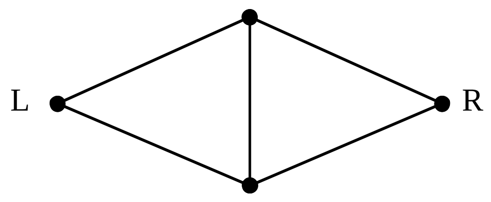
CMPE 177
Applied Graph theory and Algorithms
Summer 2009

Programming Project : Shannon's Switching Game [1]

Email Project Report to TA: Monday July 27
Source Code Submission: Monday August 3, 10:00 pm

Problem Specification:
By *network* we mean a connected graph in which there are two distinguished vertices called *terminals*. We label the terminals L (left) and R (right).

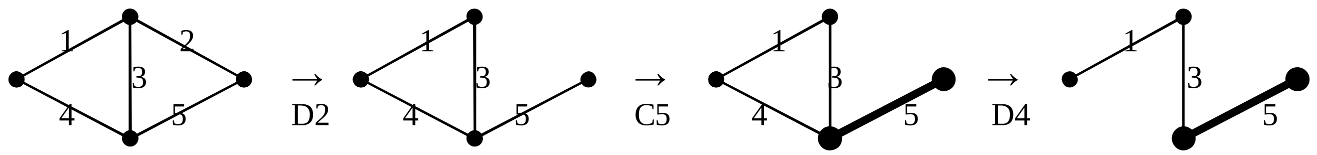


Shannon's Switching Game is played on such a network by two opposing players called *Constructor* (C) and *Destructor* (D). C and D play alternately as follows:

- C chooses an edge and reinforces it.
- D chooses an unreinforced edge and destroys it.

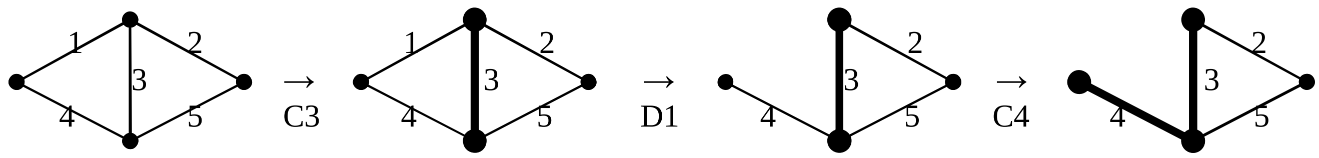
Once an edge has been played on by either player, it can no longer be played upon. The goal of C is to form a path of reinforced edges joining L to R, while D's goal is to prevent this by destroying all paths joining L to R. There are actually two versions of this game: C moves first or D moves first.

Example (D goes first)



At this point C resigns since no matter which edge is reinforced, D will destroy the other, leaving the network disconnected.

Example (C goes first)



D now resigns since whichever edge is destroyed, C will reinforce the other, winning the game.

[1] Thanks to David Huffman

Your task is to write a program which will play Shannon's Switching game on such a network. It should be capable of playing on any network, and in any of the four modes: constructor moving first, constructor moving second, destructor moving first, or destructor moving second. Your executable program will be called ssg and will be invoked on the unix command line as follows:

```
ssg mode filename
```

The first argument will be one of the four strings: `"c1"`, `"c2"`, `"d1"`, or `"d2"`, indicating in which of the above modes your program will play. The second argument will specify a text file giving the network on which the game will be played. Your program will play by simply reading integers from standard input, and writing integers to standard output. These integers will represent the edges on which play is occurring.

For instance if your program is invoked in mode `"c1"` it will read the input file, build an internal representation of the network based on that data, print an integer to standard output indicating the first edge to be reinforced, then read an integer from standard input representing the opponent's move. It will alternately read and write integers in this way until one of the players has won. Your program should know when it has won or lost. If it prints a move which wins the game, it should simply quit. If it receives a winning move from the opponent, it should acknowledge this by printing a zero, then quit. (Zero will not be the label of any edge.)

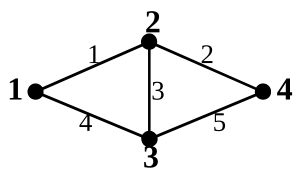
If your program prints an illegal move, such as an integer not corresponding to any edge in the network, or an edge which has already been played upon, or if it prints anything other than an integer to standard output, it will be deemed to have lost. Likewise if your program quits prematurely or core dumps in the middle of play, it will be considered a loss. A limit will be placed on the total amount of time consumed by your program. This limit will be the same for both players, and will depend on the size and complexity of the network. (This is similar to a chess match in which each player is timed while considering a move, then after playing, pushes a button which starts the opponent’s clock.) Thus if your program enters an infinite loop, it will lose as a result of this limit. **Important:** your program *must* flush the output buffer immediately after it writes an integer to standard out. Use `fflush(stdout)` in C/C++, or `System.out.flush()` in Java.

Input File Format
All vertices and edges in a network will be labeled by consecutive integers starting with 1. The first four lines of a network file will each contain a single integer giving the number of nodes *n*, the number of edges *m*, the left terminal *L*, and the right terminal *R*, respectively.

```
n
m
L
R
```

Necessarily $1 \le L \le n$ and $1 \le R \le n$. The remaining *m* lines of the input file will give the end vertices for each edge. Two integers `u v` on a line indicate that an edge joins vertex *u* to vertex *v* ($1 \le u \le n$ and $1 \le v \le n$.) The label of an edge is implied by the position in the file at which this line occurs. In other words, if the two integers appear on line $i+4$ of the input file ($1 \le i \le m$), then the corresponding edge has label *i*. For instance the following text file would represent the network in the preceding examples.

Network:



Input File:

```
4
5
1
4
1 2
2 4
2 3
1 3
3 4
```

It is important to remember that edge labels are given implicitly by the order in which the edges are listed in the input file. Proper play of the game depends on your program having the edge labels correct. You may assume that the input file will represent a simple connected graph with at most 2000 vertices and at most 5000 edges. A number of sample input files will be placed in the course locker (/afs/cats.ucsc.edu/courses/cmpe177-pt) for testing purposes, along with a perl script with which you can generate your own random input files.

A control program called `referee` will also be placed in the course locker, which will enable you to automate play of your program. You can play your program against other programs, against itself, or against a simple program which merely picks edges at random.

Project Requirements
Your project should be written in C, C++, or Java, and should compile and run without warnings or errors on the Instructional Computing (IC) Solaris environment (unix.ic.ucsc.edu). The compilers available on this system are

C compilers: `cc`, `gcc`
C++ compilers: `CC`, `g++`
Java compiler: `javac`

If you develop your project on a different platform, be sure to test it thoroughly on IC-Solaris before turning it in, since that is where it will be evaluated.

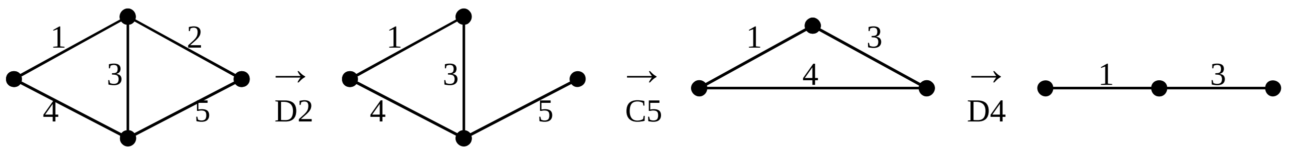
Submit your source files and a `Makefile` which compiles your code and generates an executable called `ssg`. Your `Makefile` should also include a `clean` utility. If you are working in Java it will be necessary to place your code in an executable jar file so that it is not necessary to type `java` at the command line. In any case one should be able to type `gmake` to compile your code, `ssg` to run your program, and `gmake clean` to delete all binaries created during compilation. Be sure to submit only your source code for this project. Do not submit a compiled program or extra files of any kind. To submit your program type:

```
submit cmpe177-pt.u09 project filename filename ... filename
```

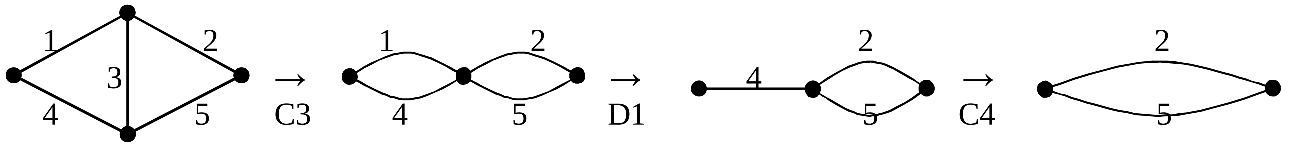
1. **Project Report: Monday July 27 (email to TA).** Explain the algorithms and strategies you used in your program. Describe your algorithms in detail, include pseudo-code and an outline of your program. Analyze its asymptotic space and time complexity where appropriate. This complexity should be polynomial in *n* and *m* (the number of vertices and edges).
2. **Electronic Submission: Monday August 3, 10:00 pm.** Submit your source code (.c, .cc, .h, or .java files), along with a `Makefile` which creates the executable `ssg` described above. Also submit a brief README file listing all the files submitted and their function in your project, as well as any notes or comments.

Evaluation
In order to get a C or higher on this project, your program must at minimum compile without warnings, run properly (i.e. provide only legal moves, not crash or quit prematurely, and not enter an infinite loop), and be able to consistently win against a program which does nothing more than select edges at random. If your program meets these minimum standards, it will participate in a tournament which will determine it’s grade, with the best program receiving the highest grade. Note that if you do not turn in the Project Report, 10 points (one full grade) will be deducted from your score.

Remarks on Strategy
An equivalent way to define the constructor’s move is as *contraction* of an edge: remove the edge and identify its two end vertices. (After an edge is reinforced, there is a permanent path joining its endpoints, so they may as well be identified.) Under this interpretation of C's move, the two previous examples of play would look like:



and



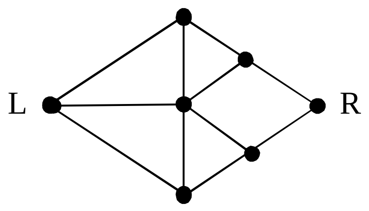
Note that after each move, the resulting network has one less edge, and either the same number of vertices (if D moves) or one less vertex (if C moves). Observe that when we represent C's move as a contraction, loops and parallel edges may be created during play. Note that any loop in our network is irrelevant to the play, since contracting and deleting a loop are the same operation. Equivalently, there is no advantage to either player in reinforcing or destroying a loop. Even though the network will be a simple graph, this alternate definition of C’s action suggests that it may be useful for your program to maintain an internal graph representation which allows for multiple edges and/or loops.

It can be proved that every network can be classified as either

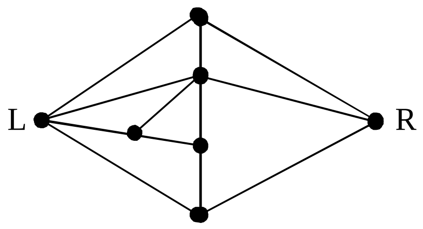
1. Favorable to the first player,
2. Favorable to C, no matter who plays first, or
3. Favorable to D, no matter who plays first.

By favorable we mean that the indicated player can force a win. This follows from some facts in game theory. Shannon's switching game is a game of *perfect information*, which means that each player is completely informed, at all times, of the full history of play (like chess, checkers, and backgammon, unlike bridge and poker.) Such games are known to have solutions in pure strategies, which means essentially that there is a single 'correct' strategy for each player.

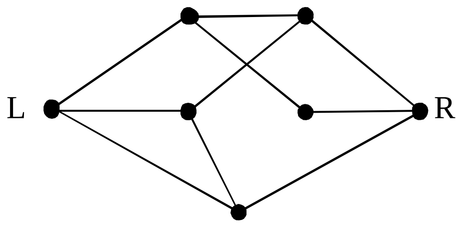
Exercise Determine which player (first, C, or D) has the advantage in the following networks:

(a) 

(Answer: first player)

(b) 

(Answer: C)

(c) 

(Answer: D)

Spend some time exploring examples such as above to increase your understanding of the problem. You should strive for a program design which is flexible enough to allow for changes as you refine and improve your strategies. Another handout on strategy for Shannon's switching game will be posted on the webpage.