HydratedHakofos Writeup

This is how my algorithm works: it traverses the tables, as represented by the `waterAvailable` and `waterRequired` arrays. At each table, it checks if `waterAvailable[table] - waterRequired[table] ≥ 0`, which ensures that starting at this table won't immediately dehydrate you. If true, it calls a helper method to check if this table could be a good starting location. The helper method traverses the tables, adding each table's `waterAvailable` and subtracting its `waterRequired` to my variable `waterTotal`, which represents how much water a person has. If this loop returns to the initial table it started at, the Hakofah was successful and my algorithm returns that table. If `waterTotal` is ever negative, the helper method returns the table at which the `waterTotal` became negative. At this point, the loop in `doIt` resumes, but it skips to the table after where `waterTotal` became negative in the helper method. The loop continues until a valid table is found, the main loop reaches the end of the arrays, or the helper method returns a table that has already been passed over in either the main loop or a pass of the helper method loop, at which point the method returns `-1`.

In the following analysis, Table A refers to the table that my helper method is examining starting Hakofos from, Table C is the table at which `waterTotal` becomes negative, and Table B is any table between those tables. My algorithm relies on the invariant that if the helper method traverses past Table B, and Table A is found to be invalid, Table B cannot be valid either. This is because since the examination started at an earlier table, the `waterTotal` when Table B is visited must be `≥ 0`. This is because if `waterTotal < 0`, traversal never would have reached Table B in the first place. This means that `waterTotal` when reaching Table C from Table B is `≤ waterTotal` when reaching Table C from Table A, so if it is negative when starting at Table A, making Table A invalid, the same must also be true for Table B. Because of this, the main loop can now skip every table between A and C and continue its search after Table C.

Because of this invariant, in the worst-case scenario where there is no solution, my algorithm examines each table no more than twice: once when examining either that table or a table with enough water to reach it, and possibly a second time when examining a table whose traversal loops back to the beginning of the table arrays. As such, the number of tables my algorithm visits is no more than 2n, where n is the number of tables, giving my algorithm an order of growth of O(n).