Exercises 6 — Trees and graphs

1.

Recall that a *directed graph* (V, E) is defined as a finite set V of vertices and a relation $E \subseteq V \times V$ between them.

This question is about the properties of that relation. In the table below, make one mark in each row for the property in the left column, depending on whether all, some, or no relations defining a graph have that property. Put the mark in the corresponding ALL box, if **all relations** defining a graph have the corresponding property, the NONE box, if **no relation** has it, and the SOME box if at least one relation does, and at least one does not.

	ALL	SOME	NONE
reflexive over V		X	
transitive		X	
symmetric		X	
antisymmetric		X	
asymmetric		X	

Graphs in our definition make no special assumptions about the relations that define them, so any (finite) relation could be a graph.

2.

Recall that a *rooted tree* is a graph (T, R) such that, if the set T of nodes is not empty, then there is a node $a \in T$ (the root) such that for every $x \in T$ with $x \neq a$ there is exactly one path from a to x. $R \in T \times T$ is a relation on the set of nodes. To make things simpler, for this question we only consider non-empty trees, that is $T \neq \emptyset$.

This question is about the properties of the relations defining trees. In the table below, make one mark in each row for the corresponding property in the left column, depending on whether all, some, or no relations defining a tree have that property. Put the mark in the corresponding ALL box, if **all relations** defining a tree have the corresponding property, the NONE box, if **no relation** has it, and the SOME box if at least one relation does, and at least one does not.

	ALL	SOME	NONE
reflexive over T			X
transitive		X	
symmetric		X	
antisymmetric	X		
asymmetric	X		

The situation is different for trees, which are much more specialized and constrained structures than graphs. Since we only consider non-empty trees, none of them can be reflexive. (If we allowed the empty tree, then its link-relation R would also be empty, which is reflexive over the empty set.)

But there are trees whose link relation is transitive, viz. all those of link height 0 or 1. (Make sure you understand why that is.) And there is a tree whose link relation is symmetric, namely the tree consisting of only a root, whose link relation is therefore empty, which is symmetric.

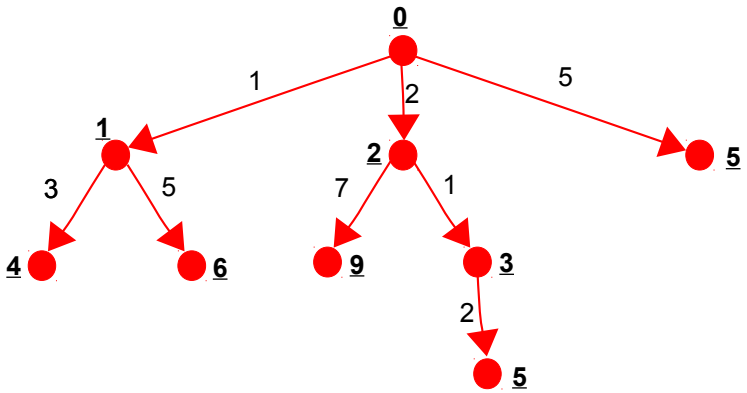
3.

Suppose we have a rooted tree (T, R) with nodes T, links $R \subseteq T \times T$.

We are also given a function $w : R \longrightarrow \mathbb{N}$ that assigns each *link* a natural number, let's call it the "weight" of that link.

1. Define a function $W : T \longrightarrow \mathbb{N}$ that maps each node in the tree to its "path weight", that is the sum of the weights of the links on the path from the root to that node.

   Example:



   The numbers next to the links are those assigned to them by the (given) function $w : R \longrightarrow \mathbb{N}$. The underlined numbers next to the nodes are those that your function $W : T \longrightarrow \mathbb{N}$ is supposed to compute in the case of this example.

$$W : n \mapsto \begin{cases} 0 & \text{if } R^{-1}(n) = \emptyset \\ w(m, n) + W(m) & \text{if } R^{-1}(n) = \{m\} \end{cases}$$

Since we are working with a tree, the image of every node under the inverse link relation is either empty (if the node is the root) or a singleton set (its parent). Once you have those conditions, the recursive call "walks upward" in the tree, computing the path weight of the parent, and adds the weight of the link from the parent to this node. Of course, the path weight of the root is 0.

2. Define the set $L \subseteq T$ of *leaf nodes* of a tree, i.e. nodes that do not have any children:

$$L = \{n \in T : R(n) = \emptyset\}$$

A node without children is one whose image under the link relation is the empty set. Note that this is the dual of the root, which is a node that has no parent, and whose image under the *inverse* link relation is empty.

3. Define the set $M \subseteq L$ of leaf nodes in T with the smallest path weight (you may, indeed should, use L from the previous subtask, and will probably find the $\min$ function useful, where for any set of numbers S, $\min S$ is the minimum number in that set, if it has a minimum):

$$M = \{n \in L : W(n) = \min\{W(m) : m \in L\}\}$$

Once you know the set of leaves, you can compute their smallest path weight as $\min\{W(m) : m \in L\}$, and then the set of all leaves with that path weight is, well, the set of all leaves whose path weight is that number.

4.

Suppose we have a directed graph (V, E). We want to define a function $r : V \longrightarrow \mathcal{P}(V)$ that computes for each vertex the set of vertices that one can reach from it by following zero or more directed edges.

We do this using a helper function $r' : V \times \mathcal{P}(V) \longrightarrow \mathcal{P}(V)$ that keeps track of the vertices we have visited already, so that we do not get stuck in cycles. Then r itself simply becomes

$$r(a) = r'(a, \emptyset)$$

The second argument of r' is the set of vertices we have visited already, initially empty.

Your task is to define r' using recursion:

$$r' : (a, S) \mapsto \begin{cases} S & \text{for } a \in S \\ S \cup \{a\} \cup \bigcup\limits_{v \in E(a)} r'(v, S \cup \{a\}) & \text{otherwise} \end{cases}$$

Hint 1: Note that the edge relation E can be used to compute all the nodes that can be reached from a given vertex a in one hop: that set is simply the image of a under E, i.e. $E(a)$.

Hint 2: You might want to recall the notion of a “generalized union”, along with the associated notation.

The first case handles the situation where we run into a node we have visited already, i.e. at the end of a cycle. In that case, we return all the nodes we have seen so far.

In the second case we deal with a node we haven't encountered before. We return the union of two things: $S \cup \{a\}$, the set of all nodes we have seen so far (plus the current one), and $\bigcup\limits_{v \in E(a)} r'(v, S \cup \{a\})$, the union of all those nodes we can reach from here along the edges going out from the current node (not forgetting to add the current node to the set of “visited nodes” we pass into r').

One might think that this second part would be sufficient, and that we would not need to explicitly add $S \cup \{a\}$ at the beginning, because all the calls to r' eventually run into the first case and will return the set S then. Except they don't: if a node has no outgoing edges, E(a) is the empty set, and no calls to r' will be made. That is why we need to include $S \cup \{a\}$. Alternatively, one could write r' to explicitly distinguish the case where the current node has outgoing arcs from when it doesn't.

5.

Suppose we have a directed graph (V, E). We want to define a function $d : V \times V \longrightarrow \mathbb{N} \cup \{\infty\}$ that computes for each pair of vertices the smallest number of directed edges required to go from one to the other. For any vertex a, $d(a, a) = 0$, and if there is no directed path from a to b, then $d(a, b) = \infty$.

We compute the function d using a helper function $d' : \mathbb{N} \times \mathcal{P}(V) \times \mathcal{P}(V) \times V \longrightarrow \mathbb{N} \cup \{\infty\}$, which keeps track of the vertices we have visited already, so that we do not get stuck in cycles, and also keeps track of the number of steps we have taken from the initial vertex. The second argument is the set of all vertices than can be reached in the next step. With this, the d becomes

$$d(a, b) = d'(0, \{a\}, E(\{a\}), b)$$

The first argument to d' is the number of steps we have taken, the second is the set of vertices we have seen already (and which are therefore at most that many steps away from the start node). The third argument is the set of nodes we can reach from the previous set in one step – note that this is **the *image* of the edge relation** E! Finally, the last parameter is the target vertex.

Your task is to define d' using recursion:

$$d' : (n, S, T, v) \mapsto \begin{cases} n & \text{if } v \in S \\ \infty & \text{if } v \notin S \text{ and } T \subseteq S \\ d'(n + 1, S \cup T, E(S \cup T), v) & \text{otherwise} \end{cases}$$

The function will need to deal with three cases:
(1) the target vertex v was found after n steps,
(2) the target vertex cannot be reached,
(3) otherwise.

Make sure you identify the conditions for the first two, as well as the return values for all three.

Hint: Note that using the edge relation E can be to compute the next nodes that can be reached at its image, you basically compute all those nodes at once. So if you have a set of nodes A, the nodes that can be reached from A in one step is $E(A)$.

The key here is to detect when the node was found ($v \in S$), and when it cannot be reached (that is $v \notin S$ and $T \subseteq S$), viz. when the nodes that can be reached from S in one step (i.e. those in T) are all in S while v is not, which means we cannot reach any new nodes.

6.

Suppose we have a rooted tree (T, R) with nodes T, links $R \subseteq T \times T$, and root a as well as a labeling function $\lambda : T \longrightarrow \mathbb{N}$ assigning each node in the tree a natural number.

We want to define a function $L : T \longrightarrow \mathbb{N}$ that computes for each node $n \in T$ the lowest number a node in the subtree rooted at n is labeled with (that subtree includes n itself). If the subtree consists only of n, its label $\lambda(n)$ is the lowest number.

As before, for any non-empty set S of numbers, $\min S$ is the lowest number in that set.

1. Define L using well-founded recursion. (Hint: You may use cases if you like, but it is possible to define this function without an explicit "base case.")

   $$L : n \mapsto \min\left(\{\lambda(n)\} \cup \{L(n') : nRn'\}\right)$$

   Note that the nRn' takes care of the "termination": if there is no child, is means there is no n', and the second set in the union above will simply be empty.

2. Define a strict partial order $\prec$ on T such that the poset $(T, \prec)$ is well-founded and your definition of L performs well-founded recursion on that poset. For all $n, n' \in T$...

   $$n' \prec n \Longleftrightarrow nR^+n'$$

   R^+ is the transitive closure of R. Specifying *only* the link relation itself would not result in a poset, since it is not a partial order (as it is not necessarily transitive, see task 2).

   There are other ways of answering that question, for example using the closure of {n} under R, i.e. R[{n}]: $n' \prec n \Longleftrightarrow n' \in R[\{n\}]$

   No proof is required. It is sufficient that the strict partial order is well-founded and your definition of L conforms to it.

   As always, make sure the partial order you define actually is one, i.e. that it has all the properties required from a strict partial order, including, for example, transitivity.

7.

Suppose we have a directed graph (V, E), as well as a weight function $w : E \longrightarrow \mathbb{N}^+$ assigning every edge in the graph a positive natural number, its *edge weight*. The *path weight* of a directed path in the graph is the sum of all edge weights of the edges in that path.

Using recursion, define a function $R : V \times \mathbb{N} \longrightarrow \mathcal{P}(V)$ such that for any vertex a and any positive integer k, the set $R(a, k)$ is the set of all vertices that can be reached from a by a path with path weight less than or equal to k.

$$R : (a, k) \mapsto \{a\} \cup \bigcup_{b \in \{b \in E(a) : w(a,b) \leq k\}} R(b, k - w(a, b))$$

8.

Suppose we have a directed graph (V, E), as well as a weight function $w : E \longrightarrow \mathbb{N}^+$ assigning every edge in the graph a positive natural number, its *edge weight*. The *path weight* of a directed path in the graph is the sum of all edge weights of the edges in that path.

We define the *distance* of two vertices in such a graph as the smallest path weight of any directed path connecting them. If there is no such path between two vertices, we say their distance is infinite, ∞. The distance of a vertex to itself is always zero, i.e. $d(a, a) = 0$ for all $a \in V$.

We want to define a function $d : V \times V \longrightarrow \mathbb{N} \cup \{\infty\}$ that computes for each pair of vertices their distance.

We do this using a helper function $d' : V \times V \times \mathcal{P}(V) \longrightarrow \mathbb{N} \cup \{\infty\}$ that keeps track of the vertices we have visited already, so that we do not get stuck in cycles. Then d itself simply becomes

$$d(a, b) = d'(a, b, \emptyset)$$

The third argument of d' is the set of vertices we have visited already, initially empty.

Your task is to define d' using recursion:

$$d' : (a, b, S) \mapsto \begin{cases} 0 & \text{for } a = b \\ \min\{w(a, x) + d'(x, b, S \cup \{a\}) : x \in E(a) \setminus S\} & \text{otherwise} \end{cases}$$

For our purposes here, you may use ∞ as a number greater than any natural number, so that you can compare with it, add to it and so on. This includes the use of functions such as $\min$, so that for example $\min\{17, 91, \infty\} = 17$ and also $\min \emptyset = \infty$.

Note that you can express the set of vertices you can reach from a vertex a in one step (i.e. via one edge) as $E(a)$, i.e. as the image of a under the edge relation E.

9.

Suppose you have a graph (V, E) with vertices V and edges $E \subseteq V \times V$.

As before, a *path* in this graph is a non-empty finite sequence $v_0 v_1 ... v_n \in V^*$, such that for an $i \in \{0, ..., n-1\}$ y we have $(v_i, v_{i+1}) \in E$. The number n, corresponding to the number of edges connecting the vertices in the path (and one less than the number of vertices in the sequence representing it), is called its *length*.

A *cycle* is a path of at least length 1 where the first and the last vertex are the same, so $v_0 = v_n$. A *simple cycle* is a cycle where every vertex occurs at most once, except for the first and last, which occurs exactly twice.

This task is about defining a function $C : V \longrightarrow \mathcal{P}(V^*)$ that for any vertex $v \in V$ computes **the set of all simple cycles** starting (and therefore also ending) at v.

We shall do so using a helper function $C' : V \times V^* \times V \longrightarrow \mathcal{P}(V^*)$, such that $C'(v, p, w)$ is the set of all simple cycles that (a) start (and end) at v, (b) then follow the path p, and (c) then continue with vertex w. In other words, $C'(v, p, w)$ is the set of all simple cycles that begin with vpw.

Using this, we can define C as follows (remember that ε represents the empty sequence):

$$C : V \longrightarrow \mathcal{P}(V^*)$$
$$v \mapsto \bigcup_{w \in E(v)} C'(v, \varepsilon, w)$$

Convince yourself that this results in all simple cycles starting at v if C' behaves as described above.

1. Define C' recursively. You may find it useful to look at the *set* of all vertices occurring in a path $p \in V^*$. You can use the notation $set(p)$ for this purpose, i.e. if p is the path $v_0 v_1 ... v_n$, then $set(p)$ is the set $\{v_0, v_1, ..., v_n\}$.

   $C' : V \times V^* \times V \longrightarrow \mathcal{P}(V^*)$

   $$v, p, w \mapsto \begin{cases} \{vpw\} & \text{if } v = w \\ \bigcup_{x \in E(w)} C'(v, pw, x) & \text{if } v \neq w \wedge w \notin set(p) \\ \emptyset & \text{if } v \neq w \wedge w \in set(p) \end{cases}$$

   The cases can also be collapsed into an elegant one-liner, like this:

   $$v, p, w \mapsto \{vpw : v = w\} \cup \bigcup_{x \in \{y \in E(w) : v \neq w \wedge w \notin set(p)\}} C'(v, pw, x)$$

2. In order to ensure that C' terminates, we require a **well-founded strict order** $\prec$ of its arguments, such that for any (v, p, w) that C' is called on, it will only ever call itself on $(v', p', w') \prec (v, p, w)$. Define such an order:

   $(v', p', w') \prec (v, p, w) \iff set(p') \supset set(p)$

Note that the order must rely on the *set* of symbols in the partial path p. It is true, of course, that p' is always also a prefix of p, but using the prefix property to establish the order does not work because there are infinite chains in it (in other words: sequences can get longer forever, but there are only a finite number of vertices, so if we add a new one at every step, we will eventually terminate).

Hint: A correct answer to this question must have three properties.

1. It must be a strict order.
2. It must be well-founded, i.e. there cannot be an infinite descending chain in that order.
3. Your definition of C' must conform to it, i.e. any recursive call in it must be called on a smaller (according to the order) triple of arguments.

10.

Suppose we have a graph (V, E), as usual with vertices V and edges $E \subseteq V \times V$, as well as a function $w : E \longrightarrow \mathbb{N}$ assigning each edge a natural number as a *weight*.

1. Define the set $E_{\leq k} \subseteq E$ consisting of all edges in E with weight not more than k:

   $$E_{\leq k} = \{e \in E : w(e) \leq k\}$$

2. Define the function $R : V \times \mathbb{N} \longrightarrow \mathcal{P}(V)$, such that $R(v, n)$ is the set of all vertices in$V$ that can be reached from v in exactly n steps, and $R(v, 0) = \{v\}$.

   $$R : V \times \mathbb{N} \longrightarrow \mathcal{P}(V)$$

   $$v, n \mapsto \begin{cases} \{v\} & \text{if } n = 0 \\ E(R(v, n-1)) & \text{if } n > 0 \end{cases}$$

3. Define the relation $P \subseteq V \times V$ such that for any two vertices $v, w \in V$, it is the case that $(v, w) \in P$ iff there is a path from v to w in the graph (V, E).

   $$P = E^{+}$$

4. Define the relation $D \subseteq V \times V$ on the vertices in V such that for any two vertices $v, w \in V$ it is the case that $(v, w) \in D$ iff there is a path p from v to w and another path q from w to v that has the same length as p.

   $$D = \{(v, w) \in V \times V : \exists n \in \mathbb{N}^{+} : w \in R(v, n) \wedge v \in R(w, n)\}$$