Priority Queue

Charles W. Kann

Overview

- Binary trees with total and partial ordering
- Complete binary tree
  - Perfect tree
  - Complete binary tree
  - Storing complete binary tree and storage in arrays
  - Heap order property
- Priority Queue
  - insert
  - deleteFirst
- Huffman tree and encoding

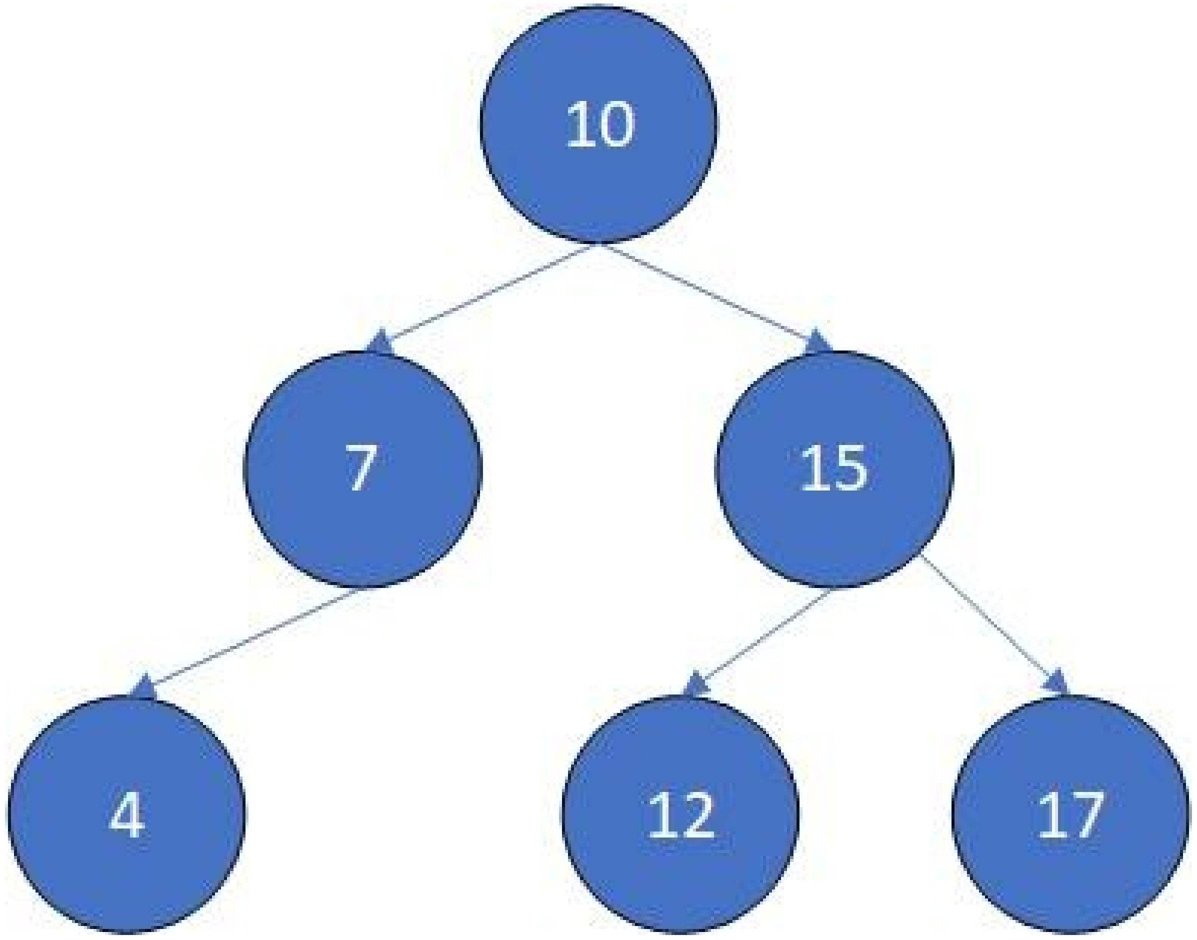
Total Ordering

- In mathematics, a total order is a binary relation on some set X, which is antisymmetric, transitive, and a connex relation.
- Formally
  - Antisymmetry : If $a \leq b$ and $b \leq a$ then $a = b$;
  - Transitivity : If $a \leq b$ and $b \leq c$ then $a \leq c$
  - Connexity : $a \leq b$ or $b \leq a$.

Total Ordering – Binary Search Tree (BST)

- **Binary Search Tree** is a node-based binary tree data structure which has the following properties:
  - The left subtree of a node contains only nodes with keys lesser than the node’s key.
  - The right subtree of a node contains only nodes with keys greater than the node’s key.
  - The left and right subtree each must also be a binary search tree.

BST Example



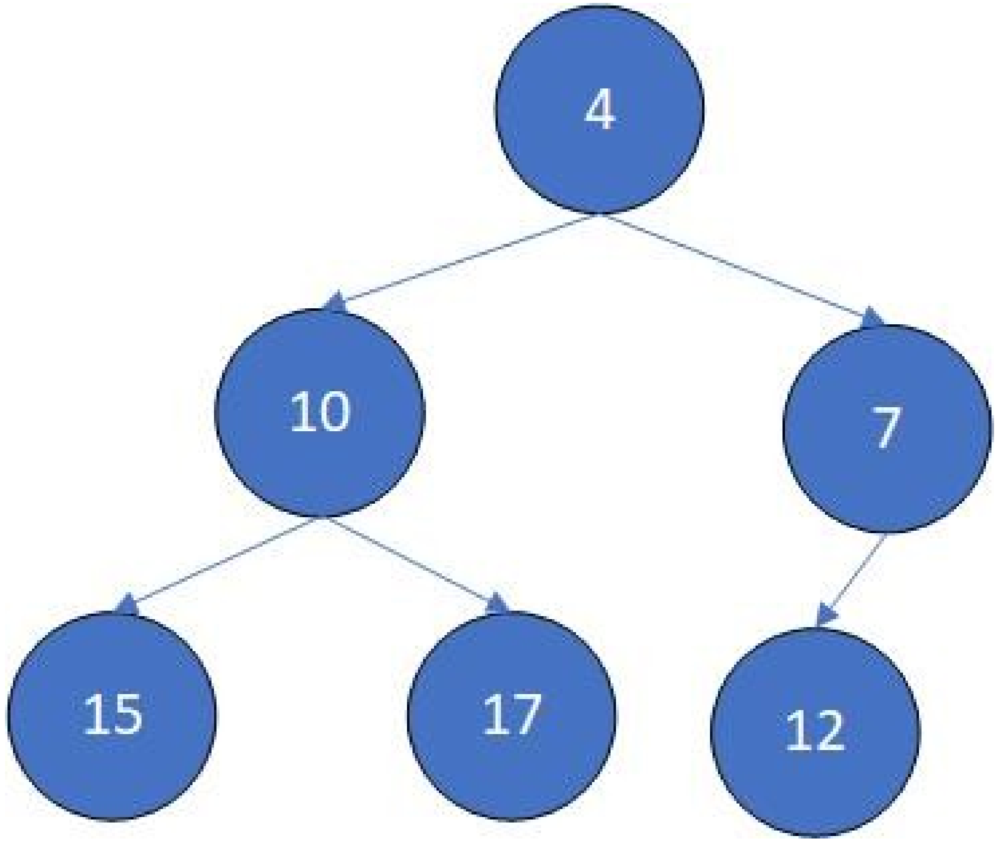
Partial Ordering

- In mathematics a partial order is any binary relation that is reflexive (each element is comparable to itself), antisymmetric (no two different elements precede each other), and transitive (the start of a chain of precedence relations must precede the end of the chain).
  - if $a \leq b$ and $b \leq a$, then $a = b$
  - if $a \leq b$ and $b \leq c$, then $a \leq c$
- Note no connexity, e.g. not all pairs are related.

Partial Ordering – Binary Heap

- A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:
  - the *min-heap property*: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
  - the *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.
- Only need one type of BinaryHeap as the difference between a min-heap and max-heap can be handled by the Comparable or an appropriate Comparator.
- Heap simply means *pile.* So you can have different types of heaps.
  - Specifically a binary heap is not related to heap memory, or any other types of heap

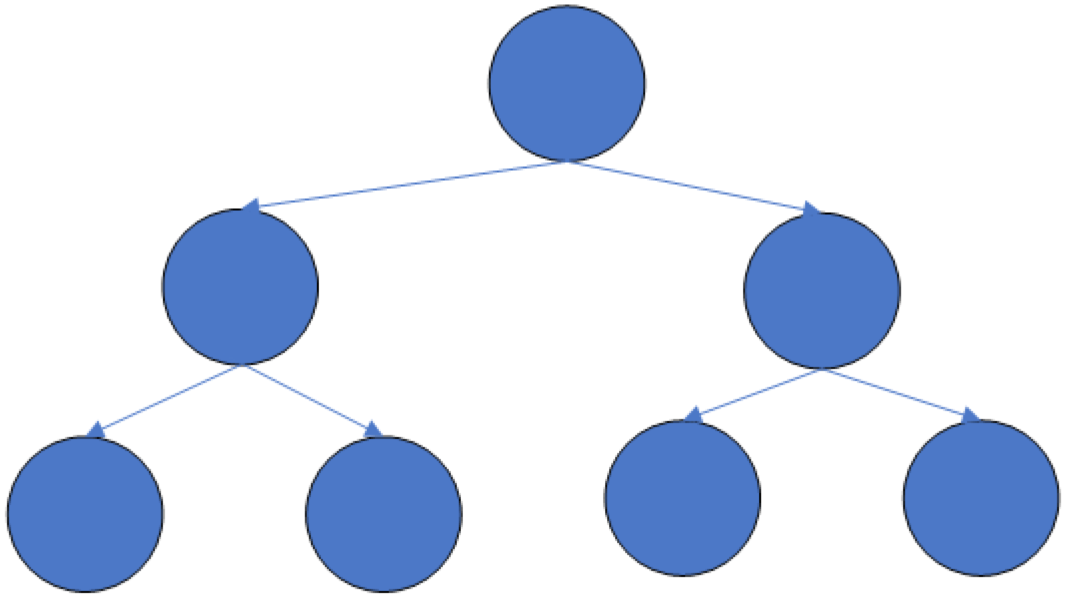
Heap Example



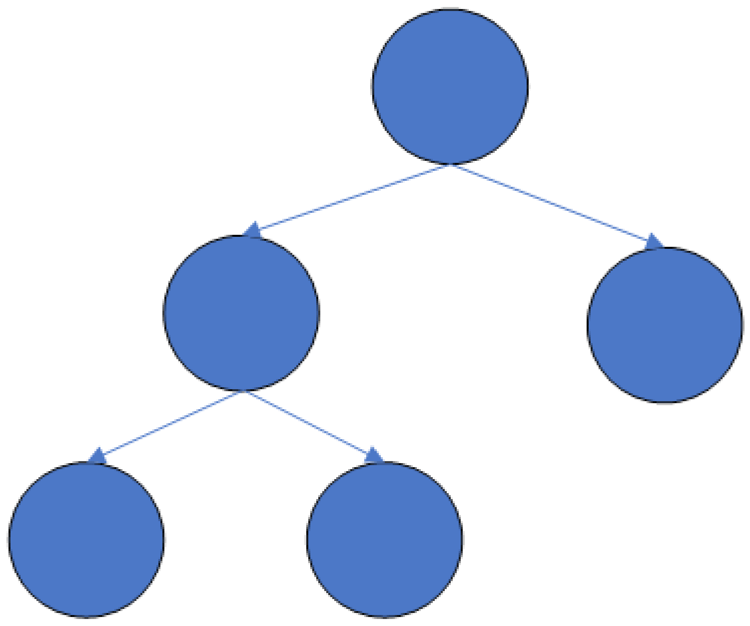
Other Binary Tree Definitions

- Perfect Binary Tree – tree of height n which contains 2^n-1 nodes.
  - Simply, a binary tree of height n where all nodes at height n-1 have 2 children, and all nodes at height n have no children.
- Complete Binary Tree – A n-height Perfect Binary Tree to which nodes are always added to level n+1 in the left most open position; and nodes are always removed at the right most position on level n+1.

Perfect Tree



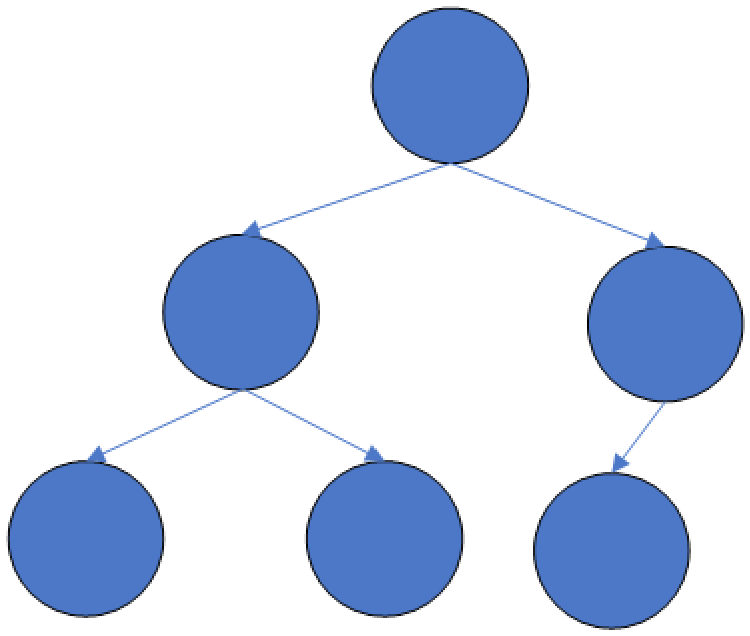
Complete Binary Tree



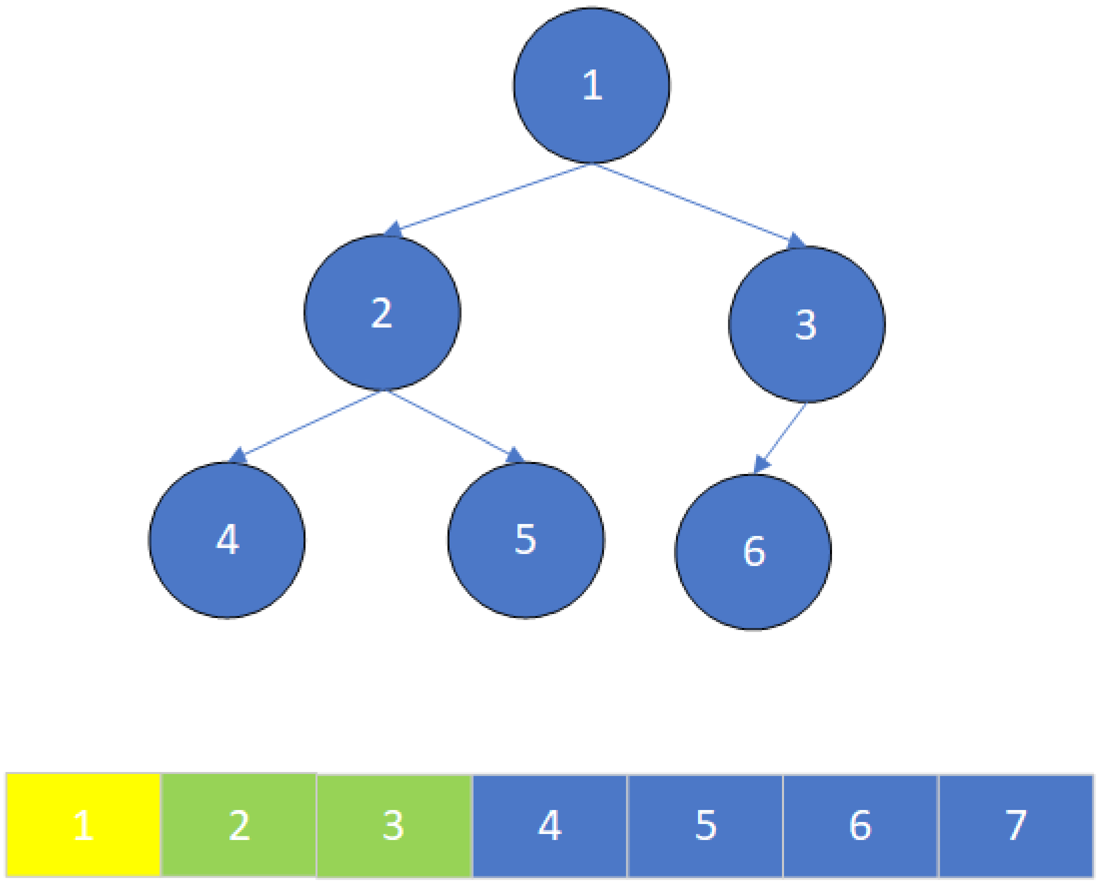
Advantage of CBT – Can be stored in array

- A tree of N nodes requires an array of size N+1
  - Always start with node 1 to more easily calculate parents and children
  - Arbitrary tree requires 2^N nodes, so do not use an array for an arbitrary tree.
- Parents are always (node number/2)
- Child(ren) are always (node number * 2) and ((node number * 2) + 1)

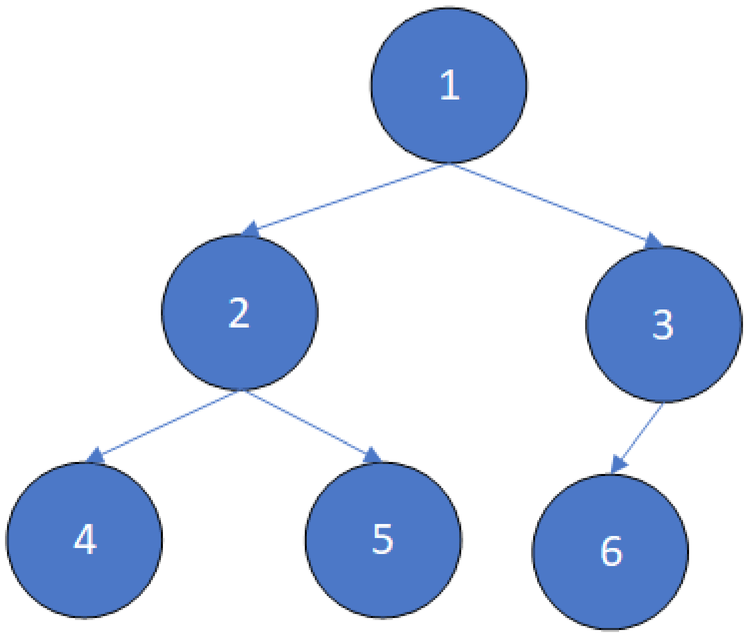
Complete Binary Tree



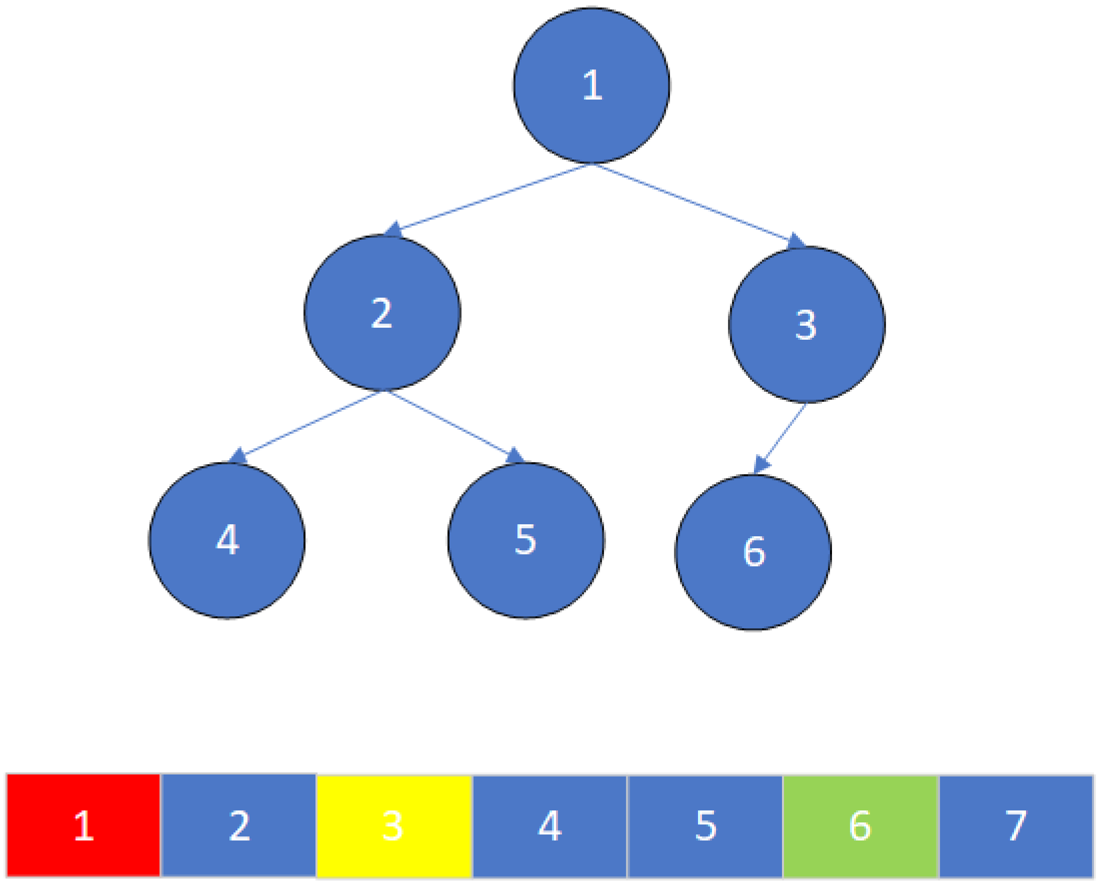
Complete Binary Tree- Parent/Child relationships in array – Node 1- 2,3 (children)



Complete Binary Tree- Parent/Child relationships in array – Node 2- 1 (parent)/2,3 (children)



Complete Binary Tree- Parent/Child relationships in array – Node 3- 1 (parent)/6 (child)



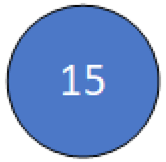
BinaryHeap

- As defined earlier, a BinaryHeap is a CBT with a heap order property, e.g. the children are always less than the parent (max-heap) or the children are always greater than the parent (min-heap)
- A binary heap is a partial ordering.
  - The axioms for a non-strict partial order state that the relation ≤ is reflexive, antisymmetric, and transitive. That is, for all a, b, and c in P, it must satisfy:
  - a ≤ a (reflexivity: every element is related to itself).
  - if a ≤ b and b ≤ a, then a = b (antisymmetry: two distinct elements cannot be related in both directions).
  - if a ≤ b and b ≤ c, then a ≤ c (transitivity: if a first element is related to a second element, and, in turn, that element is related to a third element, then the first element is related to the third element).

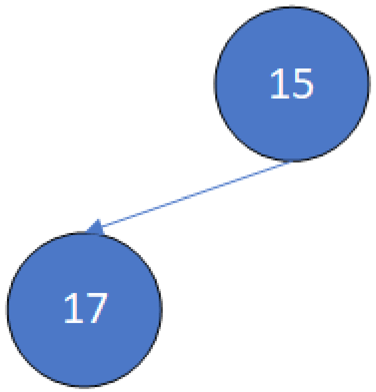
Inserting into a Binary Heap – min-heap

- Add new node at the left most node on last level.
- Bubble-up the node to the correct position in the tree.
- Problem – insert 15, 17, 10, 7, 12, and 4

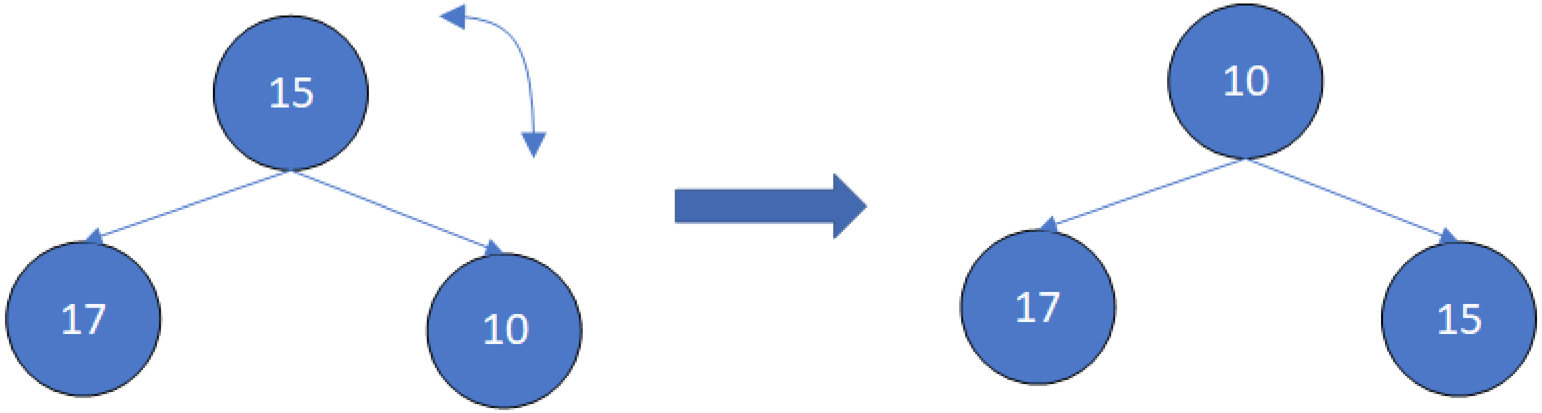
Binary Heap 1



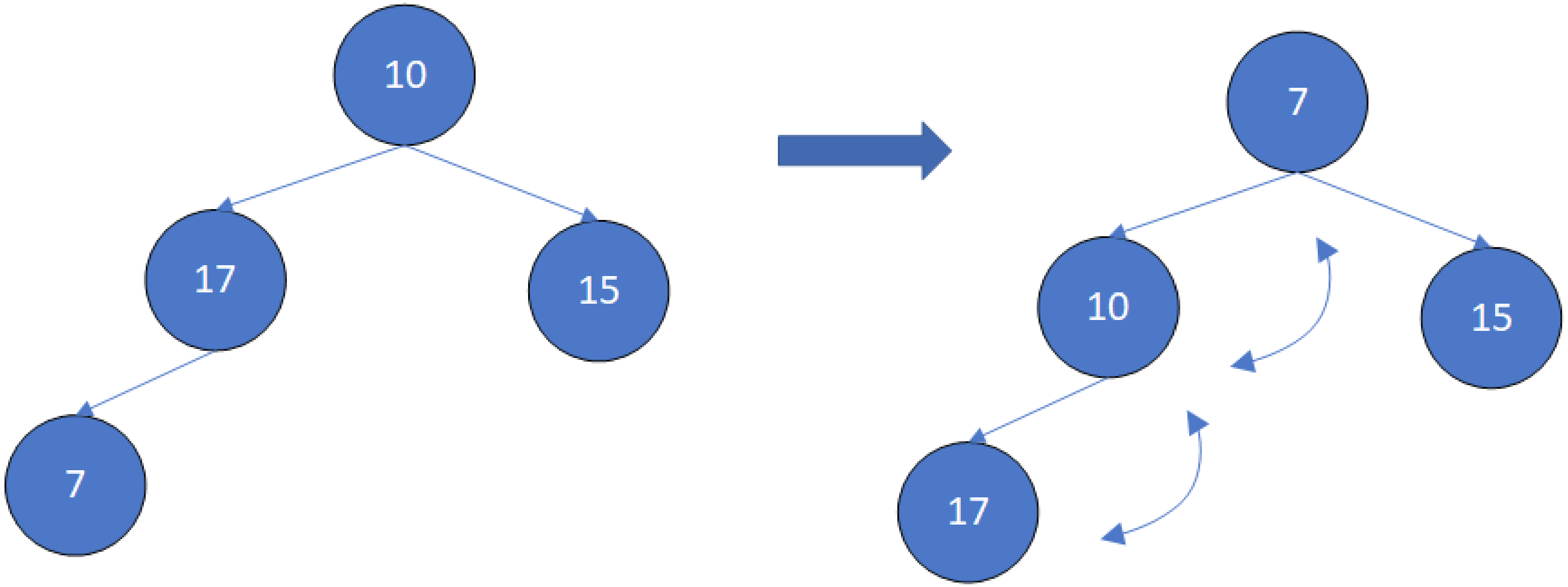
Binary Heap 2



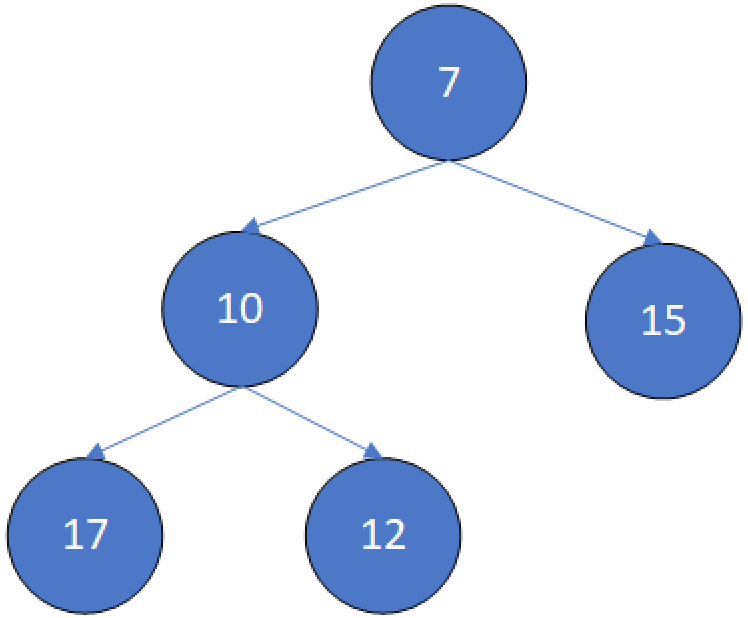
Binary Heap 3



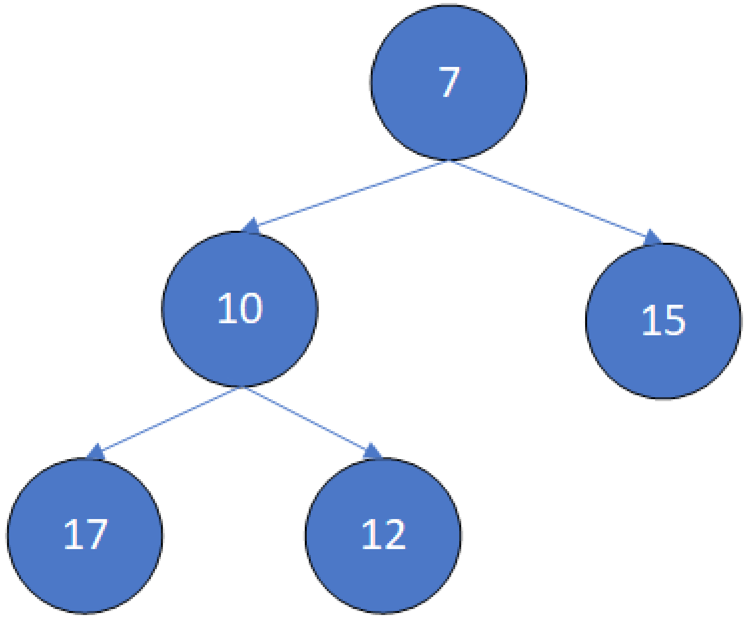
Binary Heap 4



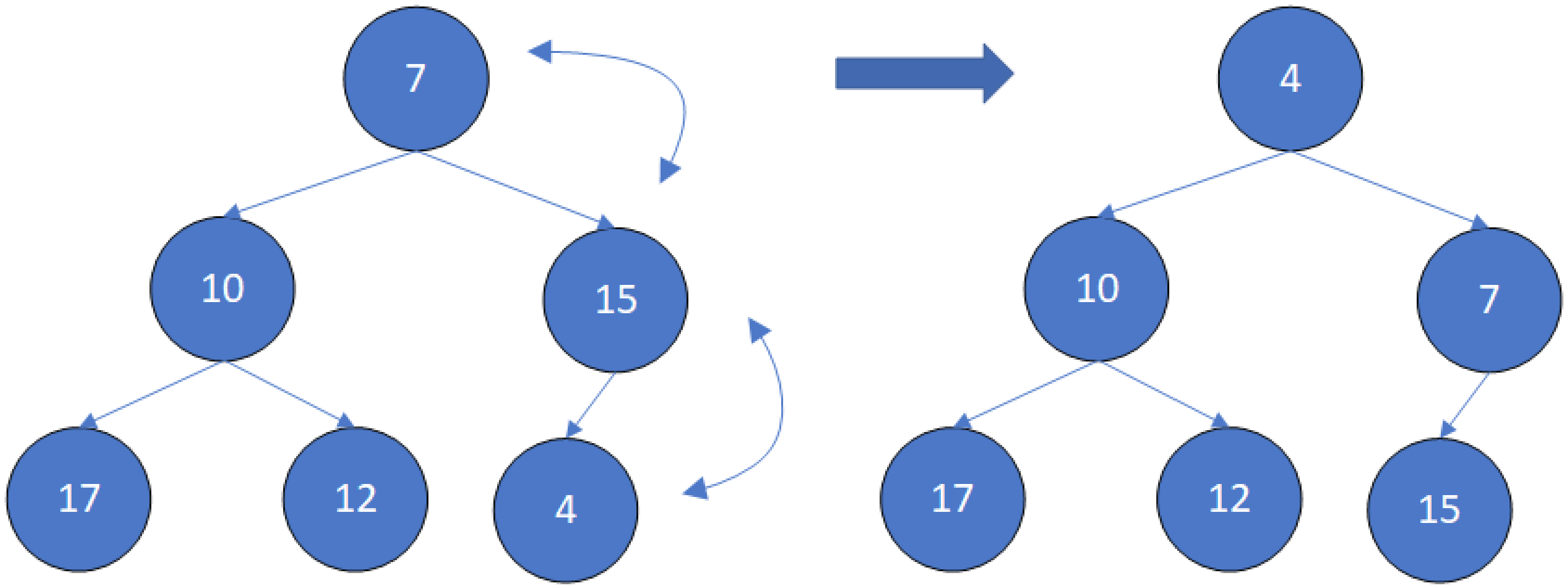
Binary Heap 5



Binary Heap 6



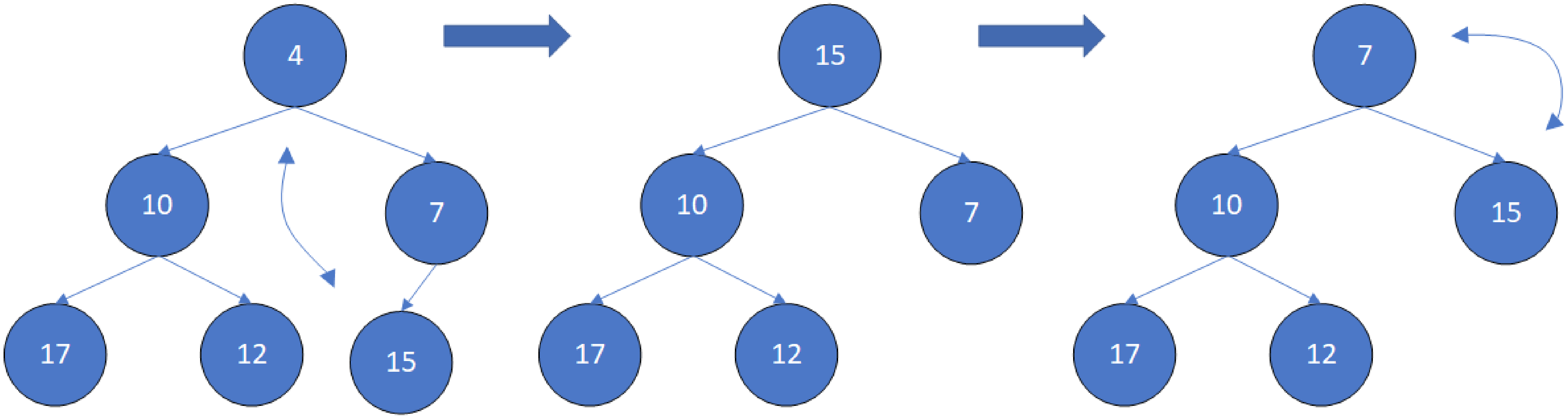
Binary Heap 7



DeleteFirst

- Dequeue the minimum (maximum) value in queue. This is the root.
- Move the last node to the root.
- Bubble-down the node to the correct place.

DeleteFirst



Huffman Code Generation

- Count the number of occurrences of each character
- Build a PriorityQueue on the letter frequency
- Do until PriorityQueue has one element
  - Pop two nodes.
  - Add frequencies
  - Push a new node, with the children being the popped nodes, and the frequency being the added frequency.
- Pop the last node, this is the root of the Huffman tree
- Recursively walk the tree and store the Huffman code with the character.

Code to count Frequency

```
public static int[] countFrequency(String s) {
        int[] frequency = new int[256];

        for(int i = 0, n = s.length() ; i < n ; i++) {
            char c = s.charAt(i);
            frequency[c]++;
        }

        return frequency;
}
```

Count Occurrences

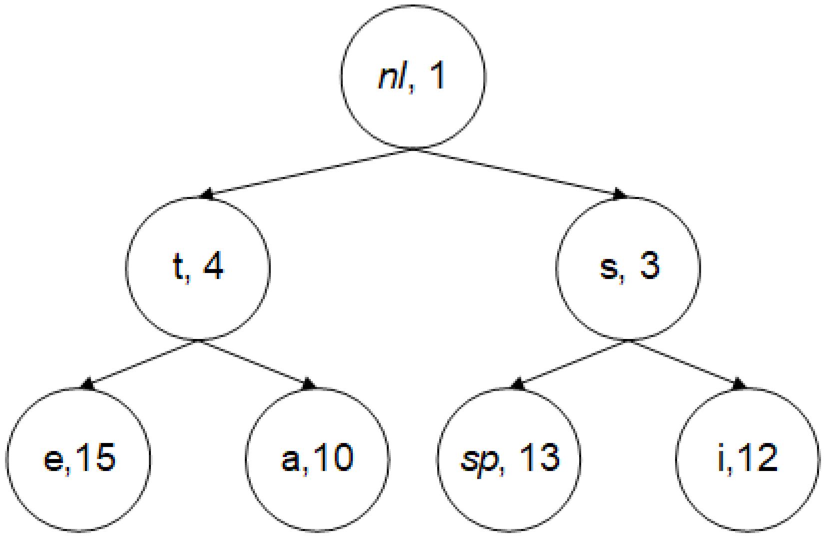
Character	Frequency
a	10
e	15
i	12
s	3
t	4
space	13
newline	1

Create Array of items

A,10	E,15	I, 12	S, 3	T,4	*sp*,13	*nl*,1

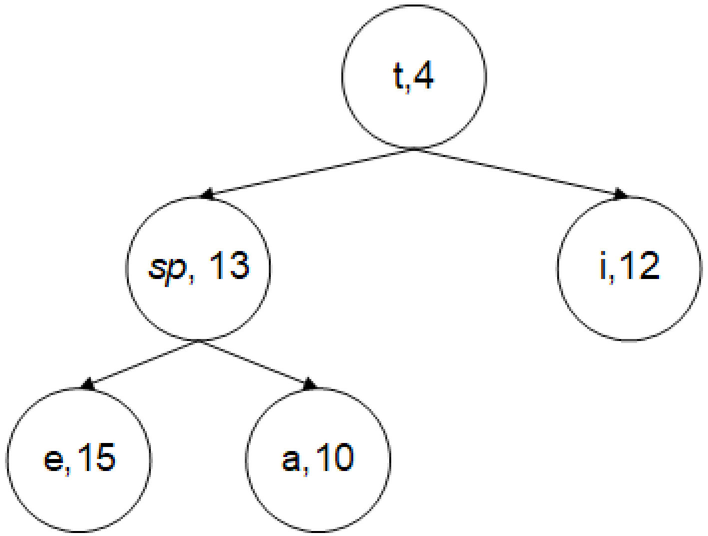
Build a min-head (priority queue) from the data

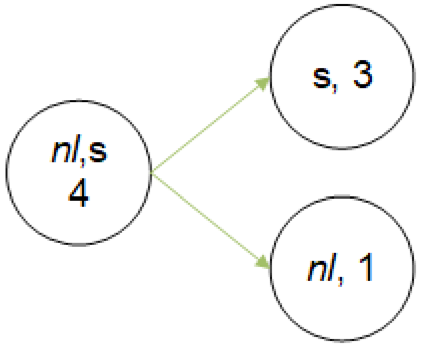
nl, 1	t, 4	s,3	3,15	a,10	*sp*,13	i,12



Delete two smallest nodes and combine

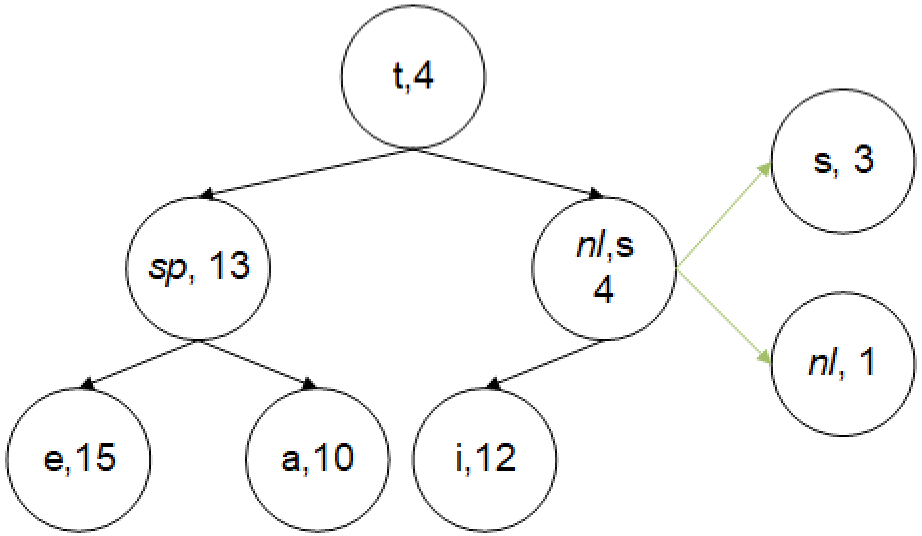
t,4	*sp*,13	i,12	e,15	A,10





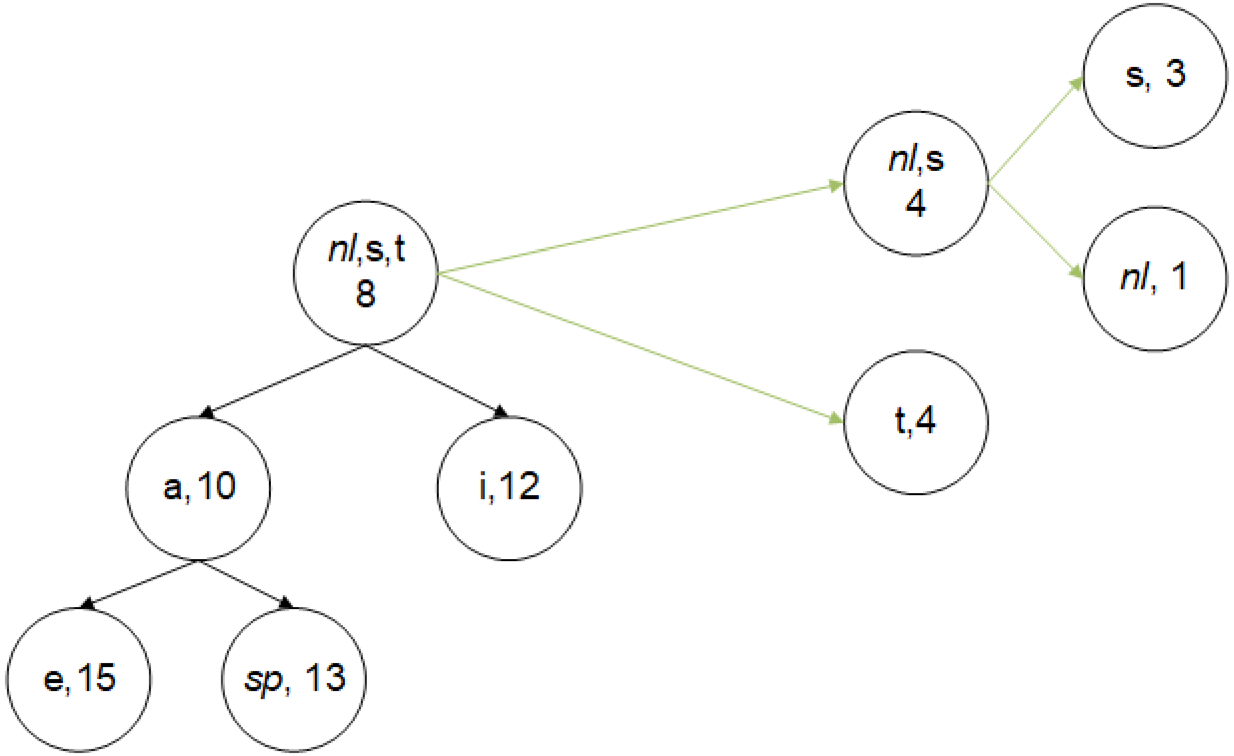
Push new node on min heap. Note that the new Node is part of head, and growing a tree

t,4	*sp*, 13	*nl*,s 4	e,15	a,10	i,12



Repeat until one node is left. This is the Huffman tree

nl,s,t 8	a,10	i, 12	e,15	*sp*,13



Walk Huffman tree to build code