Weight - let me sum up

- Directed graph that presents as City → Int → City
- Defining an undirected graph by defining the edges in one way and then unioning it with the inverse edges.
    - Use `let` to define a helper expression, and then use it as many times as you want in the scope of that expression
    - Why is this slow?
        - We're giving the solver a job - we ask it to find a graph that fits particular constraints rather than constructing a graph explicitly.
        - Even the explicit definition in pred form has a similar number of vars, even it it's faster to translate
        - Using the concrete inst syntax is much faster - we just set a bound, not actually solving anything!
            - Doesn't even need to invoke the solver - super fast
        - What are some downsides of using instances?
            - Partial instances can still be super slow - especially if we underconstrain (there are a **lot** of possible weighted edges in a directed graph)
- What do we do with weights?
    - We can try to add up all the edges
        - sing[sum[City.es.City]]
            - This doesn't work! City.es.City is a set, so if there are two edges with the same weight, one of the weights won't be counted.