Family Trees  Recall the data definition for family trees:
A family-tree is either:

- `'unknown`
- `(make-ft name eye-color mom dad)`
  where name and eye-color are symbols,
  and mom and dad are family-trees.

(define-struct ft (name eye-color mom dad))

Write a function called path-to-blue-eyes that finds a path to a blue-eyed ancestor, if one exists and returns #f if there aren’t any. A path is represented as a list of symbols, either ‘mom or ‘dad.

Here are some examples (be sure to use these as tests when developing your function):

(define tutu (make-ft ’emily ’brown ’unknown ’unknown))
(define opa (make-ft ’bruce ’blue ’unknown ’unknown))
(define mom (make-ft ’alice ’green tutu opa))
(define dad (make-ft ’bill ’brown ’unknown ’unknown))
(define me (make-ft ’robby ’hazel mom dad))

(path-to-blue-eyes ’unknown) "should be" #f
(path-to-blue-eyes tutu) "should be" #f
(path-to-blue-eyes opa) "should be" ’()  
(path-to-blue-eyes mom) "should be" ’(dad)
(path-to-blue-eyes dad) "should be" #f
(path-to-blue-eyes me) "should be" ’(mom dad)

Sets  Write a function called union that computes the union of two sets. The union of two sets is a set that contains all of the elements in both sets.

Write your data definition for sets, along with any invariants of the data definition and be sure that your function satisfies those invariants.

Here is a function header for union

;; union : set-of-numbers set-of-numbers → set-of-numbers
;; builds a set of the numbers contained in both s1 and s2
(define (union s1 s2)
  …)