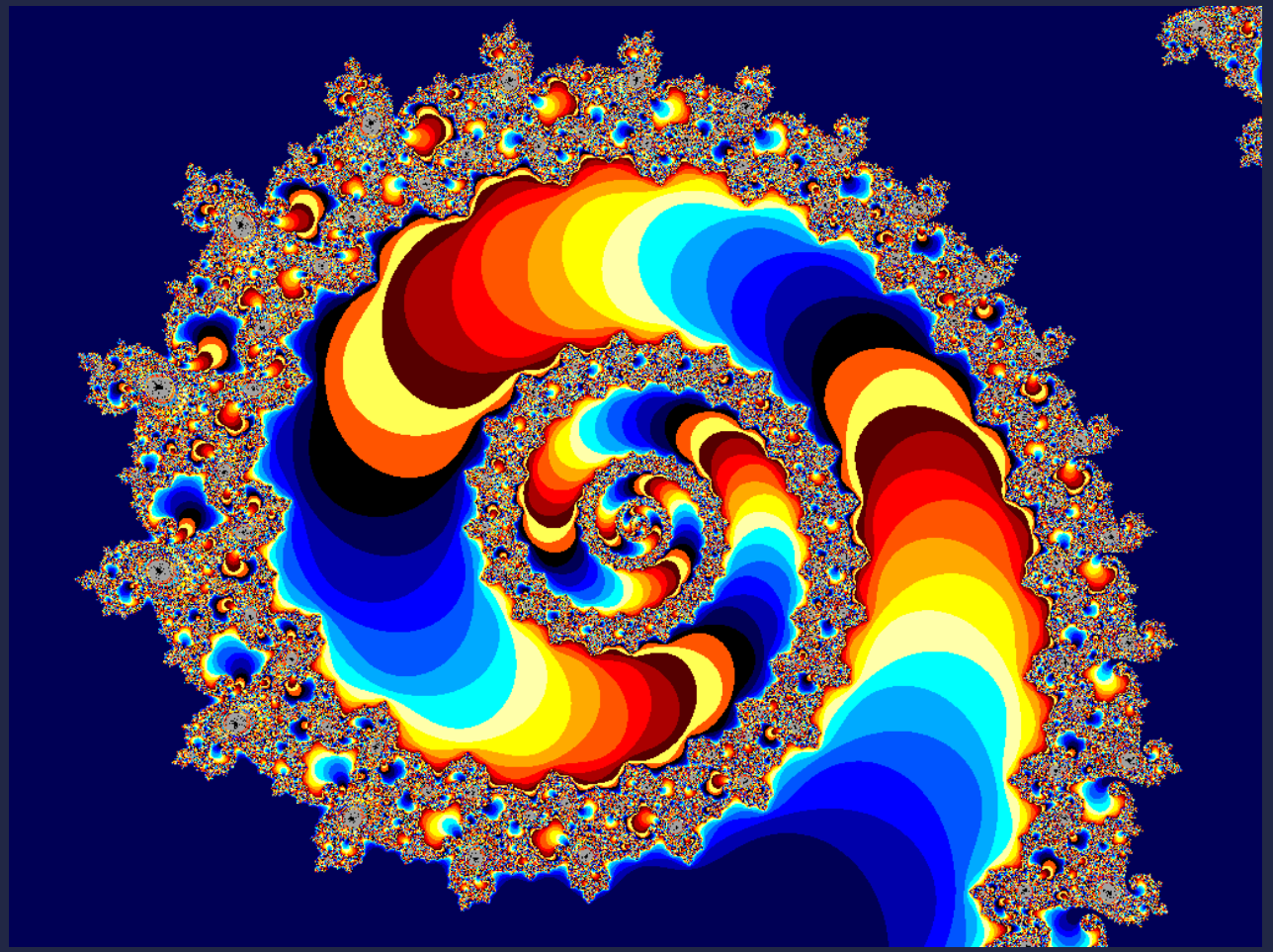
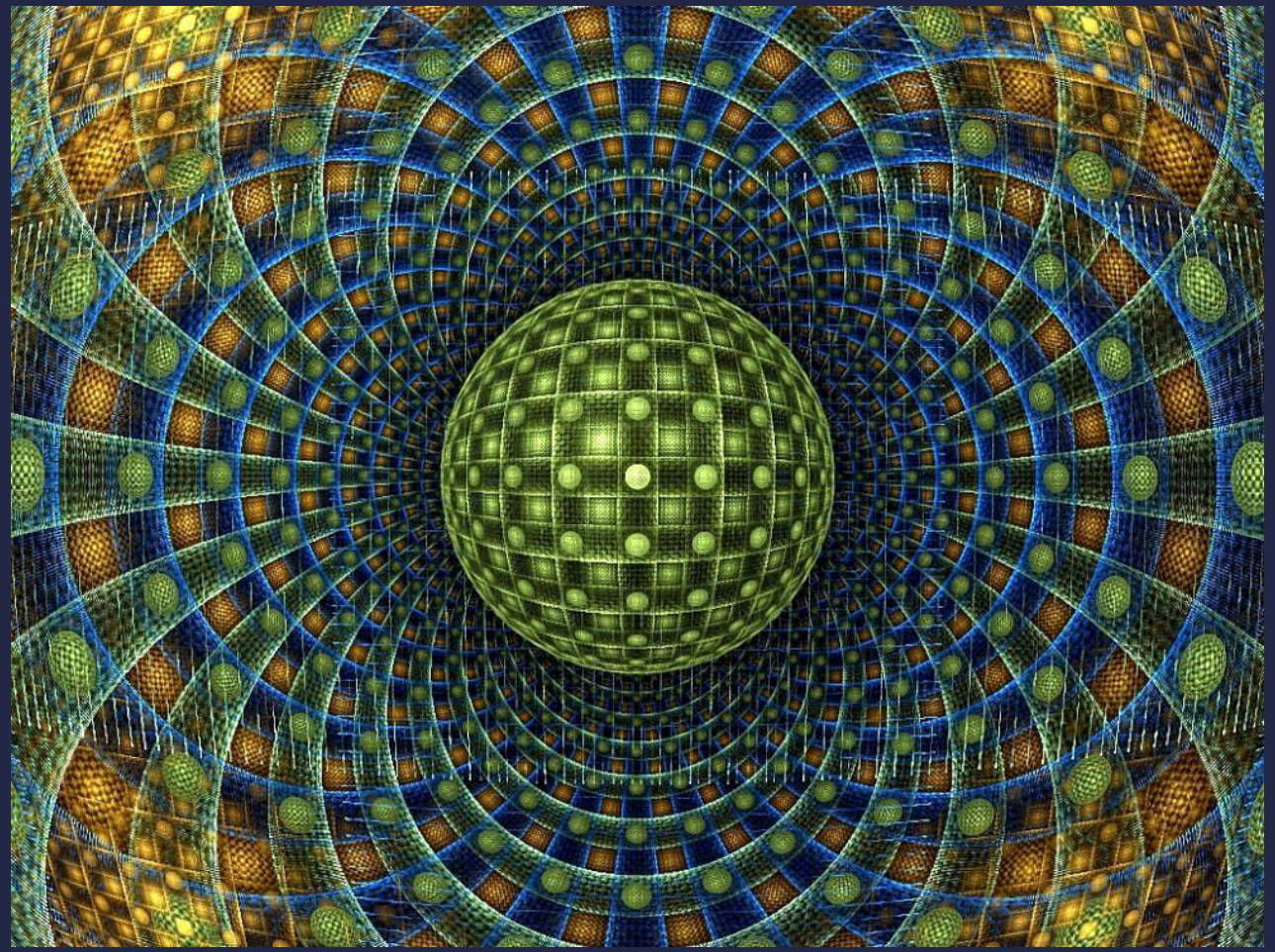
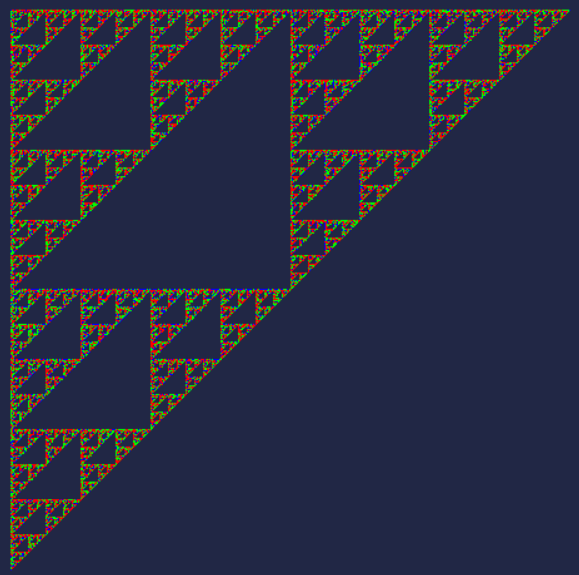
5.3 Recursive Definitions and Structural Induction

Credit
Danielle Bellavance, Max Welling,
Richard Scherl, Ching-Shoei Chiang
Paul Beame, James Lee,
Luse Cheng, Majed AlHassan,
Husni Al-Muhtaseb





Fractals



a fractal is “a fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole".

Startup

Examine each sequence and tell how you would compute the 20^{th} term if you were given the 19^{th} term:

1) 6, 15, 24, 33, … $t_{20} = t_{19} + 9$ $t_n = t_{n-1} + 9$

2) 5, 10, 20, 40, … $t_{20} = 2 \cdot t_{19}$ $t_n = 2 \cdot t_{n-1}$

3) 1, 2, 6, 24, 120, … $t_{20} = 20 \cdot t_{19}$ $t_n = n \cdot t_{n-1}$

4) –1, –3, –5, –7, … $t_{20} = t_{19} - 2$ $t_n = t_{n-1} - 2$

5) 1, 3, 7, 15, 31, … $t_{20} = 2 \cdot t_{19} + 1$ or $t_{20} = t_{19} + 2^{19}$

$t_n = 2.t_{n-1} + 1$ or $t_n = t_{n-1} + 2^{n-1}$

6) 1, 2, 8, 64, 1024, … $t_{20} = 2^{19} \cdot t_{19}$ $t_n = 2^{n-1} t_{n-1}$

These example computations involve *recursion* – defining something in terms of itself / relying on what has preceded.

Explicit vs. Recursive Definitions

Previously, most sequence definitions were explicit. You can find the value of any term given the formula.

Ex: Find the 1st six terms and the 100th term of the sequence in which $t_n = n^2 - 1$

$t_1 = 1^2 - 1 = 0$ $t_2 = 2^2 - 1 = 3$ $t_3 = 3^2 - 1 = 8$

$t_4 = 4^2 - 1 = 15$ $t_5 = 5^2 - 1 = 24$ $t_6 = 6^2 - 1 = 35$

$t_{100} = 100^2 - 1 = 9{,}999$

Explicit vs. Recursive Definitions

A recursive definition of a sequence has 2 parts:

- An initial condition that tells how the sequence starts.
- A recursive formula that tells how any term in the sequence is related to the preceding term(s).

a.k.a.
"seed" value

Explicit vs. Recursive Definitions

Ex: Find the 1st six terms and the 100th term of the sequence in which $t_1 = 3$ and $t_n = t_{n-1} + 2$ for all $n > 1$

$t_1 = 3$ $t_2 = t_1 + 2 = 5$ $t_3 = t_2 + 2 = 7$

$t_4 = t_3 + 2 = 9$ $t_5 = t_4 + 2 = 11$ $t_6 = t_5 + 2 = 13$

$t_{100} = t_{99} + 2 = ??$

Finding the 100th term requires the prior, 99th term!

In this particular case, since this is an arithmetic sequence with $t_1 = 3$ and $d = 2$, we can deduce the explicit formula.

$$t_{100} = 3 + (100 - 1)\,2 = 201$$

Explicit vs. Recursive Definitions

- It is generally straight-forward to convert between explicit and recursive sequence definitions:

Arithmetic:

Recursive:

$t_1 = 3,\ t_n = t_{n-1} + 4,\ n > 1$

$\rightarrow 3, 7, 11, \ldots.\quad (d = 4)$

to Explicit:

$t_n = t_1 + (n - 1)d = 3 + (n - 1)(4)$

$t_n = 4n - 1$

Geometric:

Explicit: $t_n = 3(2)^n$

$(t_1 = 6,\ r = 2)$

to Recursive:

$t_1 = 6,\ t_n = 2 \cdot t_{n-1},\ n > 1$

- Some sequences can only be defined recursively; e.g., The Fibonacci Sequence: $\underbrace{t_1 = 1,\ t_2 = 1}_{\text{initial condition}},\ \underbrace{t_n = t_{n-2} + t_{n-1},\ n > 2}_{\text{recursion formula}}$

Recursive Definitions

- Recursive definitions are
  - important tools in modeling
  - easier to set up since they reflect what is directly observed
  - widely used in computer programming

Recursive Definitions

- EXAMPLE: The population of a certain country is currently 8.5 million and grows as a result of two conditions:
  - The annual growth rate for those currently living in the country is 2%
  - The net migration is 50,000 people into the country every year

  Give a recursive definition for the population in *n* years.

  $P_0 = 8.5 \text{ X } 10^6, \; P_n = P_{n-1}(1 + .02) + 50{,}000, \; n > 0$

  What will the population be in 5 years from now?

  $P_1 = (8.5 \text{ X } 10^6)(1.02) + 50{,}000 = 8.720 \text{ X } 10^6$

  $P_2 = P_1(1.02) + 50{,}000$

  $P_2 = (8.720 \text{ X } 10^6)(1.02) + 50{,}000 = 8.944 \text{ X } 10^6$

  $P_3 = P_2(1.02) + 50{,}000$

  $P_3 = (8.944 \text{ X } 10^6)(1.02) + 50{,}000 = 9.173 \text{ X } 10^6$

  $P_4 = (9.173 \text{ X } 10^6)(1.02) + 50{,}000 = 9.407 \text{ X } 10^6$

  $P_5 = (9.407 \text{ X } 10^6)(1.02) + 50{,}000 = 9.645 \text{ X } 10^6$

Luckily we did not ask for 20 years out!

Recursively Defined Functions

Example:

Suppose f is defined by:

$f(0) = 3$,

$f(n + 1) = 2f(n) + 3$

Find $f(1)$, $f(2)$, $f(3)$, $f(4)$

Solution:

$f(1) = 2f(0) + 3 = 2 \times 3 + 3 = 9$

$f(2) = 2f(1) + 3 = 2 \times 9 + 3 = 21$

$f(3) = 2f(2) + 3 = 2 \times 21 + 3 = 45$

$f(4) = 2f(3) + 3 = 2 \times 45 + 3 = 93$

Recursively Defined Functions

Example:

Give a recursive definition of the factorial function $n!$.

Solution:

$f(0) = 1$

$f(n + 1) = (n + 1) \times f(n)$

Recursively Defined Functions

Example:

Give a recursive definition of: $\sum_{k=0}^{n} a_k$

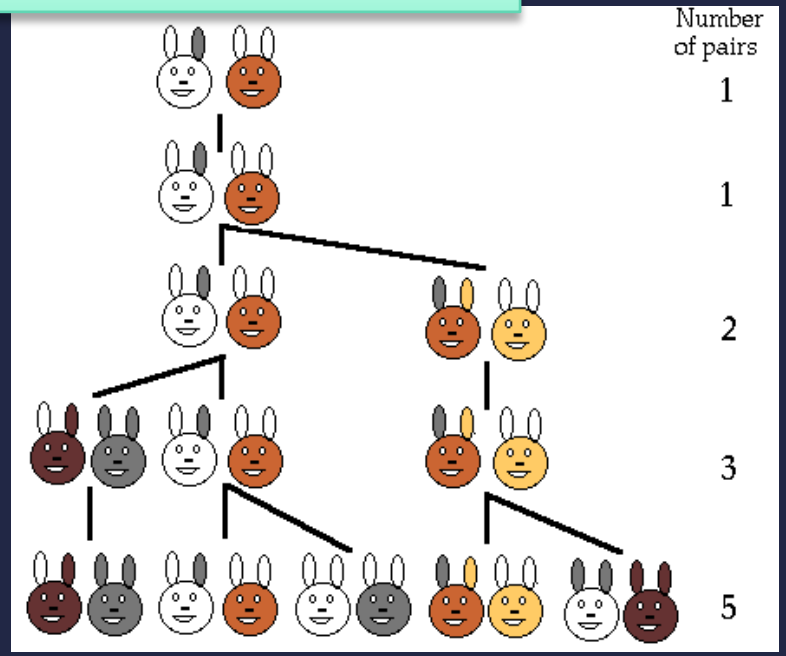
Solution:

The first part of the definition is $\sum_{k=0}^{0} a_k = a_0$

The second part is

$$\sum_{k=0}^{n+1} a_k = \left(\sum_{k=0}^{n} a_k\right) + a_{n+1}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...



Fibonacci numbers:

$f(0) = 0, f(1) = 1, f(n + 1) = f(n) + f(n - 1)$ for $n = 1, 2, 3, \ldots$

Or $f(n) = f(n - 1) + f(n - 2)$ for $n = 2, 3, 4, \ldots$

$f(2) = 1 + 0 = 1;$

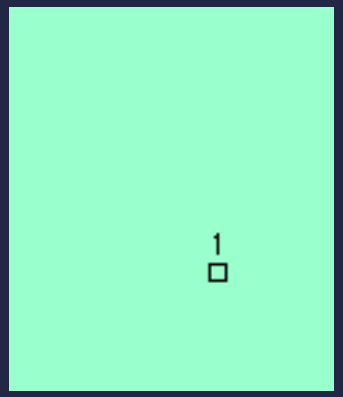
$f(3) = 1 + 1 = 2;$

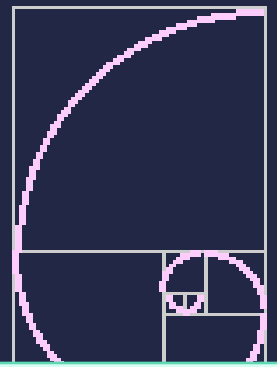
$f(4) = 2 + 1 = 3;$

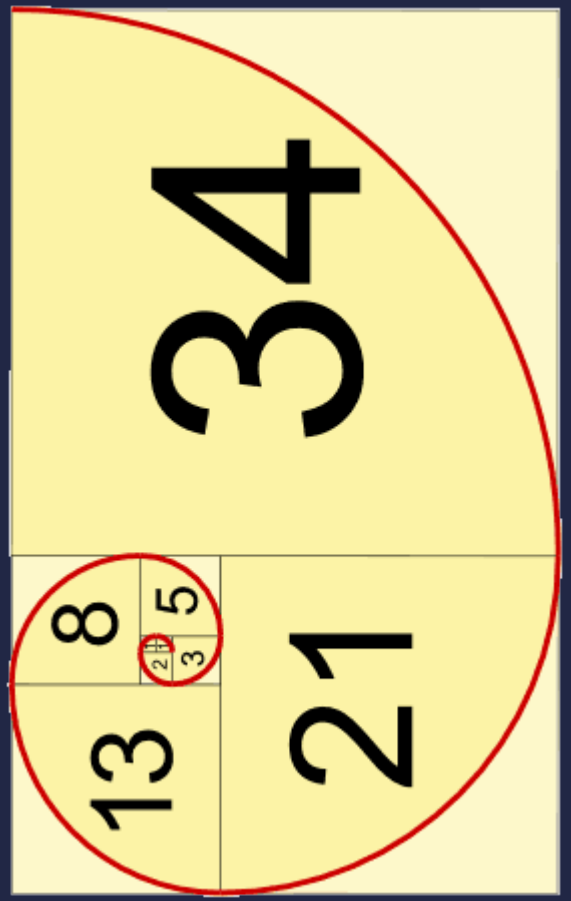
$f(5) = 3 + 2 = 5;$

Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits. Suppose that the female always produces one new pair (one male, one female) every month from the second month on. The puzzle that **Fibonacci** posed was: How many pairs do we have after one year?

More on Fibonacci

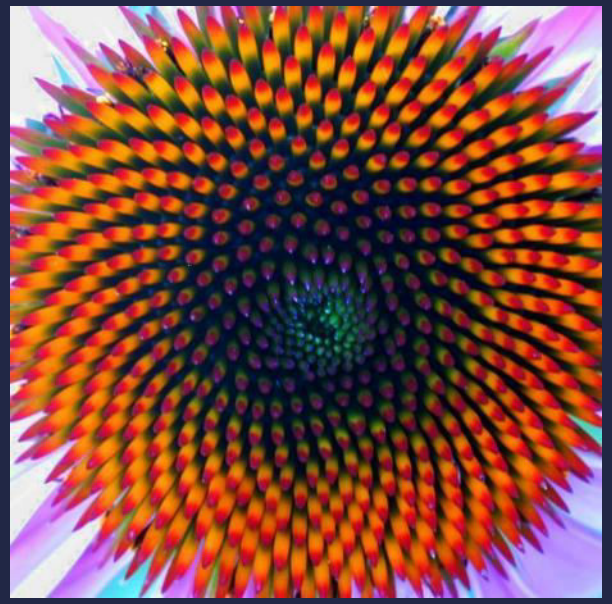




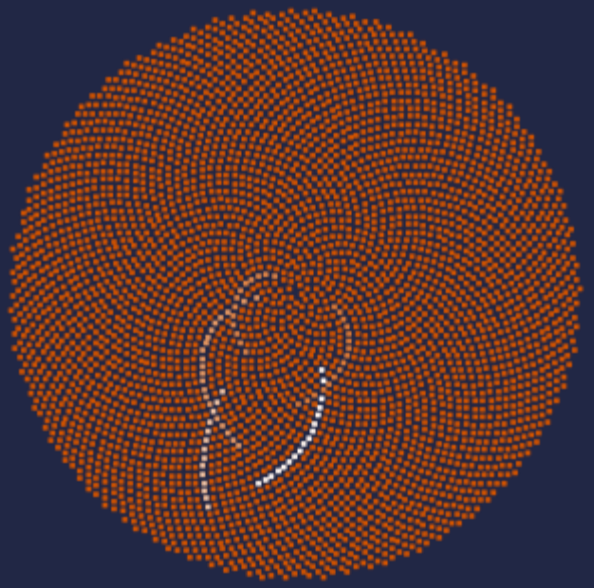


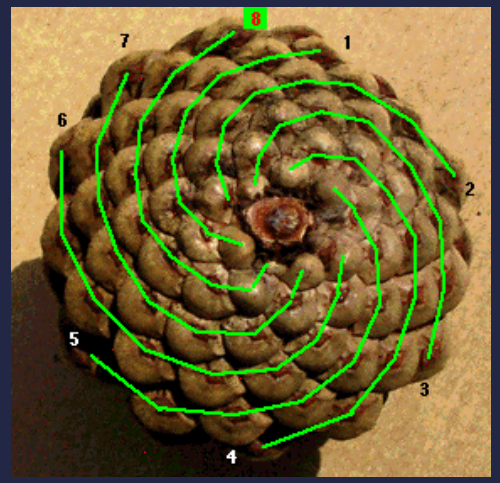
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, ...

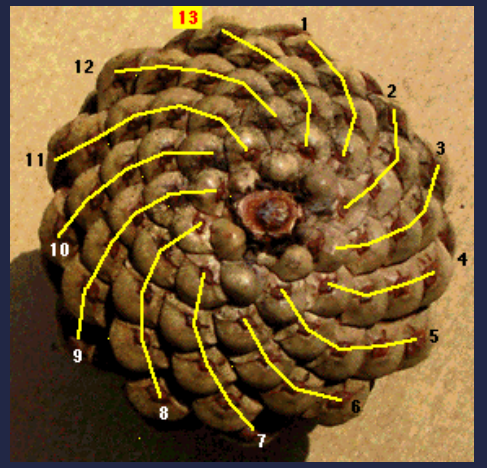
More Fibonacci



The left and right going spirals are neighboring Fibonacci numbers!





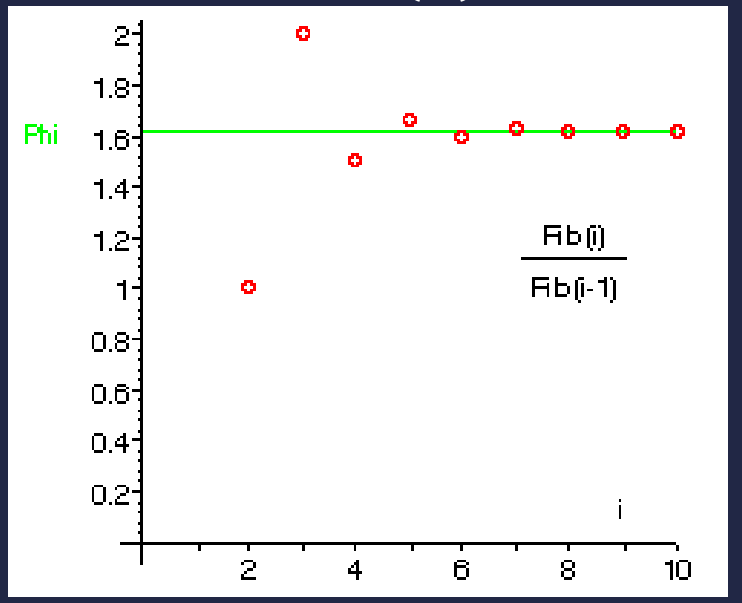


Golden Section

two quantities are in the **golden ratio** if their ratio is the same as the ratio of their sum to the larger of the two quantities

$$\frac{x+y}{x} = \frac{x}{y} = \frac{1}{2}\left(1+\sqrt{5}\right) = 1.6180 = Phi$$

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = Phi$$



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...

Recursively Defined Sets and Structures

Recursive definitions of sets have two parts:

- The *basis step* specifies an initial collection of elements.
- The *recursive step* gives the rules for forming new elements in the set from those already known to be in the set.

- Sometimes the recursive definition has an *exclusion rule*, which specifies that the set contains nothing other than those elements specified in the basis step and generated by applications of the rules in the recursive step.
- We will always assume that the exclusion rule holds, even if it is not explicitly mentioned.
- We will later develop a form of induction, called *structural induction*, to prove results about recursively defined sets.

Recursively defined sets

Basis Step: *define a basis set* (e.g. the empty set).

Recursive Step: *Define a rule to produce new elements from already existing elements.*

Example:

Basis Step: 3 is in S.

Recursive Step: if x is in S and y is in S then $x + y$ is in S.

3

3 + 3 = 6

3 + 6 = 9 & 6 + 6 = 12

...

Recursively defined sets

Strings:

S = set of strings

A = alphabet

S is the Set of strings Σ^*

A is the Alphabet Σ

Basic step: empty string is in S

Recursive step: if w is in S and x in A → wx is in S

Example: binary strings: $A = \{0, 1\}$

1) empty string
2) 0 & 1
3) 00 & 01 & 10 & 11
4) …

Recursively Defined Sets and Structures

Repeating previous slide using textbook notation

- Definition: The set $\sum^*$ of *strings* over the alphabet $\sum$ can be defined recursively by
- Basis step: $\lambda \in \sum^*$ (where λ is the empty string containing no symbols)
- Recursive step: if $w \in \sum^*$ and $x \in \sum$, then $wx \in \sum^*$

- Example: if $\sum = \{0, 1\}$, the strings found to be in $\sum^*$, the set of all bit strings, are

1. λ specified to be in the basis step,
2. 0 and 1 formed during the first application of the recursive step,
3. 00, 01, 10, and 11 formed during the second application for the recursive step, and so on.

Recursively Defined Sets and Structures

- Definition: two strings can be combined via the operation of *concatenation*.
- Let $\sum$ be a set of symbols and
- $\sum^*$ the set of strings formed form symbols in $\sum$
- We can define the *concatenation* of two strings, denoted by $\cdot$, recursively as follows
- Basis step: if $w \in \sum^*$, then $w \cdot \lambda = w$, where λ is the empty string.
- Recursive step: if $w_1 \in \sum^*$ and $w_2 \in \sum^*$ and $x \in \sum$,

  then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$
- Example: *length of a string*

Give a recursive definition of $l(w)$, the length of the string w.

Basis step: $l(\lambda) = 0$;

Recursive step: if $w \in \sum^*$ and $x \in \sum$, $l(wx) = l(w) + 1$.

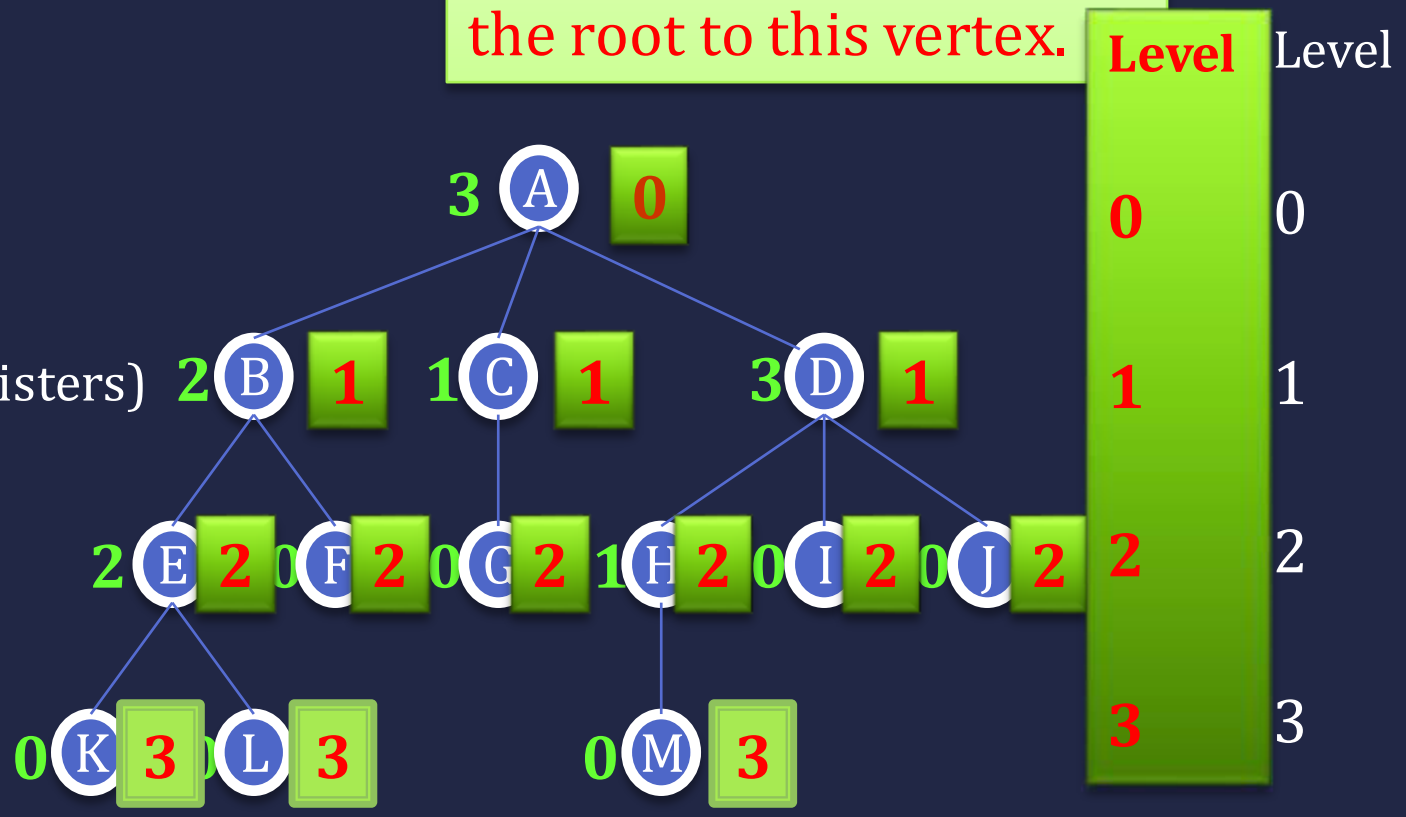
Trees

nodes (13) – vertices (vertex)
Root
edge
degree of a node
leaf (terminal)
nonterminal
parent
children
Sibling (brothers or sisters)
degree of a tree (3)
ancestor
level of a node

level of a node

level of a node

From textbook: The **level** of a vertex *v* in a rooted tree is the length of the unique path from the root to this vertex.



level of a node: Some authors prefer to set the root to be on level one.

Introduction

- **Definition** (recursively): A *tree* is a finite set of one or more nodes such that
  - There is a specially designated node called *root*.
  - The remaining nodes are partitioned into $n >= 0$ disjoint set $T_1, ..., T_n$, where each of these sets is a tree. $T_1, ..., T_n$ are called the *subtrees* of the root.
- Every node in the tree is the root of some subtree

Introduction -Some Terminology

- *Node* (vertex): the item of information plus the branches to each child.
- *Degree of a node*: the number of subtrees of a node
- *degree* of a tree: the maximum of the degree of the nodes in the tree.
- *terminal* nodes (or *leaf*): nodes that have degree zero
- *nonterminal* nodes: nodes that don't belong to terminal nodes.
- *Child:* A node directly connected to another node when moving away from the Root.
- *Parent:* The converse notion of a child.

Introduction- Some **Terminology** (cont'd)

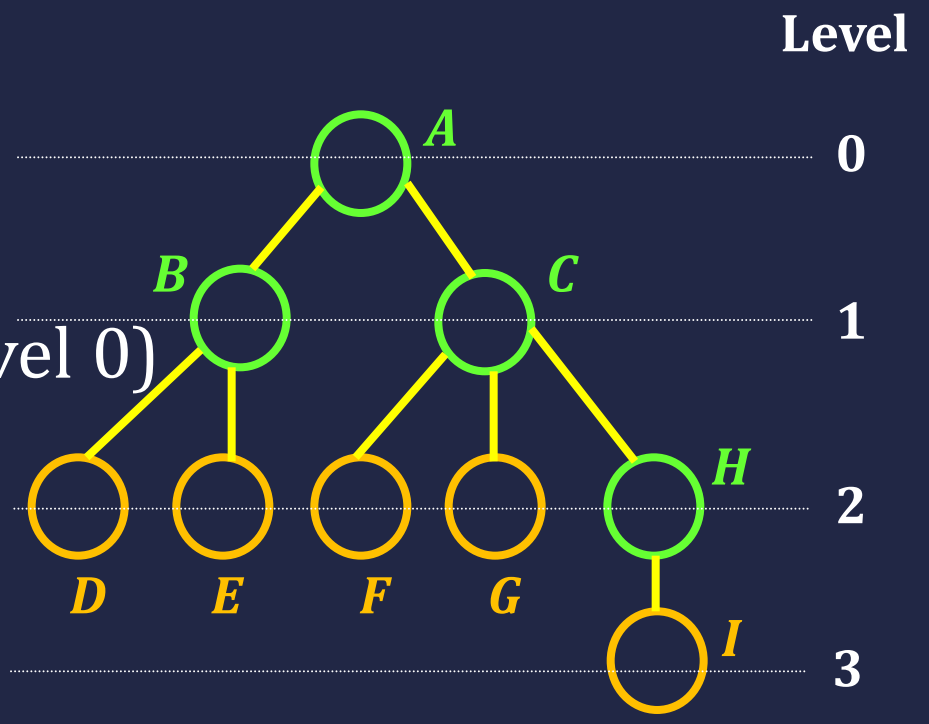
- *siblings*: children of the same parent are said to be siblings (brothers and sisters).
- *Ancestors* of a node: all the nodes along the path from the root to that node.
- The *level* of a node: defined by letting the root be at level zero (or one on some books). If a node is at level l, then it children are at level $l + 1$.
- *Height* (or *depth*) of a tree: the maximum level of any node in the tree

Introduction

- Example

A is the *root* node
B is the *parent* of *D* and *E*
C is the *sibling* of *B*
D and *E* are the *children* of *B*
D, E, F, G, I are *external nodes*, or *leaves*
A, B, C, H are *internal nodes*
The *level* of *E* is 2 (assuming root is at level 0)
The *height (depth)* of the tree is 3
The *degree* of node *B* is 2
The *degree* of the tree is 3
The *ancestors* of node *I* is *A, C, H*
The *descendants* of node *C* is *F, G, H, I*

Property: (*# edges*) = (*#nodes*) - 1



Rooted Trees

- Definition: The set of *rooted trees*, where a rooted tree consists of a set of vertices containing a distinguished vertex called the *root*, and edges connecting these vertices, can be defined recursively by these steps:

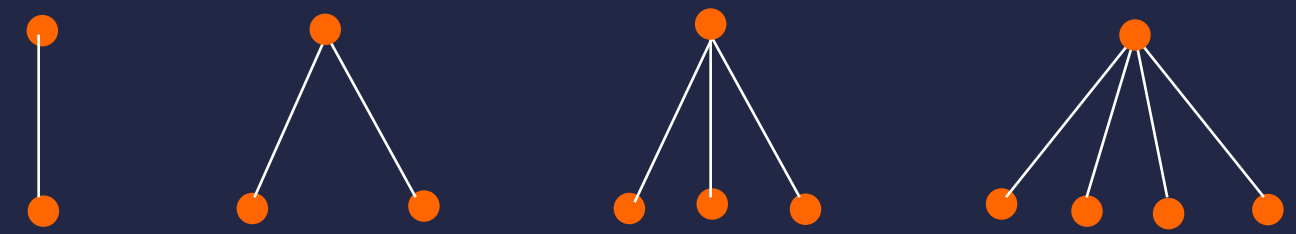
- Basis step: A single vertex r is a rooted tree.

- Recursive step: Suppose that $T_1, T_2, \ldots, T_n$ are disjoint rooted trees with roots $r_1, r_2, \ldots, r_n$, respectively.
- Then the graph formed by starting with a root r, which is not in any of the rooted trees $T_1, T_2, \ldots, T_n$, and adding an edge from r to each of the vertices $r_1, r_2, \ldots, r_n$, is also a rooted tree.

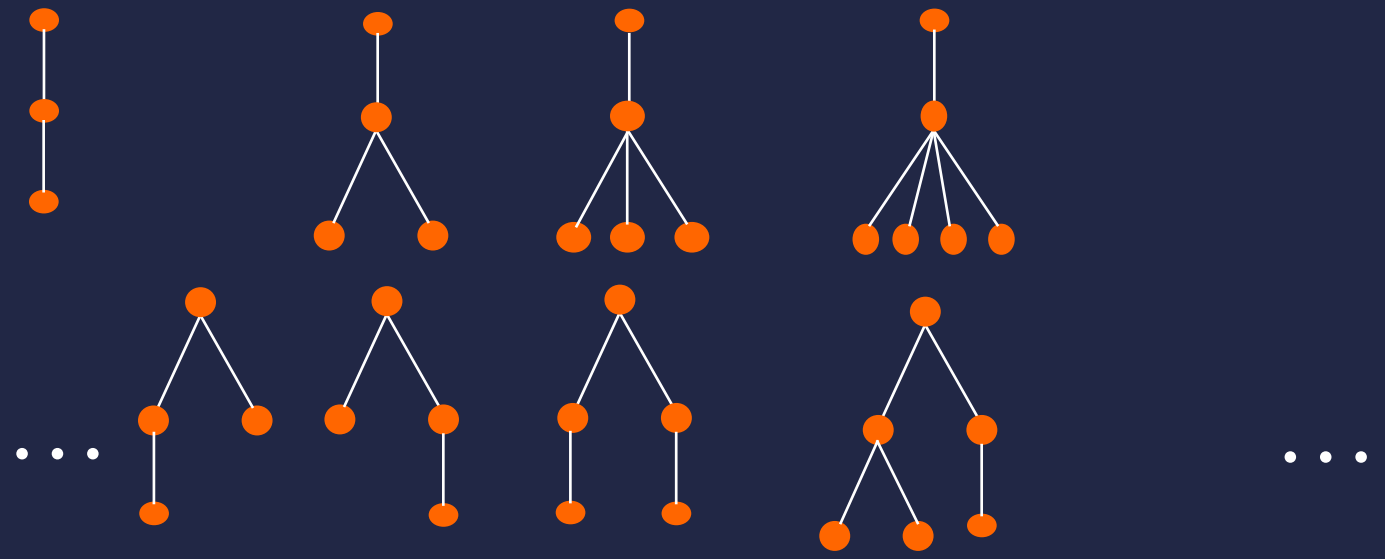
Building up rooted trees

Basis step

Step 1



Step 2



Extended Binary Trees

- Definition: The set of *extended binary trees* can be defined recursively by these step:

- Basis step: The empty set is an extended binary tree.

- Recursive step: if T_1 and T_2 are disjoint extended binary trees, there is an extended binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2 when these trees are nonempty.

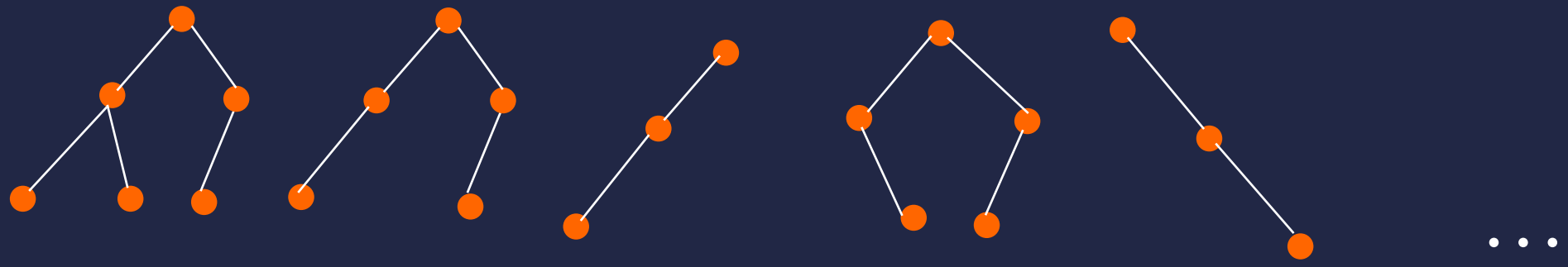
Building Up Extended Binary Trees

Basis Step ∅

Step 1

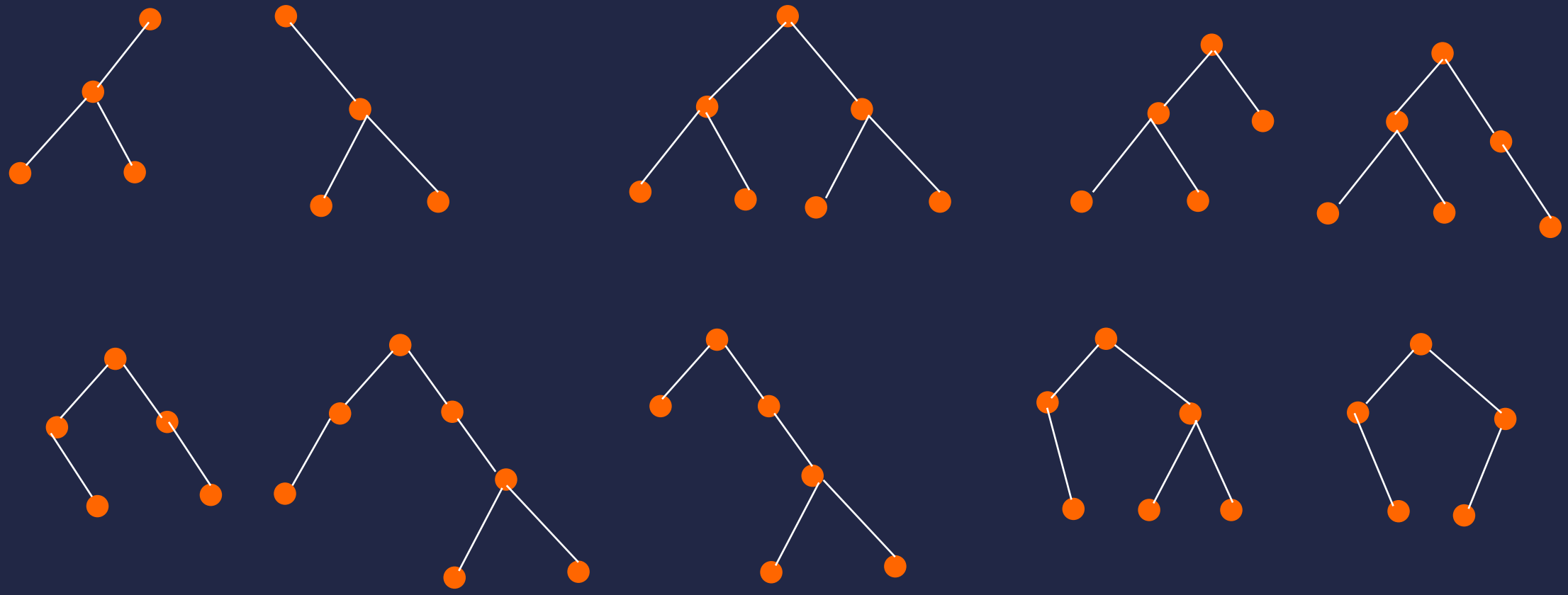
Step 2

Step 3



Building Up Extended Binary Trees (continue…)

… Step 3



Full Binary Trees

- Definition: The set of *full binary trees* can be defined recursively by these steps:

- Basis step: There is a full binary tree consisting only of a single vertex r.

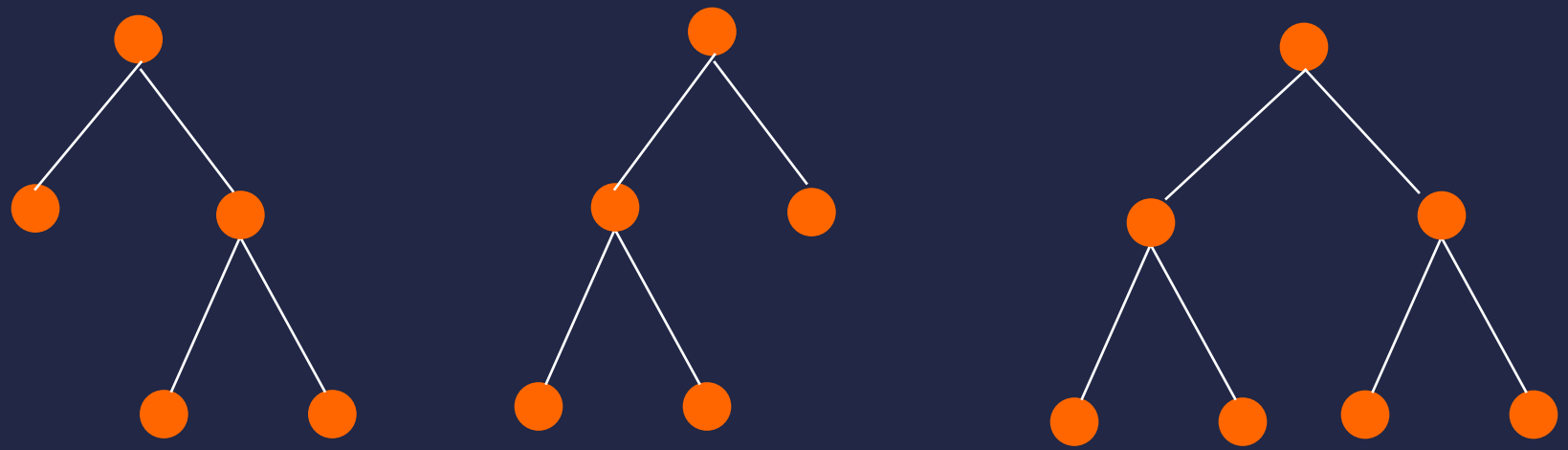
- Recursive step: if T_1 *and* T_2 are disjoint full binary trees, there is a full binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2.

Building Up Full Binary Trees

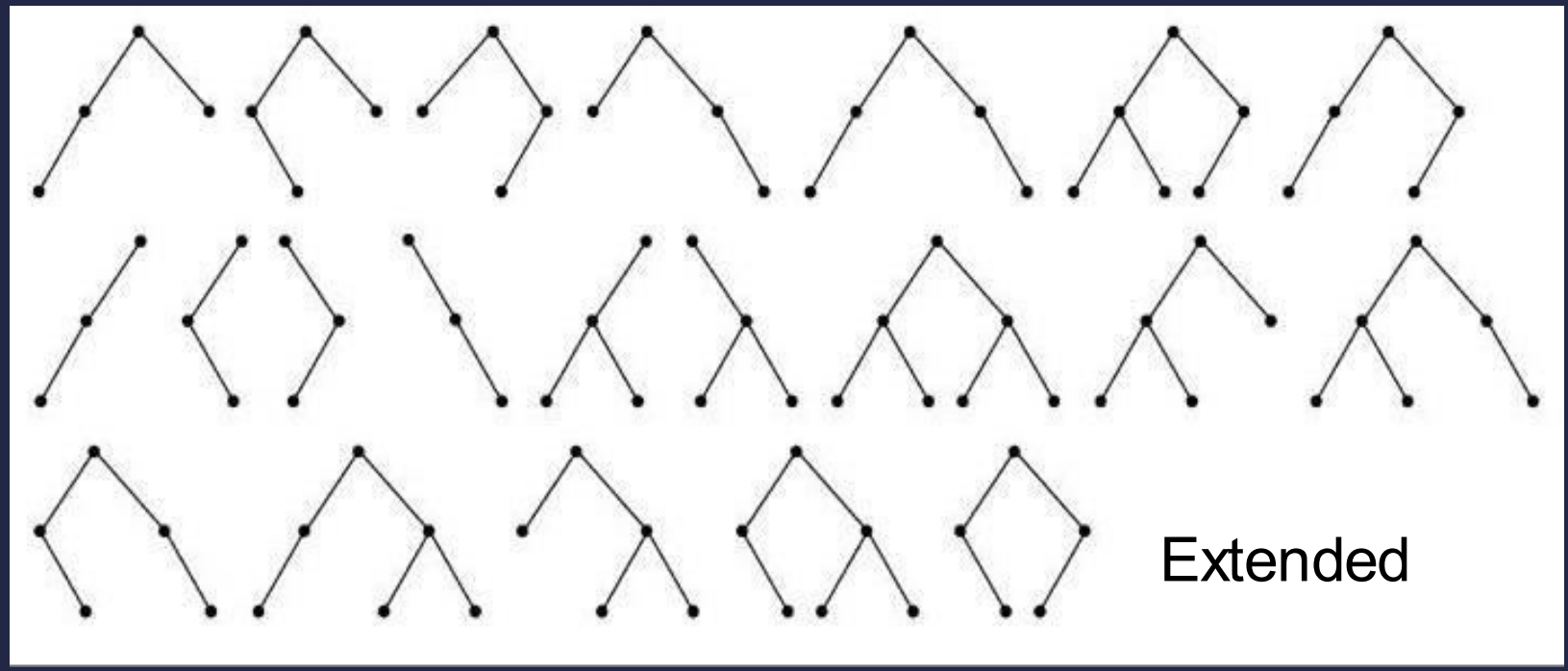
Basis step

Step 1

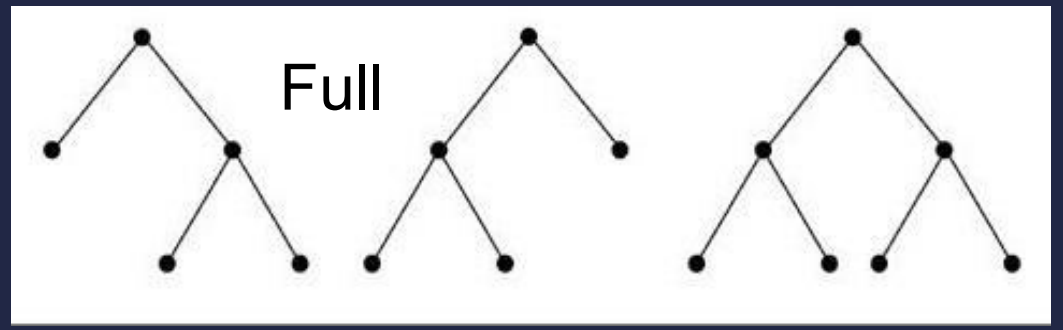
Step 2



Extended Binary Trees vs Full Binary Trees

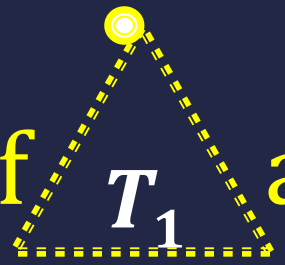
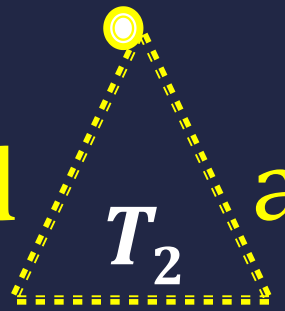


In extended binary trees, the left subtree or the right subtree can be empty, but in full binary trees this is not possible.

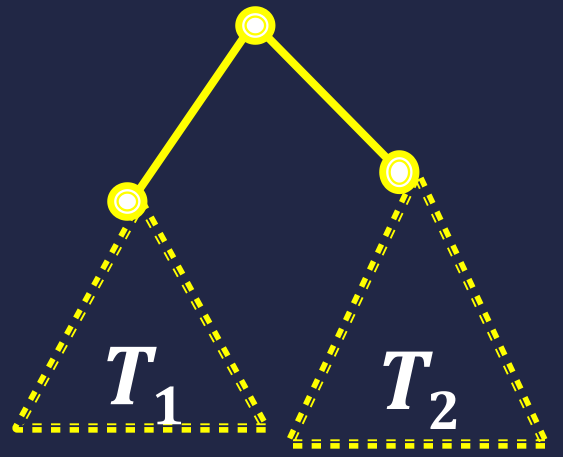


Binary trees

- **Basis:** • is a binary tree

- **Recursive step:** If  and  are binary trees

then so is:

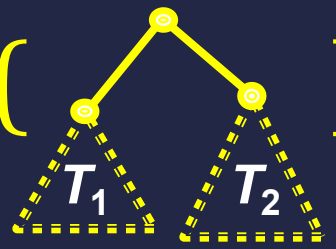


$(T_1 \cdot T_2)$

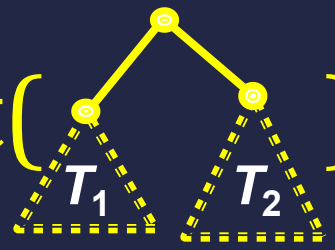
functions defined on binary trees

size of a tree is the number of nodes (vertices) in it

$\text{size}(\bullet) = 1$

$\text{size}($  $) = 1 + \text{size}(T_1) + \text{size}(T_2)$

$\text{height}(\bullet) = 0$

$\text{height}($  $) = 1 + \max\{\text{height}(T_1), \text{height}(T_2)\}$

Structural Induction

Definition:

To prove a property of the elements of a recursively defined set, we use *structural induction*.

BASIS STEP: Show that the result holds for all elements specified in the basis step of the recursive definition.

RECURSIVE STEP: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

The validity of structural induction can be shown to follow from the principle of mathematical induction.

Height of a full binary tree

$h(T)$ is the height of a full binary tree:

Recursive Definition:

Basis Step:

The height of a tree consisting of a single root node is

$$h(T) = 0$$

Recursive Step: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1.T_2$ has height

$$h(T) = 1 + \max(h(T_1), h(T_2))$$

Number of vertices (size) in a full binary tree

$n(T)$ is the number of vertices in the tree.

Recursive definition:

Basis Step: The number of vertices of a tree consisting of a single root node is:

$$n(T) = 1;$$

Recursive Step: If T_1 and T_2 are full binary trees, then the number of vertices of the tree $T_1.T_2$ is

$$n(T) = 1 + n(T_1) + n(T_2).$$

Structural Induction and Binary Trees

Theorem:

If T is a full binary tree, then

$$n(T) \leq 2^{(h(T)+1)} - 1$$

Where $n(T)$ denote the number of vertices (nodes) in a full binary tree and $h(T)$ is the height of the tree.

Structural Induction and Binary Trees

Theorem: If T is a full binary tree, then $n(T) \leq 2^{(h(T)+1)} - 1$

Proof: Use structural induction

BASIS STEP: The result holds for a full binary tree (T) consisting only of a root,

$n(T) = 1$ and $h(T) = 0$.

Hence, $n(T) = 1 \leq 2^{(0+1)} - 1 = 1$.

we have $n(T) = 1 + n(T1) + n(T2)$
and $h(T) = 1 + \max(h(T1), h(T2))$.

Structural Induction and Binary Trees

RECURSIVE STEP: Assume $n(T_1) \le 2^{(h(T1)+1)} - 1$ and $n(T_2) \le 2^{(h(T2)+1)} - 1$ whenever T_1 and T_2 are full binary trees.

$(T = T_1 \bullet T_2)$

$n(T) = 1 + n(T_1) + n(T_2)$ (*by recursive formula of* $n(T)$)

$\le 1 + (2^{(h(T1)+1)} - 1) + (2^{(h(T2)+1)} - 1)$ (*by inductive hypothesis*)

$\le 2 \cdot \max(2^{(h(T1)+1)}, 2^{(h(T2)+1)}) - 1$

the sum of two terms is at m

$1 + 2^{(h(T1)+1)} - 1 + 2^{(h(T2)+1)} - 1$
$= 2^{(h(T1)+1)} + 2^{(h(T2)+1)} - 1$

$= 2 \cdot 2^{(\max(h(T1), h(T2))+1)} - 1$ $(\max(2^x, 2^y) = 2^{\max(x,y)})$

$= 2 \cdot 2^{h(T)} - 1$ (*by recursive definition of* $h(T)$)

$= 2^{h(T)+1} - 1$

Structural Induction and Binary Trees

Theorem: If T is a full binary tree, then $n(T) \le 2^{(h(T)+1)} - 1$

Proof: Use structural induction

BASIS STEP: The result holds for a full binary tree consisting only of a root, $n(T) = 1$ and $h(T) = 0$. Hence, $n(T) = 1 \le 2^{(0+1)} - 1 = 1$.

RECURSIVE STEP: Assume $n(T_1) \le 2^{(h(T_1)+1)} - 1$ and also $n(T_2) \le 2^{(h(T_2)+1)} - 1$ whenever T_1 and T_2 are full binary trees.

$\le 1 + 2^{(h(T1)+1)} - 1 + 2^{(h(T2)+1)} - 1$

$\le 2^{(h(T1)+1)} + 2^{(h(T2)+1)} - 1$

$n(T) = 1 + n(T_1) + n(T_2)$ (*by recursive formula of $n(T)$*)

$\le 1 + (2^{(h(T_1)+1)} - 1) + (2^{(h(T_2)+1)} - 1)$ (*by inductive hypothesis*)

$\le 2 \cdot \max(2^{(h(T_1)+1)}, 2^{(h(T_2)+1)}) - 1$ *the sum of two terms is at most 2 times the larger*

$= 2 \cdot 2^{(\max(h(T_1), h(T_2))+1)} - 1$ ($\max(2^x, 2^y) = 2^{\max(x, y)}$)

$= 2 \cdot 2^{h(T)} - 1$ (*by recursive definition of $h(T)$*)

$= 2^{h(T)+1} - 1$