Improving Pseudo-Code

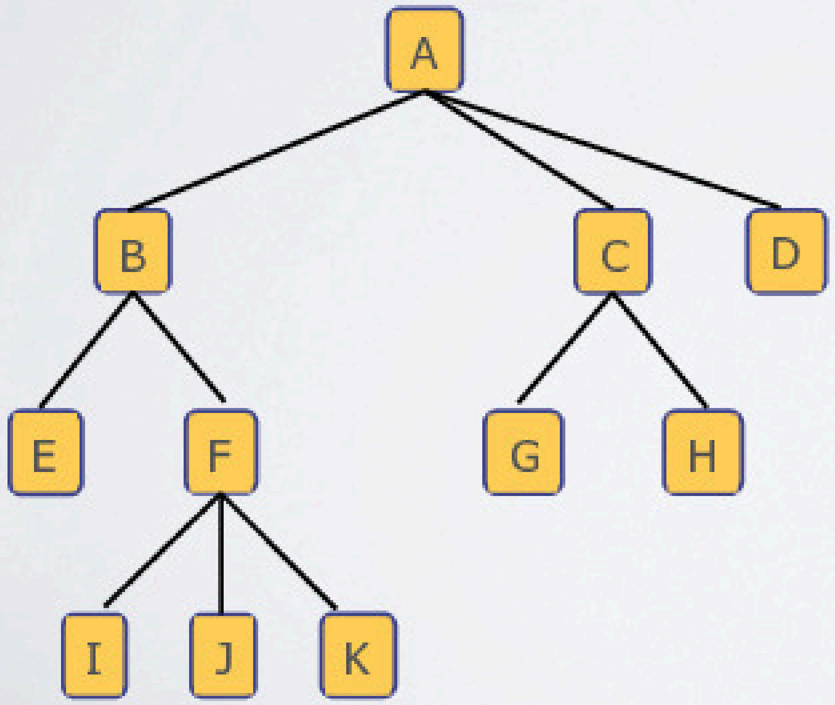
CS16: Introduction to Data Structures & Algorithms
Spring 2020

Clean Your Code!

- Errors per line is approximately constant
  - fewer lines → fewer errors overall
- Fewer lines are easier to grade
  - more likely to receive credit
- Clean code reflects clean thinking
  - and a better understanding of material
- Let's see some examples

Lowest Common Ancestor

- Given two nodes **u** and **v**
  - determine deepest node that is ancestor of both



u	**v**	**LCA(u, v)**
C	D	A
J	E	B
G	J	A
G	C	C

Lowest Common Ancestor

Activity #1

3 min

Lowest Common Ancestor

Activity #1

3 min

Lowest Common Ancestor

Activity #1

2 min

Lowest Common Ancestor

Activity #1

1 min

Lowest Common Ancestor

Activity #1

0 min

Ways to Improve Pseudo-Code

- Clarify inputs and outputs with comments
    - good habit and makes methods easier to understand
- Make sure all necessary arguments are included as parameters

Lowest Common Ancestor

```
function LCA(u, v):

   lca = null
   udepth = T.depth(u)
   vdepth = T.depth(v)
   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if udepth > vdepth then
         u = T.parent(u)
         udepth = udepth – 1
      else if vdepth > udepth
         v = T.parent(v)
         vdepth = vdepth – 1
      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth – 1
   return lca
```

Inputs & outputs ?

Lowest Common Ancestor

```
function LCA(u, v):
   // Input: two nodes u, v
   // Output: the lowest common ancestor of u and v
   lca = null
   udepth = T.depth(u)
   vdepth = T.depth(v)
   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if udepth > vdepth then
         u = T.parent(u)
         udepth = udepth – 1
      else if vdepth > udepth
         v = T.parent(v)
         vdepth = vdepth – 1
      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth – 1
   return lca
```

Lowest Common Ancestor

```
function LCA(u, v):
   // Input: two nodes u, v
   // Output: the lowest common ancestor of u and v
   lca = null
   udepth = T.depth(u)
   vdepth = T.depth(v)
   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if udepth > vdepth then
         u = T.parent(u)
         udepth = udepth – 1
      else if vdepth > udepth
         v = T.parent(v)
         vdepth = vdepth – 1
      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth – 1
   return lca
```

Where does T come from?

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   udepth = T.depth(u)
   vdepth = T.depth(v)
   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if udepth > vdepth then
         u = T.parent(u)
         udepth = udepth – 1
      else if vdepth > udepth
         v = T.parent(v)
         vdepth = vdepth – 1
      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth – 1
   return lca
```

Ways to Improve Pseudo-Code

- Get rid of unnecessary variables
- Using vars for information that is elsewhere…
  - …leads to careless errors
- In example, no need for udepth and vdepth
  - since Tree keeps track of node's depth

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   udepth = T.depth(u)
   vdepth = T.depth(v)
   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if udepth > vdepth then
         u = T.parent(u)
         udepth = udepth – 1
      else if vdepth > udepth
         v = T.parent(v)
         vdepth = vdepth – 1
      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth – 1
   return lca
```

Needlessly complex

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   udepth = T.depth(u)  \
   vdepth = T.depth(v)  /  ------------------- Now irrelevant
   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)

      else if T.depth(v) > T.depth(u)
         v = T.parent(v)

      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth – 1
   return lca
```

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null



   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)

      else if T.depth(v) > T.depth(u)
         v = T.parent(v)

      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

Ways to Improve Pseudo-Code

- If method returns boolean
  - no need to check if returned value `==true`
- Logical operators can be used on boolean returned valued
  - `!T.isroot(u)` is same as `T.isroot(u)==false`

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null


   if (T.isroot(u) == true) or (T.isroot(v) == true) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)

      else if T.depth(v) > T.depth(u)
         v = T.parent(v)

      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

Redundant equality checks

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null



   if T.isroot(u) or T.isroot(v) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)

      else if T.depth(v) > T.depth(u)
         v = T.parent(v)

      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   if T.isroot(u) or T.isroot(v) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
      return lca
```

Just removed whitespace

Ways to Improve Pseudo-Code

- As soon as you found answer, return it
  - This avoids going through unnecessary code

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   if T.isroot(u) or T.isroot(v) then
      lca = T.root
   while (lca == null) do
      if (u == v) then
         lca = u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

It's the answer. Return it!

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   if T.isroot(u) or T.isroot(v) then
      lca = T.root
      return lca
   while (lca == null) do
      if (u == v) then
         lca = u  <------------------------------------
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

It's the answer. Return it!

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   if T.isroot(u) or T.isroot(v) then
      lca = T.root
      return lca
   while (lca == null) do
      if (u == v) then
         lca = u
         return lca
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

Ways to Improve Pseudo-Code

- If variable is only used to return something
    - simply return it
- Avoids keeping track of unnecessary variables
    - and makes code shorter and cleaner

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   if T.isroot(u) or T.isroot(v) then
      lca = T.root
      return lca
   while (lca == null) do
      if (u == v) then
         lca = u
         return lca
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
   return lca
```

Condition is irrelevant

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   lca = null
   if T.isroot(u) or T.isroot(v) then
      lca = T.root
      return lca
   repeat
      if (u == v) then
         lca = u
         return lca
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
```

lca is no longer used

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v

   if T.isroot(u) or T.isroot(v) then

      return T.root
   repeat
      if (u == v) then

         return u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
```

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   if T.isroot(u) or T.isroot(v) then
      return T.root
   repeat
      if (u == v) then
         return u
      else if T.depth(u) > T.depth(v) then
            u = T.parent(u)
      else if T.depth(v) > T.depth(u)
            v = T.parent(v)
      else
         u = T.parent(u)
         v = T.parent(v)
```

Ways to Improve Pseudo-Code

- If you never enter a conditional in a while loop,
    - try to use two while loops to simplify
- If **u** is at lower depth than **v**, algorithm will not allow **u** to get to a higher depth than **v**
    - so unnecessary to check both within one while loop

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   if T.isroot(u) or T.isroot(v) then
      return T.root
   repeat
      if (u == v) then
         return u
      else if T.depth(u) > T.depth(v) then
         u = T.parent(u)
      else if T.depth(v) > T.depth(u)
         v = T.parent(v)
      else
         u = T.parent(u); udepth = udepth - 1
         v = T.parent(v); vdepth = vdepth - 1
```

Only one of these conditionals will ever be true

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   if T.isroot(u) or T.isroot(v) then
      return T.root
   repeat
      if (u == v) then
         return u
      else
         u = T.parent(u)
         v = T.parent(v)
```

Ways to Improve Pseudo-Code

- Avoid unnecessary conditional checks
- In example, after the two while loops,
    - **`u`** and **`v`** have same depth
    - if either is root, they both are the same root
- Try to be as concise as possible!

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   if T.isroot(u) or T.isroot(v) then
      return T.root
   repeat
      if (u == v) then
         return u
      else
         u = T.parent(u)
         v = T.parent(v)
```

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   if T.isroot(u) or T.isroot(v) or u == v then
      return u
   repeat
      if (u == v) then
         return u
      else
         u = T.parent(u)
         v = T.parent(v)
```

Ways to Improve Pseudo-Code

- After the two while loops,
  - **u** and **v** have same depth
  - if either is root, then they both are root
- Try to be as concise as possible!

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   if T.isroot(u) or T.isroot(v) or u == v then ◄--------
      return u
   repeat
      if (u == v) then
         return u
      else
         u = T.parent(u)
         v = T.parent(v)
```

Can be simplified

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   if u == v then
      return u
   repeat
      if (u == v) then
         return u
      else
         u = T.parent(u)
         v = T.parent(v)
```

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   if u == v then
      return u
   repeat
      if (u == v) then
         return u
      else
         u = T.parent(u)
         v = T.parent(v)
```

Condense into a single loop

Lowest Common Ancestor

```
function LCA(u, v, T):
   // Input: two nodes u, v in a tree T
   // Output: the lowest common ancestor of u and v
   while T.depth(u) > T.depth(v)
      u = T.parent(u)
   while T.depth(v) > T.depth(u)
      v = T.parent(v)
   while u != v then
      u = T.parent(u)
      v = T.parent(v)
   return u
```

From clunky 19 lines to elegant 8 lines!

Improve Pseudo-Code

- Now that you have seen how easily pseudocode can be simplified…
  - …you are expected to make similar improvements to your pseudo-code
- Good pseudo-code is both accurate & concise