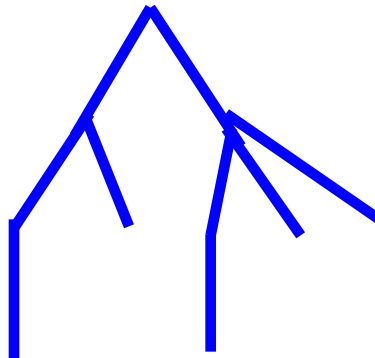


Advanced Data Structures

Succinct Data Structures

Arbitrary Ordered Trees

- Use parenthesis notation
- Represent the tree



- As the binary string $((()())((()())))$:
traverse tree as “(“ for node, then subtrees,
then “)”
- 2 Bits per node

- Transform a tree into a binary string.
- Construct a tree from a binary string.

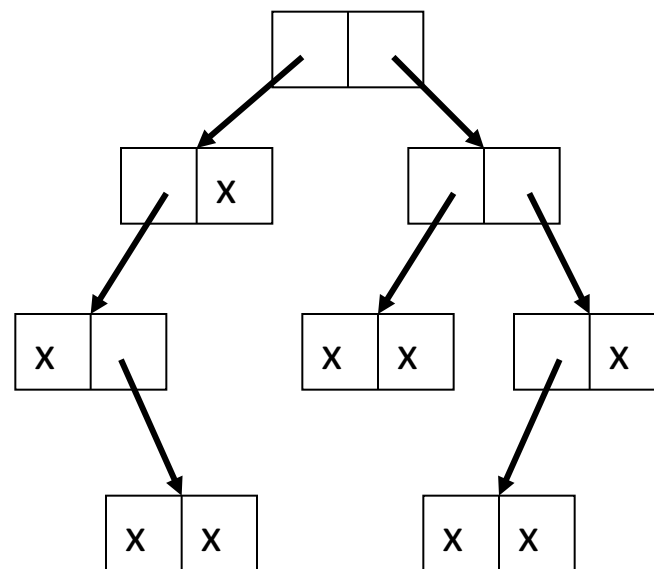
Space for trees

- The space used by the tree structure could be the dominating factor in some applications.
 - Eg. More than half of the space used by a standard **suffix tree** representation is used to store the tree structure.
- Standard representations of trees support very few operations. To support other useful queries, they require a large amount of extra space.

Standard representation

Binary tree: each node has two pointers to its left and right children

An n -node tree takes $2n$ pointers or $2n \lg n$ bits



Supports finding **left child** or **right child** of a node (in constant time).

For each extra operation (eg. **parent**, **subtree size**) we have to pay, roughly, an additional **$n \lg n$** bits.

Can we improve the space bound?

- There are less than 2^{2n} distinct binary trees on n nodes.
- $2n$ bits are enough to distinguish between any two different binary trees.
- Can we represent an n node binary tree using $2n$ bits?

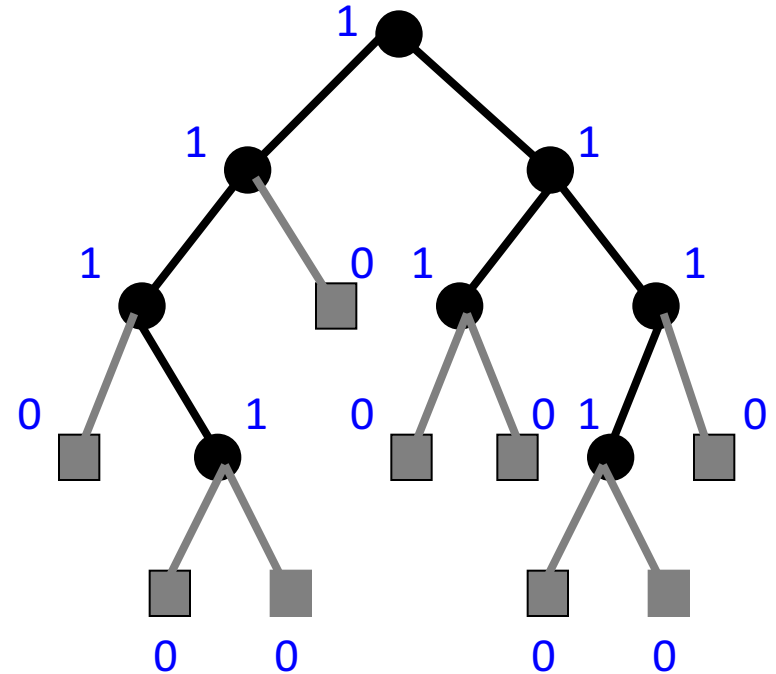
Heap-like notation for a binary tree

Add external nodes

Label internal nodes with a 1
and external nodes with a 0

Write the labels in level order

11110110100100000



One can reconstruct the tree from this sequence

An n node binary tree can be represented in $2n+1$ bits.

What about the operations?

Heap-like notation for a binary tree

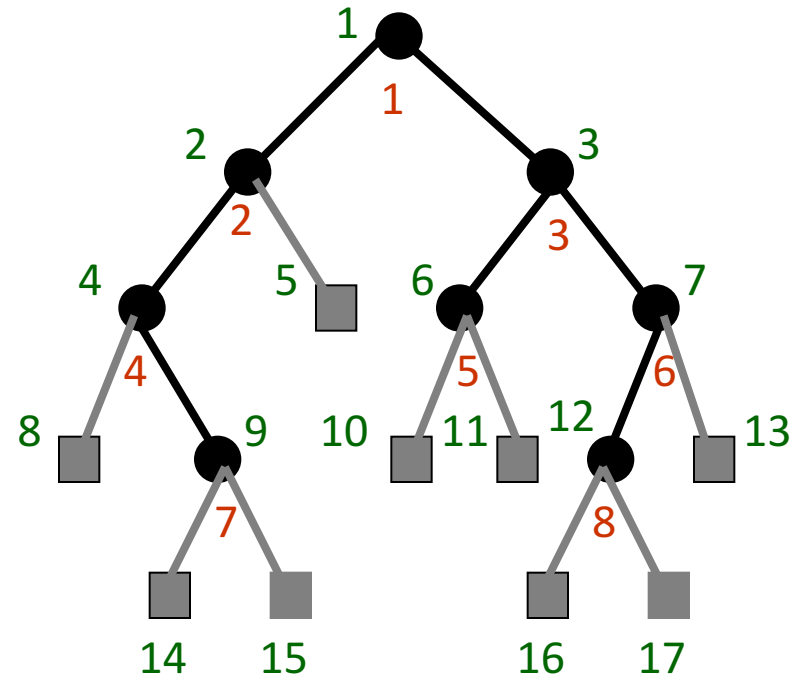
left child(x) = $[2x]$

right child(x) = $[2x+1]$

parent(x) = $[x/2]$

$x \rightarrow x$: # 1's up to x

$x \rightarrow x$: position of x -th 1



1 2 3 4 5 6 7 8

1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Rank/Select on a bit vector

Given a bit vector B

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B:	0	1	1	0	1	0	0	0	1	1	0	1	1	1	1

$\text{rank}_1(i) = \# \text{ 1's up to position } i \text{ in } B$

$\text{select}_1(i) = \text{position of the } i\text{-th 1 in } B$

(similarly rank_0 and select_0)

Given a bit vector of length n , by storing an additional $\mathcal{O}(n)$ -bit structure, we can support all four operations in constant time.

$\text{rank}_1(5) = 3$
 $\text{select}_1(4) = 9$
 $\text{rank}_0(5) = 2$
 $\text{select}_0(4) = 7$

An important substructure in most succinct data structures.

Have been implemented.

Binary tree representation

- A binary tree on n nodes can be represented using $2n+o(n)$ bits to support:

- parent
- left child
- right child

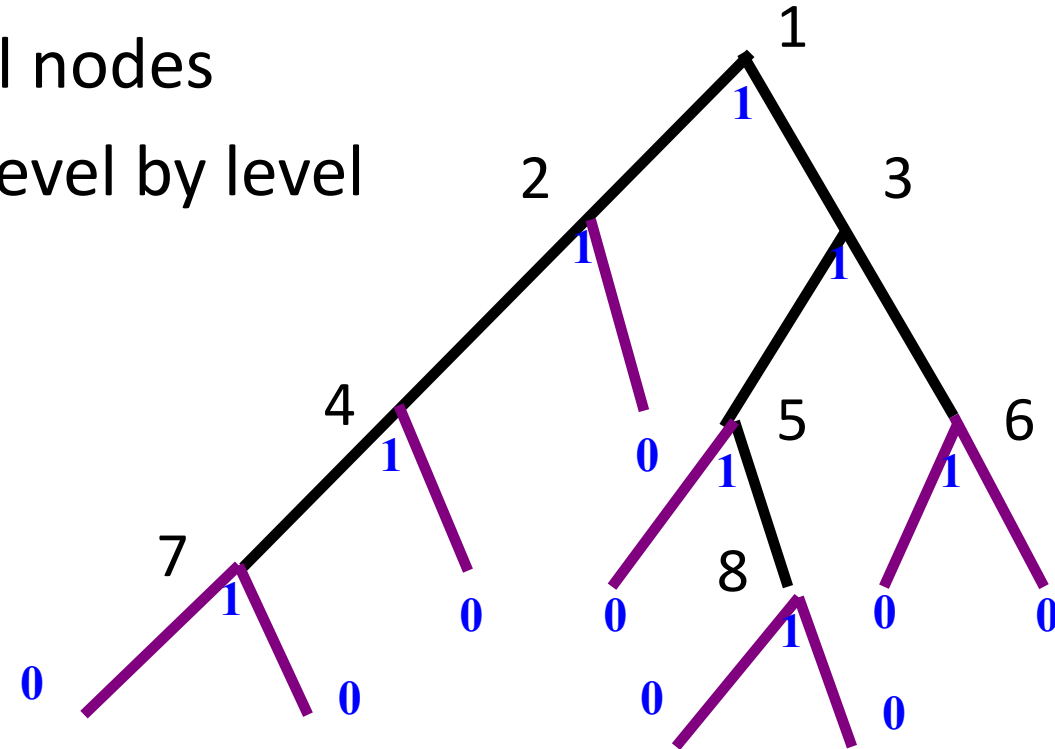
in constant time.

- 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 0

Heap-like Notation for a Binary Tree

Add external nodes

Enumerate level by level



1 2 3 4 5 6 7 8

Store vector 1 1 1 1 0 1 1 1 0 0 1 0 00000 length $2n+1$

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7

Ordered trees

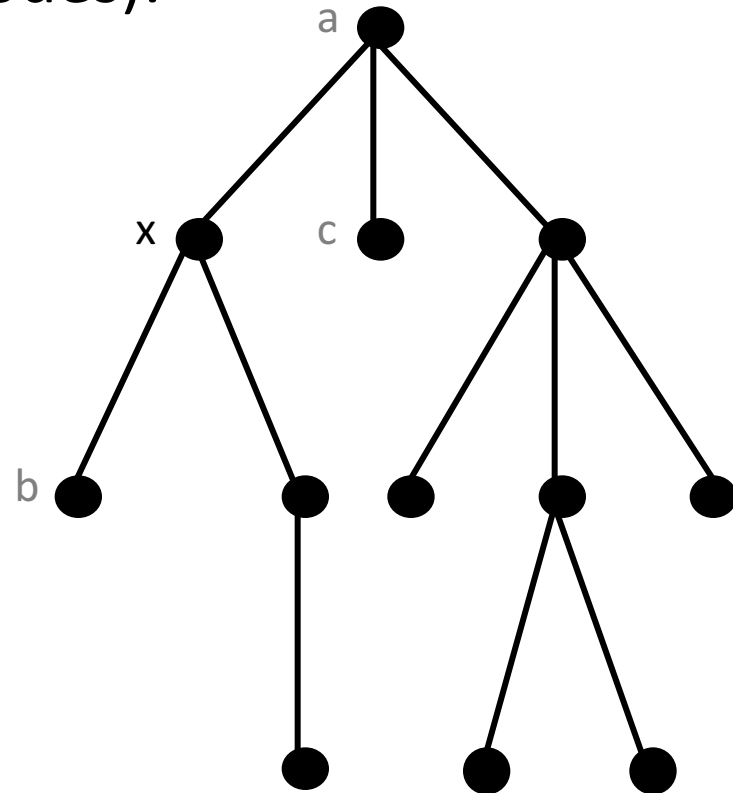
A rooted ordered tree (on n nodes):

Navigational operations:

- $\text{parent}(x) = a$
- $\text{first child}(x) = b$
- $\text{next sibling}(x) = c$

Other useful operations:

- $\text{degree}(x) = 2$
- $\text{subtree size}(x) = 4$



Ordered trees

- A binary tree representation taking $2n+o(n)$ bits that supports **parent**, **left child** and **right child** operations in constant time.
- There is a one-to-one correspondence between binary trees and rooted ordered trees
- Gives an ordered tree representation taking $2n+o(n)$ bits that supports **first child**, **next sibling** (but not **parent**) operations in constant time.
- We will now consider ordered tree representations that support more operations.

Level-order degree sequence

Write the degree sequence in level order

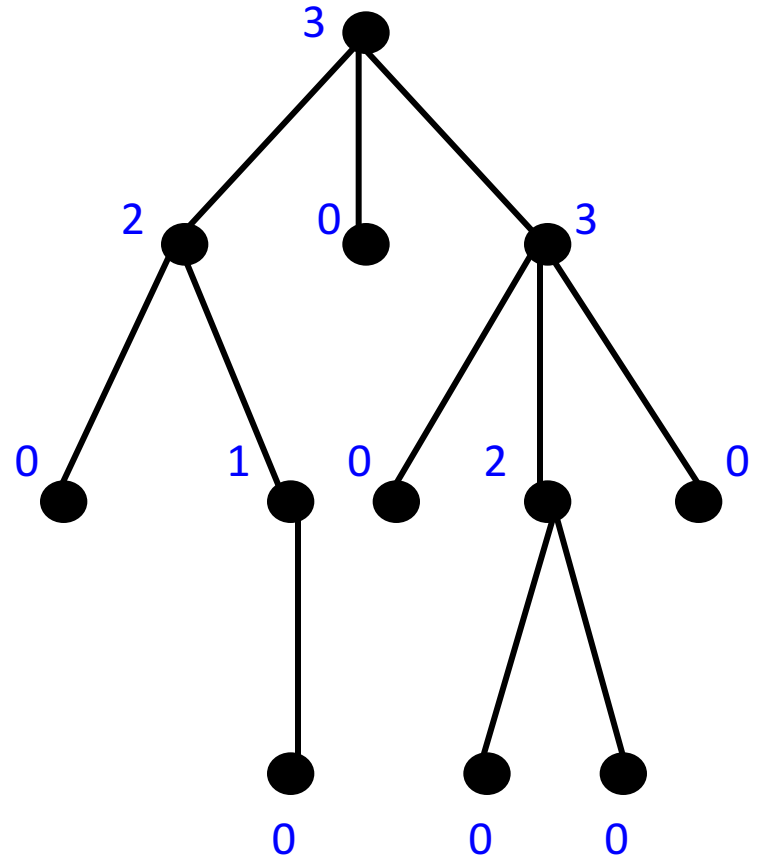
3 2 0 3 0 1 0 2 0 0 0 0

But, this still requires $n \lg n$ bits

Solution: write them in unary

1 1 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0

Takes $2n-1$ bits



A tree is uniquely determined by its degree sequence

Supporting operations

Add a dummy root so that each node has a corresponding 1

1 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0
1 2 3 4 5 6 7 8 9 10 11 12

node k corresponds to the
 k -th 1 in the bit sequence

$\text{parent}(k) = \# \text{ 0's up to the } k\text{-th 1}$

children of k are stored after the k -th 0

supports: parent , i -th child, degree

(using rank and select)

