Code Jam 2011 - Round 1B

Analysis: Revenge of the Hot Dogs

This problem might look petty tough at first glance. There are lots of hot dog vendors, and each one has a very important choice to make. How can you account for all the possibilities at once?

It turns out there are at least two completely different solutions, one of which uses a classical algorithmic technique, and one of which is purely mathematical. We will present both approaches here.

An Algorithmic Solution

There are two key ideas that motivate the algorithmic solution.

- There is no reason to ever have one hot dog vendor walk past another. Instead of doing that, they could walk up to each other and then just go back the way they came. Since everyone moves at the same speed, this is completely equivalent to having the two people cross.
- Rather than trying to directly calculate the minimum time required, it suffices to ask the following slightly easier question: Given a time **t** and a distance **D**, is it possible to move all the hot dog vendors distance **D** apart in some fixed time **t**? If we can answer that question efficiently, we can use a binary search to find the minimum **t**:

```
lower_bound = 0
upper_bound = $10^{12}$
while upper_bound - lower_bound > $10^{-8}$ * upper_bound:
  t = (lower_bound + upper_bound) / 2
  if t is high_enough:
    upper_bound = t
  else:
    lower_bound = t
return lower_bound
```

So we need to decide if **t** seconds is enough to move all the hot dogs vendors apart. Let's think about the road as going from left (negative values) to right (positive values), and focus on the leftmost person *A*. By our first observation, he is still going to be leftmost when everyone is done moving. So we might as well just move him as far left as possible. That way, he will interfere as little as possible with the remaining people. Since we have fixed **t**, this tells us exactly where *A* will end up.

Now let's consider the second leftmost person *B*. He has to end up right of *A* by at least distance **D**. Subject to that limit, he should once again go as far left as possible. (Of course once you account for the first guy, "as far left as possible" might actually be to the right!) The reason for doing this is the same as before: the further left this person goes, the easier it will be to place all the remaining people. In fact, this same strategy works for everyone. Once the time is fixed, each person should always go as far left as possible without getting less than distance **D** from the previous person.

Using this greedy strategy, we can position every single person one by one. If we come up with a valid set of positions in this way, then we know **t** is enough time. If we do not, then there is nothing better we could have possibly done.

We can now just plug this into the binary search, and the problem is solved!

A Mathematical Solution

On the Google Code Jam, we would expect our contestants to try algorithmic approaches first. After all, you guys are algorithm experts! However, we would like to also present a mathematical solution to this problem. It avoids the binary search, and so it is more efficient than the previous solution if you implement it right.

As above, sort the people from left to right. Let P_i be the position of the i^{th} person, and let $x_{i,j} = D*(j - i) - (P_j - P_i)$. Finally, define $X = \max_{i < j} x_{i,j}$. We claim max(0, X) / 2 is exactly the amount of time required.

Let's first show that you need at *least* this much time. Focus on two arbitrary people: i and j. Since nobody should ever cross (as argued above), there must still be j - i - 1 people between these two when everything is done. Therefore, they must end up separated by a distance of at least D*(j - i). They start off separated by only $P_j - P_i$, and this distance can go up by at most 2 every second, so we do in fact need at least $[D*(j - i) - (P_j - P_i)] / 2$ seconds altogether.

To prove this much time suffices, we show how X can always be decreased at a rate of 2 meters per second. Let's focus on some single person j. We will say he is "left-limited" if there exists i < j such that $x_{i,j} = X$, and he is "right-limited" if there exists k > j such that $x_{j,k} = X$. Suppose we can move every left-limited person to the left at maximum speed, and every right-limited person to the right at maximum speed. Then any term $x_{i,j}$ which is equal to X will be decreasing by the full 2 meters per second, and hence X will also be decreasing by 2 meters per second, as required.

So this strategy works as long as no single person is both left-limited and right-limited. (If that happened, he would not be able to go both left and right at the same time, and the strategy would be impossible.) So let's suppose $x_{i,j} = X = x_{j,k}$. If you just write down the equation, you'll see $x_{i,k}$ is exactly equal to $x_{i,j} + x_{j,k}$. But this means $x_{i,k} = 2X > X$, and we have a contradiction. Therefore, no single person is ever both left-limited and right-limited, and the proof is complete!

Additional Comments

- It turns out the answer for this problem is always an integer or an integer plus 0.5. Do you see why? In particular, if you multiply all positions by 2 at the beginning, you can work only with integers. This allows you to avoid worrying about floating point rounding issues, which is always nice!
- At first, the mathematical solution looks like it is $O(n^2)$, since you are calculating the maximum of $O(n^2)$ different numbers. Do you see how to do it linear time?