Data Structures

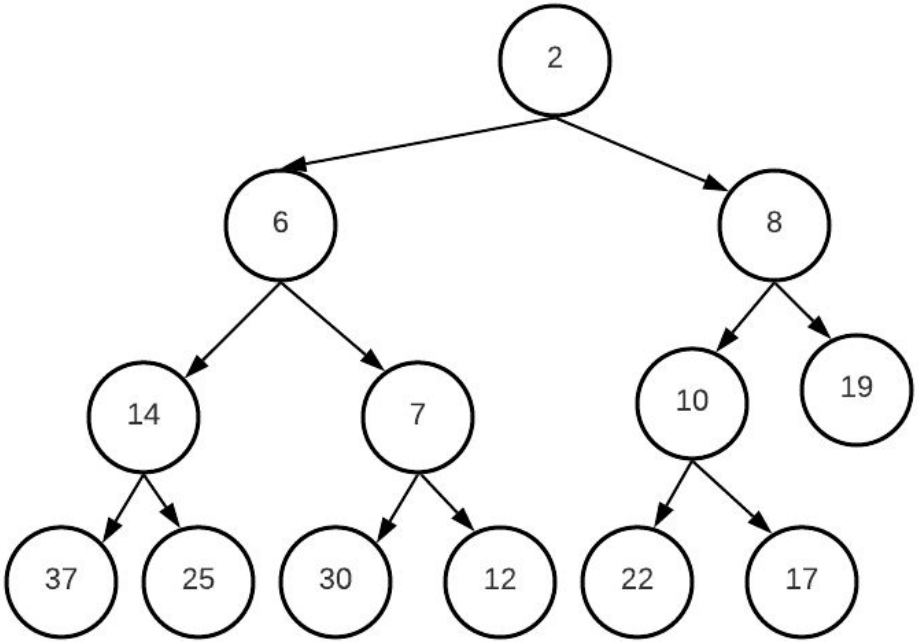
Binary tree using array representation

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

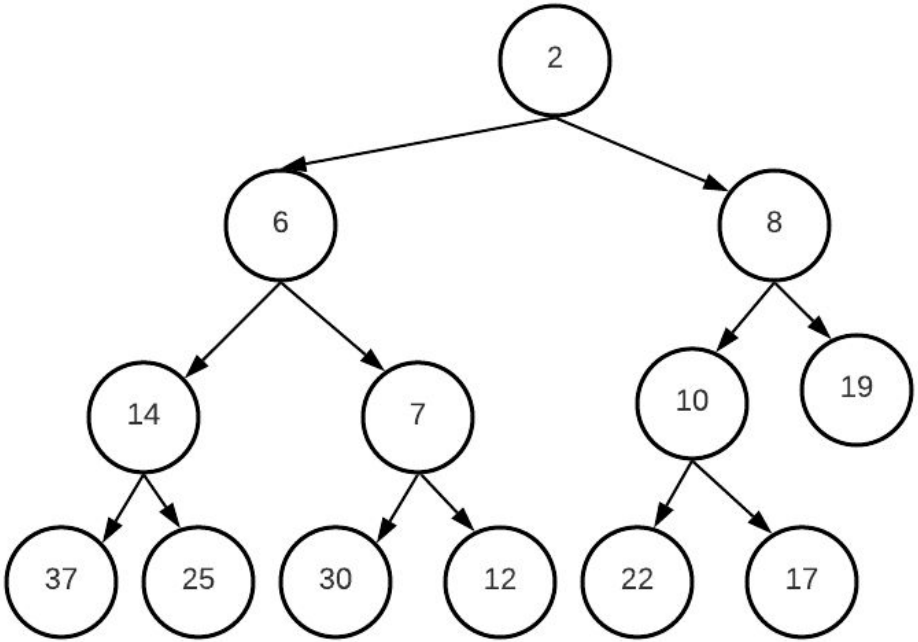
Utilizing the properties

- As clarified before, when something has extra special characteristics, there could be potential advantages/ways to handle than the general way
- What is special about perfect and complete trees?
    - Many top levels are complete



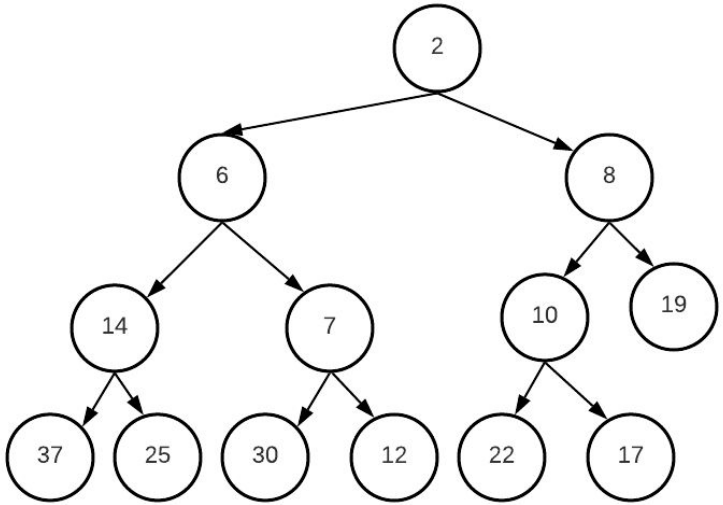
Utilizing the properties

- Nodes per level are 2^i
  - Level 0 = 1
  - Level 1 = 2
  - Level 2 = 4
  - Level 3 = 8
  - Level 4 = 16
  - Level 5 = 32
  - ….
- If we have n levels, total number of nodes 2^n -1
- We can simply put **tree level order traversal** in an array



Complete tree ⇐⇒ Level order traversal

- Interestingly: we can convert a tree level-order traversal to the original binary tree
  - Consume: 1, 2, 4, 8, 16, etc
- But what is really important:
  - Given the index of node, what is parent index?
  - Given the index of node, what are children indies?
  - Try to find simple formulas



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	6	8	14	7	10	19	37	25	30	12	22	17		

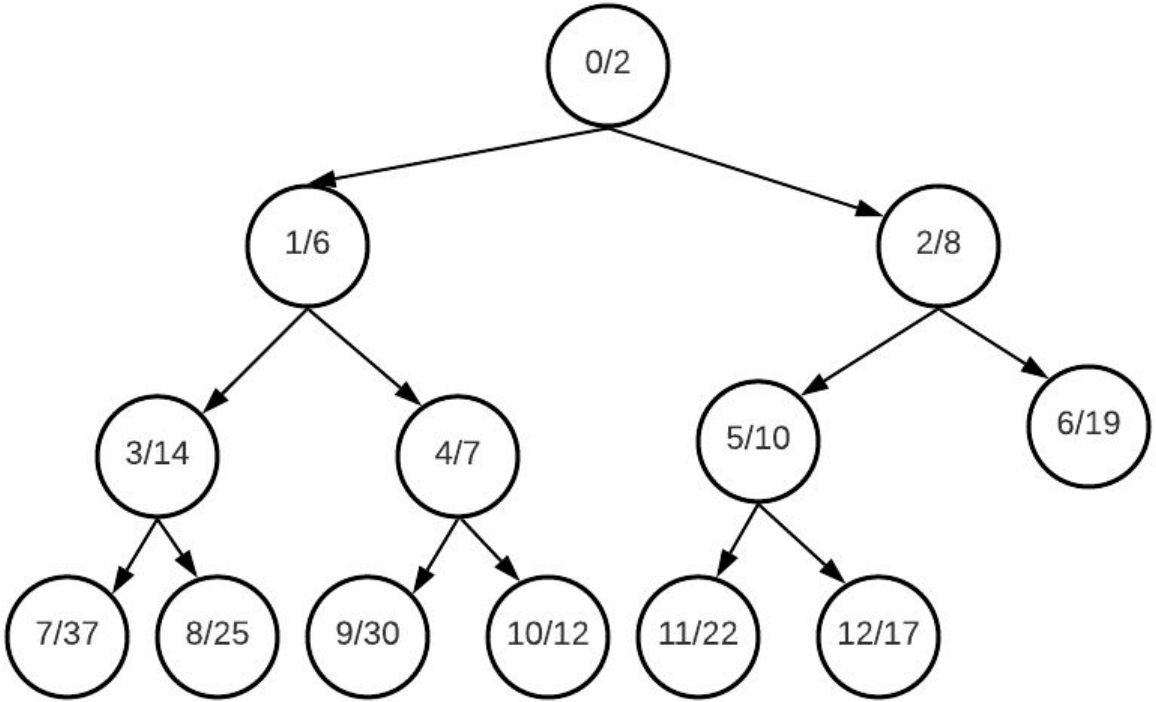
Indices relations

Index	Value	Parent index	Child 1 index	Child 2 index
0	2	NA	1	2
1	6	0	3	4
2	8	0	5	6
3	14	1	7	8
4	7	1	9	10
5	10	2	11	12
6	19	2	13	14
7	37	3	15	16
8	25	3	17	18

What are the formulas?

- Given node i, its children are
  - 2*i + 1
  - 2*i + 2
- Given node i, its parent is:
  - floor((i-1) / 2)

Indices relations



What are the formulas?

- Given node i, its children are
  - 2*i + 1
  - 2*i + 2
- Given node i, its parent is:
  - floor((i-1) / 2)

So far

- We can represent a complete tree in an array (level order traversal)
- We can trivially move from a node to its parent on the array
- We can trivially move from a node to its children on the array
- Why is that cool? For the heap

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	6	8	14	7	10	19	37	25	30	12	22	17		

Heap representation

- We can represent a heap using an array
- *Ok but we can use a normal pointer based tree?*
- Yes, but now we can find the next **available** node position trivially!
  - Below a tree of 13 nodes!
  - The next available node in the tree is simply index #13 ⇒ O(1)
- In next lecture: how is that useful for a heap!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	6	8	14	7	10	19	37	25	30	12	22	17		

Array representation

- Direct implementation
- Sometimes, the current tree doesn't have a child. Or the node is root and no parent
  - Use -1 as indicator
- bool? If_true: if_false
  - C++ ternary operator

```
int *array{};
int size {};

int left(int node) {
    int p = 2 * node + 1;
    if (p >= size)
        return -1;
    return p;
}
int right(int node) {
    int p = 2 * node + 2;
    return p >= size ? -1 : p;
}
int parent(int node) {
    return node == 0 ? -1 : (node - 1) / 2;
}
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."