Office Hours

HW 6: 1 b)

```
(: change-program : TV Integer String -> TV)
(define (change-program tv chan program)
 (local
  {
   (: new-tv : TV)
   (define (new-tv new-chan)
    (if (= new-chan chan)
          program
        (tv new-chan))) }
  new-tv))
```

tv : 1 → "AB"

2 → "XYZ"

everything else → "MNO"

(tv 10000000) ⇒ "MNO"

(change-program tv 2 "JK")

⇒ ntv

ntv : 1 → "AB"

2 → "JK"

everything else → "MNO".

(tv -1) ⇒ "MNO"

<u>Problem 4</u>

<u>Problem involving tree</u>

$\equiv$ <u>Recurse</u>

Solve the left subtree

Solve for right subtree

Combine the answers

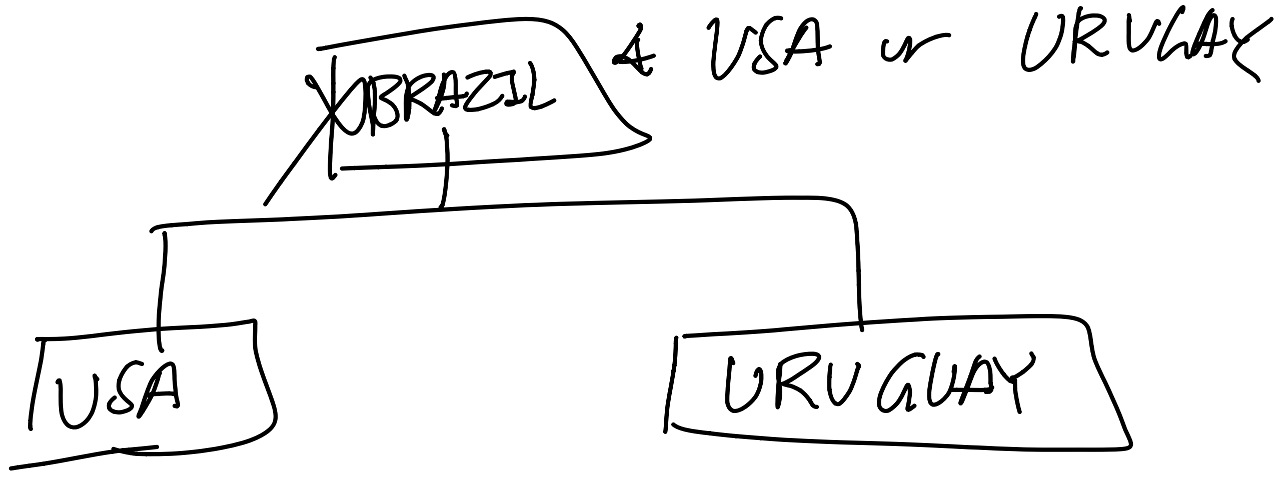
Suppose you know the answers for the left subtree and righ subtree, determine the answer for the whole tree

Suppose you know the answers for the left subtree and right-subtree.

Suppose you know left subtree is not a valid
right subtree is a valid bracket can the whole be valid bracket?

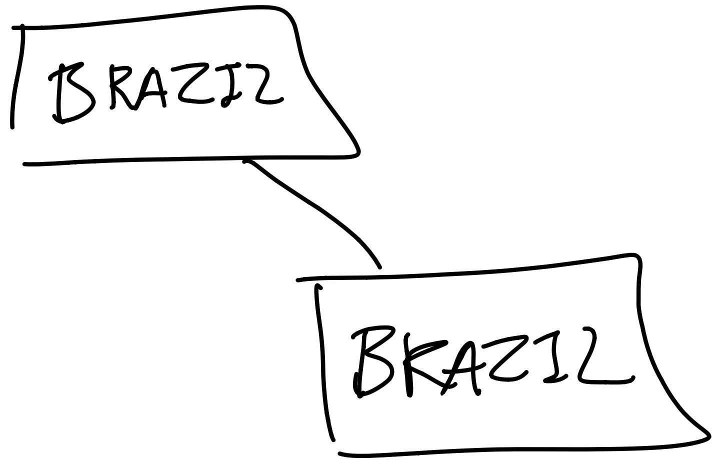
Conditions for a tree to be a valid bracket

1. Both left and right subtree must be valid bracket.



2. (Tree-val root) == (tree-val left) or (Tree-val root) == (Tree-val right)

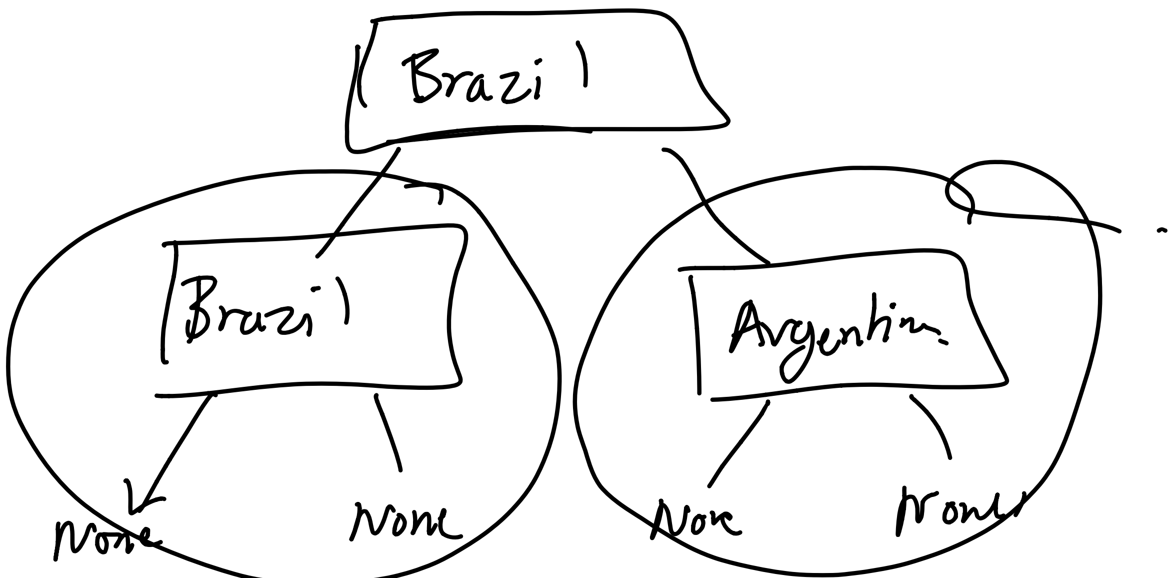
③ ⇒

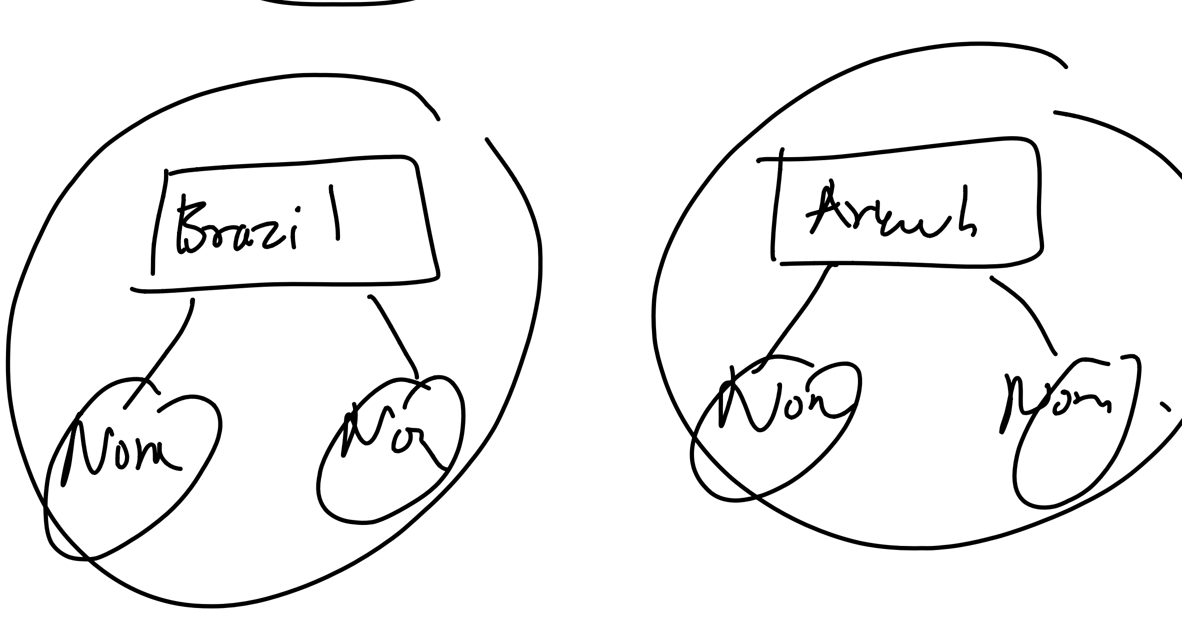


The root cannot only one child.

What is our base case?

Suppose a tree is none.
Is it a valid bracket?
Yes





```
(: valid-bracket? : (U 'none (Tree String))
        -> Boolean)
(define (valid-bracket? node)
  (local
    [
     (: left : (U (Tree String) 'none))
    (define left (Tree-left-chid node))
    (: right : (U (Tree String) 'none))
    (define right (Tree-right-child node))]
```

— Base case change to one node

```
(cond
 [(symbol? node) #t]
 [(or (not (Tree? left)) (not (Tree? right)))
     #f]

 [(or
  (string=? (Tree-value node) (Tree-value
                                 right))
 (string=? (Tree-value node) (Tree-value
                                 left))
 (and (valid-bracket? left)
       (valid-bracket? right))] )]

[else   #f].))
```

Problem 3

(a)

```
(build-list n f)

[f(0) f(1)          f(n-1)]

(: list-0-to-n : Integer ->
                 (Listof Integer))

(define (list-0-to-n  n)
  (build-list (+ n 1) int-id))
```

Note Define int-id above.

```
(: int-id : Integer → Integer)
 (define (int-id n) n )
```

Problem 2

```
(: duplicate-string : String Integer → String)
(define (duplicate-string s n)
  (local { (: constant-s : Integer → String)
     (define (constant-s n) s) }
          (build-list n constant-s)
```

× use foldr here

```
[0 3   1   2]
["ha"  "ha"  "ha"]
```

Problem 3

Square root of x.
Square root is largest n such $n^2 \le x$

1: Create a list (list-to-n function)
$[0\ 1\ 2\ \ 3\ 4 \dots x]$

2: use filter all n such that $n^2 \le x$

3: apply foldr to get max of the previous list.

$(\text{foldr max } 0 \underbrace{\qquad\qquad}_{\text{your list}})$

```
(: my-sqrt : Integer -> Integer)
 (define (my-sqrt x)
   (local [
     (: lte-sqrt-x : Integer -> Boolean)
 (define (lte-sqrt-x n)
        (<= (* n n) x)))
  )

(foldr max 0 (filter lte-sqrt-x (list-0-to-n
                x))))  )
```