Lecture 14: Drawing Binary Trees II

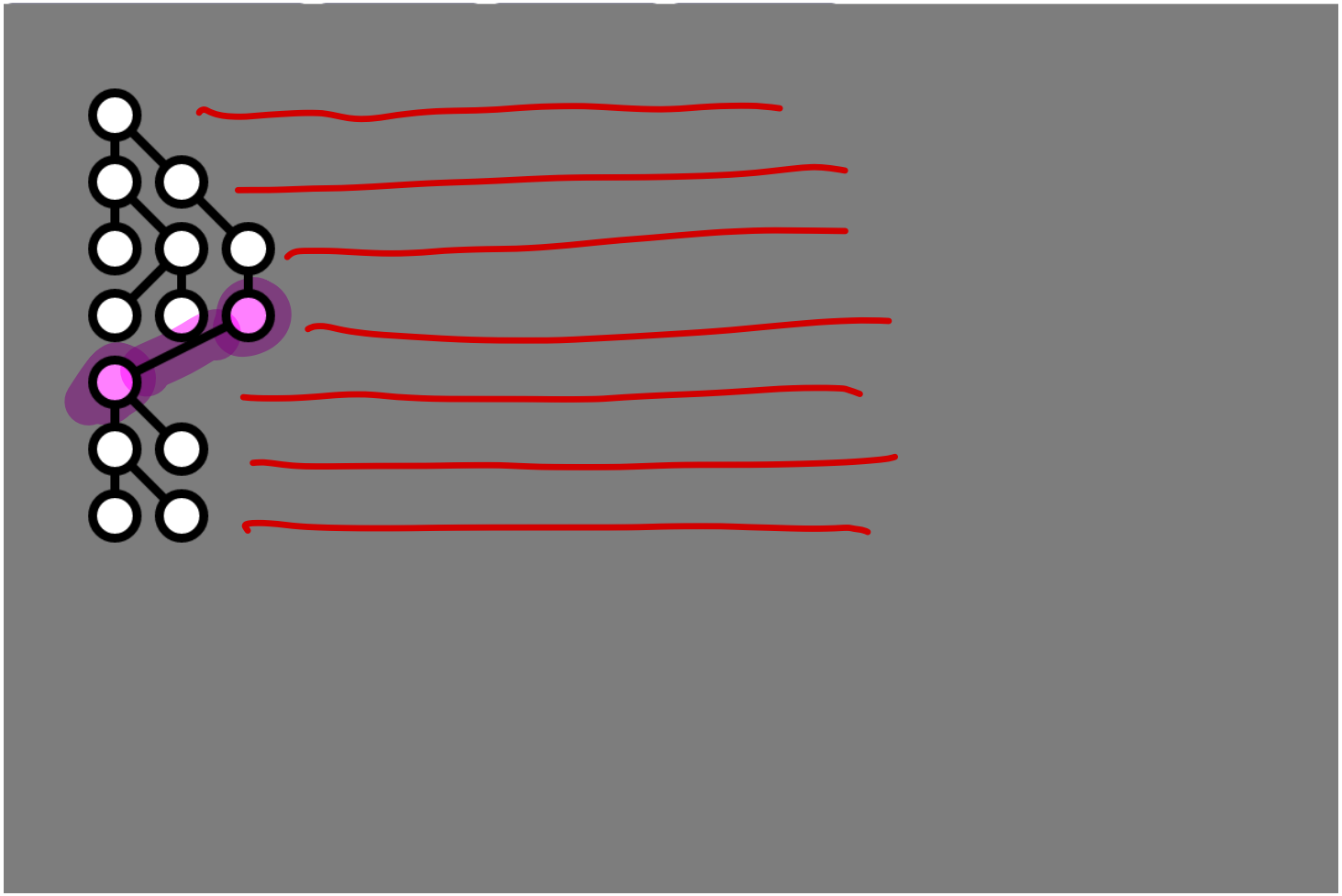
COSC 225: Algorithms and Visualization

Spring, 2023

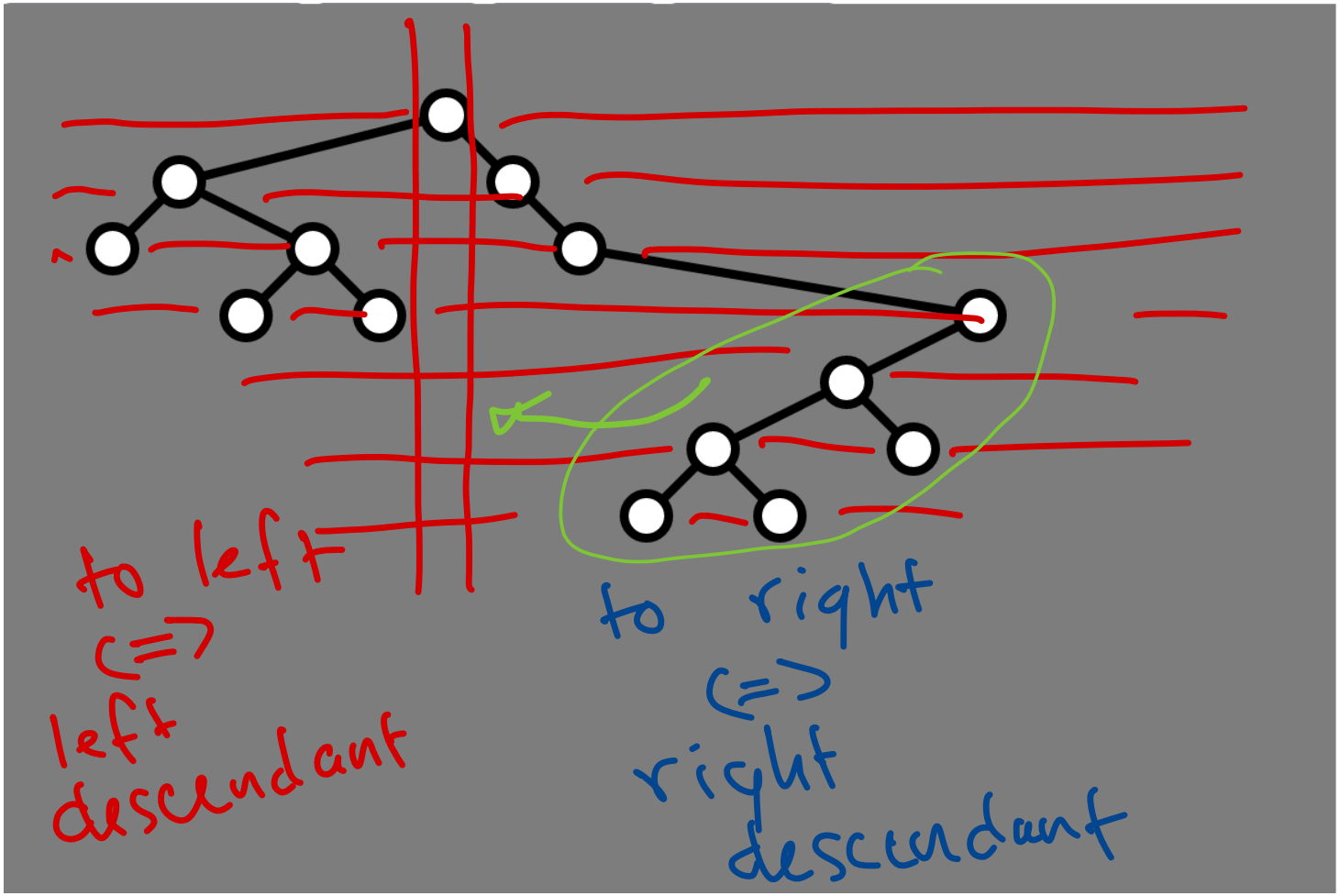
Outline

1. Knuth Layout
2. Tidy Drawing Layout

Last Time: Greedy Layout



Also: Knuth Layout



Aesthetic Principles

Aesthetic Principle 1. Vertices at the same **depth** should lie along a horizontal line with deeper nodes lower than shallower nodes.

Aesthetic Principle 2. The left child of any node should appear to the left of its parent, and a right child should appear to the right of its parent.

↪ Knuth: all left descendants to left of parent, and sim. for right.

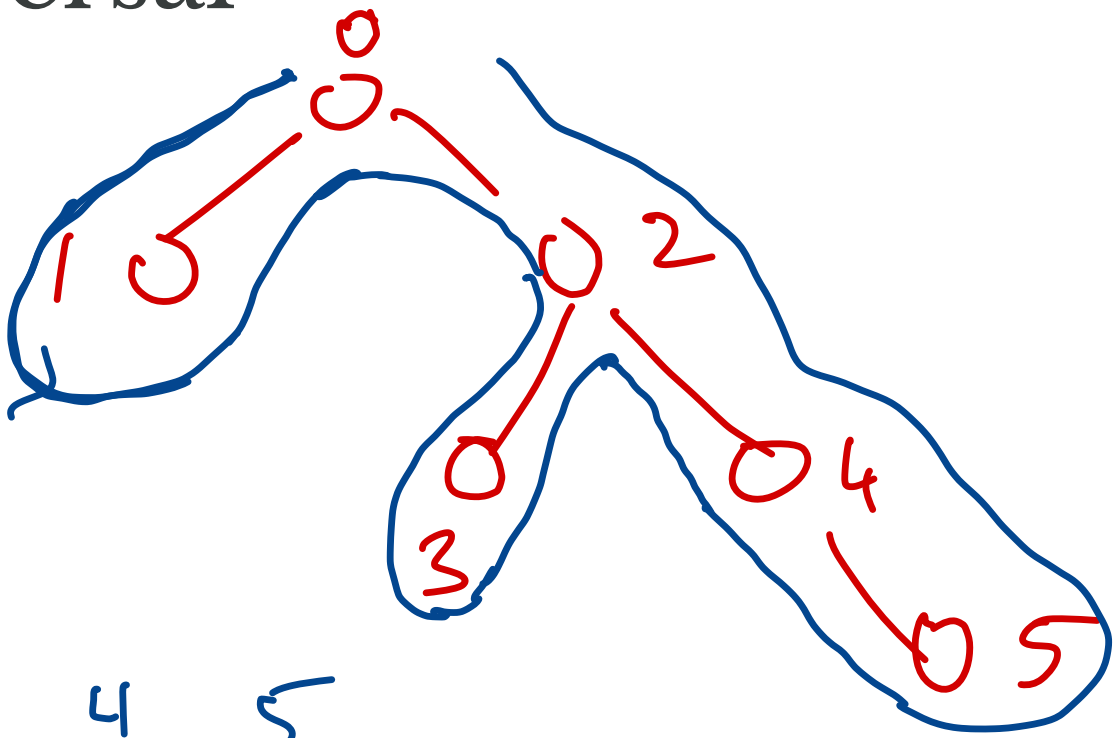
Knuth's Layout Algorithm

Rows and Columns

- rows are defined by depth (Aesthetic Principle 1)
- columns are "in-order" traversal order
  - each vertex gets own column
- guarantees
  - left descendants to the left
  - right descenadants to the right

] Aesthetic Principle 1

In-order Traversal



col	0	1	2	3	4	5
	1	0	3	2	4	5

Lect 13 code on website

In-order Traversal in Code

starting vtx

```
this.verticesInOrder = function (from = this.root) {
  let vertices = [];
  if (from.left != null)
    vertices = vertices.concat(this.verticesInOrder(from.left));
  vertices.push(from);
  if (from.right != null)
    vertices = vertices.concat(this.verticesInOrder(from.right));
  return vertices;
```

in BinaryTree class

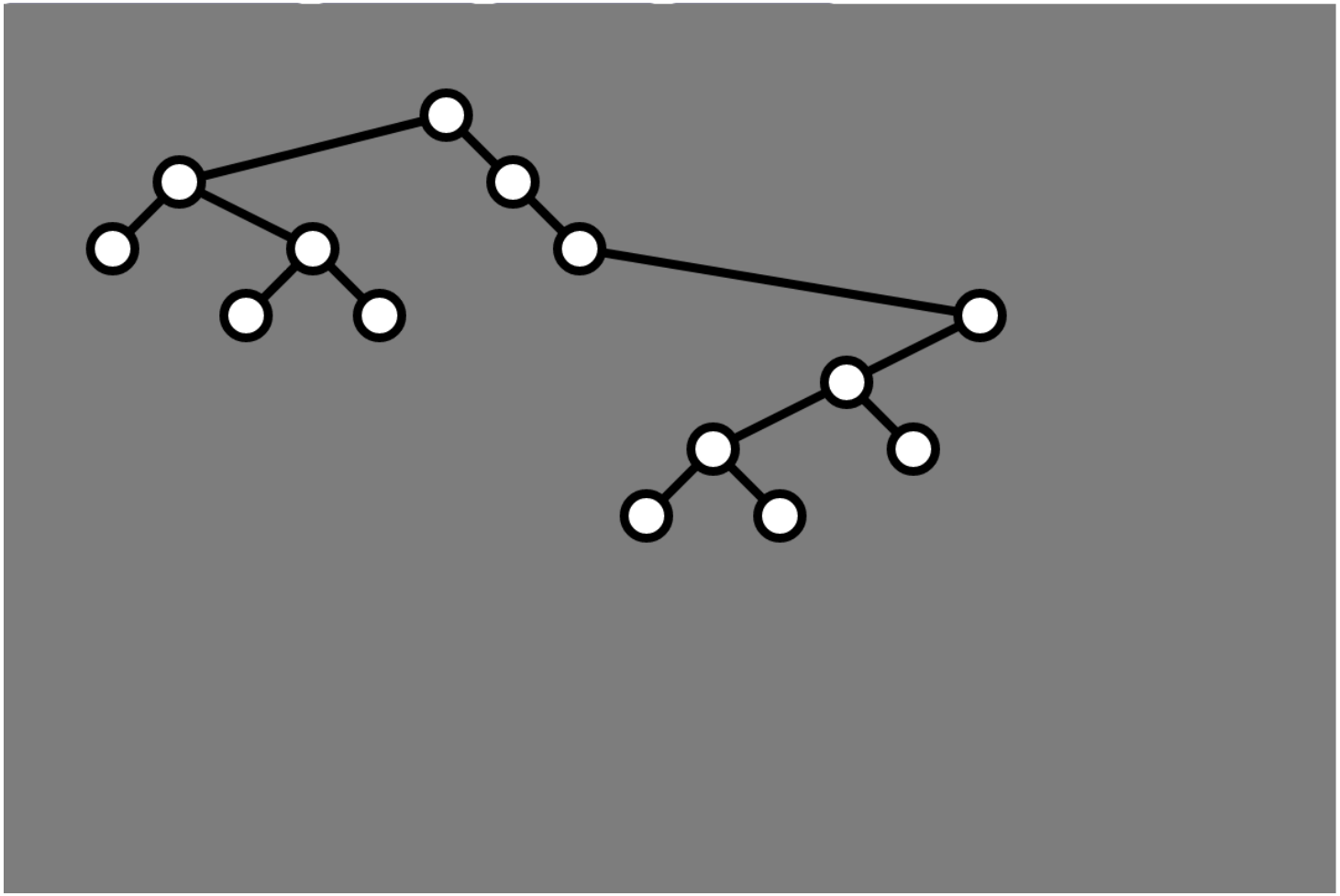
all descendants of "from" list according to in-order traversal.

Knuth's Layout in Code

```
this.setLayoutKnuth = function () {
  const vertices = this.tree.verticesInOrder();
  const depths = this.tree.depths;
  for (let i = 0; i < vertices.length; i++) {
    let vtx = vertices[i];
    let depth = depths.get(vtx.id);
    /* set vtx location to row depth, column i */
  }
}
```

from Monday a Map

Result



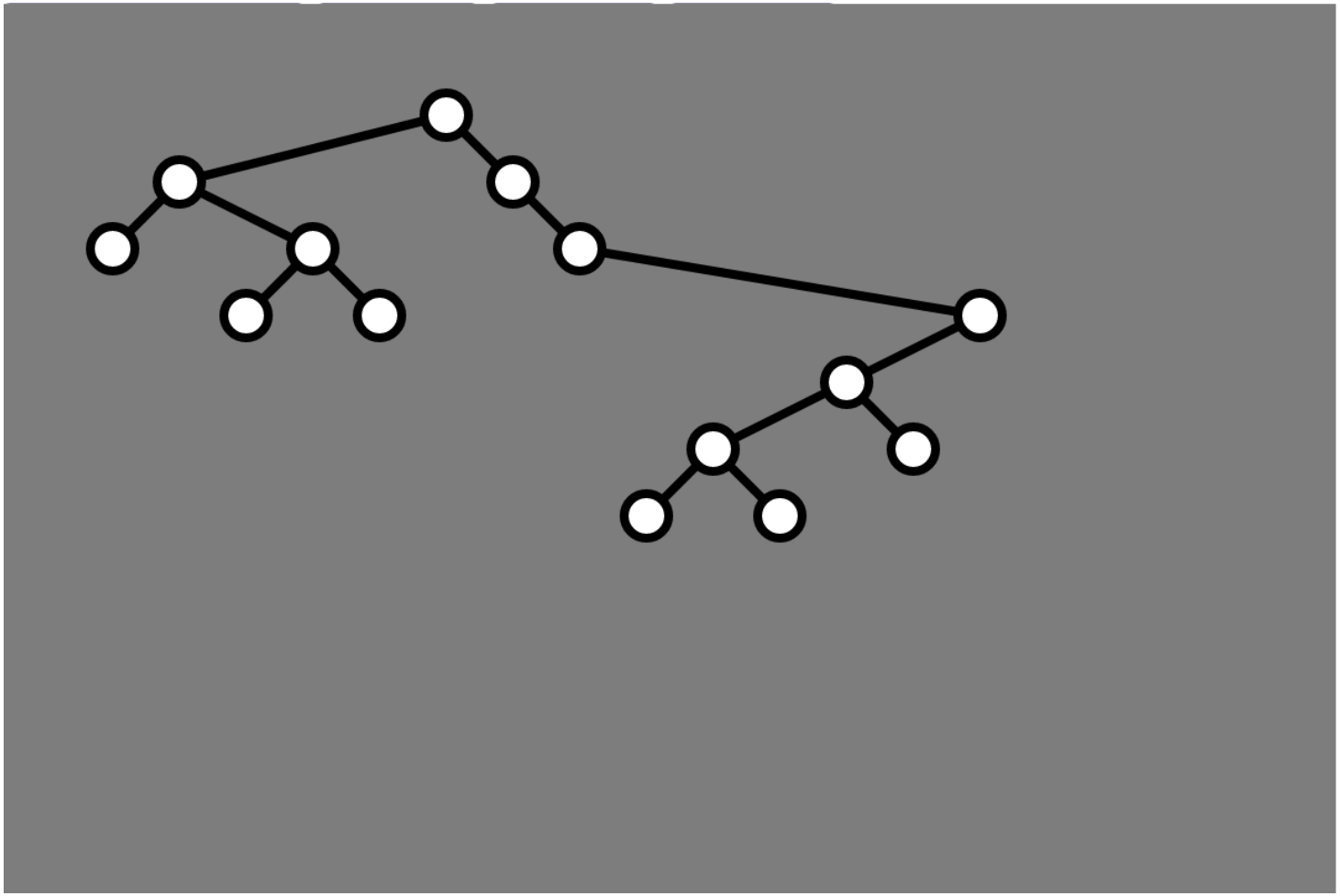
Demo, Again

- `lec13-binary-tree.zip`

What's Not to Like?

- Sometimes too much horizontal space?
- centering — more balance?
- process graph more quickly?

Result Again



Third Principle

Aesthetic Principle 1. Vertices at the same **depth** should lie along a horizontal line with deeper nodes lower than shallower nodes.

Aesthetic Principle 2. The left child of any node should appear to the left of its parent, and a right child should appear to the right of its parent.

Aesthetic Principle 3. If a node has two children, it's x-coordinate should be the midpoint of its childrens' x-coordinates

How Can We Achieve All Three?

Fundamental change:

Process both children first before parent

Post-order traversal

A First Attempt

Idea. Place children first, then place parent above midpoint of children.

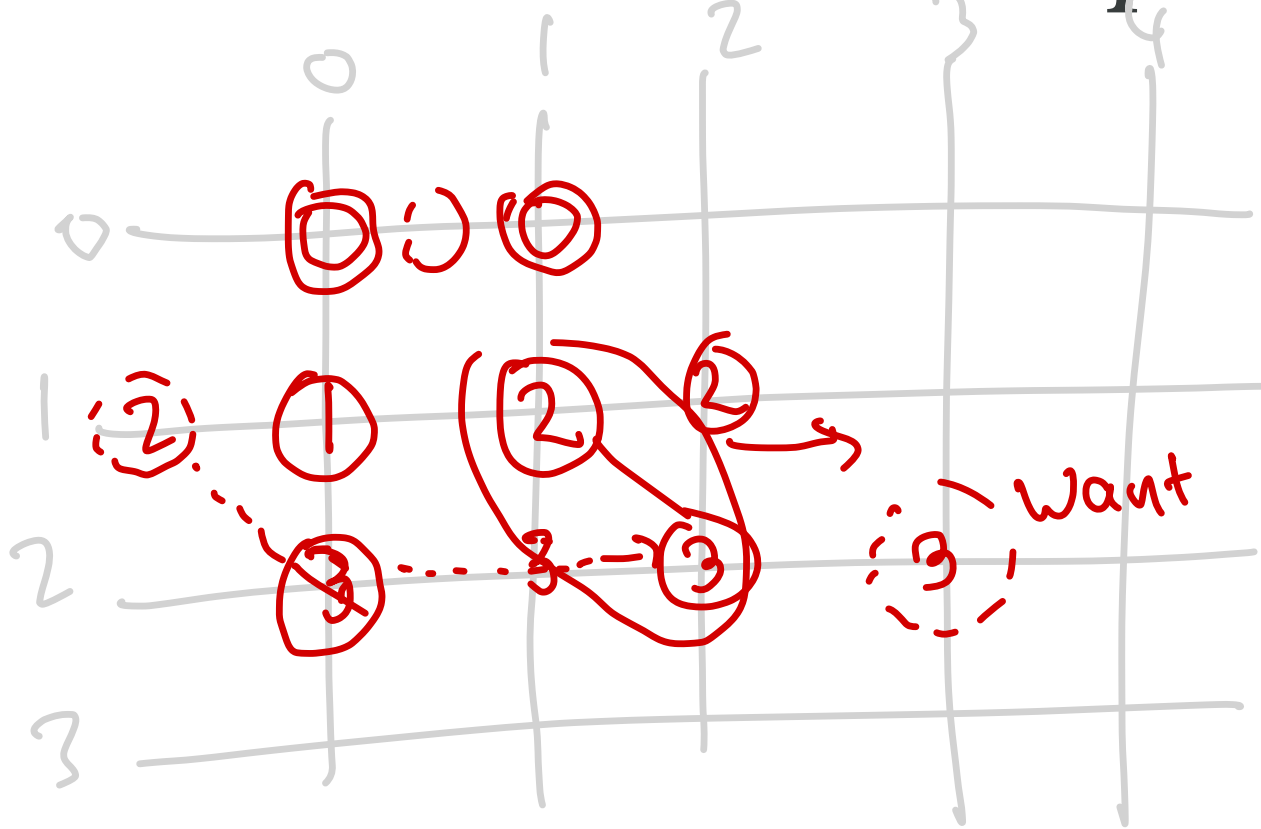
- if one child, must respect Aesthetic Principle 2.

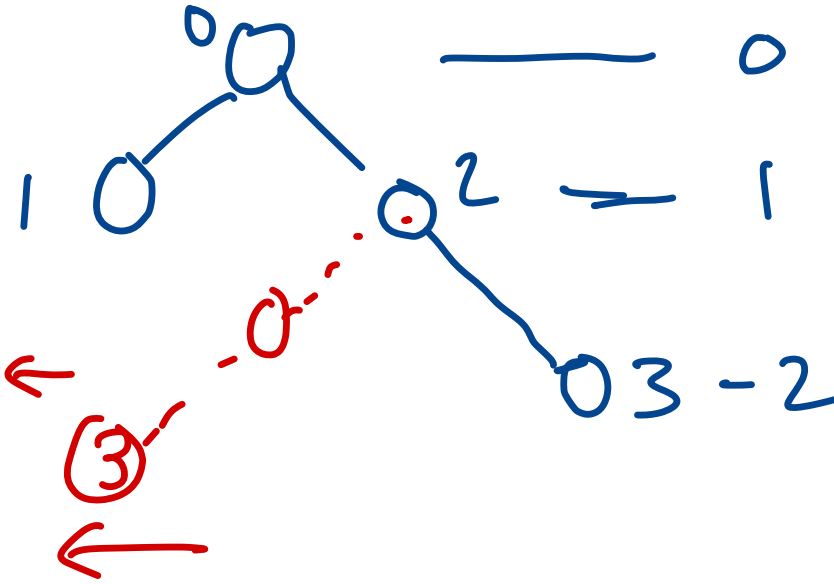
Question. In what order should we place vertices?

↪ Post order

Post: left desc., then right, then self

A Problematic Example





1 3 2 0

Provisional Placement:
as far to left as possible, while respecting:
(1) AP 3 and
(2) vertices in row

A Solution?

- suppose children are provisionally placed
- place parent:
  - correct relative to children, or
  - left-most availble position at parent's depth

Then what?

record difference between actual placement and desired placement rel. to children

→ then shift all descendants

Tidy Drawings of Graphs

- Wetherell and Shannon, 1979

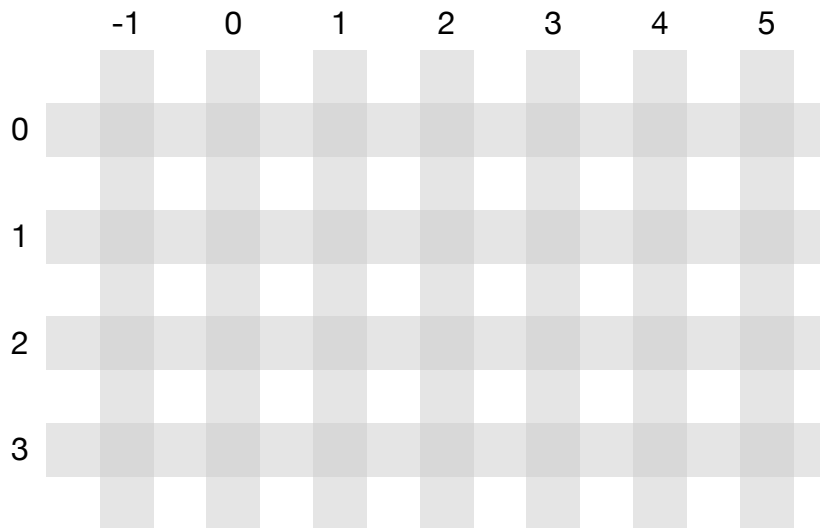
Phase 1. Get initial placement

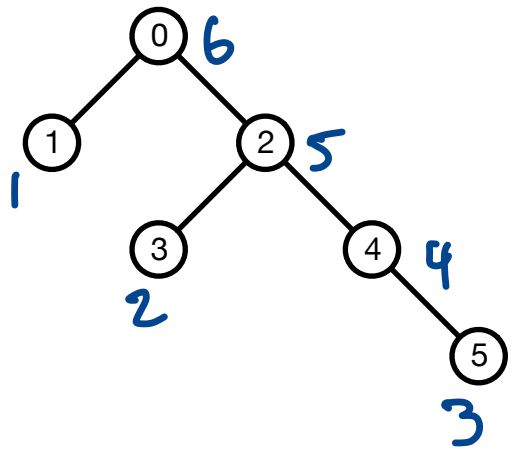
- process vertices in *post-order*
- place each vertex according to maximum of
  - child-aware placement & first available column
- keep track of *offset* if placed vertex to right of child-aware placement

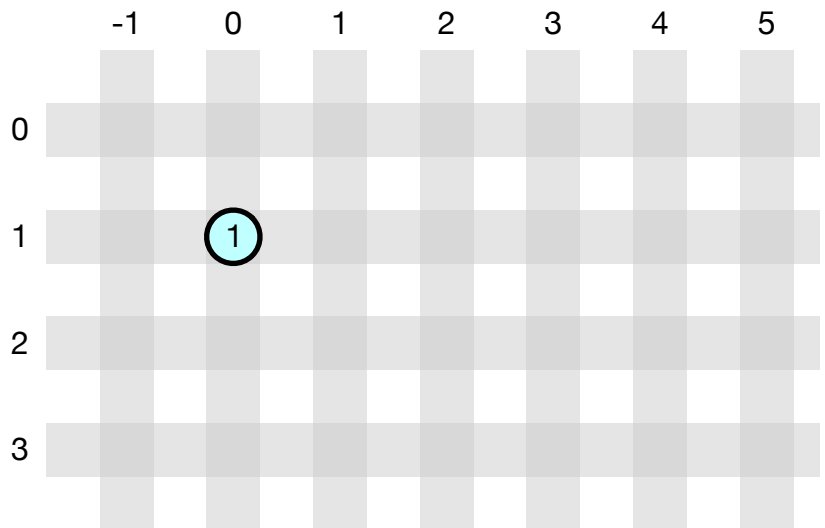
Phase 2. Finalize placement

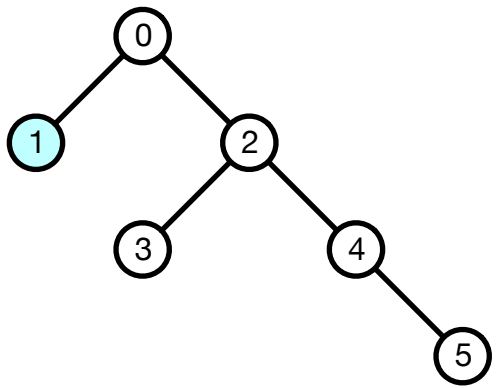
- process vertices in *pre-order*
- place vertex at current position + sum of ancestors' offsets

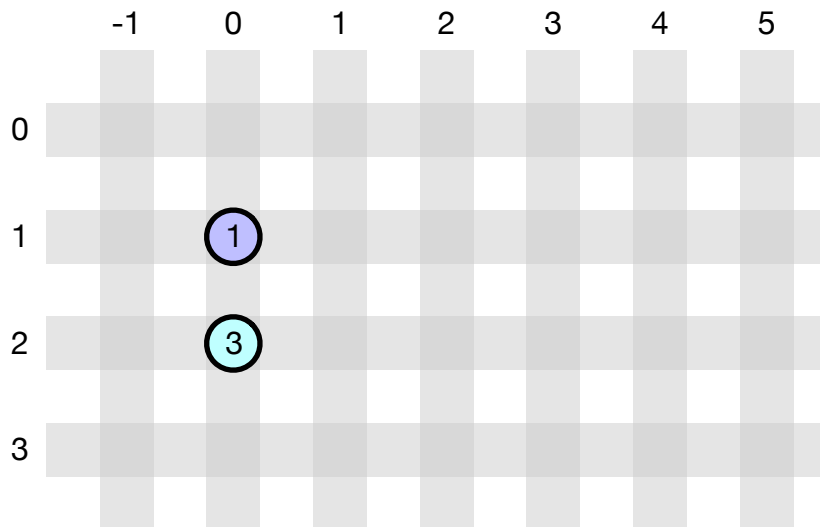
Tidy Drawing Example

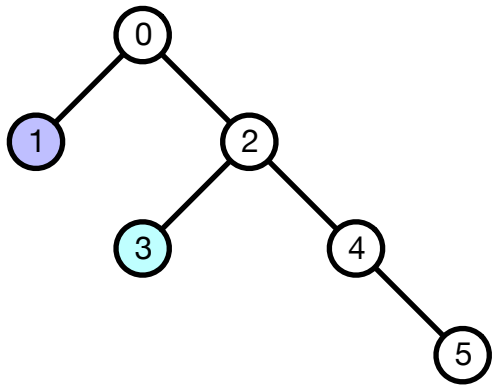

-1
0
1
2
3
4
5
0
1
2
3

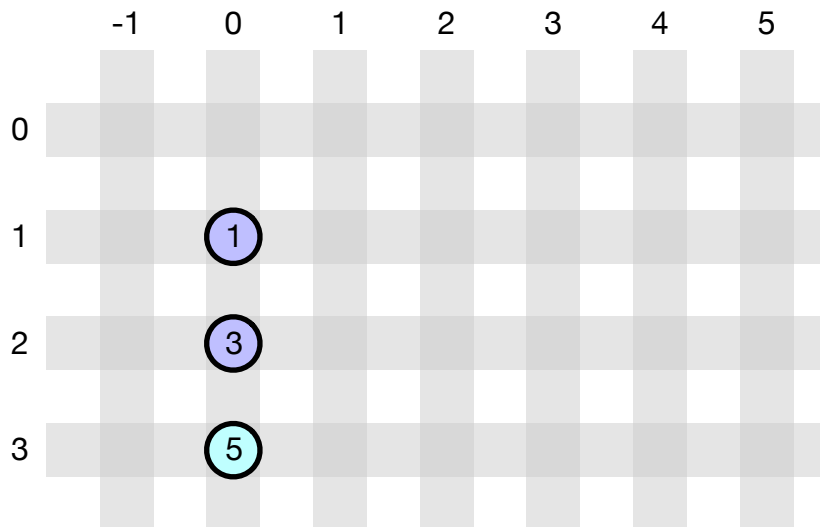

0
1
2
3
4
5
6
1
5
2
4
3

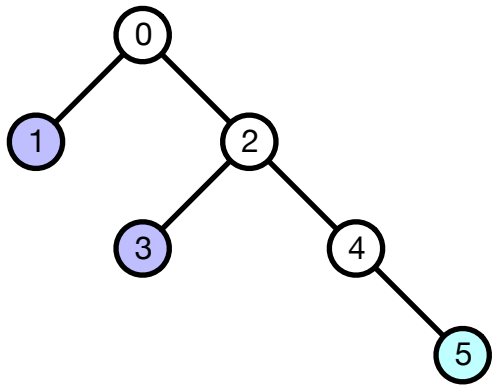

-1
0
1
2
3
4
5
0
1
2
3
1

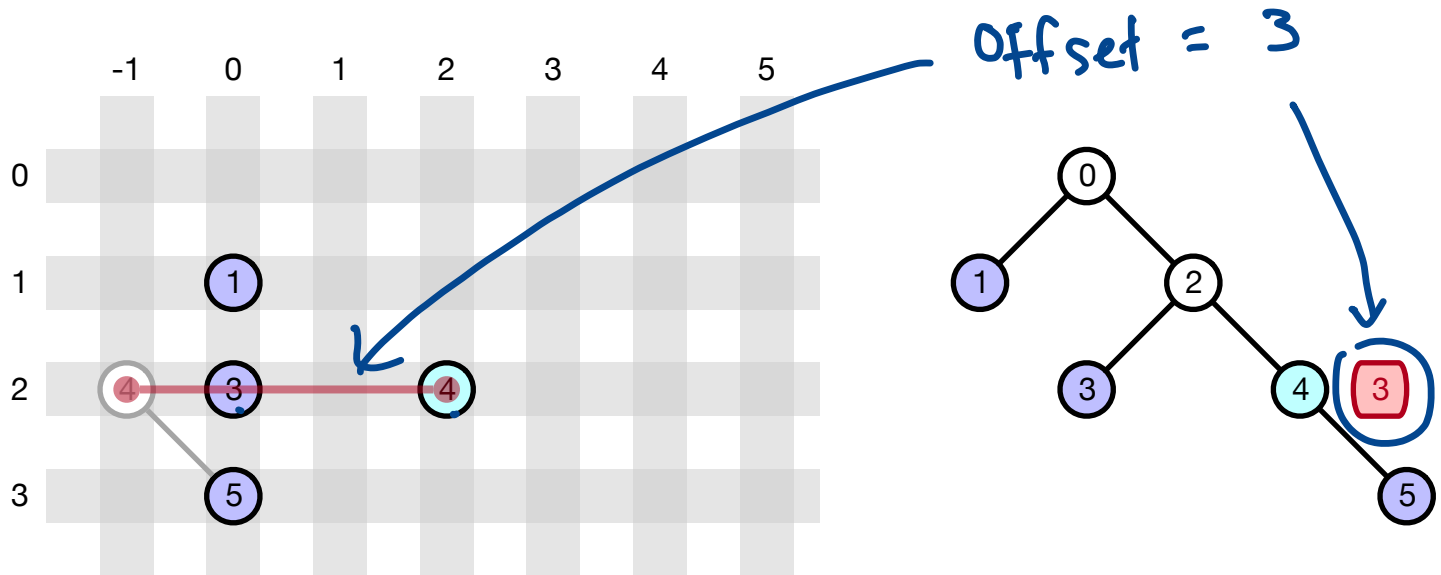

0
1
2
3
4
5

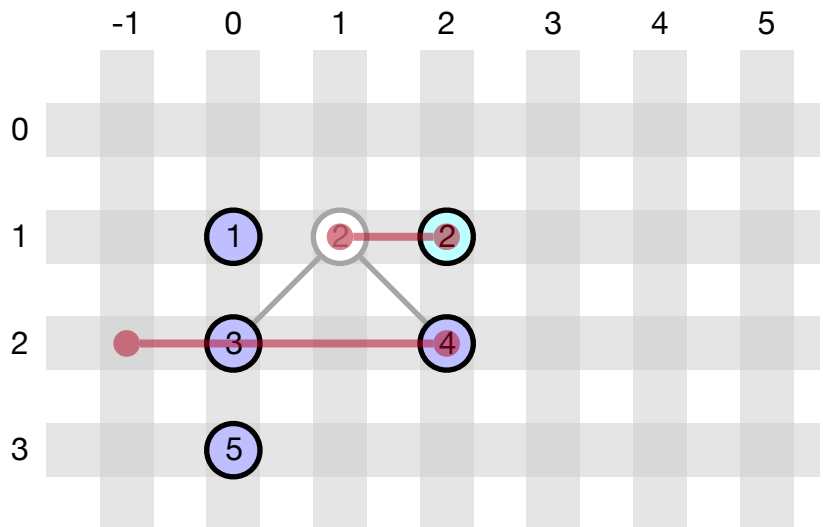

-1
0
1
2
3
4
5
0
1
2
3
1
3

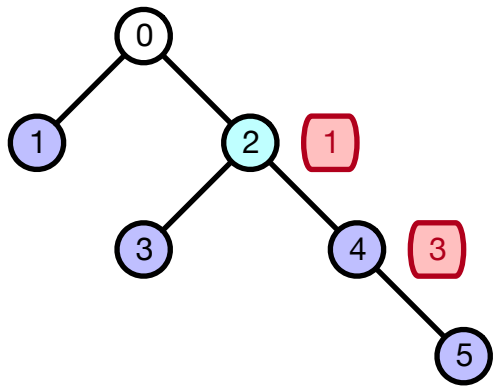

0
1
2
3
4
5

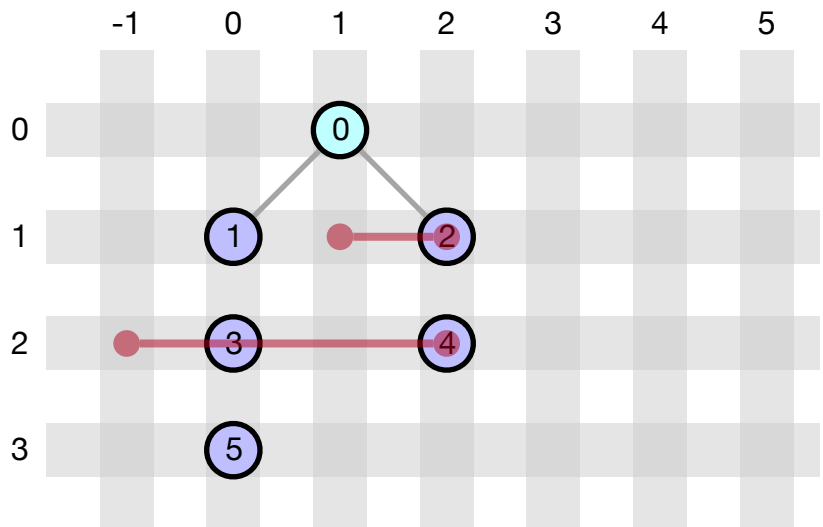

-1
0
1
2
3
4
5
0
1
2
3
1
3
5

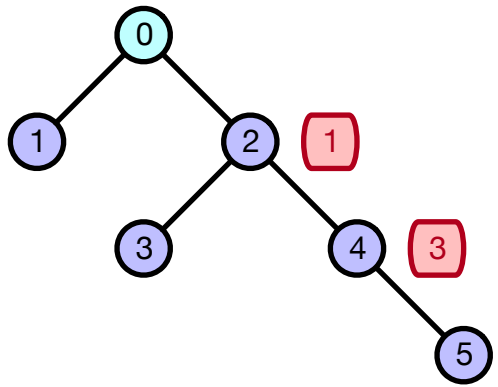

0
1
2
3
4
5

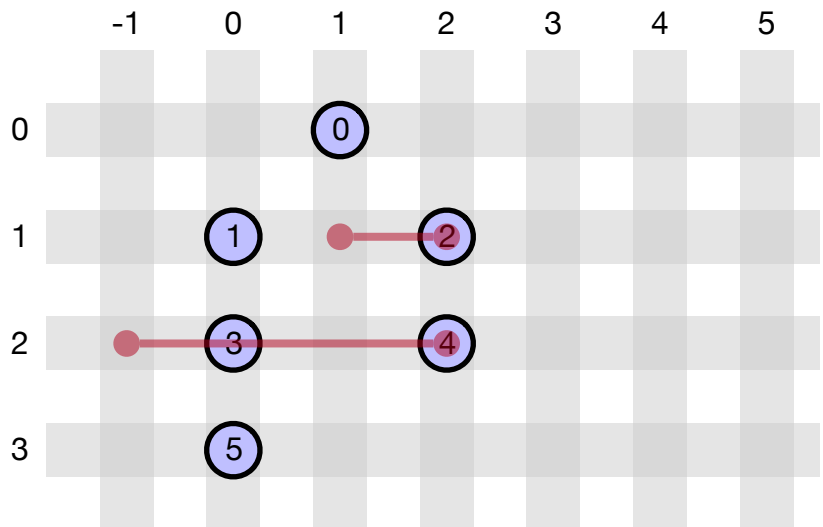

-1
0
1
2
3
4
5
0
1
2
3
1
4
3
5
offset = 3
0
1
2
3
4
3
5

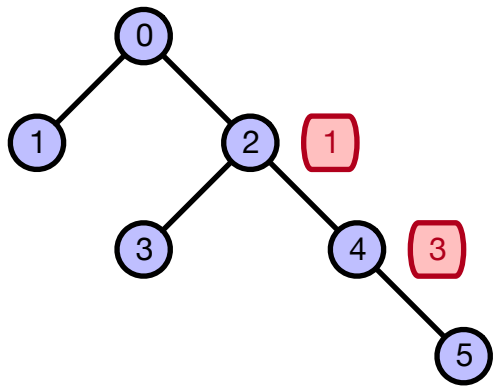

-1
0
1
2
3
4
5
0
1
2
3
1
2
2
3
4
5

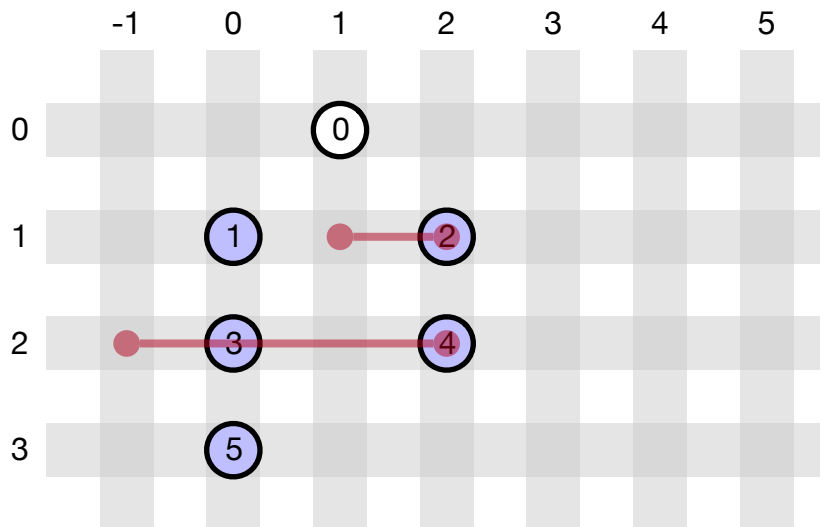

0
1
2
1
3
4
3
5

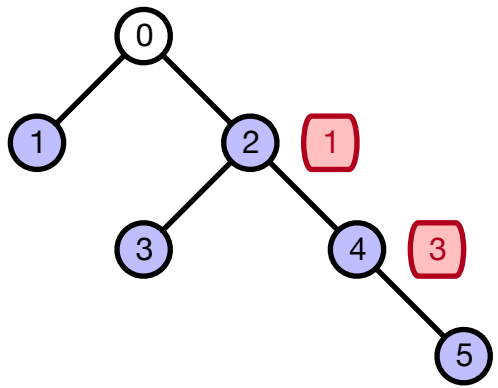

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5

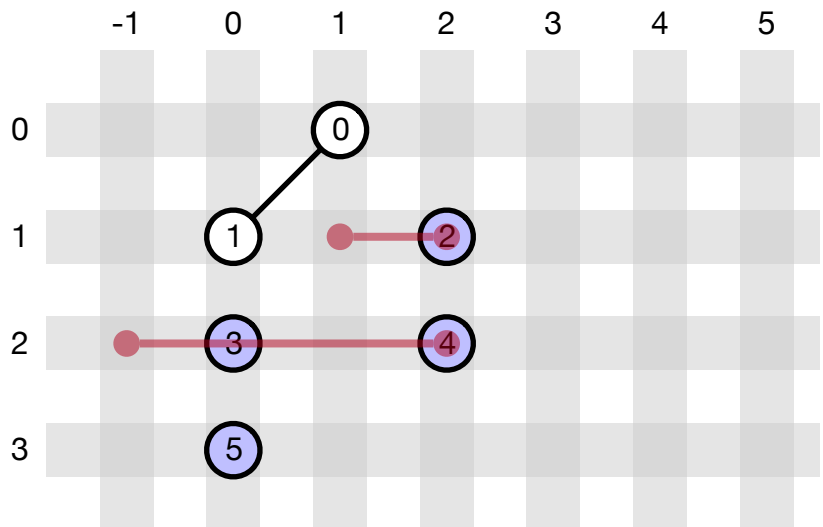

0
1
2
1
3
4
3
5

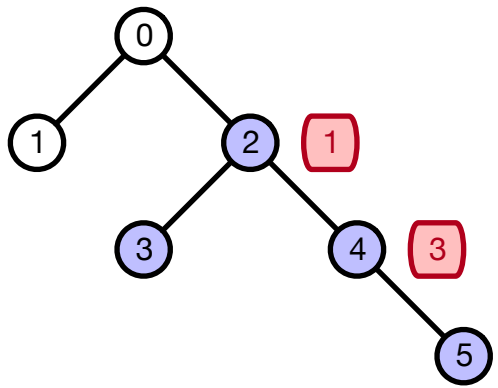

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5

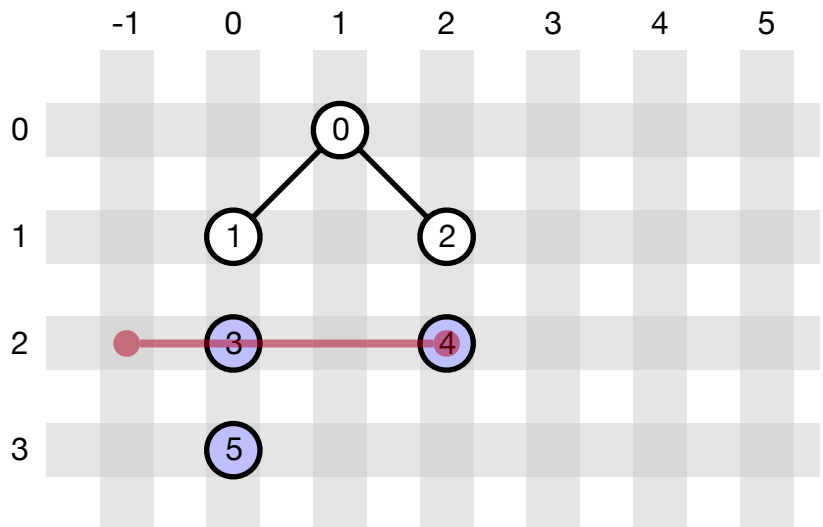

0
1
2
1
3
4
3
5

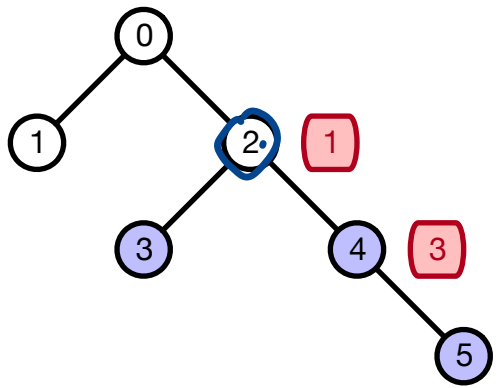

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5

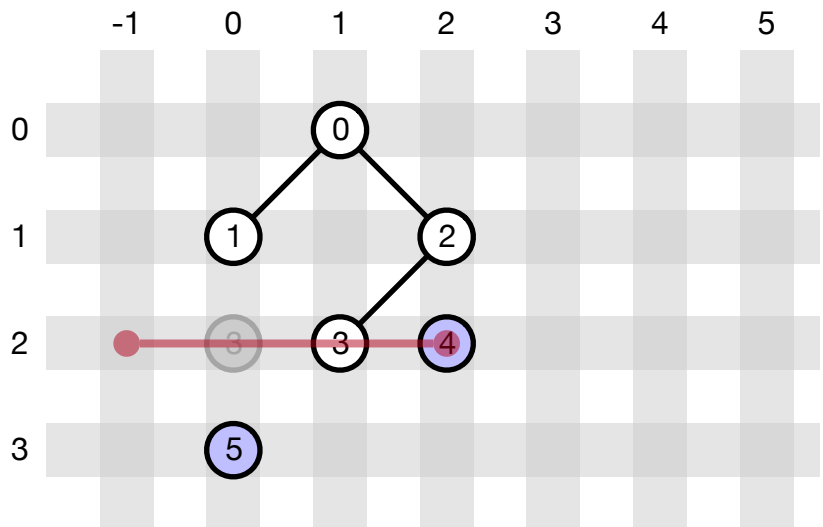

0
1
2
1
3
4
3
5

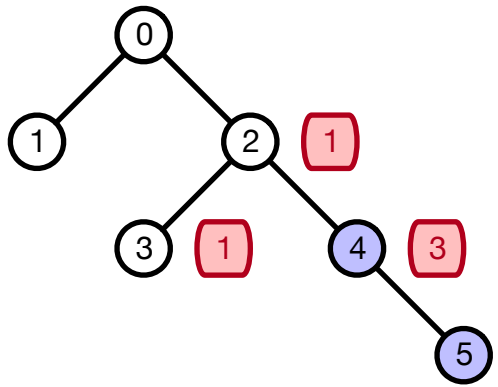

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5

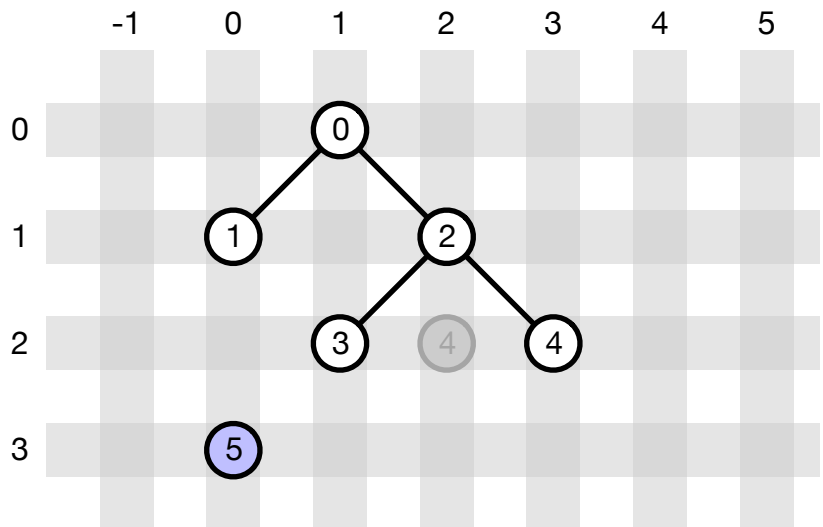

0
1
2
1
3
4
3
5

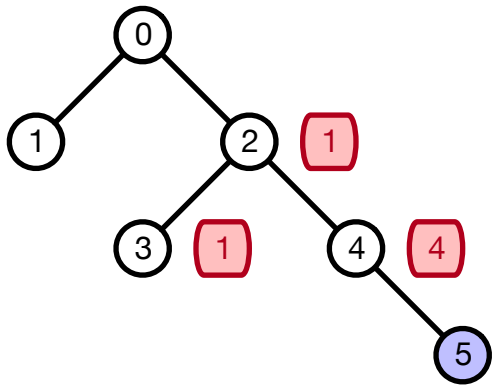

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5

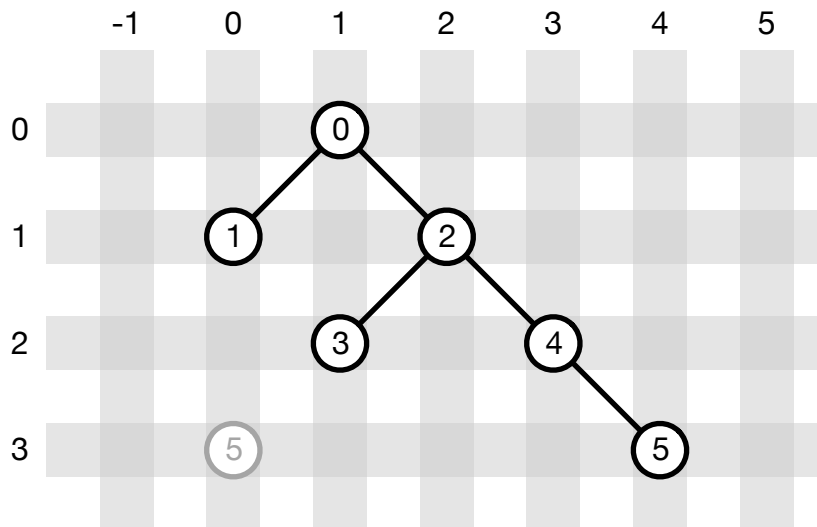

0
1
2
1
3
4
3
5

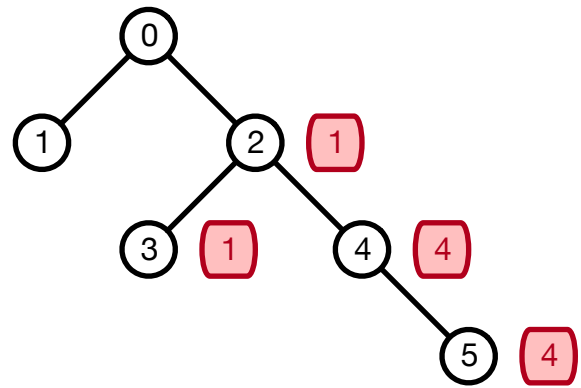

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5

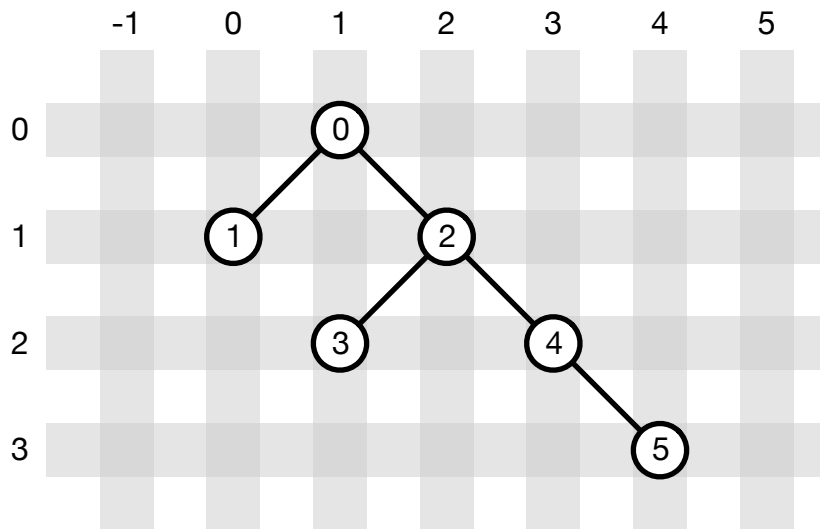

0
1
2
1
3
1
4
3
5

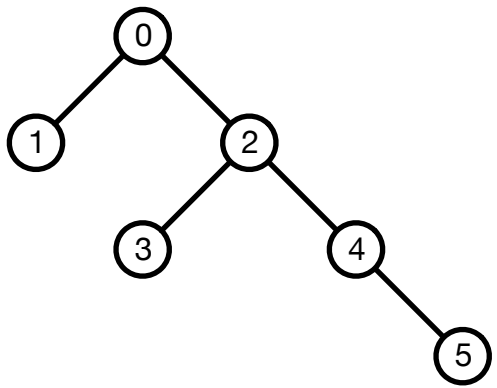

-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
4
5


0
1
2
1
3
1
4
4
5


-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5


0
1
2
1
3
1
4
4
5
4


-1
0
1
2
3
4
5
0
1
2
3
0
1
2
3
4
5


0
1
2
3
4
5

Phase One Setup

Setup:

- get `vertices` in post-order
- store next available column at each depth, `col`
- a `Map` `pos` for each (horizontal) position
- a `Map` `offest` for each horizontal offset

sim. to greedy pos.

key = vtx id,
val = col.

Post-order Iteration over v

Child aware position, `curPos`:

- if leaf, set to next available column at v's depth
- if only left child, v is col to right of child
- if only right child, v is col to left of child
- if two children, v's col is midpoint of children

If not leaf, update `offset` of v to

- max of next availble col, `curPos`

Set v's position to `curPos + offset` (if non-leaf)

- update col at v's depth to be v's position + 2 ← make room for a parent

Phase 2

Pre-order Iteration over v

Set final position of v to

- row = v's depth
- col = v's provisional position + *sum* of ancestor's offsets

Tidy Drawing Demo

Homework 08

Implement the Tidy Drawing procedure yourself!

Input:

- A `BinaryTree`

Output:

- The row/column of each vertex according to Tidy Tree procedure