The output should be the route from start to finish, including both junctions and controls. However, the wording "a list of all the junctions visited" was ambiguous on this count so there was no penalty for omitting controls. Some people also removed apparent duplicate visits that resulted from going from a junction to a control and then exiting the control back via that same junction. That was not intended, but also not penalized.

Efficiency considerations —

- Watch out for hidden $O(n)$ costs, such as checking whether a particular vertex is a control. If you have a list of controls, `controls.contains(c)` is $O(n)$. This should be a $O(1)$ operation...
- There are two versions of `run` for the provided Dijkstra's implementation — `run()` finds the shortest path to all controls, while `run(f)` runs only until the shortest path to `f` is found. Since most of the map probably won't be visited on any given leg, stopping as soon as the shortest path to the next control is found can save a bunch of time.
- It isn't necessary to create a new `DijkstraDirected` object for each leg of the course. It's not an expensive operation, but even the cheap ones add a little to the running time.

Tips —

- `Scanner` can be more convenient for parsing input than reading a whole line at a time and then having to split it up and convert `String`s read into numeric values.
- Instead of reading in everything from the files before starting to build the graph, consider building the graph as you go. This can avoid unnecessary bookkeeping and data structures to store stuff that you don't need to keep around.