Priority Queues

Priority Queue

A Priority Queue is an Abstract Data Type that maintains a collection of items which has an associated *priority* value and can get (and remove) the item with highest priority efficiently. New items with arbitrary priorities can be added at any time.

There are a number of obvious implementation strategies we could use:

- Unsorted Array: Inserting an item is $O(1)$, `get` is $O(n)$
- Sorted Array: Insert is $O(n)$, `get` is $O(1)$
- AVL Tree: Insert and `get` are both $O(\log n)$

We can do better than that.

Priority Queue applications

In general, priority queues are useful whenever we repeatedly need the maximum of a changing collection.

For example, if do a search on the web, the underlying search algorithm will find potential matches, each with a score that estimates how good a match it is. It may put those matches into a priority queue, and then extract the 10 best matches from the queue. In the background, further matches might be searched for and added to the queue. When the go to the next page of results, then the next 10 best matches are extracted and displayed.

Similar examples can be found in finding the best next move in a computer game, online flight search websites, hotel booking systems, comparative pricing websites

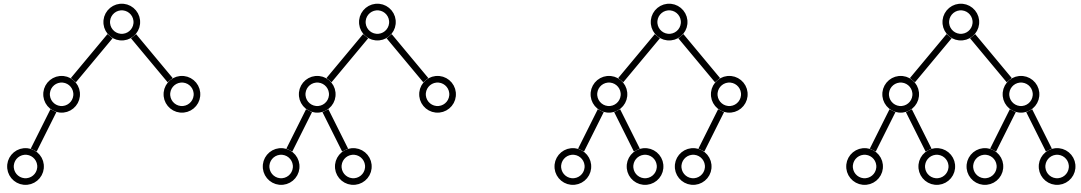
Complete Binary Tree

Our first implementation of Priority Queues will use a type of **Complete Binary Tree**, called a **Binary Heap Tree**, which, in turn, we will implement using an array.

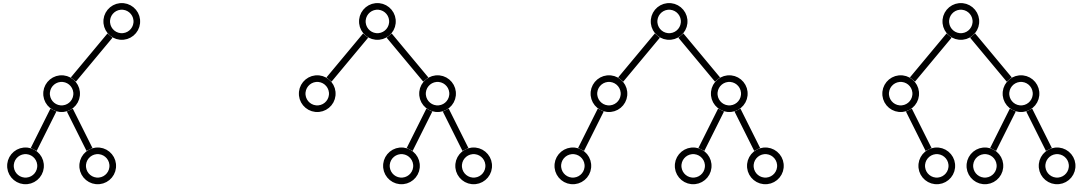
Definition
A binary tree is *complete* if every level, except possibly the last, is completely filled, and all the leaves on the last level are placed as far to the left as possible.

This simply defines, for a binary tree with a given number of nodes, the tidiest, most balanced and compact form possible

Complete:

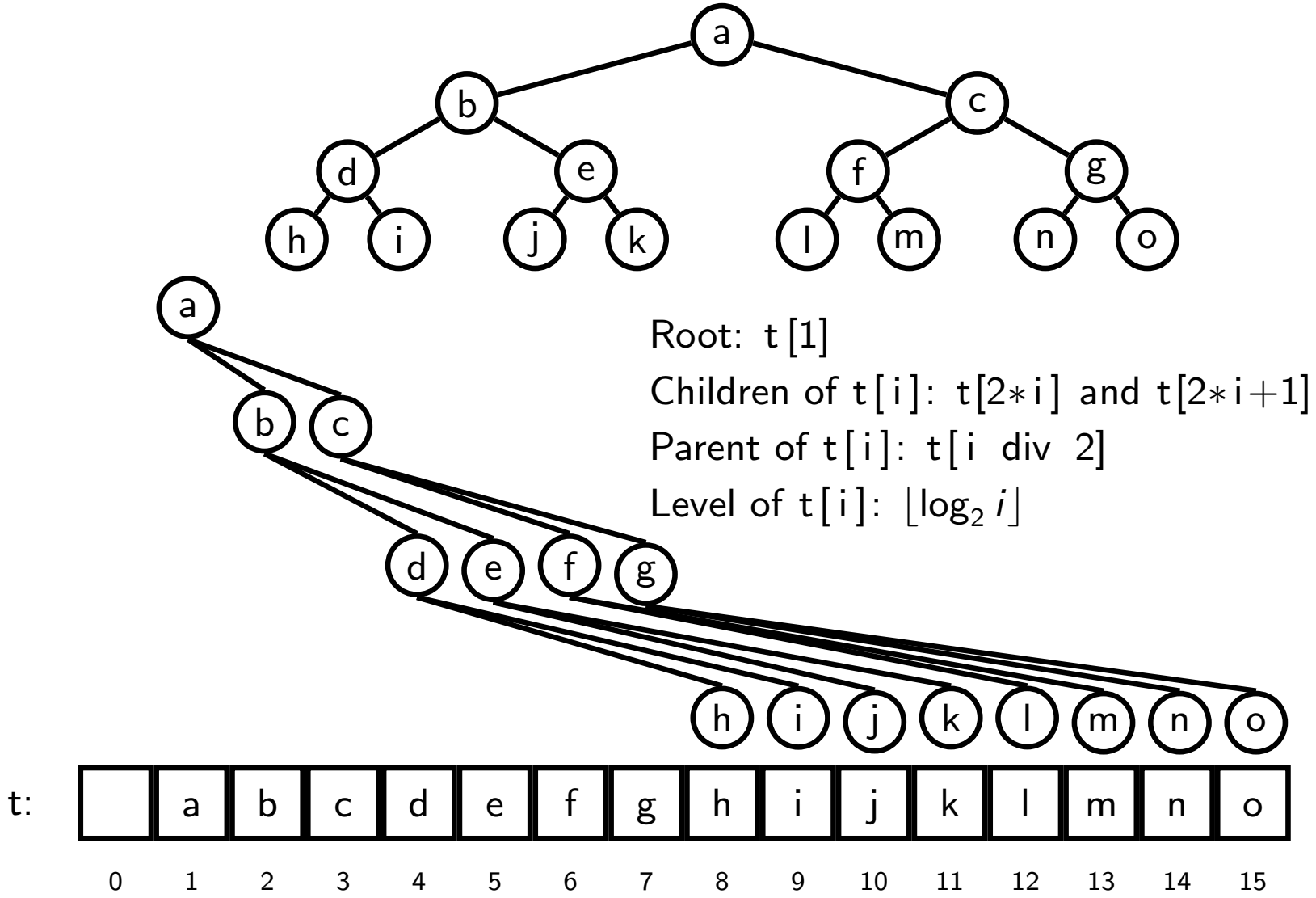


Not Complete:



Array Implementation of Complete Binary Trees

We can store Binary Trees in a simple array structure:



If the binary tree is not complete, then:

- this is a wasteful implementation strategy because space is still reserved for the missing nodes
- when a node is missing, the array cell needs to be marked to indicate that it is not part of the tree
  - use an invalid value for this application (null, -1, etc.), or
  - use a separate data structure (e.g. a bit array — 1 bit per cell of the binary tree array) to indicate whether the corresponding cell contains a real value

If the binary tree is always complete, then:

- A single integer to record number of nodes in the tree is sufficient to identify the end of the tree
- You do not need to mark missing nodes because there are none before the end of the tree