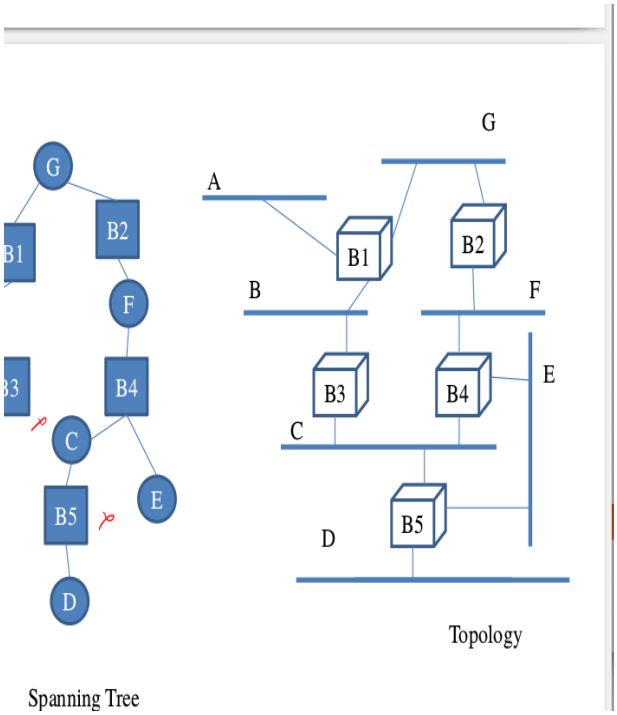
Lab 05 - Spanning Tree Protocol Implementation and Bridge Simulation

In this lab you will implement the spanning tree protocol on a given LAN and bridge topology, and then simulate the functioning of the learning bridges for a sequence of given data transfers.

For example, consider the following LAN topology



This will be specified to you as follows:

1
5
B1: A G B
B2: G F
B3: B C
B4: C F E
B5: C D E

Here, 1 is a trace flag, which if set to 1 should write a detailed trace to stdout, and if set to 0 should produce no trace. 5 specifies the number of bridges whose details will be specified. Each Bridge is then listed in the given syntax showing the LANs to which it is connected directly. You may assume that bridges names will be B1, B2, B3… and LAN names will be single Characters. The bridge list will be specified in order of its ID.

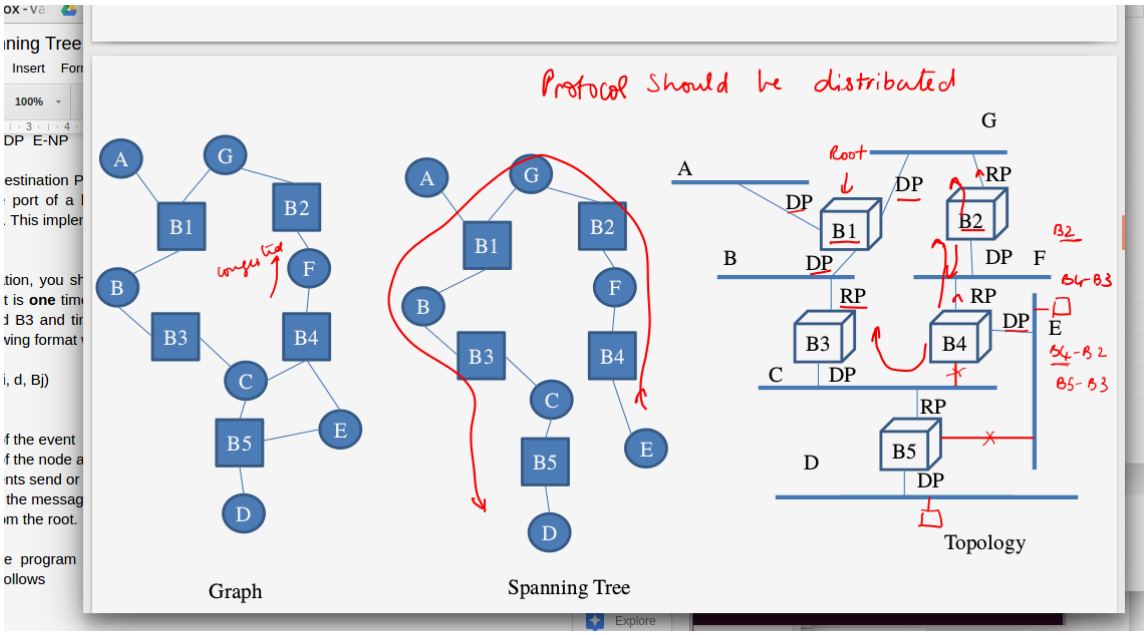
You have to write a program that first reads the above output, creates some internal representation of the LAN topology, and then starts with states of a Bridge's ports as active on all ports. It should then simulate the running of the spanning tree protocol - thus at t=0 all

bridges will send their advertisements and then as time progresses will behave according to the protocol. After the protocol converges, messages will stop, and then your program should output the state of each port as follows:

B1: A-DP B-DP G-DP
B2: F-DP G-RP
B3: B-RP C-DP
B4: C-NP E-DP F-RP
B5: C-RP D-DP E-NP

(Print bridges and port IDs in increasing order)

Here DP = Designated Port, RP = Root Port, and NP = Null Port (deactivated port). Note here that the port of a bridge is simply referred to by the name of the LAN which it is connected to. This implements the spanning tree as shown below.



In the simulation, you should assume that the time required for data transfer across any LAN segment is **one** time unit. E.g., in this topology, a message from B1 sent at time *t* will reach B2 and B3 and time *t+1*. When the trace flag is on, you should produce a trace with the following format while the simulation is going on.

t Bk s|r (Bi, d, Bj)

where
t is the time of the event
Bk is the ID of the node at which the event has happened
s or r represents send or receive event
(Bi, d, Bj) is the message indicating that Bridge Bj thinks Bridge Bi is the root and it is at a distance *d* from the root.

Suggested ordering of trace messages (I will not test the trace message ordering itself in the output):

- Increasing order of time
- If tied on time, increasing order of Bk
- If tied, r first, then s
- If tied, increasing order of Bi
- If tied, increasing order of d
- If tied, increasing order of Bj

After this, the program should read a list of host IDs per LAN and a set of data transfer end points which will be specified as follows

A: H1 H2 H3
B: H4 H5
C: H6 H7 H8
D: H9 H10
E: H11
F: H12 H13
G: H14
3
H9 H2
H4 H12
H3 H9

Explanation of above input:
3: This means that 3 transfers will be specified.
H9 H2 : means Host H9 is sending to Host H2
And so on

You can assume that each host is connected to exactly one LAN segment.

Exact timing of messages: You can assume that the spanning tree protocol has finished convergence before the first data transmission starts. You can also assume that the flooding/routing of the first packet is complete before the second one starts. To be more specific, assume that the network diameter is less than 100, the spanning tree protocol operation starts at t=0, the first message transmission starts at t=100 (by which time the spanning tree protocol could have converged), the second message transmission at t=200, and so on.

Your program should print out the forwarding tables at each bridge after each data transfer, in the following syntax. E.g. after reading H9 H2, it should print out:

B1:
HOST ID | FORWARDING PORT
H9 | B

//Above means that a packet arriving with destination address H9, will be forwarded on the port connected to LAN B. The rest of the output will be:

B2:
HOST ID | FORWARDING PORT
H9 | G
B3:
HOST ID | FORWARDING PORT
H9 | C
B4:
HOST ID | FORWARDING PORT
H9 | F
B5:
HOST ID | FORWARDING PORT
H9 | D

Note: There will be a space between HOST ID, | and Forwarding port
If there are multiple entries in the forwarding table, print in order of increasing host IDs.

If trace flag is set to one, produce the following trace for this simulation (assume again that "crossing" each LAN segment will incur one time unit delay.

t Bk s|r X --> Y

This means at time t, at Bridge Bk, a packet arrived (r) or was sent (s), where the packet source address was on LAN X and packet destination address was on LAN Y.

Suggested ordering of trace messages (I will not test the trace message ordering itself in the output):

- Earliest time first
- If tied, least Bk first
- If tied, r first, then s
- If tied, sort increasing by X
- If tied, sort increasing by Y

Do not simulate other delays (e.g. processing delays at bridge), behaviours (e.g. bridge failures), MAC protocol (e.g. backoff, collisions), etc etc etc etc complications that are not mentioned in this problem statement. The **unit transfer delay is constant.** The bridges do not fail so after convergence of spanning tree your simulation should stop (the root bridge does not need to keep sending configurations). These assumptions are so that the assignment remains simple and doable in 7-8 hours.

This is an individual lab. Ensure that the input and output syntax is STRICTLY followed. Please note we will use auto-grading for functionality check and you WILL lose marks if output syntax is not strictly followed. You may discuss concepts and approaches with friends but every single line of the program should be yours and yours alone. We will run sophisticated plagiarism detectors and even the slightest hint of copying will result in a report to DDAC.

Overall Input for the above example

Your program should read input such as above from STDIN (you can redirect from an input file). The overall file for the above example will contain:

```
0
5
B1: A G B
B2: G F
B3: B C
B4: C F E
B5: C D E
A: H1 H2 H3
B: H4 H5
C: H6 H7 H8
D: H9 H10
E: H11
F: H12 H13
G: H14
3
H9 H2
H4 H12
H3 H9
```

The overall output should be written to STDOUT (you can redirect to a file for testing). For the above example (since trace flag is 0) it will be:

```
B1: A-DP  B-DP  G-DP
B2: F-DP G-RP
B3: B-RP  C-DP
B4: C-NP  E-DP   F-RP
B5: C-RP  D-DP  E-NP
B1:
HOST ID | FORWARDING PORT
H9 | B
B2:
HOST ID | FORWARDING PORT
H9 | G
B3:
HOST ID | FORWARDING PORT
H9 | C
B4:
HOST ID | FORWARDING PORT
H9 | F
B5:
HOST ID | FORWARDING PORT
H9 | D
```

B1:
HOST ID | FORWARDING PORT
H4 | B
H9 | B
B2:
HOST ID | FORWARDING PORT
H4 | G
H9 | G
B3:
HOST ID | FORWARDING PORT
H4 | B
H9 | C
B4:
HOST ID | FORWARDING PORT
H4 | F
H9 | F
B5:
HOST ID | FORWARDING PORT
H4 | C
H9 | D

B1:
HOST ID | FORWARDING PORT
H3 | A
H4 | B
H9 | B
B2:
HOST ID | FORWARDING PORT
H4 | G
H9 | G
B3:
HOST ID | FORWARDING PORT
H3 | B
H4 | B
H9 | C
B4:
HOST ID | FORWARDING PORT
H4 | F
H9 | F
B5:
HOST ID | FORWARDING PORT
H3 | C
H4 | C
H9 | D

Submission files:
Name your main simulator as spanningTree.cpp. You will find it useful to define a class for a bridge, which you can name bridge.h with the implementation of that class in bridge.cpp.

Add details regarding class and method design, and algorithm sketch to a README.txt file. Tar these four files in the format 12345678.tar.gz (where 12345678 is an example roll number) and upload.

We will be using the EvalPro part of BodhiTree for auto-evaluating this lab.