Recursion on Trees

CMPT - 225

Recursion: A **definition** of a function is recursive if the body contains an application of itself.

Consider: $$S(n) = \sum_{i=0}^{n} i$$

or: $$S(n) = \begin{cases} 0 & \text{if } n = 0 \\ n + S(n-1) & \text{if } n > 0 \end{cases}$$

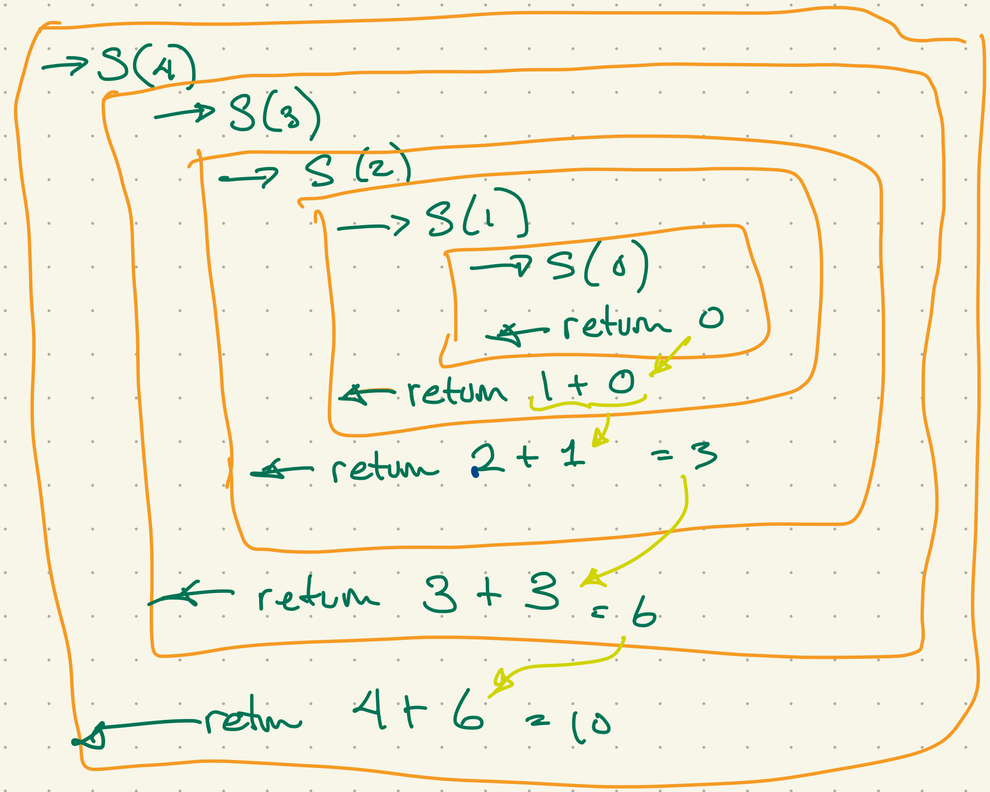
These two descriptions of $S(n)$ suggest two implementations:

Eg

```
S(n){
    s = 0
    for i = 1..n
        s = s + i
    return s
}
```

```
S(n){
    if n = 0
        return 0
    else
        return n + S(n-1)
}
```

Recursive Version:



Iterative Version: $s = 0 + \sum_{i=1}^{n} i = 1 + \sum_{i=2}^{n} i = 3 + \sum_{i=3}^{n} i = \dots$

$$= 0 + 1 + 2 + 3 + 4$$

- The <u>same computation</u>, but a different control strategy.

Recursion & The Call Stack

```
S(n){
  if(n=0){
    r=0
  }else{
    r=n+S(n-1)
  }
  return r
}
```

```
P(){
  :
  S(2)
* :
```

Compute S(2)

→ Call S(2)
n = 2
r = ?

→ Call S(1)
n = 1
r = ? → Call S(0)
(1+S(0))

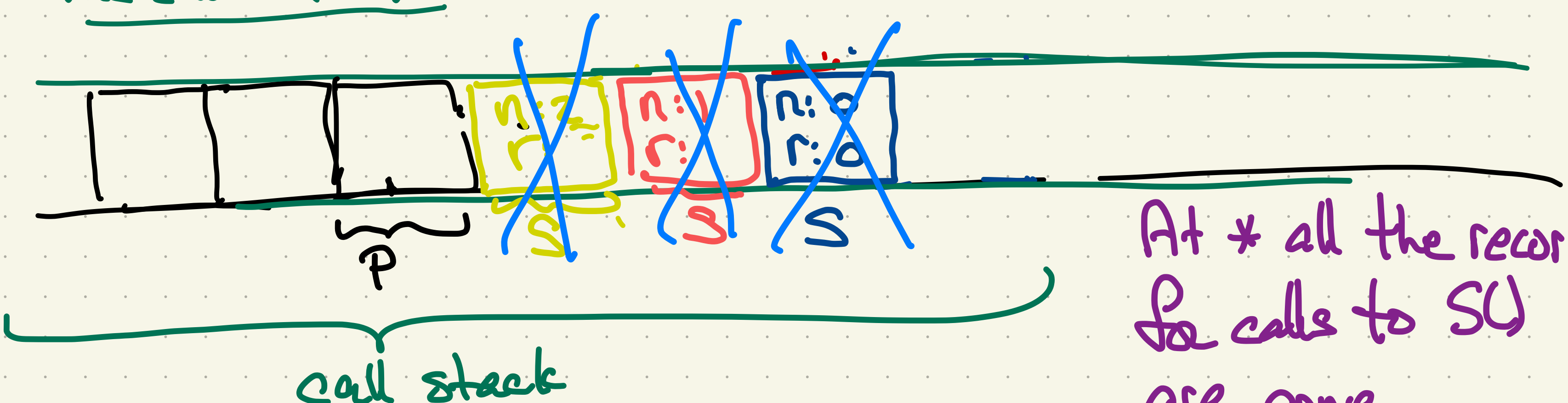
n=0
r=0
return 0

r = 1 + S(0)
return 1

return 2+S(1)

returns 3

The call stack:



At * all the records for calls to S() are gone

Recursion & The Call Stack

```
S(n){
  if(n=0){
    r=0
  }else{
    r=n+S(n-1)
  }
  return r
}
```

```
P(){
  :
  S(2)
  :
```

Compute S(2)

Call S(2)
n = 2
r = ? → Call S(1)
n = 1
r = ? → Call S(0)
1+S(0)
n=0
r=0
return 0
r = 1 + S(0)
return 1
return 2+S(1)
returns 3

The call stack:

```
S(n){
  if(n=0){
    r=0
  }else{
    r=n+S(n-1)
  }
  return r
}
```

```
P(){
  ⋮
  S(2)
* ⋮
}
```

Compute S(2)

→ Call S(2)

n = 2

r = ? → Call S(1)

n = 1

r = ? → Call S(0)

1+S(0)

n=0

r=0

return 0

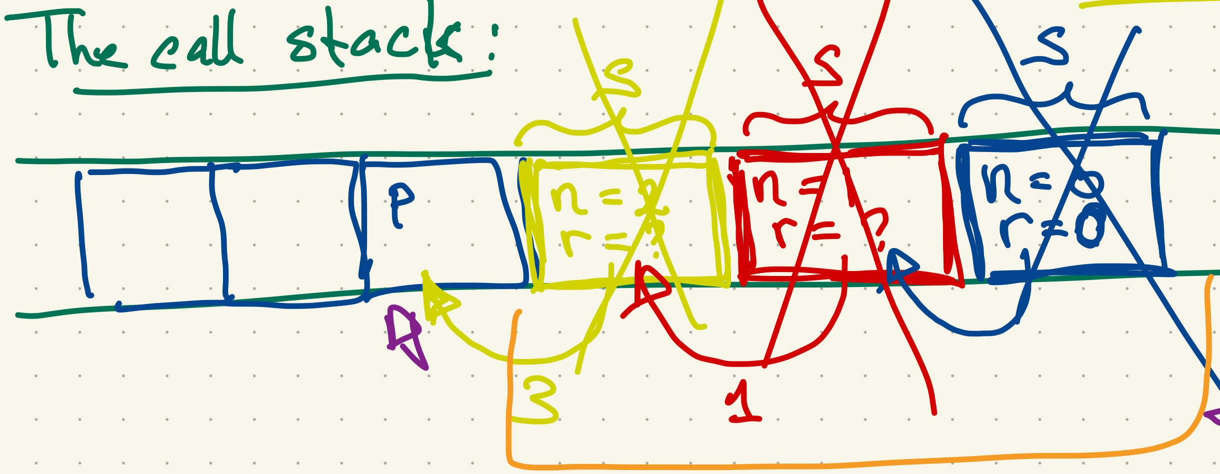
r = 1+S(0)

return 1

return 2+S(1)

returns 3

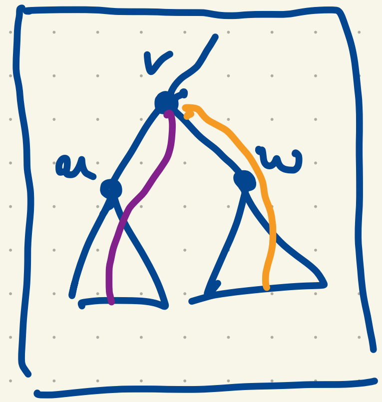
The call stack:



At *, all this is gone.

Recursion on Trees

- We will often use recursion & induction on trees.
  - eg. the tree rooted a v has some property if its subtrees have some (related) property.



- Eg: The height of a node v in a binary tree may be defined by:

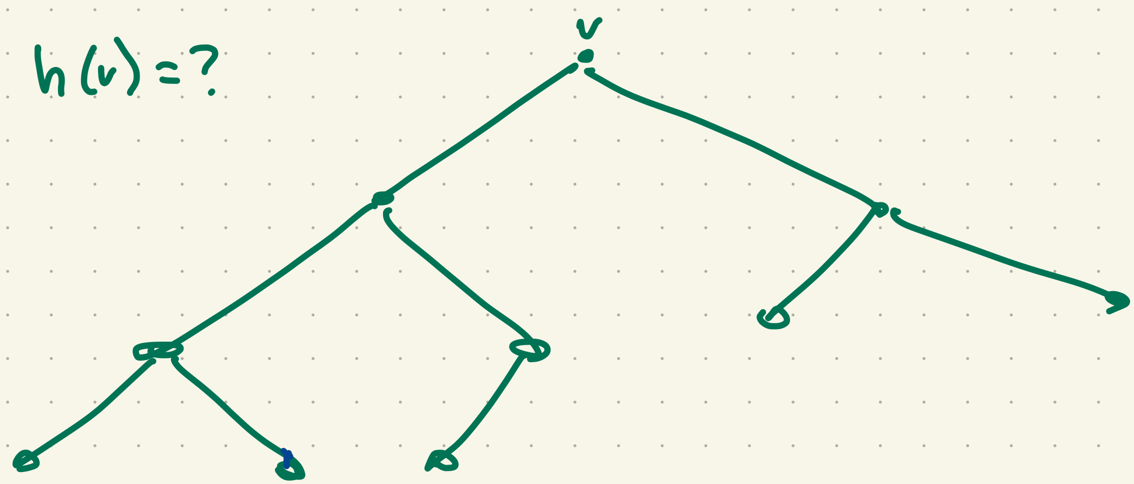
$$h(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ 1 + \max\{h(\text{left}(v)), h(\text{right}(v))\} & \text{o.w.} \end{cases}$$

(We can define $h(\text{left}(v))$ to be -1 if $\text{left}(v)$ does not exist, and sim. for $\text{right}(v)$).

Recursion on Trees Example

height of node v in T:

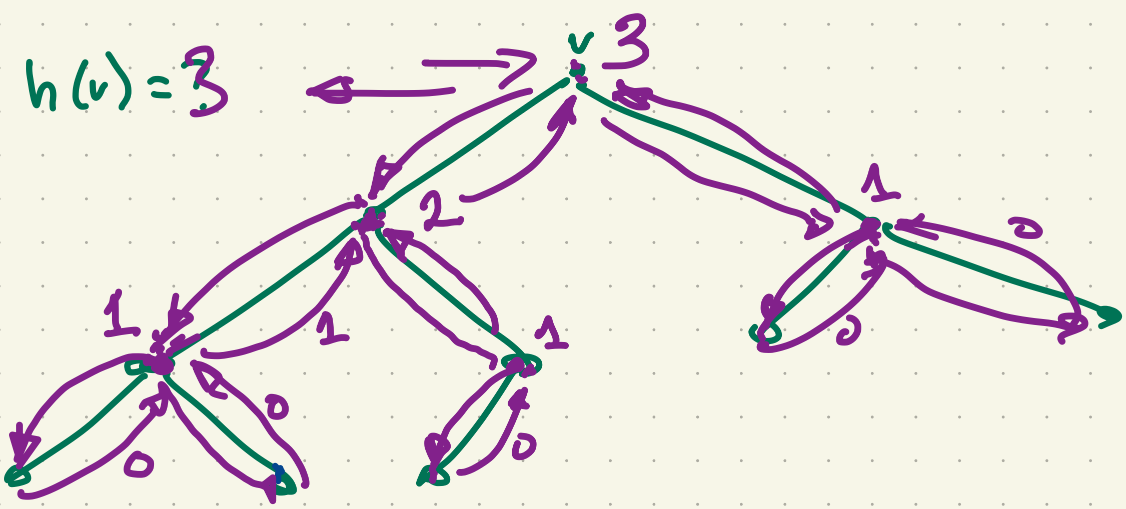
$$h(v) = \begin{cases} 0 & \text{if } v \text{ a leaf} \\ 1 + \max\{h(\text{left}(v)), h(\text{right}(v))\}, & \text{ow}. \end{cases}$$



Recursion on Trees Example

height of node v in T:

$$h(v) = \begin{cases} 0 & \text{if } v \text{ a leaf} \\ 1 + \max\{h(\text{left}(v)), h(\text{right}(v))\}, & \text{ow}. \end{cases}$$



Pseudo-code version

```
height(v){
    if v is a leaf
        return 0
    if v has one child u
        return 1+ height(u)
    else
        return 1+ max(height(left(v)),
                      height(right(v)))
}.
```

Traversals of Binary Trees

- A traversal of a graph is a process that "visits" each node in the graph once.
- We consider 4 standard tree traversals:

  1. level order
  2. pre-order
  3. in-order
  4. post-order.

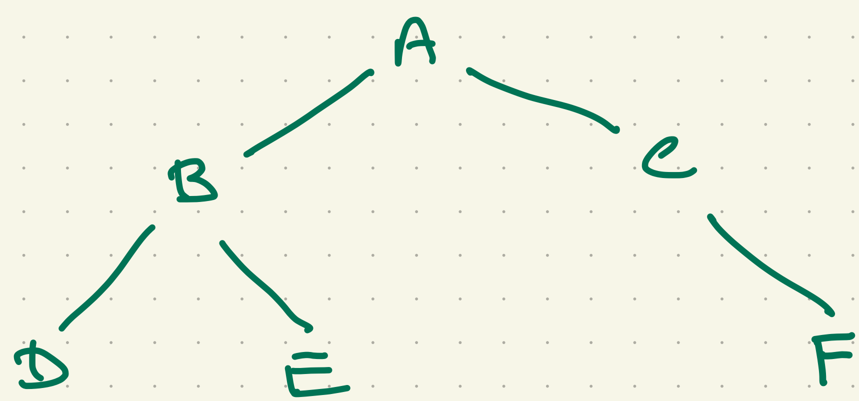
2,3,4 begin at the root & recursively visit the nodes in each subtree & the root. They vary in the relative order.

(Level order later).

```
pre-order-T(v){
    visit v
    pre-order-T(left(v))
    pre-order-T(right(v))
}
```

pre-order-T(v) does nothing if v does not exist.

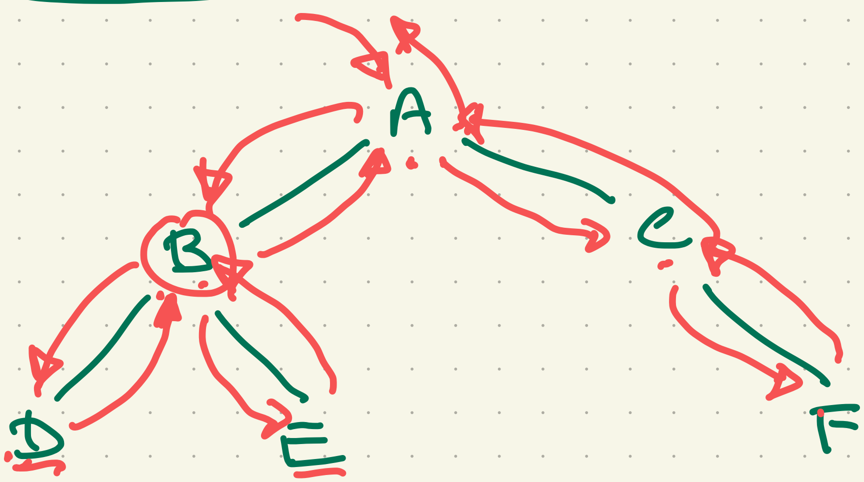
- v is visited before any of its descendants
- every node in the left subtree is visited before any node in the right subtree.



```
pre-order-T(v){
    visit v
    pre-order-T( left(v))
    pre-order-T (right(v))
}
```

$\boxed{\text{pre-order-T (r) does nothing if r does not exist.}}$

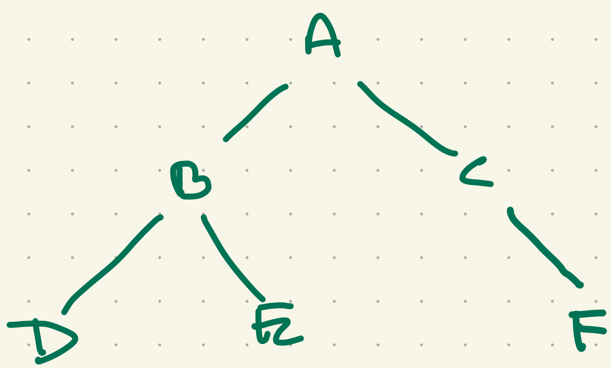
- v is visited before any of its descendants
- every node in the left subtree is visited before any node in the right subtree.

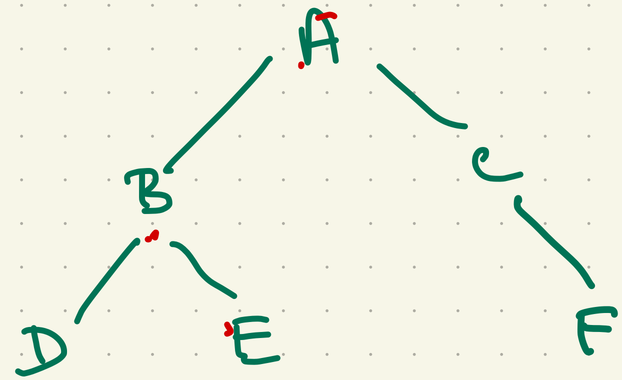


A, B, D, E, C, F

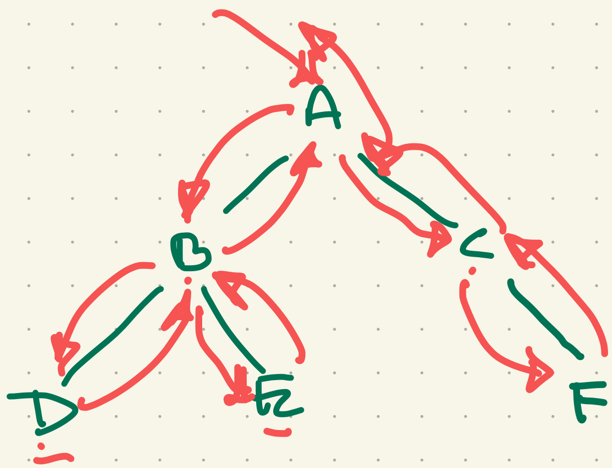
```
In-order-T(v){
    in-order-T( left(v))
  - visit v
    in-order-T( right(v))
}
```

```
post-order-T(v){
    post-order-T(left(v))
    post-order-T(right(v))
  - visit v
}
```



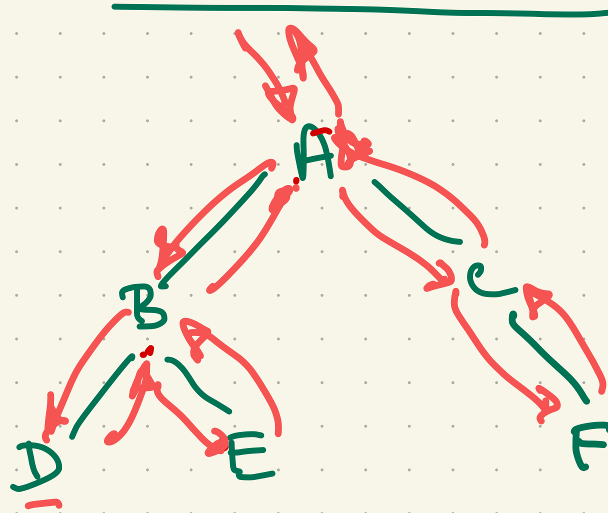


```
In-order-T(v){
    - in-order-T( left(v))
    - visit v
      in-order-T( right(v))
}
```



D, B, E, A, C, F

```
post-order-T(v){
    post-order-T(left(v))
    post-order-T(right(v))
  - visit v
}
```



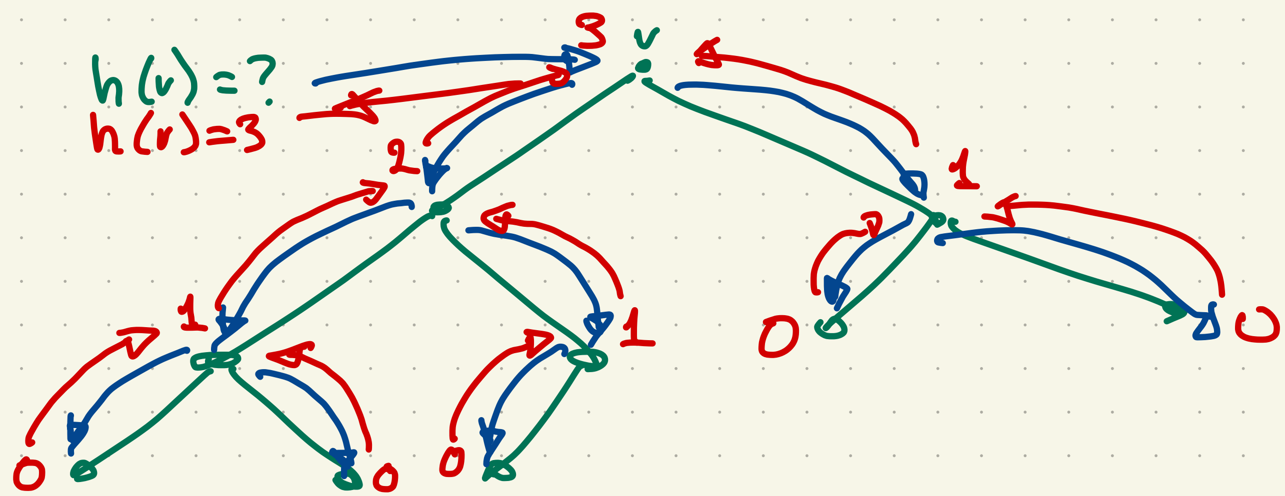
D, E, B, F, C, A.

End

Recursion on Trees Example

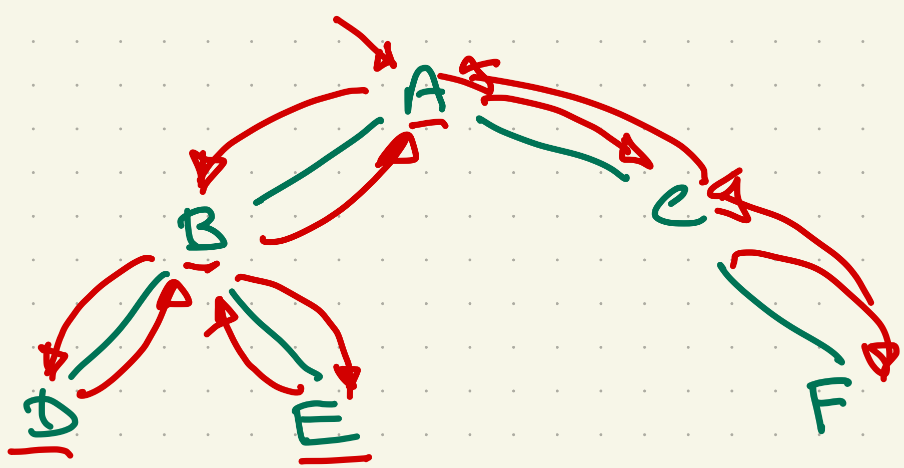
height of node v in T:

$$h(v) = \begin{cases} 0 & \text{if } v \text{ a leaf} \\ 1 + \max\{h(\mathrm{left}(v)), h(\mathrm{right}(v))\}, & \text{ow}. \end{cases}$$



```
pre-order-T(v){
    visit v
    pre-order-T(left(v))
    pre-order-T(right(v))
}
```

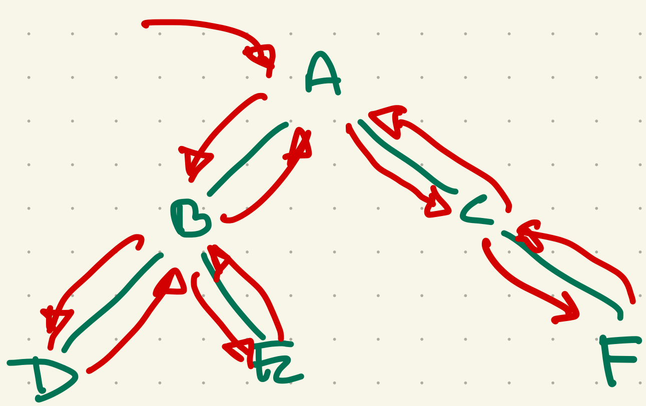
- v is visited before any of its descendats
- every node in the left subtree is visited before any node in the right subtree.

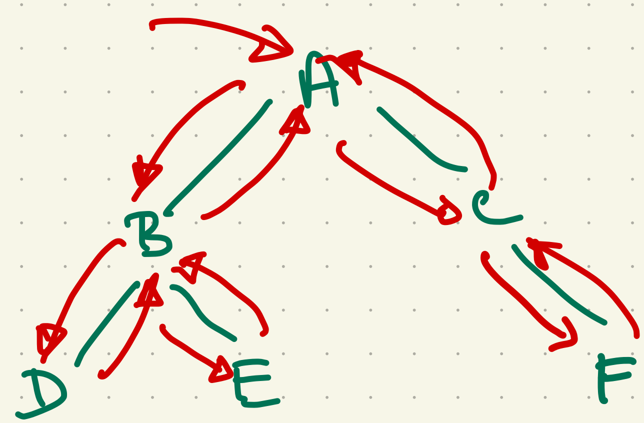


A, B, D, E, C, F

```
In-order-T(v){
    in-order-T(left(v))
  - visit v
    in-order-T(right(v))
}
```

```
post-order-T(v){
    post-order-T(left(v))
    post-order-T(right(v))
  - visit v
}
```





D, B, E, A, C, F

D, E, B, F, C, A