Introduction to Algorithms

CELEN086

Seminar 7
(w/c 25/11/2024)

Outline

In this seminar, we will study and review on following topics:

- Basic tree concepts
- Commands for binary trees
- Designing recursive algorithms on binary trees
- Binary tree with minimal height/depth

You will also learn useful Math/CS concepts and vocabularies.

Tree basics

Size of tree: 10

Height (depth) of tree: 3

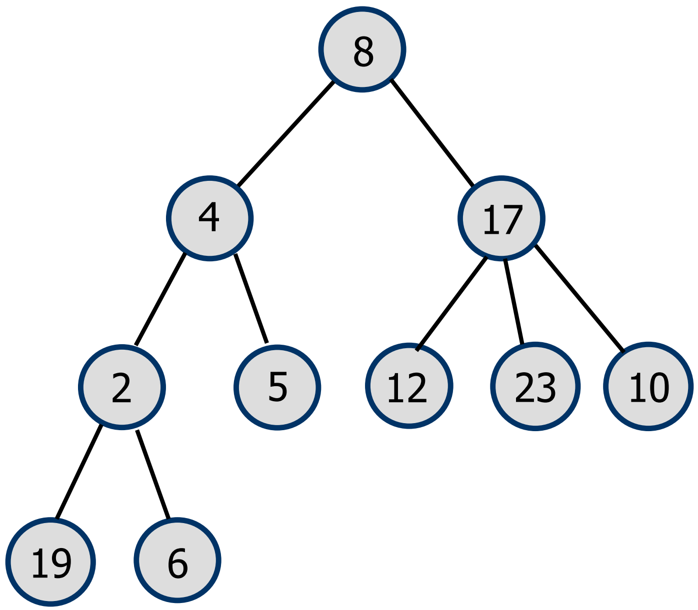
Depth of node with value 4: 1

Height of node with value 4: 2

Leaf nodes: 19 6 5 12 23 10

Binary tree? Why?

No. The node with value 17 has degree 3.



Binary tree (vs. list) commands

[tree commands]

- leaf
- node(leaf, x, leaf)
- node(left-subtree, x, right-subtree)
- isLeaf(tree)
- root(tree)
- left(tree), right(tree)
- isLeaf(left(tree))&&isLeaf(right(tree))

to check single-element tree
(leaf node)

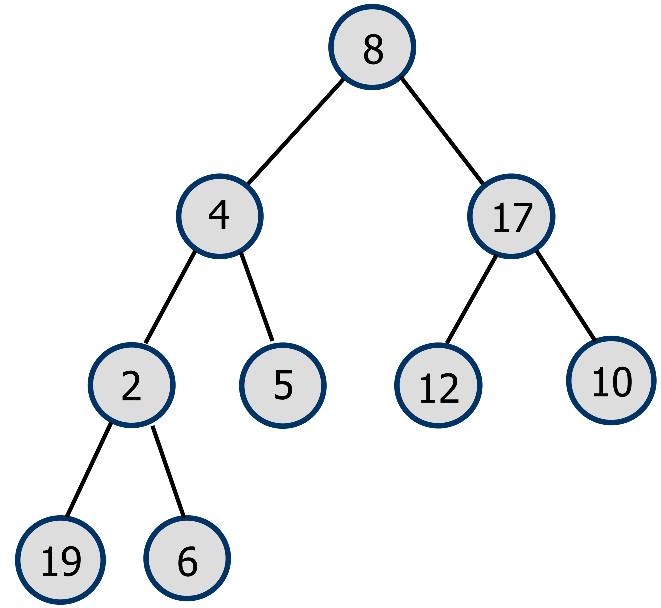
[list commands]

- nil
- cons(x, nil)
- cons(x, list)
- isEmpty(list)
- value(list)
- tail(list)
- isEmpty(tail(list))

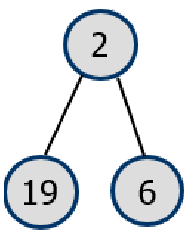
to check single-element list

Binary tree

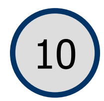
Let T be the given binary tree.



Write the pseudo code for:

i. 

ii. 12

iii. 

i. left(left(T))

ii. root(left(right(T)))

iii. right(right(T))

vi. Draw the following tree:

node(node(node(leaf,19,leaf),2,node(leaf,4,node(leaf,6,leaf))),5,leaf)

Is it a subtree of T?

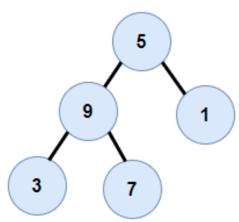
Algorithm: tree size

Is the following alternative algorithm for finding the tree size correct?

```
if  isLeaf(left(T))&& isLeaf(right(T))
      return  1
else
      return  size(left(T))+size(right(T))+1
endif
```

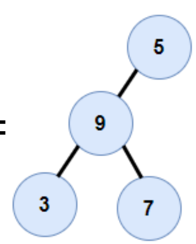
```
Algorithm: size(T)
Requires: a binary tree T
Returns: total number of nodes in T (size of T)

if  isLeaf(T)
      return  0
else
      return    size(left(T))+size(right(T))+1
endif
```

Trace it with T=



Working well.

Trace it with T=



Missing base case.

Algorithm: tree size (alternative ver.)

Algorithm: size(T)
Requires: a binary tree T
Returns: total number of nodes in T (size of T)

```
if  isLeaf(left(T))&& isLeaf(right(T))
    return  1  // base case
elseif  isLeaf(left(T))
    return  1+size(right(T))  // recursive call case#1
elseif  isLeaf(right(T))
    return  1+size(left(T)) // recursive call case#2
else
    return  1+size(left(T))+size(right(T)) // recursive call case#3
endif
```

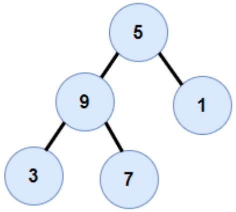
Practice: tree height

Write a recursive algorithm called height(T) that compute the height of a non-empty binary tree.
You can call the max() function to compute the maximum of two values.

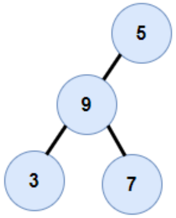
Algorithm: height(T)
Requires: a non-empty binary tree T
Returns: height of T

```
if  isLeaf(left(T)) && isLeaf(right(T))
    return  0
elseif  isLeaf(left(T))
    return  1+height(right(T))
elseif  isLeaf(right(T))
    return  1+height(left(T))
else
    return  1+ max(height(left(T)), height(right(T)))
endif
```

Trace it with

T =


and

T =


Practice: delete leaf node

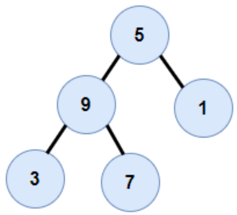
Write a recursive algorithm called delete(T) that deletes all the leaf nodes in a binary tree.

Algorithm: delete(T)
Requires: a non-empty binary tree T
Returns: a tree after deletions of all leaf nodes in T

```
if  isLeaf(left(T)) && isLeaf(right(T))
     return  leaf
elseif  isLeaf(left(T))
     return  node(leaf, root(T), delete(right(T)))
elseif  isLeaf(right(T))
     return  node(delete(left(T)),root(T),leaf)
else
     return  node(delete(left(T), root(T), delete(right(T)))
endif
```

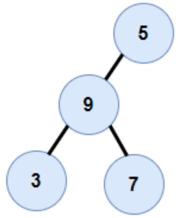
Trace it with

T =



and

T =

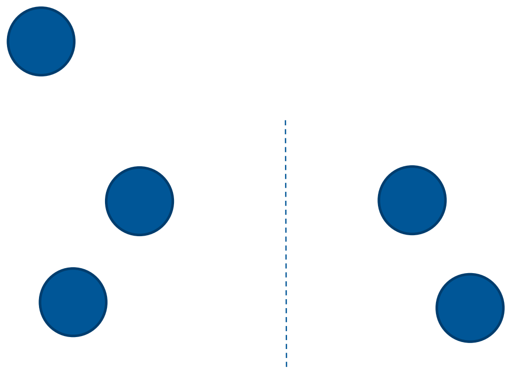


Binary tree with minimal height

How many binary trees can be made with n (same) nodes?

n=1

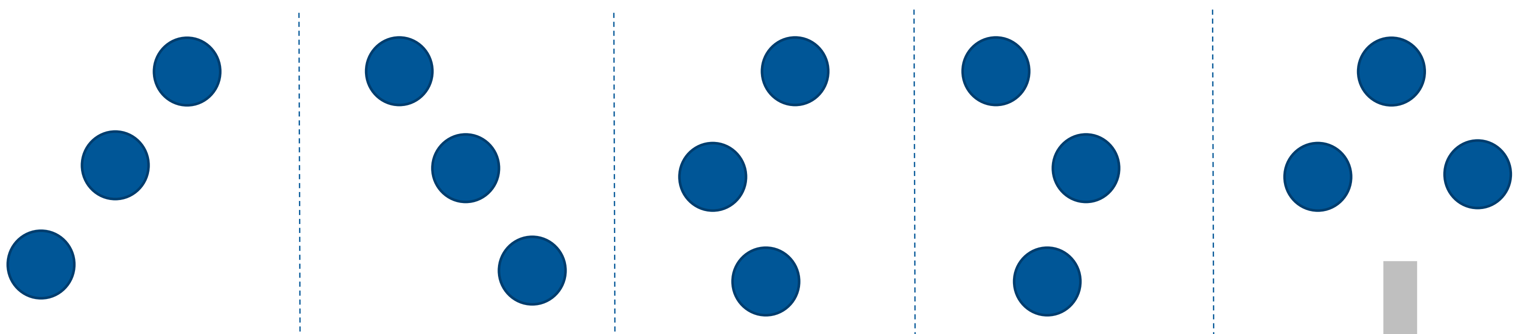
n=2



In general:

$$\frac{(2n)!}{n!\,(n+1)!}$$

n=3

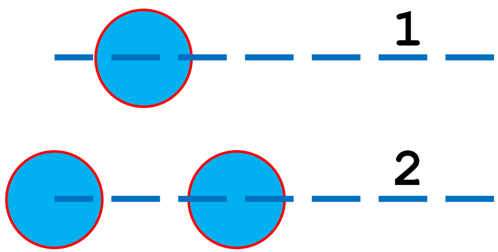


Binary tree with minimal height/depth.

Binary tree: size vs. height

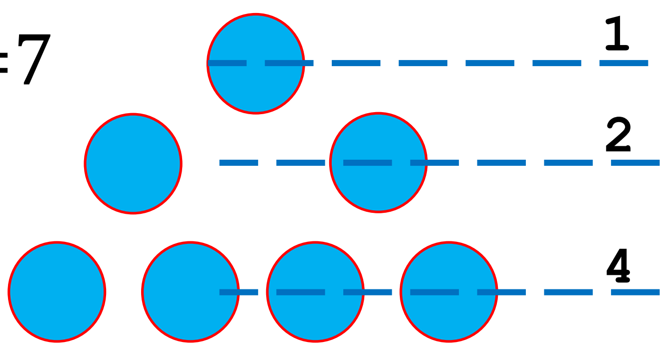
What is the minimal height of a binary tree with n nodes? $h = \lfloor \log_2 n \rfloor$

n=3



$3 = 2^0 + 2^1$ $h = 1$

n=7



$7 = 2^0 + 2^1 + 2^2$ $h = 2$

For a binary tree with height h, maximum number of nodes can be stored is:

$$2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1$$