp->root_port = non_designated_ports[0];
    } else {
        for(int i = 0; i < num_non_ports -1; i++) {
            if (judge == 0) {
                if(!compare_ports_pirority(non_designated_ports[i],
non_designated_ports[i+1])) {
                    stp->root_port = non_designated_ports[i];
                    judge = 1;
                }
            } else {
                if(!compare_ports_pirority(non_designated_ports[i], stp-
>root_port)) {
                    stp->root_port = non_designated_ports[i];
                }
            }
        }
    }
```

```
    if (stp->root_port == NULL) {
        stp->designated_root = stp->switch_id;
        stp->root_path_cost = 0;
    } else {
        stp->designated_root = stp->root_port->designated_root;
        stp->root_path_cost = stp->root_port->designated_cost + stp->root_port-
>path_cost;
    }
}
//更新其他节点状态
void update_other_ports(stp_t *stp) {
    for (int i =0 ; i < stp->nports; i++) {
        if(stp_port_is_designated(&stp->ports[i])) {
            stp->ports[i].designated_root = stp->designated_root;
            stp->ports[i].designated_cost = stp->root_path_cost;
        }
    }
}

static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
        struct stp_config *config)
{
    // TODO: handle config packet here
    fprintf(stdout, "TODO: handle config packet here.\n");
    if (recv_has_higher_pirority(p, config)) {
        update_port_config(p, config);
        update_root(stp, config);
        update_other_ports(stp);
        if(!stp_is_root_switch(stp)) {
            stp_stop_timer(&stp->hello_timer);
            stp_send_config(stp);
        }
    }
}
```

运行py文件

```
nowcoder@nowcoder:~/ucas_network/3-stp$ sudo python2 four_node_ring.py
mininet> net
b1 b1-eth0:b2-eth0 b1-eth1:b3-eth0
b2 b2-eth0:b1-eth0 b2-eth1:b4-eth0
b3 b3-eth0:b1-eth1 b3-eth1:b4-eth1
b4 b4-eth0:b2-eth1 b4-eth1:b3-eth1
```

4个节点分别运行stp程序，将输出重定向到b*-output.txt文件



结果显示

```
nowcoder@nowcoder:~/ucas_network/3-stp$ sudo ./dump_output.sh 4
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.
```

构造7个节点，2条冗余线路的代码

```python
import os
import sys
import glob

from mininet.topo import Topo
from mininet.net import Mininet
```

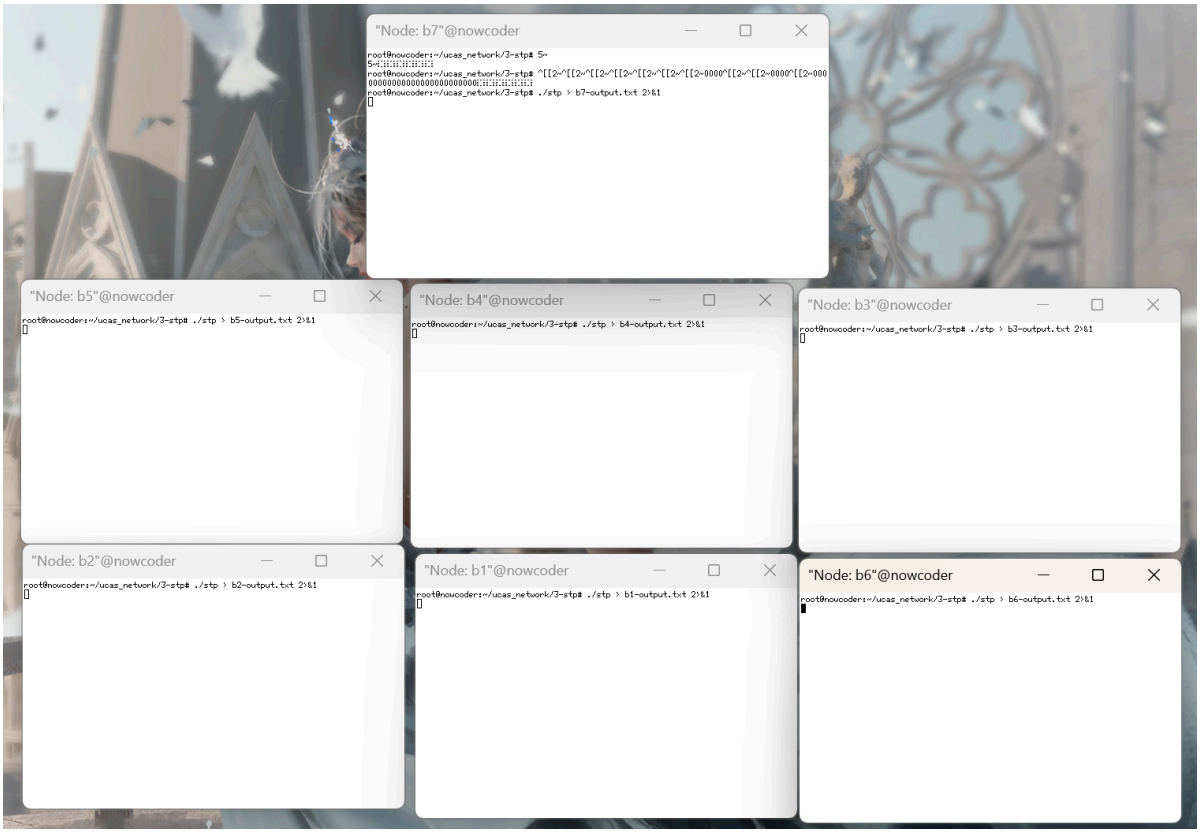
```
from mininet.cli import CLI

script_deps = [ 'ethtool' ]

def check_scripts():
    dir = os.path.abspath(os.path.dirname(sys.argv[0]))

    for fname in glob.glob(dir + '/' + 'scripts/*.sh'):
        if not os.access(fname, os.X_OK):
            print '%s should be set executable by using `chmod +x $script_name`'
% (fname)
            sys.exit(1)

    for program in script_deps:
        found = False
        for path in os.environ['PATH'].split(os.pathsep):
            exe_file = os.path.join(path, program)
            if os.path.isfile(exe_file) and os.access(exe_file, os.X_OK):
                found = True
                break
        if not found:
            print '`%s` is required but missing, which could be installed via
`apt` or `aptitude`' % (program)
            sys.exit(2)

def clearIP(n):
    for iface in n.intfList():
        n.cmd('ifconfig %s 0.0.0.0' % (iface))

class RingTopo(Topo):
    def build(self):
        b1 = self.addHost('b1')
        b2 = self.addHost('b2')
        b3 = self.addHost('b3')
        b4 = self.addHost('b4')
        b5 = self.addHost('b5')
        b6 = self.addHost('b6')
        b7 = self.addHost('b7')

        self.addLink(b1, b2)
        self.addLink(b1, b3)
        self.addLink(b2, b4)
        self.addLink(b3, b5)
        self.addLink(b4, b6)
        self.addLink(b5, b7)
        self.addLink(b6, b7)
        self.addLink(b4, b5)

if __name__ == '__main__':
    check_scripts()

    topo = RingTopo()
    net = Mininet(topo = topo, controller = None)

    for idx in range(7):
        name = 'b' + str(idx+1)
```

```
        node = net.get(name)
        clearIP(node)
        node.cmd('./scripts/disable_offloading.sh')
        node.cmd('./scripts/disable_ipv6.sh')

        # set mac address for each interface
        for port in range(len(node.intfList())):
            intf = '%s-eth%d' % (name, port)
            mac = '00:00:00:00:0%d:0%d' % (idx+1, port+1)

            node.setMAC(mac, intf = intf)

        #node.cmd('./stp > %s-output.txt 2>&1 &' % name)
        node.cmd('./stp-reference > %s-output.txt 2>&1 &' % name)

    net.start()
    CLI(net)
    net.stop()
```

执行py文件

```
nowcoder@nowcoder:~/ucas_network/3-stp$ sudo python2 seven_node_ring.py
mininet> net
b1 b1-eth0:b2-eth0 b1-eth1:b3-eth0
b2 b2-eth0:b1-eth0 b2-eth1:b4-eth0
b3 b3-eth0:b1-eth1 b3-eth1:b5-eth0
b4 b4-eth0:b2-eth1 b4-eth1:b6-eth0 b4-eth2:b5-eth2
b5 b5-eth0:b3-eth1 b5-eth1:b7-eth0 b5-eth2:b4-eth2
b6 b6-eth0:b4-eth1 b6-eth1:b7-eth1
b7 b7-eth0:b5-eth1 b7-eth1:b6-eth1
mininet>
```

7个节点分别运行stp程序，将输出重定向到b*-output.txt文件



结果如下

```
nowcoder@nowcoder:~/ucas_network/3-stp$ sudo ./dump_output.sh 7
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: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 2.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 2.

NODE b5 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ROOT.
```

```
NODE b5 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 2.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 2.

NODE b6 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 2.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 02, ->cost: 3.

NODE b7 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 02, ->cost: 3.
```