Code Jam 2015 - Round 1B

Analysis: Hiking Deer

Some programming contest problems have their origins in everyday life. This year, the Qualification Round's "Standing Ovation" was written during a concert, and Round 1A's "Haircut" was written while waiting in a long line for a sandwich at Google's Go Cafe. You may not be surprised to learn that this problem was inspired by a circular hike on a day with an unusually large number of hikers on the trail!

Let H be the total number of hikers. One important observation is that the answer can't be larger than H, since no matter what the hikers' initial positions and speeds are, Herbert can just go around so fast that no hiker gets a chance to move very far, so he encounters each of them once.

Another key observation is that allowing Herbert to wait or vary his speed makes no difference to the answer. If we know the time he arrives at the end, we can put a lower bound on the answer based on the number of hikers he must have overtaken and the number of times hikers must have overtaken him. This lower bound is exactly the answer we get by having Herbert move at a constant speed, so no strategy of waiting or changing speed can improve on it.

So now, we will find a finish time that minimizes the number of encounters.

Consider the case of a single hiker. As the trail is circular and the hiker walks endlessly, the hiker repeatedly reaches Herbertâ€™s starting point. Call the time at which Herbert finishes his hike X, and the times when the hiker reaches Herbertâ€™s starting point T1, T2, T3, etc.

- If X <= T1, then there is one encounter with the hiker as Herbert passes.
- If T1 < X < T2, then there are no encounters with the hiker.
- If T2 <= X < T3, then the hiker passes Herbert once.
- If T3 <= X < T4, then the hiker passes Herbert twice.
- etc.

So as we increase Herbert's finish time, there are multiple "events" where the number of encounters changes. It decreases by 1 for the first event, then it increases by 1 for each event after that.

Now when we have multiple hikers, there is a series of events for each of them that can increase or decrease the total number of encounters.

There will never be a reason for Herbert to take so long that we see more than H events for a single hiker, because then the number of encounters with that single hiker will be at least H. So a solution that will work for the small datasets is to make a sorted list of the first H events for all of the hikers, and then find the optimal number of encounters in this way:

- Initialize the number of encounters to H.
- For each event, in order of time, subtract 1 from the number of encounters if it is the first event for the hiker, otherwise add 1.

The answer is the minimum number of encounters for any time.

For the large dataset, we need a more efficient solution. Note that there are only H events that subtract 1 from the number of encounters, and the rest add 1. So once we have processed 2H events, we will never find a time with fewer than H encounters, so we can stop searching.

We still need to avoid storing H^2 events. To do this, we can maintain a priority queue of events. We initialize the queue with one event for each hiker -- the event for that hiker's T1. Whenever we process an event, we replace that event on the queue with the next event for that hiker. In this way we only need O(H) memory, and O(H log H) time.

One subtlety to note is that if multiple hikers reach Herbertâ€™s starting point at the same time, we must process the events that add an encounter (those for a hiker's T2, T3, etc.) before the events that subtract an encounter (those for a hiker's T1) since the number of encounters only really decreases once Herbert's finish time is *larger* than T1.

52 people successfully solved the large input for this problem during the contest. See, for example, Belonogov's solution, which you can download from the scoreboard.