Graph properties —

- *Simple* means no self-loops and no multiedges — make sure you address both.
- Labeled vs unlabeled graph — *labeled* refers to vertex identifiers, not other sorts of information that may be associated with vertices or edges. In addition, "labeled" typically means that the vertex identifiers are meaningful beyond distinguishing one vertex from another. A way to test whether the identifiers are meaningful is to consider whether changing the identifiers without changing anything about the graph structure itself results in a different graph. Any application where you need to refer to specific vertices (e.g. to find the shortest path from A to B) requires a labeled graph — changing the identifiers around means that "A" and "B" no longer refer to the same vertices, and the result may be a completely different path. On the other hand, MST doesn't rely on the vertex labels — the MST will connect the same vertices regardless of how they are identified.
- Implicit vs explicit graph — how do you determine if two vertices are adjacent? For an *explicit* graph, the adjacency information is stored and you look up whether two vertices are adjacent. (This doesn't require edges in a graph data structure — you could look up adjacency info in some other data structure, but the key point here is having to look up the info.) For an *implicit* graph, adjacencies are computed rather than stored — there is a function or a rule defining the adjacencies. For the four-digit configuration graph, you only need the rule about button presses changing one of the digits by one value higher or lower in order to determine if any two states are connected or which states are adjacent to a particular state *abcd*.
- Embedded vs topological graph — an *embedded* graph is one where there is a specific drawing in the plane (vertices have positions, edges follow curves, and edges occur in a particular cyclic order around each vertex), while only the adjacency of vertices matters for a *topological* graph. Keep in mind that a graph is just a representation the thing being modeled, so the "topological" or "embedded" relates to the representation rather than the underlying thing. For example, a road network certainly has a drawing in the plane — intersections occur at particular geographical locations, roads trace out curves on the ground, and that determines a cyclic ordering of roads at each intersection. For a shortest path application, edge weights capture the distance between two intersections and so would be derived from that physical layout of the road network, but once you have those edge weights, the graph is treated as being purely topological — nothing about the shortest path algorithm relies on a particular layout or makes use of the cyclic ordering of edges around vertices. For driving directions, however, the routing and ordering of edges around each intersection matters — the directions need to say "turn right" or "turn sharp left" or "go straight" and more of the geometric information from the embedding must be captured in the graph. So, slightly

different graph representations are needed for the two tasks, one topological and one embedded.

For #2, how many of the poles are within stepping distance of a particular pole does depend on the arrangement of the poles. But when thinking about sparse vs dense for this situation, what do you expect? Would you expect each pole to have a few other poles nearby, or to be within stepping distance of most other poles? Also, since "stepping distance" is defined by a particular range, does it even seem possible to get most of the other poles between d and $2d$ of a particular pole without them being too close to each other? "Dense" requires a high average degree, not just a few vertices with high degree.

For #3, the problem asks for an algorithm to find a one-showing-each schedule if one exists, so be sure to address both sides of the coin — finding a 2-coloring gives a schedule, but also cover what not finding a 2-coloring means in terms of a schedule.

Also, a technical note: *bipartite* is a property of a graph — the vertices can be split into two groups such that no edge connects two vertices in the same group. A *coloring* is an assignment of colors to the vertices of the graph so that no two adjacent vertices are the same color. "2-coloring" and "bipartite" are closely related, but the tasks aren't exactly the same — checking if the graph is bipartite will tell you *if* there is a one-showing-each schedule but it won't tell you which day to show each movie.

For #4, also address (b) — how many vertices and edges will your graph have?