Homework 4, CSCE 350, Fall 2016
Due 29 November 2016

Your assignment is to do a simulation of broadcast (from node 0) in various networks.

Example

Consider a 2-d toroid of 16 nodes, that is, a 4 by 4 mesh grid with toroidal wraparound (image further down). If this represents the connections of a network of computers, then one version of the broadcast question is: How many steps does it take to send a message from node 0 to every other node in the graph?

Your program should simulate the sending of these messages and should continue until all nodes have received the message.

Now, in any such problem, there are rules that one assumes to be in place.

- You should assume that the edges are directed. The input files consist of lines of data that are

  ```
  source  dest1 dest2 dest3 ...
  ```

  and I have provided a read function in C++ because reading graphs in can be tedious.

- You must implement some fixed scheduling from any node to the other nodes to which it is directly connected. That is, each node must have a fixed scheduling algorithm and may not rely on any global information (which we would assume would itself take messages and time to obtain). You may assume that the fixed scheduling algorithm is simply that you will send messages to the destinations in the order in which the destinations appear in the input data.

- Each node is permitted to send one message in each clock tick to one destination. You can assume that there is some sort of global clock, in that clock N is the same for all nodes in the network.

- Each node is permitted to receive any number of messages from any number of nodes in any given clock tick. Since we assume that the message is the same regardless of which node is sending it, it doesn't matter that a node gets multiple identical messages in the same tick. (It does affect a record of who sent the message, but that is for this assignment only an informational thing, and the simplest thing is to keep writing on top of the "from whom" variable, so the last write is the one that sticks.)

- You are not allowed to hard code node numbers. You must rely on the graph as input, so you can do multiple graphs of different configurations.

Goal

The goal in such a program would be to be able to input a generic list of nodes and their connections and then to determine the time steps necessary to accomplish broadcast for that network. You should, therefore, be able to output at the end the information about the number of time steps USING YOUR ALGORITHM that were necessary to accomplish broadcast.

You have several input graphs and examples of the output you might get from them.

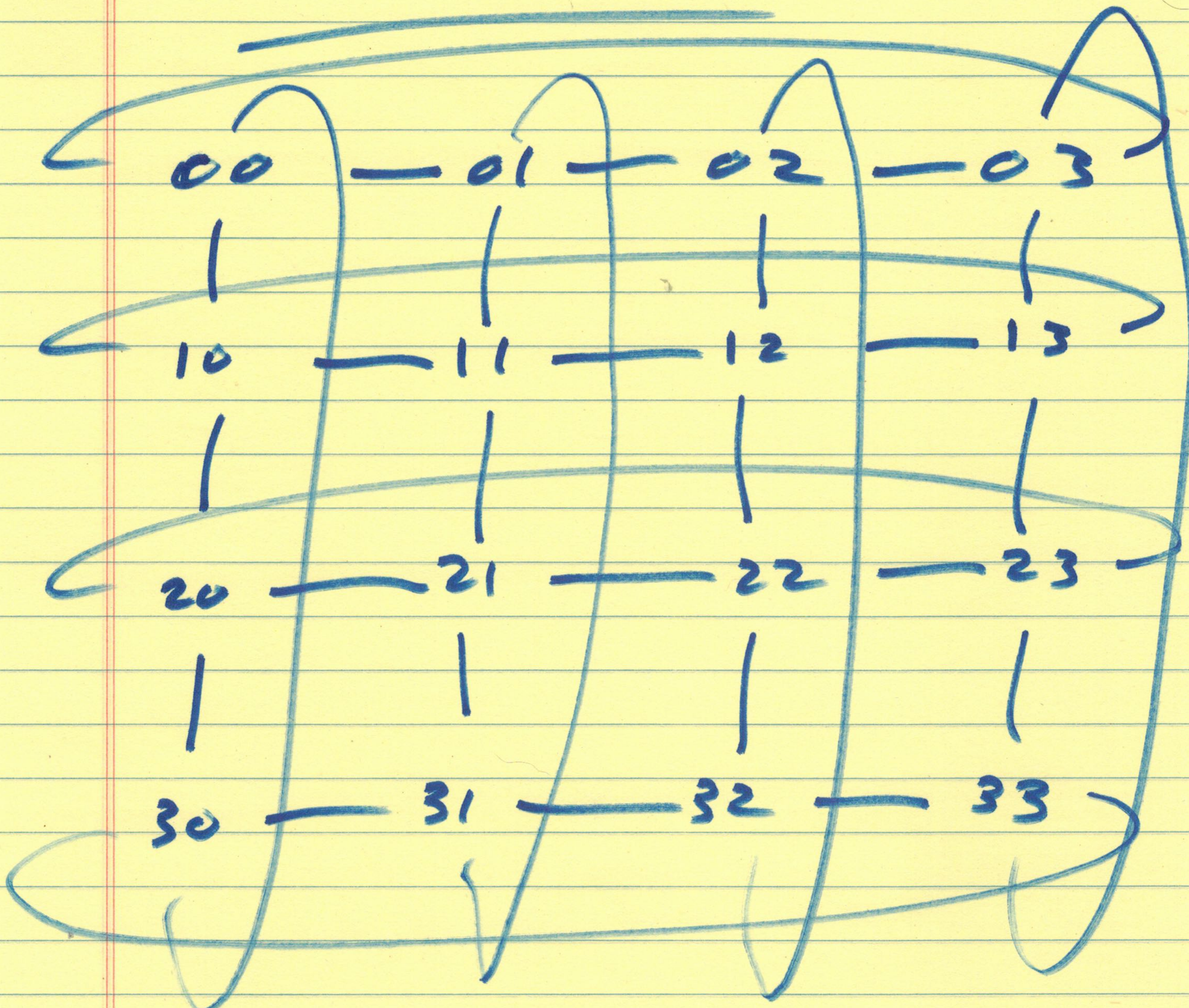
BONUS POINTS–10 points possible

If you get this right, then ...

What happens if you are not the only process in the network? What happens if the network is congested?

Pull in a random number generator and run the program under the assumption that with uniform probability 0.25 a message will not be sent. (i.e., that 25% of the time some other process will seize the single outgoing link and will prevent you from sending.) You will need to adopt a fallback algorithm. Perhaps if the message is to be sent to one link, and the link is busy (according to the random number generator), you go ahead and sent to the next link on the list (if it isn't busy). Or perhaps you just stall and wait till the next tick and try again.

TOROID



path of length $2^n - 1$.

For the sake of convenience, we use the following notations in this paper. For any vertex $u_1u_2\cdots u_n \in V(S_k^n)$ and $i \in [1, n-1]$, *if* $u_{i+1} = u_{i+2} = \cdots = u_j = l$, then we let $u_1 \cdots u_i l^{j-i} u_{j+1} \cdots u_n$ denote $u_1u_2 \cdots u_n$. In particular, if $u_{i+1} = u_{i+2} = \cdots = u_n = l$, then we denote $u_1u_2 \cdots u_i l \cdots l$ by $u_1 \cdots u_i l^{n-i}$ for $i \in [1, n-1]$. Hence extreme vertices $ii \cdots i$ of S_k^n can be denoted by i^n.

Given a vertex $u_1 \cdots u_i l^{j-i} u_{j+1} \cdots u_n$, we consider $u_1 \cdots u_0$ and l^0 to be empty strings. Thus, the edge set of S_k^n can be given by $E(S_k^n) = \{(u_1 \cdots u_{r-1} j i^{n-r},\ u_1 \cdots u_{r-1} i j^{n-r}) : u_1 \cdots u_{r-1} \in [1,k]^{r-1},\ r \in [1, n-1],\ j \neq i;\ i,\ j \in [1,k]\} \cup \{(u_1 \cdots u_n,\ u_1 \cdots u_{n-1} l) : u_1 \cdots u_{n-1} \in [1,k]^{n-1},\ u_1 \cdots u_n \in [1,k]^n,\ l \in [1,k] \setminus \{u_n\}\}$. In fact, the first edge set in the formula above is the set consisting of all bridging edges and the second edge set consists of all edges lying in induced K_k.

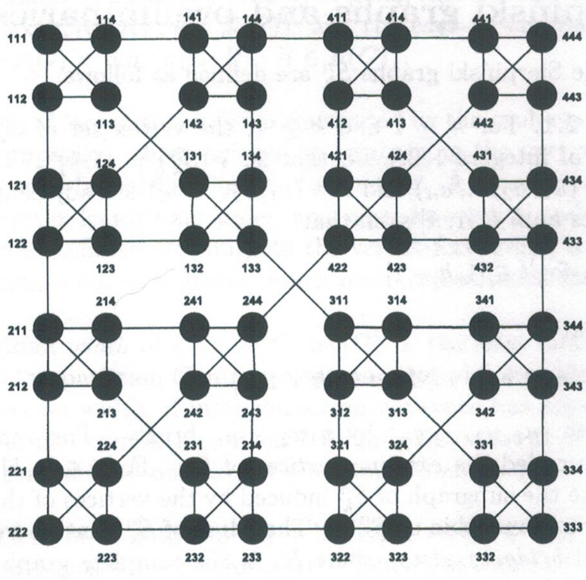


Fig 1: Sierpiński graph S_4^3

Because of the recursive nature of Sierpiński graphs, we can obtain S_k^{n-1} from S_k^n by replacing each K_k of S_k^n by a single vertex and making two such vertices adjacent if their corresponding K_k's are joined by an edge in S_k^n.