Note

I will use "lower level" to mean a level that is higher up in the BFS tree (root node is level 0), and "higher level" to mean a level that is lower down in the BFS tree (root node is level 0). In other words, "lower" and "higher" refer to the numerical number we assign to a level, where the root node is assigned level 0 and so on.

Problem 1

(a) The minimum possible value that $\alpha(G, s)$ can have is $|V| - 1$. The graphs that yield this are those graphs that just have one root/source node with only leaf nodes (nodes with no child nodes) and complete graphs. For the former, for the α value to be minimized, the BFS must be done from the node I specified as root/source. The maximum possible value that $\alpha(G, s)$ can have is $\sum_{i=1}^{|V|-1} i = \frac{|V|(|V|-1)}{2}$. The graphs that yield this are those graphs that have just one node at each level (so basically graphs that just look like lines), and the source node would be a node at the very end.

(b) See the attached page for the counterexample disproving this.

Problem 2

(a) A graph edge that is not in the BFS tree:

- Can be between vertices of the same level
- Can be between vertices in consecutive levels (from higher levels to lower levels and from lower levels to higher levels)
- Can be edges from higher levels to lower levels (these do not have to be between vertices in consecutive levels)
- Can be those edges going outwards from vertices that are not connected to the source node

The type that should appear in the shortest paths graph is the second type (from lower levels to higher levels, as the opposite is going backwards), as edges between vertices of the same level add uneccessary distance between vertices, and edges of the third type are directed back towards lower levels (going backwards). Edges of the fourth type obviously should not appear in the shortest paths graph since they are not even connected to the source node. The fact that an edge e between vertices u, v of respective levels $i - 1, i$ is on a shortest path was proved in lecture 7.

(b) NOTE: IGNORE THE G' NOTATION IN THE PROBLEM. WE WILL USE OUR OWN NOTATION. The algorithm is as follows

1. We run BFS on our original graph G from s to obtain a BFS tree G'
2. We alter G' by adding back in those edges that are between consecutive levels and that go from lower levels to higher levels. These edges need to be in G.
3. We now make a new BFS tree G'' by running a BFS on a graph G_R from t, where G_R is the same as G but with all the edges reversed.
4. We alter G'' by adding in those edges that are between consecutive levels and that go from lower levels to higher levels. These edges need to be in G_R.
5. We then reverse all the edges of G''.
6. Now we have that the shortest paths graph $G_S = G' \cap G''$.

We can see that this works for the following reason. G', after step 2, has only the edges (and vertices) that could possibly be in a shortest path from s to t (edges between vertices in consecutive levels, going from lower levels to high levels). That is, it contains all the shortest paths from s to t. But it also contains some extra edges and vertices. Then, intersecting this with G'' gets rid of those edges (and vertices) that are not actually on a path to t. This is because those edges and vertices will not be included in the BFS tree of G_R. Thus we are just left with our shortest paths graph.

Problem 3

(a) See the attached page for the counterexample disproving this.

(b) See the attached page for the counterexample disproving this.

Problem 4

The algorithm is as follows

1. We make a graph with 6 vertices. Each vertex represents a number from 1-6, and no two vertices can represent the same number. So we have 6 vertices, where one vertex represents 1, one vertex represents 2, etc.
2. For each domino with numbers x and y, we draw a line between the vertices that represent x and y.
3. Then we have that if there is a Eulerian path through this graph, Biscuit can place all the dominos in one row. And we learned in lecture that a connected undirected graph contains a Eulerian path if and only if it contains exactly two vertices with odd degrees. So we can just check for this to see if there is a Eulerian path, and if there is, we return true, else, we return false.

We can see that this works for the following reason. An Eulerian path in a graph G is a path that passes through every edge of G exactly once. And in our graph, passing through an edge is equivalent to using a domino, and passing through consecutive edges is equivalent to placing dominos next to each other (within the rules). Thus it follows that a Eulerian path through G means that we can place all the dominos next to each other, within the rules, using each domino exactly once.

Problem 5

See the attached page for the counterexample disproving this.