Code Jam 2010 - World Finals

Analysis: Travel Plan

As with City Tour, this problem is closely related to Hamiltonian cycles. The good news is that here, the graph consists only of points on a line, and is therefore much easier to analyze. The bad news is we aren't looking for just any Hamiltonian cycle, or even the shortest one; we are looking for one of a particular length. A question this precise requires a complete search of some kind.

The simplest approach is to try all possible travel plans one at a time. This is much too slow for the large input, but it is fine for the small input.

Given that $N \leq 30$ throughout, you might next think to try a dynamic programming solution that tracks how far you have gone and which nodes you have visited so far. Unfortunately, this is also too slow - large values of **F** are a big problem!

A 3^N solution

The first key step to solving this problem is to go from $N!$ time to 3^N time.

We first divide the line into intervals $I_1, I_2, ..., I_{N-1}$. Let t_j denote the number of times that the travel plan crosses interval I_j, and let d_j denote the length of interval I_j. Then the total length of the travel plan is exactly $t_1 * d_1 + t_2 * d_2 + ... + t_{N-1} * d_{N-1}$. So this means that all we need to do is figure out what each t_j should be.

The big question is what t_j values are possible? Well, let's take a look.

- Each t_j must be positive.
  Reason: If $t_j = 0$, then the travel plan never crosses the corresponding interval, and therefore cannot visit every planet.
- For each j, $t_j - t_{j-1}$ must be -2, 0, or 2.
  Reason: Let P denote the planet between the intervals I_{j-1} and I_j. In our travel plan, we can stop at P at most once. Every other time, we must travel through, which contributes 1 to both t_{j-1} and t_j. The one time we do hit the planet, we come in along one interval, and then leave along one interval. (These two intervals may or may not be the same.) This part of the travel plan contributes either (a) 1 to both t_{j-1} and t_j, or (b) 2 to one of these values, and 0 to the other. Regardless, $t_j - t_{j-1}$ must be -2, 0, or 2.
- t_0 and t_{N-1} must both be 2.
  Reason: Consider the "outside interval" I_{-1} that is past the furthest planets. The travel plan cannot traverse this interval, so $t_{-1} = 0$. The observation here now follows immediately from the previous two.
- Every choice of t_j satisfying the previous conditions can be achieved by a legal travel plan.
  Reason: We can construct such a travel plan by scanning from left to right across the intervals. At any given time during the construction, we will have decided what the travel plan does to the left of some planet P. Specifically, for each trip left of P, we will know what the trip does until it goes right of P again. We will not however know what happens between these visits, or even what order they occur in.
  Now, suppose we want to extend our partial travel plan past the next interval I_j. If $t_j =$

$t_{j-1} - 2$, then we need to merge two of our current trips via a stop to P, and extend the rest along I_j. If $t_j = t_{j-1} + 2$, then we add a new trip that goes along I_j, stops at P, and then goes back again. In the final case we just extend each trip along I_j, having one stop at P without changing direction.
If you think about this for a while, you should be able to convince yourself that this method does indeed generate a valid travel plan.

Okay! We can now describe a 3^N time solution to the original problem. Given t_{j-1}, there are at most 3 possible choices for t_j. Trying each one in turn, we can iterate over all possible assignments for the t_j's, and then pick the one that leads to the optimal length travel plan.

A $3^{N/2}$ solution

Even this 3^N time algorithm is too slow however. Fortunately, there is a very handy trick for situations like this.

Let's consider the middle interval $I_{N/2}$ and some fixed value for $t_{N/2}$. Using the approach described above, we can enumerate in $3^{N/2}$ time all choices for $t_1, t_2, \ldots, t_{N/2-1}$ that are consistent with $t_{N/2}$. Let **A** be the set of all possible values $t_1 * d_1 + t_2 * d_2 + \ldots + t_{N/2} * d_{N/2}$ that can be achieved in this way.

Similarly, we can calculate **B**, the set of all possible values of $t_{N/2+1} * d_{N/2+1} + t_{N/2+2} * d_{N/2+2} + \ldots + t_{N-1} * d_{N-1}$ that are consistent with $t_{N/2}$. All this takes $3^{N/2}$ time. What's left is to find two numbers in **A** and **B** whose sum is as close as possible to **F** without exceeding it. One efficient way to do this is to sort **B**, and then for each element in **A**, use binary search to decide which element in **B** you should try matching it with. That gives a $O(3^{N/2} * N)$ solution.

This can be improved to $O(3^{N/2})$ by rearranging the formula a little: $t_1 * (d_1 + \ldots + d_{N/2}) + (t_2 - t_1) * (d_2 + \ldots + d_{N/2}) + \ldots + (t_{N/2} - t_{N/2-1}) * d_{N/2}$. The list of these sums can be generated in sorted order iteratively by repeated merging while adding new terms. The difference $t_{j+1} - t_j$ is always one of -2, 0, 2, so we need to merge 3 sorted lists to add a new term.

When we have the two lists **A** and **B** in sorted order, we can process them in linear time with a single scan using two pointers.