CS4115 Week11 Lab Exercise

Lab Objective: In the past few weeks of labs we have been using sparse matrices to keep track of those entries of a matrix that are non-zero. I have argued that for lots of situations saving just the non-zero entries turns out to be a big win, even if it makes our coding lives more unpleasant temporarily.

In last week's lab we wrote a program, `mmult`, that multiplied a matrix, A, by itself to compute A^2. In this final lab we will consider how we can generalise this. That is, our task for this week is to write a program, `mpow`, to raise matrices to some given power.

Here's the lab summary:

❶ The program you write should, like last week, read a `formatz` matrix from standard input, and should compute the matrix raised to a given power. Read below for further details.

❷ Understand the significance of the task you are being asked to complete. Even though you will be able to complete the lab without reading any of this it will help you to understand *why* youa are doing it and also it will help you to understand better graphs and graph algorithms.

❸ Submit your `mpow` program for marking using `handin`.

In Detail

❶ You might want to use the released code to last week's lab as a starting point for this week's. Even if you don't want to use this provided code please have a look at it to compare how your problem decomposition into C++ functions compares with this.

We still need to "tell" our program what exponent (power) to which we will raise A, the matrix we have read in. Although we could insist that another line appear in the input that holds the single integer exponent I think a better design choice is to specify the exponent from the command line; it's easier too. So this means that you will need to modify your command-line processing code so that it must be given an exponent. It is an error if you are not.

The program would be run as follows:

```
mpow 3 < m5.formatz
```

Your program should issue an error message if the exponent read is not an integer or if it is less than 0. Please use my sample executable as a guide for the exact error message to report back. Can you make your program respond to the case of 0?

Your program should run as efficiently as you can make it.

❷ Why should we care about computing powers of matrices?[1]

Matrices are used to keep track of interconnections. A matrix can keep track of what web pages link to what other web pages and a matrix can represent a road network by storing what towns have a road connection to other towns. In the street network data we saw before, each of the rows in the *.graph* files represents the geographical points on a map that are immediately connected to each known point. Using the information in the corresponding *.graph.xyz* file you could find out how far apart geographically these points are to each other.

As a final example, in a computer networking situation where packets must be routed all over the world, we could build a matrix of what routers are connected to what other routers by direct transmission links. So in this case, forgetting about sparse matrices and issues of space and all of that, the boolean entry of A at row i, column j tells us whether router i is connected to router j.

Routing algorithms often need to keep track of a packet's *hop count*. This is the number of times that a packet has been forwarded from one computer to another, and it is stored as part of the packet itself. It gets updated by one each time the packet goes through another router. If a packet's hop count goes above a certain value it is usually discarded as being lost and too far from home. So an important question, then, when attempting to route a packet is whether it can get to its destination before the hop count reaches its upper limit.

A lot of information about hop count can be found from looking at powers of the initial adjacency matrix, A. If $A_{i,j} = 1$ then there is a one-hop connection between router i and router j. What happens when we square A? The entry $A_{i,j}$ of A^2 is found by doing a dot product of row i and column j. This dot product will be contributed to if there is a k so that $A_{i,k} = A_{k,j} = 1$. In terms of the network, this translates to a connection between $i--k$ *and* a connection between $k--j$. This is a hop count of 2: i to k to j. The value of the dot product is the number of such ks that are found and this is $A^2_{i,j}$ the entry i, j of A^2.

Similarly, $A^3_{i,j}$ gives us the number of 3-hop paths from i to j. If the most number of hops that a network will allow is h then you would certainly want $A^h_{i,j} > 0$ if you wanted to get a packet from i to j.

This gives us an important requirement for a viable network and a reason for looking at powers of adjacency matrices. If a network has adjacency matrix A then A^h should have no zero entries because otherwise there will be a pair of nodes who can never talk.

❸ The command to submit your program is

```
handin -m cs4115 -p w11
```

[1] Never do anything if you don't know why you're doing it. Put another way, never trust anybody. However, if you *are* the trusting type who just goes along with things and have a "whatever" outlook on life then you can save yourself the 10 minutes it would have taken to read and understand the significance of what you are being asked to do. End of (current) patronising footnote.

Labs that are to be handed in are open for handing in at 09.00 on Friday of the week it is *assigned* and, in order to get full marks, are due by 09.00 on Friday of the *following* week; a lateness penalty applies to submissions made until 18.00, Monday after that. This gives you one week to work on each lab that is to be assessed and handin without penalty. The lab sheet will be available for reading earlier in the week so that you can read it beforehand and come to the lab with questions.

Subject to last-minute changes, the planned schedule of lab assignments due for handing in are:

Lab. Week	Assessed	DueDate
Week02	✗	
Week03	✓	Fri, Week04
Week04	✗	
Week05	✗	
Week06	✓	Fri, Week07
Week07	✓	Fri, Week08
Week08	✗	
Week09	✓	Fri, Week10
Week10	✗	
Week11	✓	Fri, Week12