Garbage collection

Many modern programming languages have a *garbage collector*, which is a procedure that frees memory that is guaranteed not to be used in the program anymore. Your task is to derive a simple algorithm for garbage collection.

We can assume that everything that is stored are *objects*, where an object can refer to other objects. We let $G = (V, E)$ be an object graph, where V represents all the objects created, and a (directed) edge from u to v means that the object u has a reference to the object v. In addition, we have a set R with all the objects that can be referenced directly (typically objects which are referenced by program variables). All objects in R are also in V. Objects to which there is *no* reference via objects in R are guaranteed not to be used in the program, and should be freed.

This means that none of the objects in R should be freed, nor should any objects that are reachable through references from objects in R. The objects that should be freed are not in R, and cannot be reached from any object in R.

Suppose you have a procedure `Free` that frees an object. You should give a procedure `GarbageCollect` which takes a graph $G = (V, E)$ and a set R as input, and calls `Free` on all objects that *cannot* be reached from R.

Input: An object graph $G = (V, E)$ and a set R of objects
Output: Free all objects to which there is no reference

```
1 Procedure GarbageCollect(G, R)
  |   // ···
```