Trains

Subtask 1 ($N \leq 15$)

A simple brute-force approach every single journey is tried is enough to solve this subtask. The maximum number of journeys is 2^{14} if all $x_i \geq 15, d_i = 1$, since we always visit Vilnius, and then can choose any set of other cities to visit. As 2^{14} is only 16384, so even very slow implementations should be fast enough to solve this subtask.

Subtask 2 ($N \leq 10\,000$)

We can use dynamic programming to avoid recomputing a large number of journeys. Let's define P_i as the number of journeys that could be started from city i. We can then use the following recurrence relation:

$$P_i = 1 + \sum_{t=1}^{\substack{i+d_i \times t \leq N \\ t \leq x_i}} P_{i+d_i \times t}$$

The 1 in the expression above corresponds to the journey that is just the i-th city on its own, and if we do go to another city j, we can take any of the P_j journeys from there.

We can compute P_i from the biggest i to the smallest, and we need $O(N)$ time to compute each P_i, or $O(N^2)$ time in total.

Subtask 3 ($d_i = 1$ for all i)

Let's see what we get when we plug in $d_i = 1$ for all i into the formula from the previous subtask:

$$P_i = 1 + \sum_{t=1}^{\substack{i+t \leq N \\ t \leq x_i}} P_{i+t}$$

Notice that the sum always contains subsequent P_i values. If we define $S_i = \sum_{j=i}^{N} P_j$, then the formula becomes:

$$P_i = 1 + S_{i+1} - S_{i+x_i+1}$$

Note: some care needs to be taken that S_{i+x_i+1} evaluates to 0 when $i + x_i + 1 > N$, but that is an implementation detail.

We also have the following recurrence relation for S_i:

$$S_i = P_i + S_{i+1}$$

With these two formulas we can compute P_i and S_i from in decreasing order i, which has $O(N)$ complexity.

Extension: $d_i = k$ for all i

A natural extension to the 3rd subtask is to think about what happens when we set all d_i to some other constant value, say $d_i = k$. We then have:

$$P_i = 1 + \sum_{t=1}^{\substack{i+k\times t\leq N\\ t\leq x_i}} P_{i+k\times t}$$

We can now adjust our S_i definition to be the sum of every k-th P_i value from i to N, after which we end up with these formulas and can compute a solution similarly as in the 3rd subtask:

$$S_i = P_i + S_{i+2} \quad k$$
$$P_i = 1 + S_{i+k} - S_{i+(x_i+1)\times k}$$

General case

Now we know how to compute the solution if we have a single d value. We could try to solve the general case by keeping a different array S_i for each different value d in the input. The issue is that each S has size $O(N)$, and there can be $O(N)$ distinct d_i values, so populating this many values would immediately result in an $O(N^2)$ solution, which is not fast enough.

We can also investigate how different d values affect our other solutions, in particular our solution for the 2nd subtask. We used the following formula to solve the 2nd subtask:

$$P_i = 1 + \sum_{t=1}^{\substack{i+d_i\times t\leq N\\ t\leq x_i}} P_{i+d_i\times t}$$

It should be quite obvious that a larger d_i value for some city i means less elements in the summation. In fact, as d_i increases, the number of elements in the sum decreases very rapidly: just going from $d_i = 1$ to $d_i = 2$ decreases the number of elements in the sum by a factor of 2, and by the time we reach $d_i = \sqrt{N}$, the number of elements becomes at most $\sqrt{N}$.

So we can try to combine these two approaches: for large d values use the formula for the 2rd subtask, and maintain a separate S array for each small d value. It's not immediately clear how small is small and how large is large, but if we try a few values we can quickly see that if we split these at $\sqrt{N}$, we end up with:

- $\sqrt{N}$ S arrays of size N for small d values. Computing each element takes constant time, so the total time complexity is $O(N\sqrt{N})$.
- For small d_i values we can compute P_i directly from S_i with the formula from the previous section ($P_i = 1 + S_{i+k} - S_{i+(x_i+1)\times d_i}$). This takes constant time for each i.
- For large d_i values we use the formula for the 2nd subtask to compute P_i. As $d_i \geq \sqrt{N}$, there are $O(\sqrt{N})$ elements in the sum, and it takes $O(\sqrt{N})$ time for each i.

We need to spend $O(\sqrt{N})$ amount of time per city, and since there are N cities, the total time complexity is $O(N\sqrt{N})$. This is enough to solve the general case.

Credits

- Task: Bohdan Feysa (Ukraine)
- Solutions and tests: Aldas Lenkšas, Zigmas Bitinas (Lithuania)