Project 2 - Spanning Tree Protocol

In the lectures, you learned about Spanning Trees, which can be used to prevent forwarding loops on a layer 2 network. In this project, you will develop a simplified distributed version of the Spanning Tree Protocol that can be run on an arbitrary layer 2 topology. This project is different from the previous project in that we're not running the simulation using the Mininet environment (Don't worry, Mininet will be back in later projects!). Rather, we will be simulating the communications between switches until they converge on a single solution, and then output the final spanning tree to a file.

Project Setup

Download the project from T-square to the VM and unzip.

Project Layout

In the `Project2` directory, you can see quite a few files. You only need to (and should only) modify one of them, `Switch.py`. It represents a layer 2 switch that implements our simple Spanning Tree Protocol. You will have to implement the functionality described in the lectures and at the links above in the code sections marked with `TODO` comments.

The other files that form the project skeleton are:

- `Topology.py` - Represents a network topology of layer 2 switches. This class reads in the specified topology and arranges it into a data structure that your switch code can access.
- `StpSwitch.py` - A superclass of the class you will edit in `Switch.py`. It abstracts certain implementation details to simplify your tasks.
- `Message.py` - This class represents a simple message format you will use to communicate between switches. The format is similar to what was described in the course lectures.
- `run_spanning_tree.py` - A simple "main" file that loads a topology file (see `XXXTopo.py` below), uses that data to create a Topology object containing Switches, and starts the simulation.
- `XXXTopo.py`, etc - These are topology files that you will pass as input to the `run_spanning_tree.py` file.
- `sample_output.txt` - Example of a valid output file for `Sample.py` as described in the comments in `Switch.py`.
- `ValidateAnswer.py` - Similar to the autograder for this project, it will compare a reference output file against a student's to ensure correctness

Here is an outline of the few sections of code that you will have to complete in `Switch.py`:

1. Decide on the data structure that you will use to keep track of active and/or disabled links. The collection of active links across all switches is the resultant spanning tree.
2. Implement the Spanning Tree Protocol by writing code that sends the initial message to neighbors of the switch, and responds to a message from an immediate neighbor. The comments describe how to send messages to immediate neighbors using the provided superclass.
3. Write a logging function that is specific to your particular data structures. The format is simple, and should output only the links active in spanning tree.

To run your code on a specific topology (SimpleLoopTopo in this case) and output the results to a text file (out.txt in this case), execute the following command:

```
python run_spanning_tree.py SimpleLoopTopo out.txt
```

For this project, you will be able to create as many topologies as you wish and share them on Piazza. We encourage you to share new topologies, and your output files to confirm correctness of your algorithm. We have included several topologies for you to test your code against.

To make sure your code is outputting in the correct format generate output for the Sample.py topology file and validate against the sample_output.txt. Feel free to use this against shared output files as a testing tool as well.

```
python run_spanning_tree.py Sample student_out.txt
```

```
python ValidateAnswer.py -s student_out.txt -r sample_out.txt
```

Key assumptions and clarifications

In order to avoid confusion, there are some assumptions we will make about behaviors of switches in this project:

1. You should assume that all switch IDs are positive integers, and distinct. These integers do not have to be consecutive and they will not always start at 1.
2. Tie breakers: All ties will be broken by lowest switch ID, meaning that if a switch has multiple paths to the root at the same length, it will select the path through the lowest id neighbor. For example, assume switch 5 has two paths to the root, through switch 3 and switch 2. Assume further each path is 2 hops in length, then switch 5 will select switch 2 as the path to the root and disable forwarding on the link to switch 3.

3. Combining points one and two above, there is a single distinct solution spanning tree for each topology.
4. You can assume all switches in the network will be connected to at least one other switch, and all switches are able to reach every other switch.
5. You can assume that there will be no redundant links and there will be only 1 link between each pair of connected switches.
6. You can assume that the topology given at the start will be the final topology and there won't be any changes as your algorithm runs (i.e adding a new switch).
7. Note that when a switch deactivates/blocks a port, this port is not completely discarded. While the switch treats it as inactive, it will still be communicated with during the simulation.

What to turn in

You only need to turn in the file you modify, `Switch.py` to T-Square.

There are some very important guidelines for this file you must follow:

1. **Your submission must terminate!** If your submission runs indefinitely (i.e. contains an infinite loop, or logic errors prevent solution convergence) it will not receive full credit. Termination here is defined as self-termination by the process. Manually killing your submission via console commands or interrupts is not an acceptable means of termination. The simulation stops when there are no messages left to send, at which time the logging function is called for each switch.
2. **Pay close attention to the logging format!** Our autograder expects your logs to be in a very specific format, specified in the comments. We provide you an example of a valid output log as well. Also pay close attention to how the links in the spanning tree should be ordered, separated in your log file, and what information belongs on it's own line. Don't log anything other than what we ask you to!
3. **Remove any print statements from your code before turning it in!** Print statements left in the simulation, particularly for inefficient but logically sound implementations, have drastic effects on run-time. Your submission should take well less than 10 seconds to process a topology. If your leave print statements in your code and they adversely affect the grading process, your work will not receive full credit. Feel free to use print statements during the project for debugging, but please remove them before you submit to T-Square.

Spirit of the Project

The goal of this project is to implement a simplified version of a network protocol using a **distributed** algorithm. This means that your algorithm should be implemented at the network switch level. Each switch only knows its internal state, and the information passed to it via messages from its direct neighbors - the algorithm **must** be based on these messages.

The skeleton code we provide you runs a simulation of the larger network topology, and for the sake of simplicity, the `StpSwitch` class defines a link to the overall topology. This means it is possible using the provided code for one Switch to access another's internal state. This goes against the spirit of the project, and is not permitted. Additional detail is available in the comments of the skeleton code.

Additionally, you are not permitted to change the message passing format. We will not accept modified versions of `Message.py`, nor are you permitted to subclass the `Message` class.

When we grade your code, we will use a special version of the skeleton code that will have a randomly generated variable name for the topology object. If you access it directly in order to generate your spanning tree, your code will throw a runtime error, and receive no credit. In short - do not use *topolink* or *self.topology* objects in your code. Pay careful attention to the code comments. If you have questions about whether your code is accessing data it should not, please ask on Piazza or during office hours!

What you can (and cannot) share

Do **not** share code from `Switch.py` with your fellow students, on Piazza, or publicly in any form. You **may** share log files for any topology, and you may also share any code you write that will *not be turned in*, such as new topologies or other testing code. (It may be a good idea to share a "correct" logs for a particular topology, if you have one, when you share the code for that topology.)

Grading

25 pts	Correct Submission	for turning in all the correct files with the correct names, and significant effort has been made in each file towards completing the project.
30 pts	Provided Topologies	for correct Spanning Tree results (log file) on the provided topologies.

45 pts	Unannounced Topologies	for correct Spanning Tree results (log file) on three topologies that you will not see in advance. They are slightly more complex than the provided ones, and may test for corner cases.

GRADING NOTE: Partial credit is not available for individual topology spanning tree output files. The output spanning tree must be correct to receive credit for that input topology. Additionally, we will be using many topologies to test your project, including but not limited to the topologies we provide.