Code Jam 2018 - Round 1B

Analysis: Mysterious Road Signs

Test Set 1 (Visible)

First, let's break down what the problem is asking us to find. We are asked to identify sets of contiguous signs, where each set is defined by four variables:

1. The index of the first sign in the set, *i* (inclusive)
2. The index of the last sign in the set, *j* (exclusive)
3. The destination for eastbound travelers, *M*
4. The destination for westbound travelers, *N*

In order for a set to be valid, each sign in the set must be truthful to eastbound travelers, westbound travelers, or both. Given the four variables above, we can evaluate a set of signs for validity in O(*j*–*i*) time. We can bound the number of possible values for *M* and *N* by **S** by taking the set of all westbound or eastbound destinations displayed by at least one sign. Since *i* and *j* are also bounded by **S**, we would need O($\mathbf{S}^5$) time for this solution, which is not sufficient for Test Set 1.

To improve our brute-force solution, we can be more clever with how we pick *M* and *N*. The first sign in a set tells us a lot of information: namely, it defines what either *M* or *N* must be (one of the two destinations on the sign, or else the sign wouldn't be able to be in the set). Suppose we fix *M* based on the first sign. Now, we can walk through the rest of the signs in the set until we find a sign whose eastbound destination is not *M*; use that sign's westbound destination as *N*. Continue walking until we reach a sign that does not share either *M* or *N*, in which case the set is invalid, or until we reach the end of the set, in which case the set is valid. We can perform the analogous process by fixing *N* based on the first sign. This “two-pass” set evaluation algorithm runs in O(**S**) time, since sets have size O(**S**).

Since there are O($\mathbf{S}^2$) sets of signs and we can evaluate each set for validity in O(**S**) time, we have a O($\mathbf{S}^3$) algorithm, which is fast enough for Test Set 1.

Test Set 2 (Hidden)

Clearly the cubic solution won't work on Test Set 2. It turns out that a quadratic solution coded in a fast language with a sufficient amount of pruning can pass Test Set 2. However, in this analysis, we will describe a O(**S** log **S**) solution followed by a linear-time O(**S**) solution, both of which comfortably finish in time when evaluating Test Set 2.

The O(**S** log **S**) solution is a classic application of divide and conquer. Cut the list of signs in half, except for a single sign at the center, which we will call the “midpoint” sign. Recursively apply the algorithm to the western half and the eastern half of signs. Then, use a modified version of the previously described two-pass algorithm to find the best set containing the midpoint: first, fix *M* to the midpoint's eastbound destination. Since we are looking for long segments, we can greedily add as many signs to the set as possible. Walk the list of signs both westward and eastward until reaching a sign on each end that does not share the same eastbound destination (*M* value) with the midpoint. Let N_1 equal the westbound destination of the western boundary (the first sign to the west that did not match *M*) and let N_2 equal the westbound destination of the eastern boundary (the firsts sign to the east that did not match *M*). Find M_1 and M_2 using an analogous procedure: return to the midpoint, walk westward and eastward until reaching signs that do not share a westbound destination (*N* value) with the midpoint, and let M_1 and M_2 be the eastbound destination of the signs at the western boundary and eastern boundary, respectively. We now have four possible M/N pairs: (M, N_1), (M, N_2), (M_1, N), and (M_2, N). We can greedily walk east and west from the midpoint with each of the four pairs to find the longest set containing the midpoint. This “four-pass” step for D&C runs in linear time. Now that we have found the longest set(s) containing the midpoint as well as (recursively) the longest sets from the western and eastern halves of signs, we can combine these results to get the longest sets for the whole dataset. This yields an algorithm requiring T(**S**) operations to compute an input of size **S**, for a function T that satisfies T(**S**) = 2T(**S**/2) + O(**S**). Using the Master Theorem we get that T(**S**) = O(**S** log **S**).

The linear-time solution does a single pass, identifying all possible long segments. Start by walking forward from the first sign. Maintain two “candidates”, called the “M-candidate” and the “N-candidate”, with the following properties:

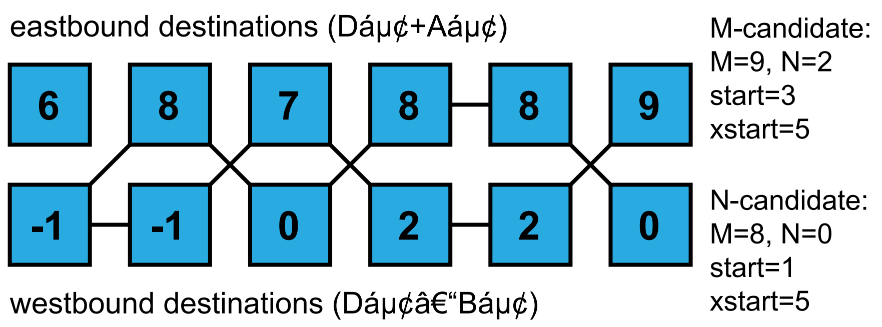
- Two destinations *M* and *N*.
- An index *start* corresponding to the westernmost sign in the contiguous segment of signs that includes the current sign and that satisfies *M* or *N*.
- An index *xstart* corresponding to the westernmost sign in the contiguous segment of signs that includes the current sign and whose eastbound destinations are all *M* (for the M-candidate) or whose westbound destinations are all *N* (for the N-candidate).

With these invariants, the set of signs starting at *start* and ending after the current index is guaranteed to be a valid set.

To maintain the invariants when reading a new sign, we can use the following procedure to create the new the M-candidate (the procedure for creating the new N-candidate is analogous):

- If new sign's eastbound destination equals previous sign's eastbound destination, copy the previous M-candidate to become the new M-candidate.
- If the new sign's eastbound destination equals the previous N-candidate's *M* value, copy the previous N-candidate to become the new M-candidate, and set *xstart* to the new sign's index.
- Otherwise, copy the previous N-candidate to become the new M-candidate, set *M* to the new sign's eastbound destination, set *start* to *xstart*, and set *xstart* to the new sign's index.

Consider the following illustration:



The illustration shows six zero-indexed signs and the destinations for those six signs. The signs are connected by lines that show which candidates are used when calculating the candidates for the next step. The candidates after reading the sixth sign (index 5) are shown at the end. On top we have the M-candidate with *M*=9 and *N*=2. The candidate starts at index 3, because the sign at index 2 does not have a compatible westbound or eastbound destination for that candidate. The illustration also demonstrates how *xstart*=5 is the start of the most recent string of westbound destination *M* matches, which in this case is only the current sign. The N-candidate goes all the way back to start index 1 with *M*=8 and *N*=0.

Since calculating the new candidates takes constant time, and we have to read each sign in sequence, this is an O(**S**) solution.

There are two more tips that can aid in solutions to this problem. First is that we don't ever need to remember $\mathbf{A_i}$, $\mathbf{B_i}$, and $\mathbf{D_i}$ directly; we only care about the westbound and eastbound destinations. You could therefore compute those destinations upon reading in the data and store them in a list of pairs. The second tip is that there is a lot of opportunity for pruning: if you maintain a global list of the best known sets of signs, you never need to look at any candidate sets that are smaller than the current best, allowing you to prune away a lot of segments that you no longer need to evaluate. All of the solutions described here except for the O(**S**) solution can take advantage of pruning.