Course › 7a: Two One-Of Types › Cross Product Code › Questions 7-11

Questions 7-11

Here is our design of `has-path?` so far:

```
                | false |      (make-node Nat Str BT BT)
----------------|-------|-------------------------------------
      empty     | false |                 true
----------------|-------|-------------------------------------
(cons "L" Path) | false | (has-path? <left-child> (rest path))
----------------|-------|-------------------------------------
(cons "R" Path) | false | (has-path? <right-child> (rest path))

;; BinaryTree Path -> Boolean
;; produce true if following p through bt leads to a node; false otherwise
(check-expect (has-path? false empty) false)
(check-expect (has-path? false P2) false)
(check-expect (has-path? false P3) false)
(check-expect (has-path? BT1 empty) true)
(check-expect (has-path? BT4 (list "L")) true)
(check-expect (has-path? BT4 (list "R")) true)
(check-expect (has-path? BT4 (list "L" "L")) true)
(check-expect (has-path? BT4 (list "L" "L" "R")) false)
```

If we template the function from the table without thinking about the results we will end up with 6 cases:

```
(define (has-path? bt p)
  (cond [(and (false? bt)(empty? p))                false]
        [(and (false? bt)(string=? "L" (first p)))  false]
        [(and (false? bt)(string=? "R" (first p)))  false]
        [(and (node? bt)(empty? p))                  true]
        [(and (node? bt)(string=? "L" (first p)))   (has-path? (node-l bt) (rest bt))]
        [(and (node? bt)(string=? "R" (first p)))   (has-path? (node-r bt) (rest bt))]))
```

This will work, but there is a MUCH simpler way of writing the function definition for `has-path?`.

Question 7

1/1 point (graded)

Look at the table again:

```
                | false |      (make-node Nat Str BT BT)
----------------|-------|-------------------------------------
      empty     | false |                 true
----------------|-------|-------------------------------------
(cons "L" Path) | false | (has-path? <left-child> (rest path))
----------------|-------|-------------------------------------
(cons "R" Path) | false | (has-path? <right-child> (rest path))
```

How many cases can we reduce it to?

4

✓ **Answer:** 4

4

Explanation

The table can be reduced to these four cases:

```
                | false |      (make-node Nat Str BT BT)
----------------|-------|-------------------------------------
      empty     | false |                 true
----------------|       |-------------------------------------
(cons "L" Path) | false | (has-path? <left-child> (rest path))
----------------|       |-------------------------------------
(cons "R" Path) | false | (has-path? <right-child> (rest path))
```

Submit

ℹ Answers are displayed within the problem

Question 8

1/1 point (graded)

Here is a partially completed function body for `has-path?`:

```
(define (has-path? bt p)
  (cond [__(1)__ false]
        [__(2)__ true]
        [__(3)__ (has-path? (node-l bt) (rest p))]
        [__(4)__ (has-path? (node-r bt) (rest p))]))
```

What should the question for case (1) be?

○ `(empty? p)`

● `(false? bt)`

○ `(and (empty? p) (false? bt))`

○ `(string=? "L" (first p))`

○ `(string=? "R" (first p))`

○ `(node? bt)`

✓

Explanation

The entire column has a bt that is `false`, so `(false? bt)` is the correct question for case (1).

Submit

Answers are displayed within the problem

Question 9

1/1 point (graded)

What should the question for case (2) be?

● `(empty? p)`

○ `(false? bt)`

○ `(and (empty? p) (false? bt))`

○ `(string=? "L" (first p))`

○ `(string=? "R" (first p))`

○ `(node? bt)`

✓

Explanation

We know that `(false? bt)` is not true, so `bt` must be a node, so we can simply check `(empty? p)`.

Submit

Answers are displayed within the problem

Question 10

1/1 point (graded)

What should the question for case (3) be?

○ `(empty? p)`

○ `(false? bt)`

○ `(and (empty? p) (false? bt))`

● `(string=? "L" (first p))`

○ `(string=? "R" (first p))`

○ `(node? bt)`

✓

Explanation
Again, we know tha if we reach this case, `bt` must be a node, so we can simply check that the first element of `p` is `"L"`.

Submit

ℹ Answers are displayed within the problem

Question 11

1/1 point (graded)
What should the question for case (4) be?

○ `(empty? p)`

○ `(false? bt)`

○ `(and (empty? p) (false? bt))`

○ `(string=? "L" (first p))`

● `(string=? "R" (first p))`

○ `(node? bt)`

✓

Explanation
Again, we know tha if we reach this case, `bt` must be a node, so we can simply check that the first element of `p` is `"R"`.
The simplified function definition is:

```
(define (has-path? bt p)
  (cond [(false? bt) false]
        [(empty? p) true]
        [(string=? "L" (first p))(has-path? (node-l bt) (rest p))]
        [(string=? "R" (first p))(has-path? (node-l bt) (rest p))]))
```

Submit

ℹ Answers are displayed within the problem