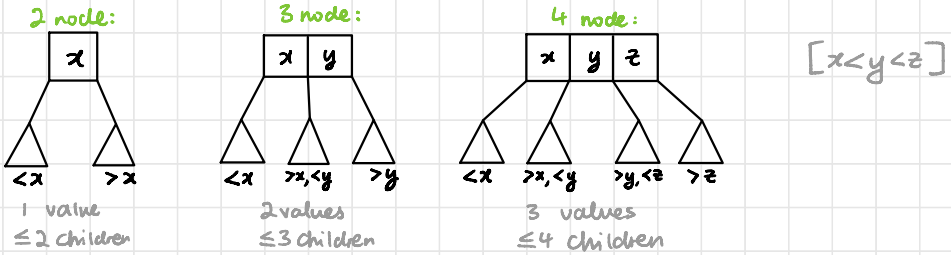
TUTORIAL 7

Recap: 2-3-4 trees.

We have already seen one way to make self-balancing trees: AVL trees. These relied on rotations to correct height imbalances created during insertion. Crucially, however, the node structure was binary (i.e. ≤ 2 children), and the efficiency of the tree was determined by how efficiently we could compute tree heights.

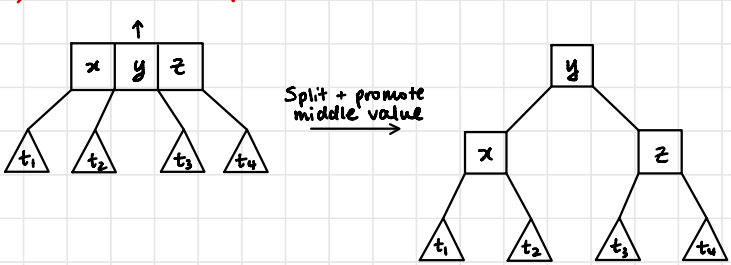
2-3-4 trees are another take on the idea of self-balancing trees that uses a multi-way node structure:



The maximum #. of children that a node can have is called its order.

To keep balance, we instead use a system of node promotion when a node becomes too full:

1. Find the node where the target value belongs
2. If the order of the node is <3, then just add the value into the node and increase its order
3. Otherwise, the node is of order 4:

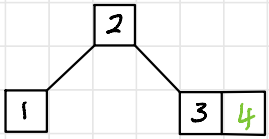


Then insert into the appropriate of the new subtrees.

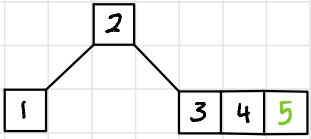
When inserting you might have to promote multiple times, but only $O(\log n)$ many times, so the complexity of an insert is $O(\log n)$ in the worst case.

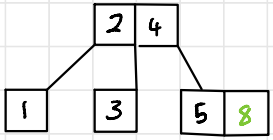
Defining characteristic: all leaf nodes are equidistant from the root node (i.e. all of the same height).

1. Insert ~~1~~ ~~2~~ ~~3~~ 4 5 8 6 7 9 10
   Search for 1 7 9 13

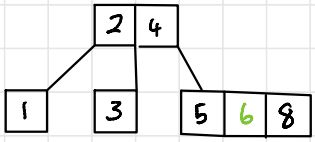


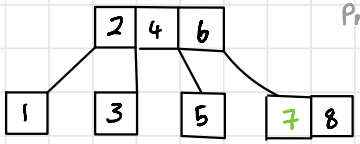
Promote + split b.c. root was full



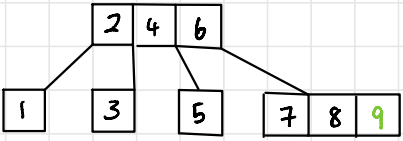


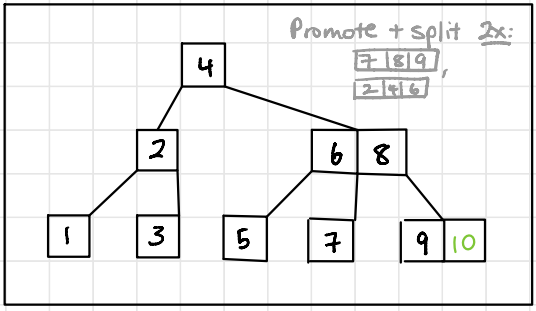
Promote + split
3 4 5





Promote + split
5 6 8





Recap: Graph traversal.

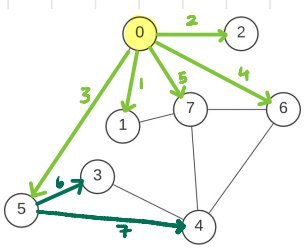
When it comes to visiting the vertices of a graph, there are 2 main approaches:

1. Depth-first traversal, which explores a path as deep as it goes before backtracking on offshoots, repeating until all reachable vertices are visited
2. Breadth-first search, which explores all available paths one edge at a time, fanning out and repeating until all reachable vertices are visited.

So DFS goes deep first, but BFS goes wide first.

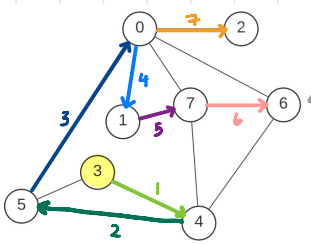
Implementation-wise they are both very similar: a DFS uses a stack (LIFO) to induce the deep-first preference for neighbours, whereas BFS uses a queue (FIFO) to induce the fanning out preference. Note however that DFS is most commonly implemented recursively, as it does not require a stack. BFS is hard to implement like this.

2. BFS starting at vertex 0:



Iteration	Choice	Visited	Queue (head → tail)
1	0	0	1 2 5 6 7
2	1	0 1	2 5 6 7 7
3	2	0 1 2	5 6 7 7
4	5	0 1 2 5	6 7 7 3 4
5	6	0 1 2 5 6	7 7 3 4 4 7
6	7	0 1 2 5 6 7	7 3 4 4 7
7	3	0 1 2 5 6 7 3	3 4 4 7 4
8	4	0 1 2 5 6 7 3 4	4 4 7 4
⋮	⋮	⋮	⋮

DFS starting at vertex 3:



Iteration	Choice	Visited	Stack (top → bottom)
1	3	3	4 5
2	4	3 4	5 6 7 5
3	5	3 4 5	0 6 7 5
4	0	3 4 5 0	1 2 6 7 6 7 5
5	1	3 4 5 0 1	7 2 6 7 6 7 5
6	7	3 4 5 0 1 7	6 2 6 7 6 7 5
7	6	3 4 5 0 1 7 6	2 6 7 6 7 5
8	2	3 4 5 0 1 7 6 2	6 7 6 7 5

NB: We typically represent visited as an array where visited[v] = 1 if we have visited v, and 0 otherwise. Above representation is for convenience.