Fall 2003

Programming Project #4 — Network/Chain

CMSC 420
Hanan Samet

The use of computer networks has seen a dramatic rise in recent years. Historically, it is based on the use of the telephone as a communications medium between different computers. The network concept is a solution to the interconnection problem. In greater detail, suppose that 7 people at different locations wish to be able to talk to each other. One solution is to establish a separate direct line from each person to the other. This requires 21 different lines (e.g., Figure 1). Clearly, the more people, the greater the cost. The number of lines (and their length) can also be decreased by insisting that at any instance of time each person can only talk to one other person. In fact, this is how your home telephone works with respect to phone numbers having the same prefix — i.e., if your number is 345-xxxx, then your phone is connected by a line to the 345 switching office as is everyone else with that prefix (e.g., Figure 2). If you have a two-party line, then you and another person share that line to the switching office and thus at any instance of time only one of the two of you can have a conversation with someone else. Notice, that with a single party line we require only 7 lines — one per person going from that person to the switching office.

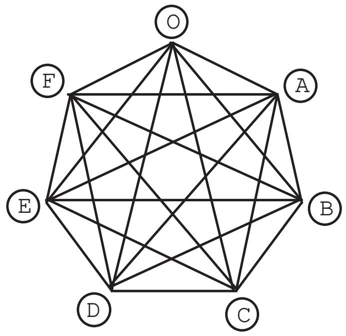

Figure 1: Full network

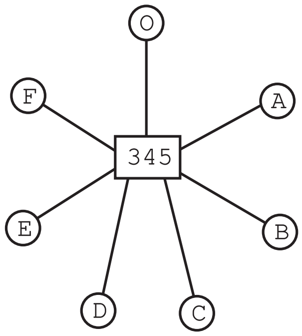

Figure 2: Switched network

In order to be able to talk to people having phone numbers with a different prefix than yours, the phone company dedicates some lines from your switching office to the switching office of the other prefix. This is based on the premise that not everyone with prefix A will want to simultaneously speak to someone with prefix B. For example, see Figure 3 where there are 7 people with prefix 345, 10 with prefix 405, and 3 lines between the two switching offices. Thus if `A`, `B`, `C` with prefix 345 are currently speaking to `D`, `E`, and `F` respectively with prefix 405 and `X`, having prefix 345, wants to call `Y` with prefix 405, then `X` will get a busy signal despite the fact that `Y` is not currently engaged in a conversation with anyone else. This phenomenon is not that frequent since the number of lines between switching offices is generally chosen to be sufficiently high to handle peak traffic. Nevertheless, it is quite common for long distance on days such as Mother's Day.

What we have just described is termed a network. One disadvantage of the telephone network is that once you initiate a conversation with another person, a line is allocated to connect the two of you thereby preventing anybody else from using it. This is termed circuit-switching. In the case of voice communication, this is not so bad because most people are susceptible to "telephonitis" and thus there is always some activity on the line. However, in the case of computer communications this is not the case. In particular, once

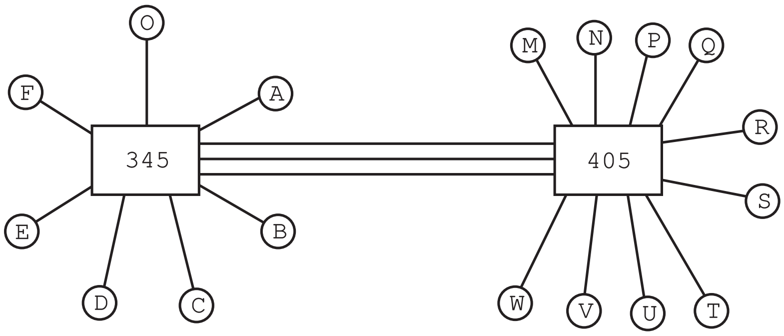

Figure 3: Connected switches

a connection has been established, the ensuing conversation is apt to be characterized by bursts of activity. This is especially true in the case of interactive data processing. Thus what is really needed is something analogous to time-sharing with respect to the lines. This way better use can be made of the network. The concept that has been developed is termed message-switching. Basically, each conversation's data is split up into one or more packets to which an identifying header is attached and it is deposited in the network. The network program makes sure that the message gets to its destination and in the process chooses a path which may involve going through several switching offices. Actually, in the network, each computer may be thought of as a switching office. The important thing to note is that each packet may make use of a different line. Furthermore, once a packet is delivered, the line can be used by someone else. Thus we see that message-switching results in reducing the effect of idleness and thereby leading to a higher utilization of the available lines.

As an example, consider the network consisting of nodes (i.e., sites) `O`, `A`, `B`, `C`, `D`, `E`, and `F` in Figure 4. We have labeled the lines with numbers to signify their costs (analogous to length).

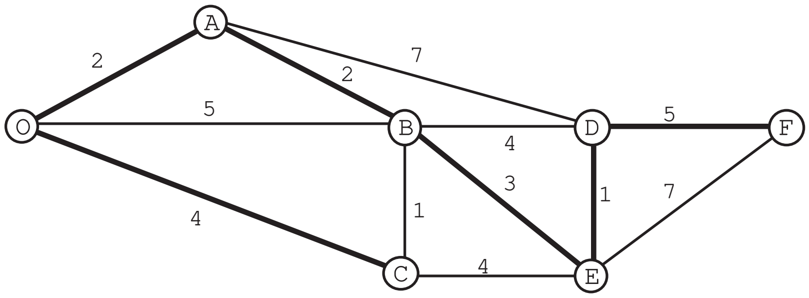

Figure 4: A network

You have just been hired as a network consultant. Your employer has just gotten his communications bill for the previous month and has decreed the situation intolerable — telephonitis is his diagnosis. He wants you to help him reduce his communication costs by getting rid of his excess lines and to insure that all conversations that could have taken place in the past will still be possible. An added constraint is that he wants the cheapest connections from the home office to any other location. Thus you will have to eliminate lines. You will want to reduce the graph, say `G`, corresponding to the network to a tree, say `T`. This tree will have a minimal cost in the sense that given a particular node in the graph, say `x0`, the unique path of `T` which joins `x0` with any other node of `G` is the cheapest path (often called "shortest path" in the literature) joining these two nodes. For example, Figure 5 is the desired tree (assumed to be rooted ad `x0`) corresponding to the network of Figure 4 when `x0` is `O`.

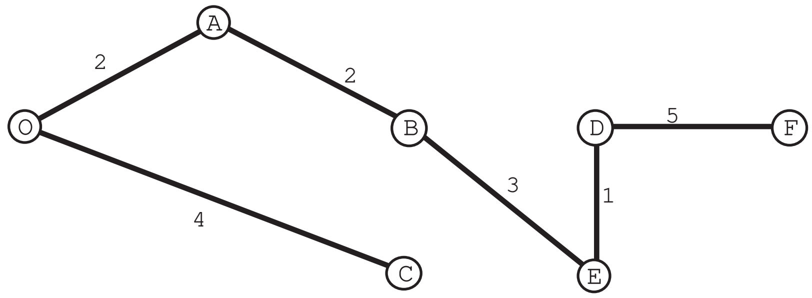

Figure 5: Cheapest paths

An algorithm for obtaining the solution is:

Let `C` = set of connected nodes, i.e., branches have been chosen to connect them
`U` = set of unconnected nodes
`T` = set of edges comprising the desired tree
`c[e]` = the cost of edge `e` (also called weight or length)
`d[x]` = tentative cost of the cheapest path from `x` to `x0` (i.e., the cheapest path from `x` to `x0` where `x` is the only node from `U` on the path)
`n[x]` = for `x` in `U`, the next to last node on the path corresponding to `d[x]`

Initially `C` = {`x0`}
`U` = {all nodes but `x0`}
`T` = $\emptyset$
`d[x0]` = 0
`d[x]` = `c[(x0, x)]`, for all `x` in `U` (∞ if no edge from `x0` to `x`)
`n[x]` = `x0`, for all `x` in `U`

1. Find the node `x` in `U` such that `d[x]` is minimum (this is the node in `U` that has the cheapest path to `x0`)

2. Move `x` from `U` to `C` and add `(n[x], x)` to `T`

3. For each edge `(x, y)` such that `y` is in `U`, check if `d[x]`+`c[(x, y)]` is less than `d[y]`. If it is, set `d[y]` to `d[x]`+`c[(x, y)]` and `n[y]` to `x`. If `U` is empty, then halt; otherwise, go to step 1.

For the example shown we started with node `O` and the tree `T` was built up by choosing the following nodes and edges in sequence:

`O (O,A), A (O,C), C (A,B), B (B,E), E (D,E) or (B,D), D (D,F), F.`

You are to write a LISP program to solve this problem. Choose a suitable representation for the graph and indicate what it is when you turn in your solution. For the more ambitious, a heap implementation can be used to make the search in step 1 efficient. Note, that, optimally, the heap structure also should support efficiently the changes in `d` values that occur in step 3.

Call the main function `cheapest_paths` that takes two arguments. The first one is the starting vertex (denoted `x0` in the description), the second one is a list whose elements are triples which in turn are lists containing the following three items:

```
(node1 node2 number)
```

i.e., there is an edge of cost `number` from `node1` to `node2`. So the function call may look like this:

```
( cheapest_paths 'A '((A B 10) (B C 20) (C A 30)))
```

Your output should consist of a list of edges comprising the desired tree, the cost of the cheapest path from `x0` of the node associated with each edge, and the sum of the path costs. As an example, a correct output corresponding to the cheapest path tree in figure 5 is:

```
((O A 2) (O C 4) (A B 4) (B E 7) (E D 8) (D F 13) 38)
```