Problem 1

(1) We should perform a BFS on G starting at a and halt if we reach b. Then we should perform a BFS on b and halt if we reach c. Sum $dist(a, b)$ and $dist(b, c)$. Of course, if BFS fails in either (or both) cases, the sum will give ∞. **(2)**The algorithm is correct because, by *Theorem 22.5* in the text, BFS always computes the shortest path from the source vertex s to every other vertex in the graph. Also, BFS discovers every vertex v which is reachable from s and assigns a finite distance to it. The shortest distance between a and b, $\delta(a, b)$, is the path from a to b in BFS. Also, the shortest path from b to c is of length $\delta(b, c)$ which is the length of the path computed by BFS. If there is a way from a to b and b to c, BFS will find it optimally. **(3)**This algorithm will run worst if $dist(a, b) = |E|$, and then there is no path from b to c, making the algorithm traverse the entire graph twice. This will still be an $O(|V| + |E|)$ algorithm, however.

Problem 2

$V = A \cup B \cup C$. The sets A, B, C may have non empty intersection, or they may be empty. First check the vertices of A. For each vertex, gather the edges which go from A to a vertex not in A. Furthermore, check if any one of these vertices is also a member of B, in which case we stop searching and say that dist(A,B) = 0. Then we collapse the vertices $v \in A$ into one vertex a with the outgoing edges we just found. We then perform a BFS on $\{a\} \cup B \cup C$ with source a. As soon as we come across a vertex $b \in B$ we record its distance number. Since BFS always returns the shortest path, we can be sure that the dist(a,b) is minimal. If we do not come across any b, then the distance in infinite.
Since $|\{a\} \cup B \cup C| \leq |V|$ and for the corresponding edge set $\hat{E}$, $|\hat{E}| \leq |E|$ we can comfortably assert the upperbound $O(|V| + |E|)$.

Problem 3

Only one edge (v, w) may exist for a pair of vertices, but there may be multiple ways of arriving at v. In addition, there may be more than one vertex v_i which connects to w. Therefore, the number of paths to a vertex is the sum over all of the predecessor vertices of the number of paths to each predecessor. In other words:

$$w.num\_paths = \sum_{v \in w.predecessor} v.num\_paths$$

To calculate this value , we perform a traditional BFS but we store a number of paths value for each vertex. Initially, all vertices have 0 paths. Then the source vertex is assigned one path, as there is one path from it to itself of distance 0. We allow vertices to be visited by almost all incoming edges, and increment the number of paths as described above for each acceptable incoming path. All incoming paths to a vertex are acceptable so long as they do not come from a vertex in the same level set as the receiving vertex. This is because these edges will not lie on a shortest path from the source, but will be at a distance of $min + 1$. Lastly, my algorithm checks before enqueueing to make sure that a vertex is not enqueued twice.
The complexity of this algorithm is the same as that for BFS. We may visit more edges than BFS due to the fact that we allow more than predecessor per vertex, but we will not exceed $|E|$ edges. In addition, each vertex will still be enqueued at most once due to the check we put in the code. The complexity is still $O(|V| + |E|)$.