ECS 60 Programming Assignment #5, 50 points (<2 hours) Spring 2014

Due: Tuesday, June 3rd at 11:59pm to p5 directory
Primary class file names: router.cpp, router.h Executable name: router.out
Makefile name: Makefile

An Internet streaming company asked a programmer to determine the maximum number of megabytes that can be transferred from Los Angeles to New York City at one time. She is given a mapping of routers with their bandwidths (in megabytes) and connections. You are to write a program that determines the maximum number of megabytes.

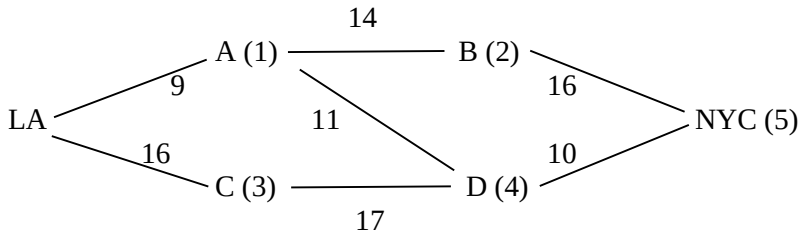
You will find CreateFile.out, a barebones router.h, a barebones router.cpp, routeRunner.h, routeRunner.cpp, CPUTimer.h, a basic Makefile, my router.out, CreateFile.cpp, and route files in ~ssdavis/60/p5.

Here are the specifications:

1. Connection Matrix
   - 1.1. Your Router class constructor will be given a matrix of routers.
   - 1.2. The matrix will have the bandwidths between every pair of routers.
     - 1.2.1. The bandwidths are based on the populations in countyDist3200.csv. The file is sorted by population, with Los Angeles first, and New York City second.
     - 1.2.2. You may assume that the bandwidth is the same for both directions. For example, if the router connecting counties A and B has a bandwidth of 40 megabytes, then 40 megabytes can be carried from A to B, and 40 megabytes can be carried from B to A at the same time.
   - 1.3. Each router will only be directly connected to up to twenty other routers.
     - 1.3.1. Most of the connections are made to nearby routers. countyDist3200.csv contains the distances between the 3200 routers.
     - 1.3.2. Beside connections to nearby routers, there are up to 14 trunk lines that connect distant routers.
   - 1.4. The first router in the matrix is always Los Angeles.
   - 1.5. The position of New York City is provided as a parameter to the solve() function.
   - 1.6. All routers are guaranteed to be connected to the Internet.
2. router.out takes as its parameter of the name of a route file.
3. Route files
   - 3.1. Names are route-<number of routers-<seed>.csv
   - 3.2. The first line has <number of routers>, <index of NYC>, <solution>
   - 3.3. The remaining lines provide information about each router starting with router [0], Los Angeles, and continuing sequentially. The information is a list of pairs of number: first a router number, and then the bandwidth of the connection to the current router.
4. Grading
   - 4.1. Performance will be tested with three route files with 3200 routers.
   - 4.2. I will copy routesRunner.h, routesRunner.cpp, and CPUTimer.h into your handin directory, and then call make, so please make sure you handin all necessary files. Students often forget to handin dsexceptions.h, as well as other secondary Weiss files. "handin cs60 p5 *.cpp *.h Makefile" would probably be wise after a successful clean remake.
   - 4.3. (25 points) If your program provides the proper number of megabytes, then it will earn 25 points for proper operation.
   - 4.4. (25 points) CPU time: min (30, 25 * Sean's Total CPU Time / Your Total CPU Time).
     - 4.4.1. CPU time may not exceed 100 for one file.
     - 4.4.2. Programs must be compiled without any optimization options. You may not use any precompiled code, including the STL and assembly.
     - 4.4.3. You may not have any significant static, or global variables since they would be created before the CPU timer begins. Note that you may have constants, and may load information from your own data files in your constructor.
5. Suggestions
   - 5.1. Keep things simple, and get things running first, and only then use gprof to learn where things are going slowly.
   - 5.2. Use Weiss code where possible. His files are good starting points for any ADT you wish to use.
   - 5.3. Remember to put the authors' names on the first line of router.h, and to turn in dsexceptions.h if your program needs it!

```
int main(int argc, char** argv) {
  int numRouters, solution, NYCPos, totalFlow;
  double startTime, endTime;
  int **capacities = readFile(argv[1], &numRouters, &solution, &NYCPos);
  startTime = getCPUTime();
  Router *router = new Router((const int**)capacities, numRouters);

  for(int i = 0; i < numRouters; i++)
    delete[] capacities[i];

  totalFlow = router->solve(NYCPos);
  endTime = getCPUTime();

  if(totalFlow != solution)
    cout << "Yours: " << totalFlow << " Solution: "  << solution << endl;

  cout << "CPU time: " << endTime - startTime << endl;
  return 0;
} // main())
```

Unlike normal route files, I created route-6-special.csv to reflect a graph like the one in class. It requires the use of a backflow (A to D) to find the solution.




```
[ssdavis@lect1 p5]$ cat route-6-special.csv
6,5,24
1,9,3,16,
0,9,2,14,4,11,
1,14,5,16,
0,16,4,17,
1,11,3,17,5,10,
2,16,4,10,
[ssdavis@lect1 p5]$ router.out route-6-special.csv
CPU time: 3.5884e-05
[ssdavis@lect1 p5]$ router.out route-3200-15.csv
CPU time: 0.571028
[ssdavis@lect1 p5]$
```

One final note: The solutions provided in the route files are based on my own router.out. Though I did do pencil and paper checking of my solutions for small files, it is not a guarantee that my program is correct for larger files. We will have to wait until some students have a finished program to be sure that my solutions are correct.