The first page of your homework submission must be a cover sheet answering the following questions. Do not leave it until the last minute; it's fine to fill out the cover sheet before you have completely finished the assignment. Assignments submitted without a cover sheet, or with a cover sheet obviously dashed off without much thought at the last minute, will not be graded.

- How many hours would you estimate that you spent on this assignment?

- Explain (in one or two sentences) one thing you learned through doing this assignment.

- What is one thing you think you need to review or study more? What do you plan to do about it?

Question 1. Consider the family of undirected graphs $\mathcal{H}_k$ defined as follows. $\mathcal{H}_k$ has 2^k vertices labelled with the integers 0 through $2^k - 1$. Vertices u and v are connected by an edge if and only if the binary representations of u and v differ in exactly one bit position. For example, in $\mathcal{H}_4$, the vertices 5 and 13 are connected by an edge since $5 = 0101_2$ and $13 = 1101_2$ differ in the first bit position, but the rest of the bits are the same.

Consider doing a BFS in $\mathcal{H}_{10}$ starting at node 0. How many vertices are in L_6, that is, the sixth layer generated by the BFS? Give your answer together with either a proof, or the program you used to calculate the answer. Either approach will receive full credit. (*Hint* if you choose to write a program: to flip the `j`th bit of an integer `n`, you can use `n ^ (1 << j)`, that is, the bitwise XOR of `n` with the result of shifting 1 left `j` times, that is, $2^{\mathtt{j}}$. These operators are valid in many languages such as Java, Python, and C/C++.)

Question 2. On the course website you will find a file called `graph-F24.txt` which describes a large undirected graph. The first line of the file contains a single integer which is the number of edges in the graph. Each subsequent line of the file describes one (undirected) edge, and contains two space-separated strings which are the names of the two vertices at the endpoints of the edge.

Write a program (in a programming language of your choice) to find a shortest path from the vertex labelled with your name to the vertex labelled `END` (if there are multiple shortest paths you can find any one of them). You should submit a text file containing the list of vertices along this shortest path, starting with your name and ending with `END`. Each vertex should be on a separate line. For example, my solution looks like this:

To avoid any ambiguity, the list of names I used was as follows: Caden, Colten, Fin, Ian, JP, Jack, Jacob, Jake, Jory, Kate, KB, Kolya, Logan, Mason, Miguel, Nard, Noah, Thomas, Tucker.

```
Brent
dzpm0l
j2s56i
719yay
klwghv
v1sol4
zhyvxg
END
```

You should also turn in the code you used to find your path.

Question 3. In class, we saw the definition of an undirected graph which is *connected*: a graph is connected if there is always a path between any two vertices. When we add directions to the edges, the concept of connectedness becomes more interesting.

A directed graph is *strongly connected* if for any two vertices x and y, there is always a *directed path* from x to y. Note this must also be true for the vertices y and x, so there must be a directed path from y to x as well. In other words, a strongly connected graph is like a network of one-way streets where it is always possible to drive from any location to any other location while obeying the one-way signs; you can never get stuck.

This problem will walk you through the process of developing an algorithm for determining whether a directed graph is strongly connected.

(a) Give an example of a strongly connected graph with at least 4 vertices (you may either draw it, or list its vertices and edges).

(b) Prove: a directed graph G is strongly connected if and only if there is some vertex s such that s is mutually connected to every other vertex; that is, for every other vertex v there is both a directed path from s to v and also from v back to s.

(c) Explain how to use a graph search algorithm like DFS or BFS to determine whether there exist directed paths from a particular starting vertex s to every other vertex.

(d) Explain how to determine, in only $O(V + E)$ time, whether there exist directed paths *to* a particular vertex s *from* every other vertex.

3(d)

(e) Put all these observations together into an algorithm to determine whether a given directed graph is strongly connected. You should:

1. Describe your algorithm. You may use pseudocode, but your description should be detailed enough that someone could take it and turn it into working code.
2. Prove/justify the correctness of your algorithm. In other words, why does your algorithm correctly determine whether a graph is strongly connected? Of course, you may freely cite the results from the previous parts of this problem.
3. Analyze the asymptotic running time of your algorithm.