Nonlinear Data Structure

Unit#3

Marwadi University

Department of Computer Engineering

Data Structure
01CE0301 / 3130702

Ravikumar Natarajan

Highlights

- Trees
- Graphs

Till Now

- **Trees**
- Graphs

- **Tree definitions and their concepts**
- **Representation of binary tree**
- **Binary tree traversal**
- **Binary search trees**
- **General trees vs binary trees**
- **Threaded binary tree**
- **Applications of Trees**
- Balanced tree and its mechanism
- Height and Weight Balanced Trees

Up Next

- **Trees**
- Graphs

- Tree definitions and their concepts
- Representation of binary tree
- Binary tree traversal
- Binary search trees
- General trees vs binary trees
- Threaded binary tree
- Applications of Trees
- **Balanced tree and its mechanism**
- **Height and Weight Balanced Trees**

Binary Search Tree - Worst Case

- Worst case running time is O(N)
  - What happens when you Insert elements in ascending order?
  - Insert: 2, 4, 6, 8, 10, 12 into an empty BST
  - Problem: Lack of "balance":
  - compare depths of left and right subtree

Height Balanced Binary Trees

- A height-balanced binary tree is one for which at every node, the absolute value of the difference in heights of the left and right children is no larger than one.

- Note: Every complete binary tree is height-balanced.
- An AVL Tree is a height-balanced binary search tree.

Balance Factor

- The balance factor of a binary tree is the difference in heights of its two subtrees (hL - hR).
- The balance factor (bf) of a height balanced binary tree may take on one of the values -1, 0, +1.
- An AVL node is
  - "left-heavy" when bf = 1,
  - "equal-height" when bf = 0, and
  - "right-heavy" when bf = +1

AVL Tree

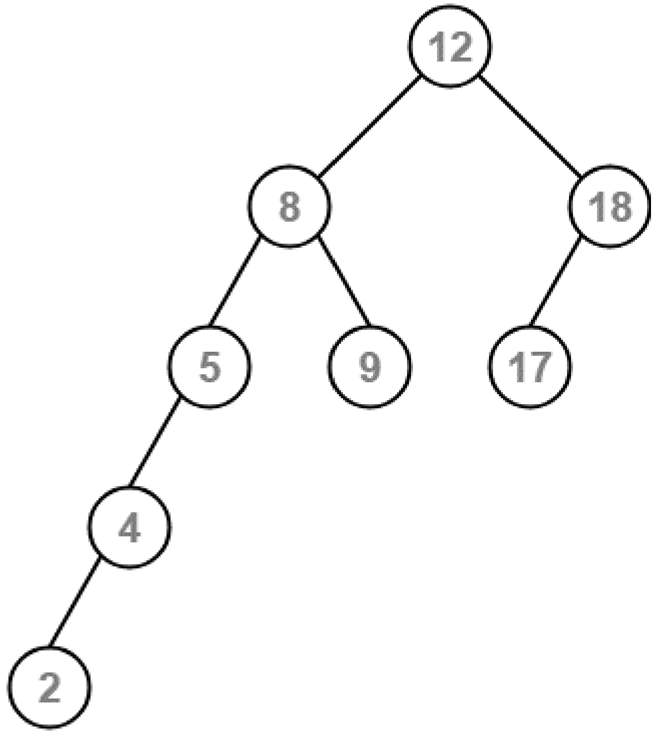
- An AVL tree is a self-balancing binary search tree.
- In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.
- Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.
- An AVL tree has balance factor calculated at every node.
- A binary tree is said to be balanced, if the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1.
- Balanced Factor,
- BF = [Height of the left subtree – Height of right subtree] <= 1

AVL Tree

- Binary search trees (BSTs) are one of the most efficient data structures with their O(n) time complexity while performing operations. But in the realm of computers and processing, constant optimization has brought about the concept of AVL trees.

- The AVL tree, named after its two inventors, Adelson-Velskii and Landis, who published it in their 1962 paper "An Algorithm for the Organization of Information" has anchored its position as a need-to-understand data structure due to its performance increase from a regular BST.

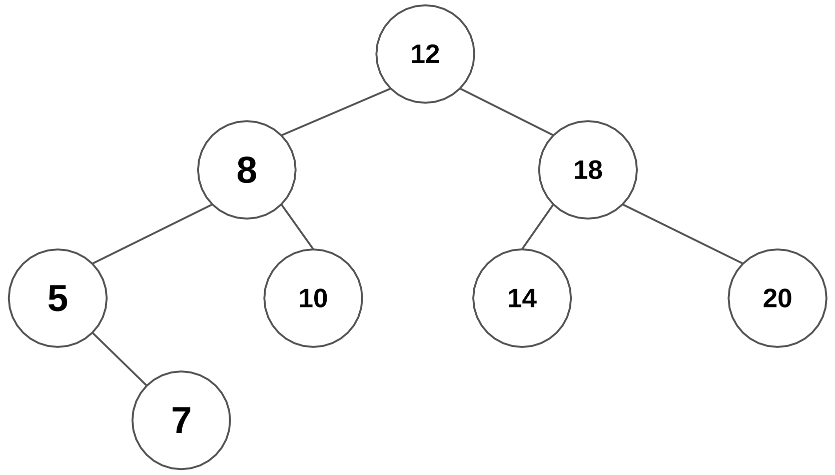
AVL Tree



Is this an AVL tree?

No.
The height of left subtree is 4 and height of right subtree is 2
So bf =(HL - HR) i.e 4 – 2 = 2
The difference should be -1, 0 0r 1. Here it is 2 so it is not an AVL tree

AVL Tree



Is this an AVL tree?

Yes.
The height of left subtree is 3 and height of right subtree is 2
So bf =(HL - HR) i.e 3 – 2 = 1

AVL Tree Insert Operation

- An AVL tree is a self-balancing binary search tree.
- Inserting a new value will be same as binary search tree.
- After inserting a new value, balance factors of every nodes will be re-calculated.
- If resulting AVL tree becomes imbalanced then rebalancing is done by one or more tree rotations.

AVL Tree Insert Operation

- Insert: 50, 35, 70, 20, 66, 40, 80, 37, 68

AVL Tree Insert Operation

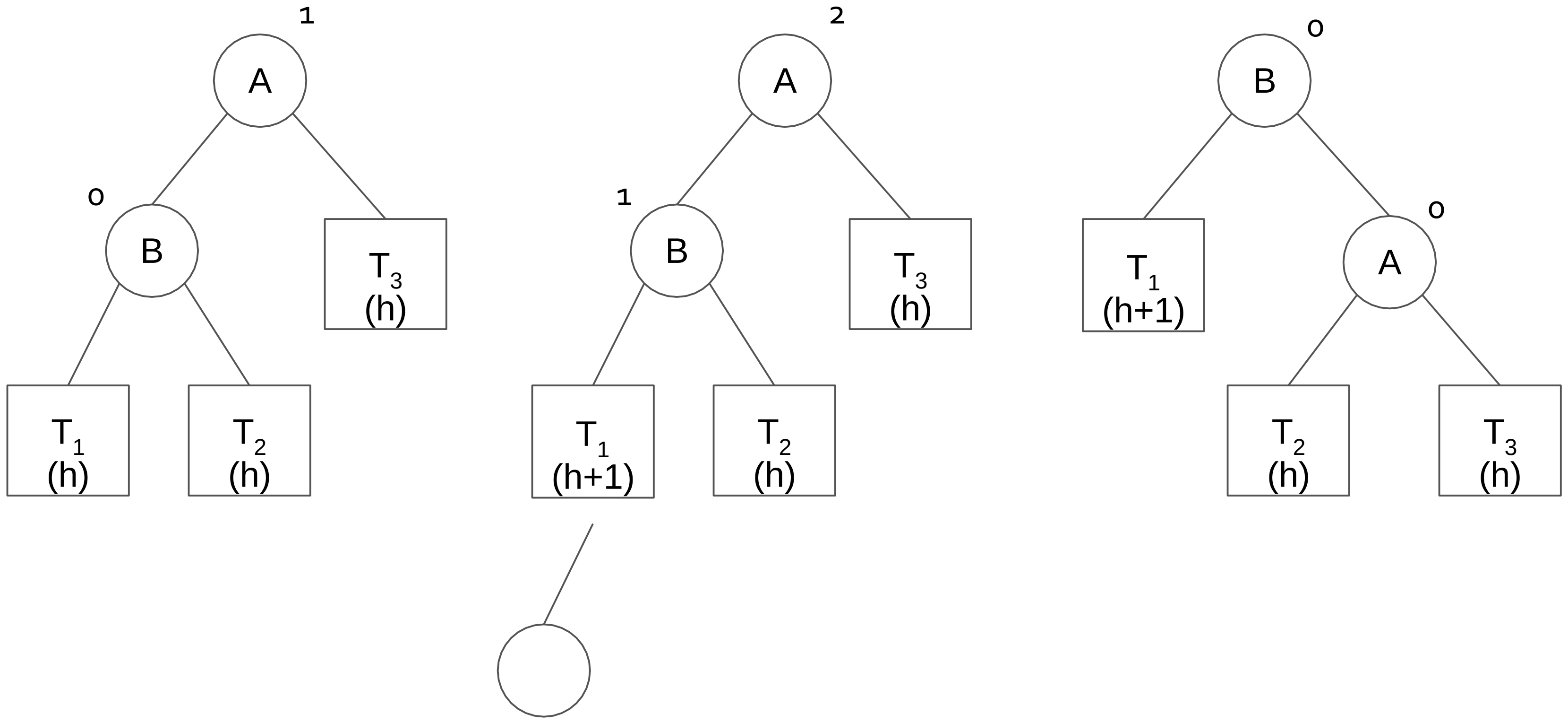
- Insert: 50, 35, 70, 20, 66, 40, 80, 37, 68

Imbalance Types

- After an insertion, when the balance factor of node A is –2 or 2, the node A is one of the following four imbalance types:
  1. LL: new node is in the left subtree of the left subtree of A
  2. RR: new node is in the right subtree of the right subtree of A
  3. LR: new node is in the right subtree of the left subtree of A
  4. RL: new node is in the left subtree of the right subtree of A

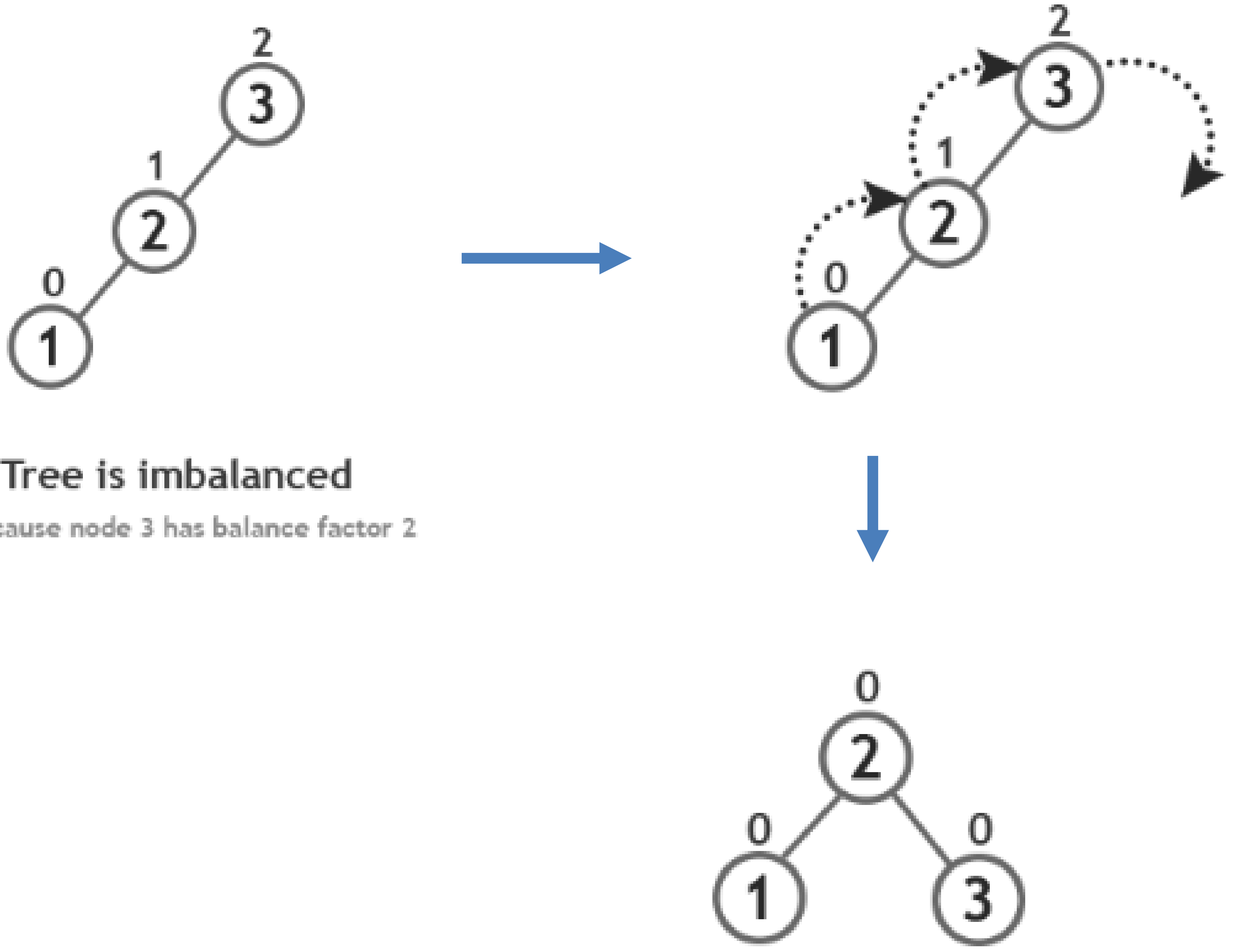
LL Imbalance

- LL: new node is in the left subtree of the left subtree of A



LL Imbalance Example

insert 3, 2 and 1

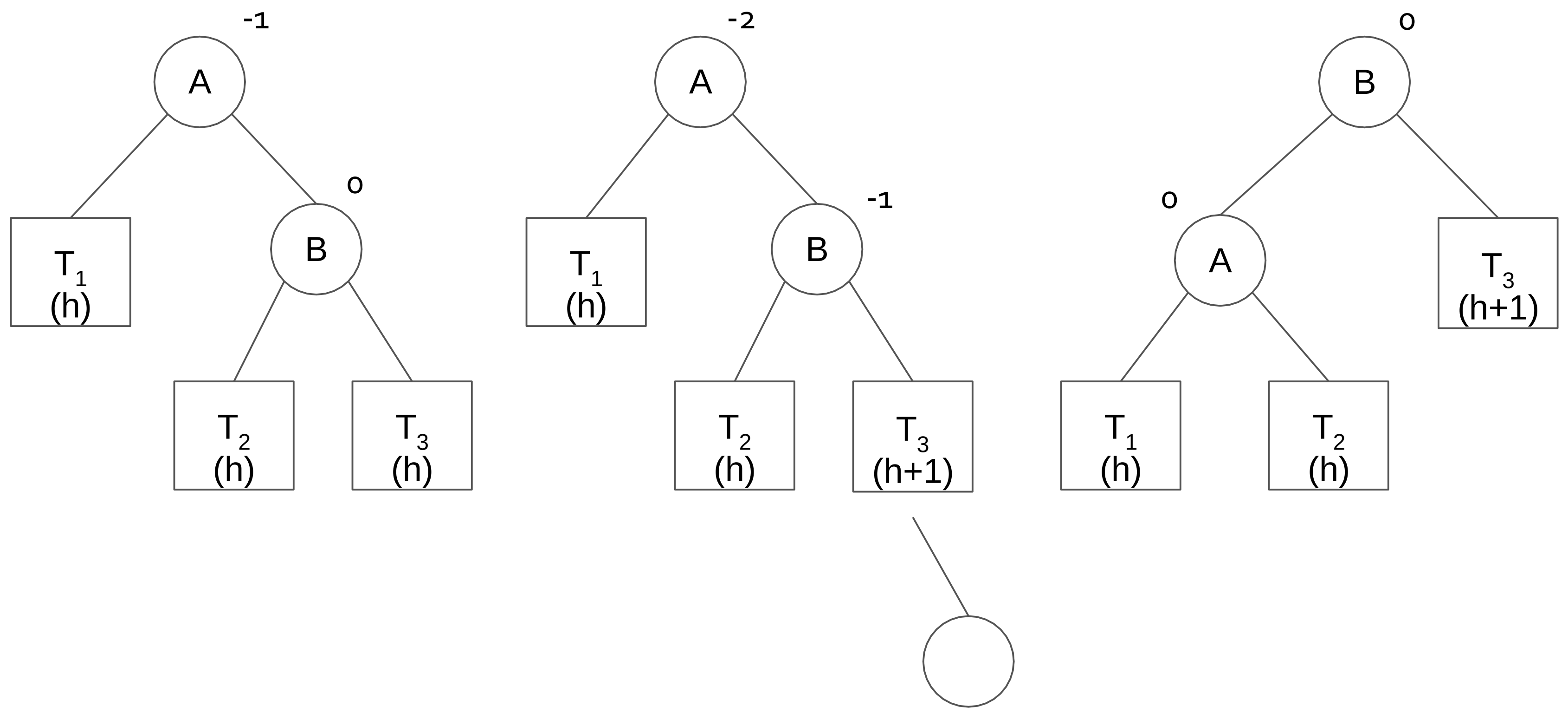


Tree is imbalanced

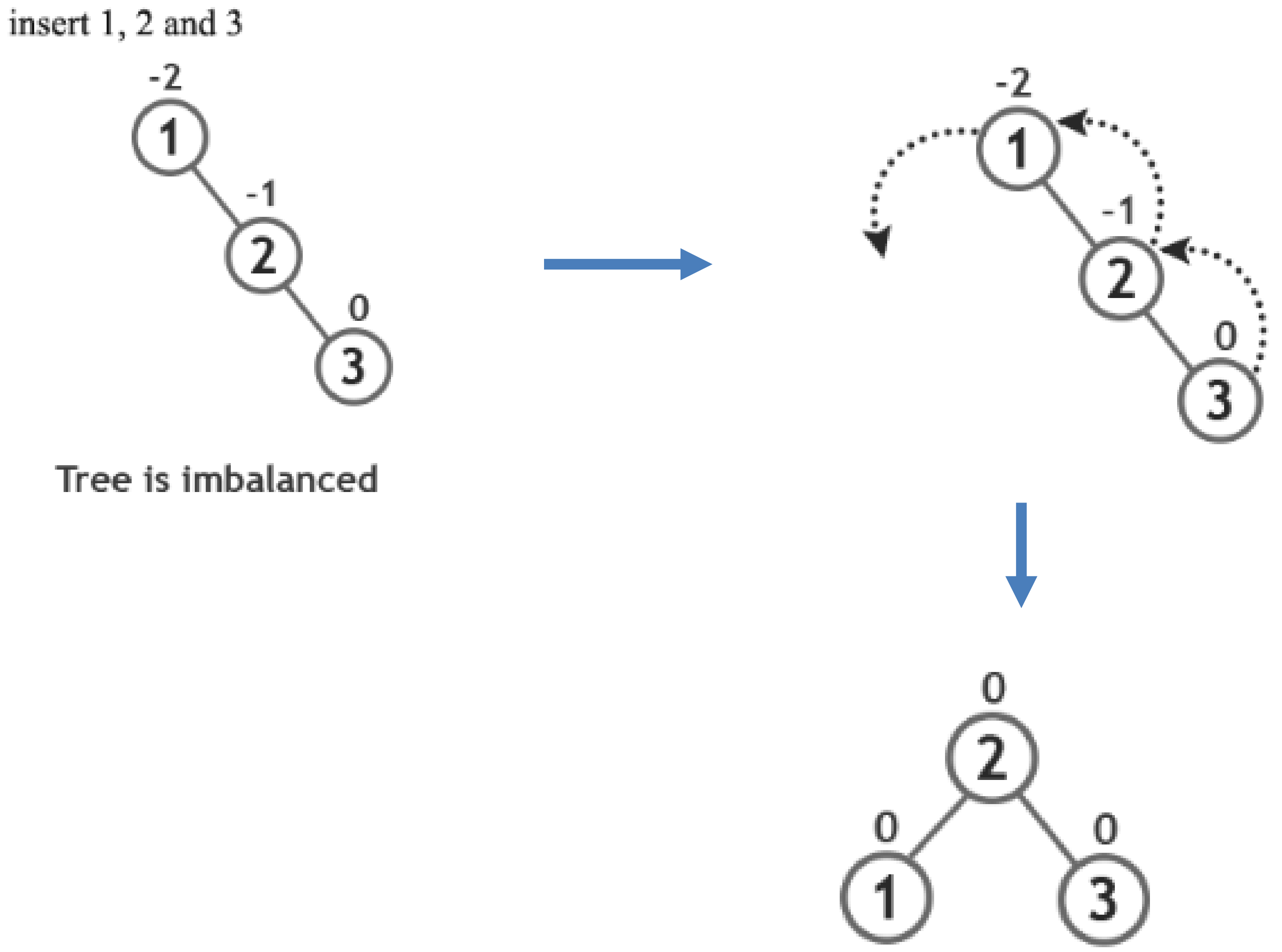
because node 3 has balance factor 2

RR Imbalance

- RR: new node is in the right subtree of the right subtree of A

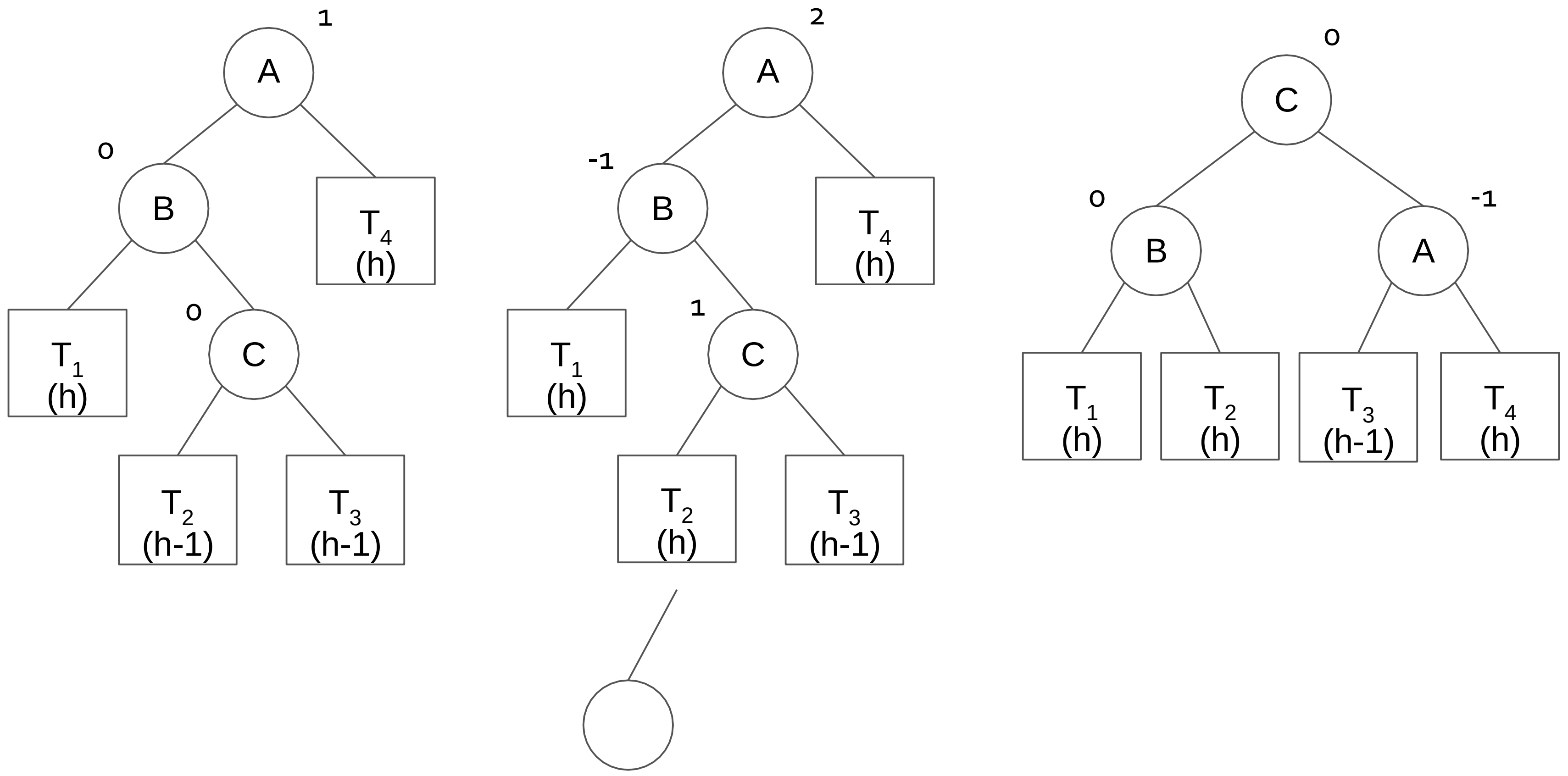


RR Imbalance Example

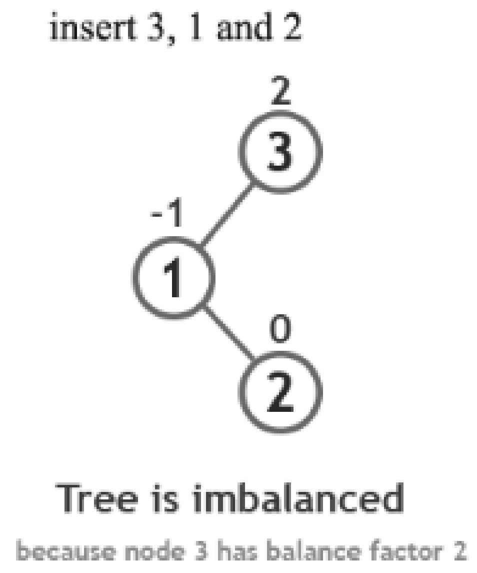


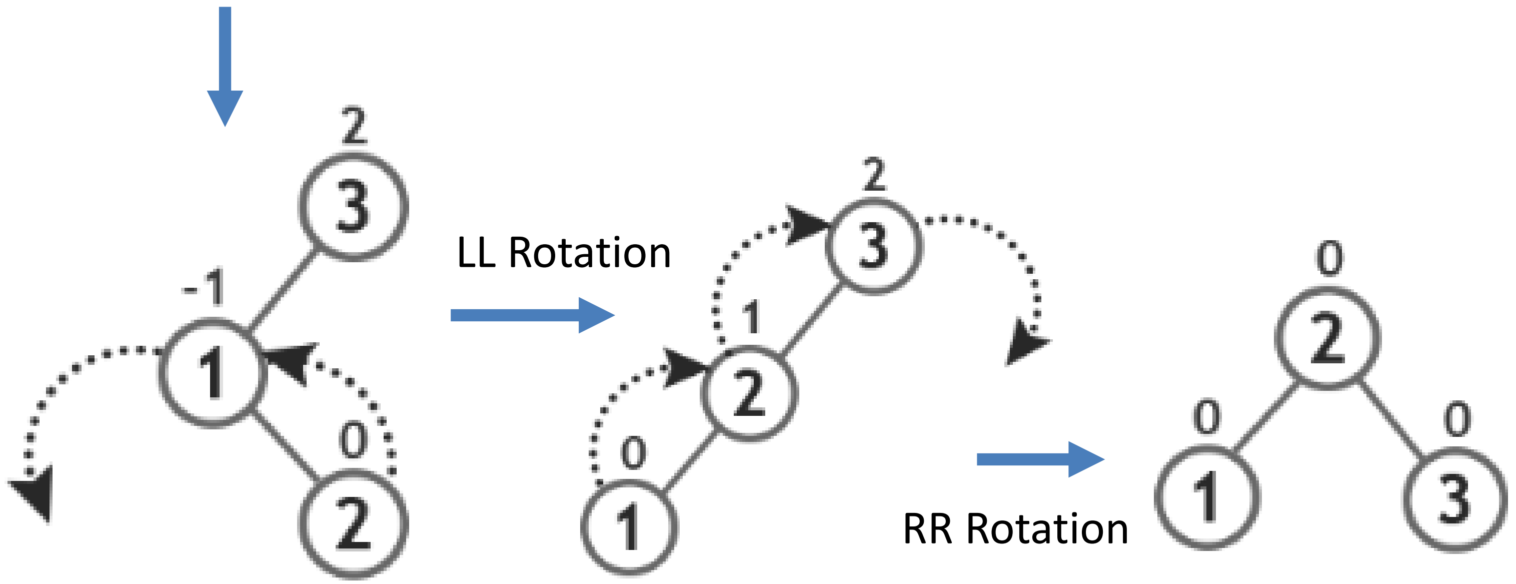
LR Imbalance

- LR: new node is in the right subtree of the left subtree of A



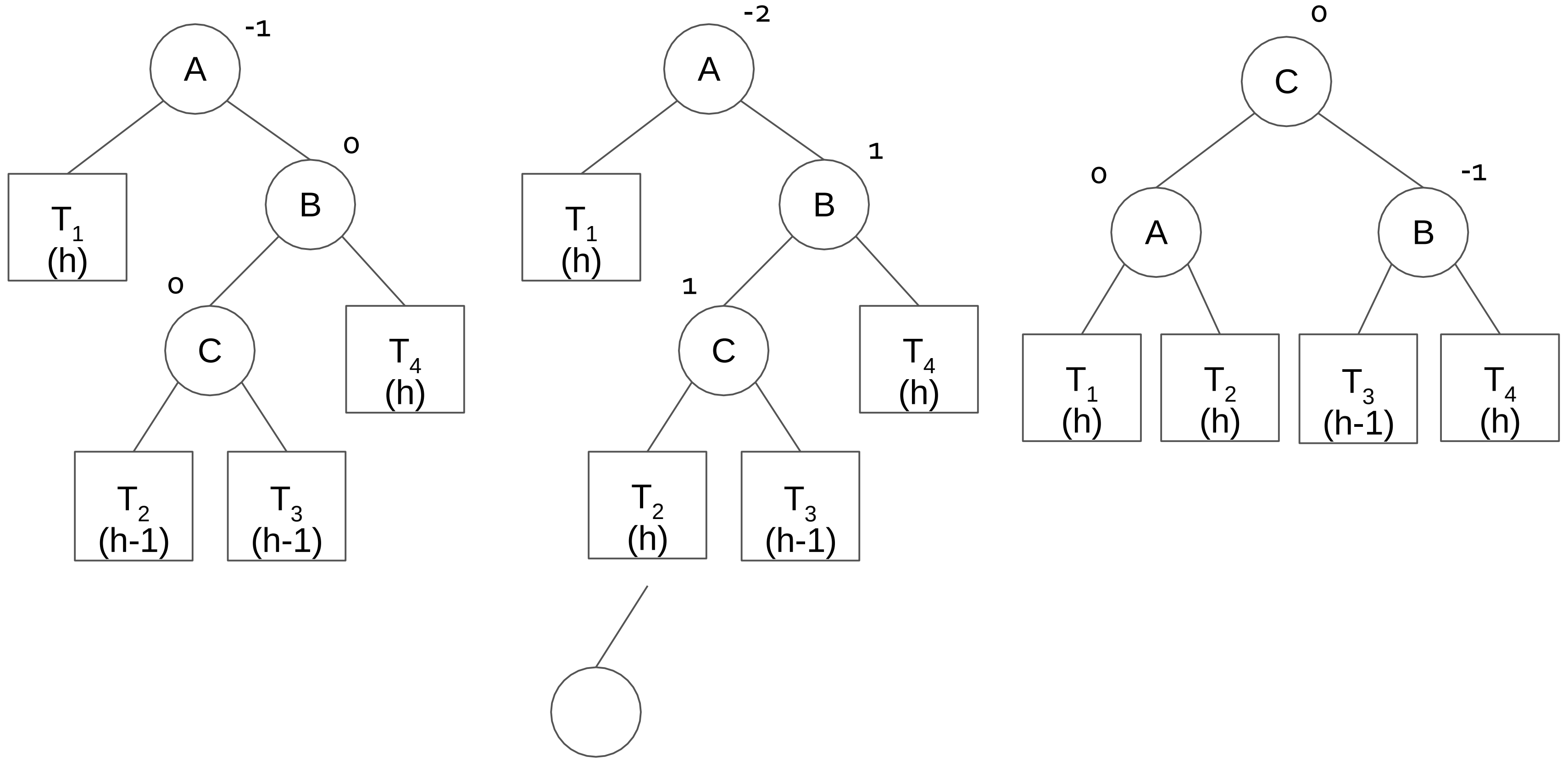
LR Imbalance Example



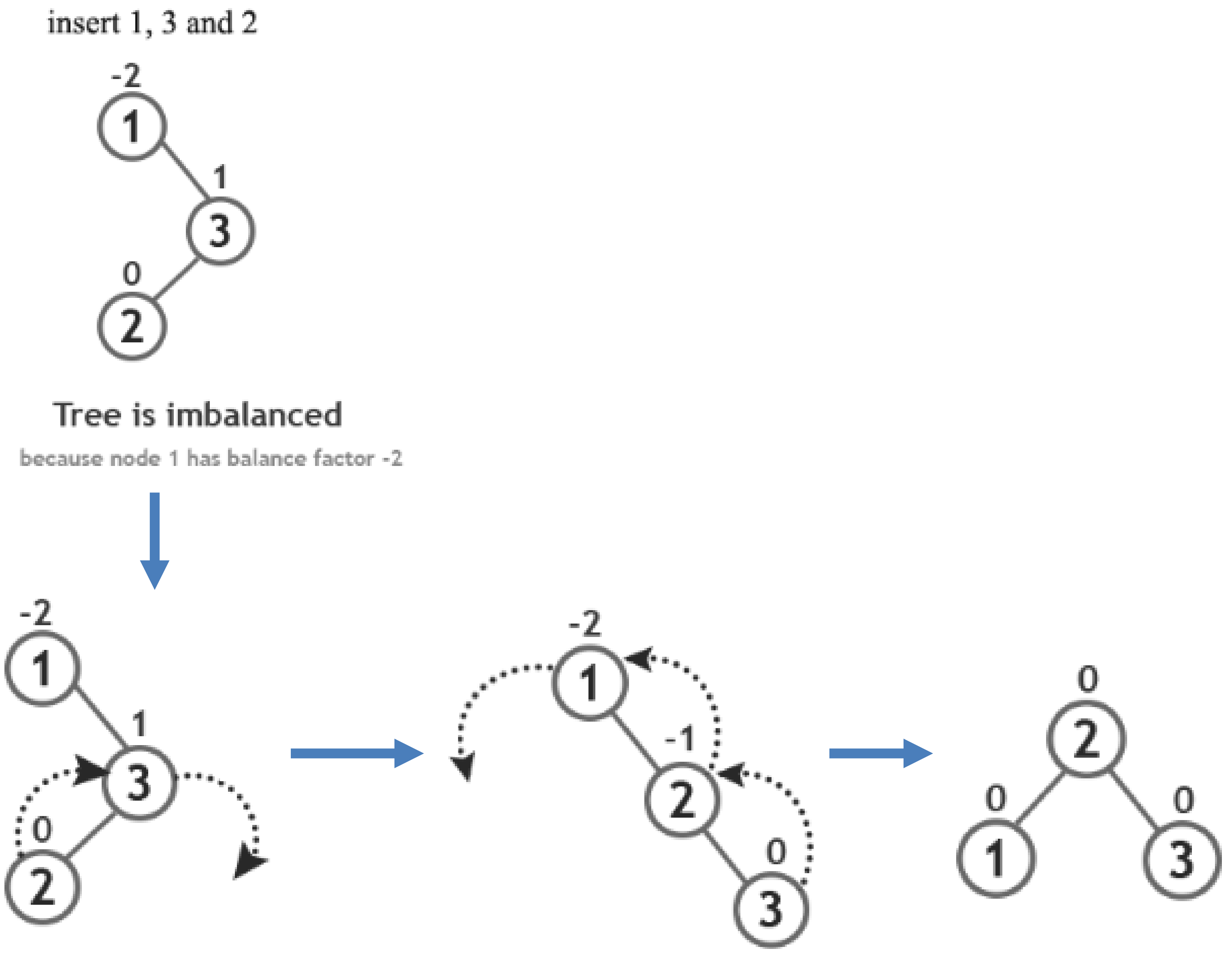


RL Imbalance

- RL: new node is in the left subtree of the right subtree of A



RL Imbalance Example



Construct AVL Tree

- Construct AVL tree with 63, 9, 19, 27, 18, 108, 99, 81

Construct AVL Tree

- Construct AVL tree with 63, 9, 19, 27, 18, 108, 99, 81

AVL Tree Delete Operation

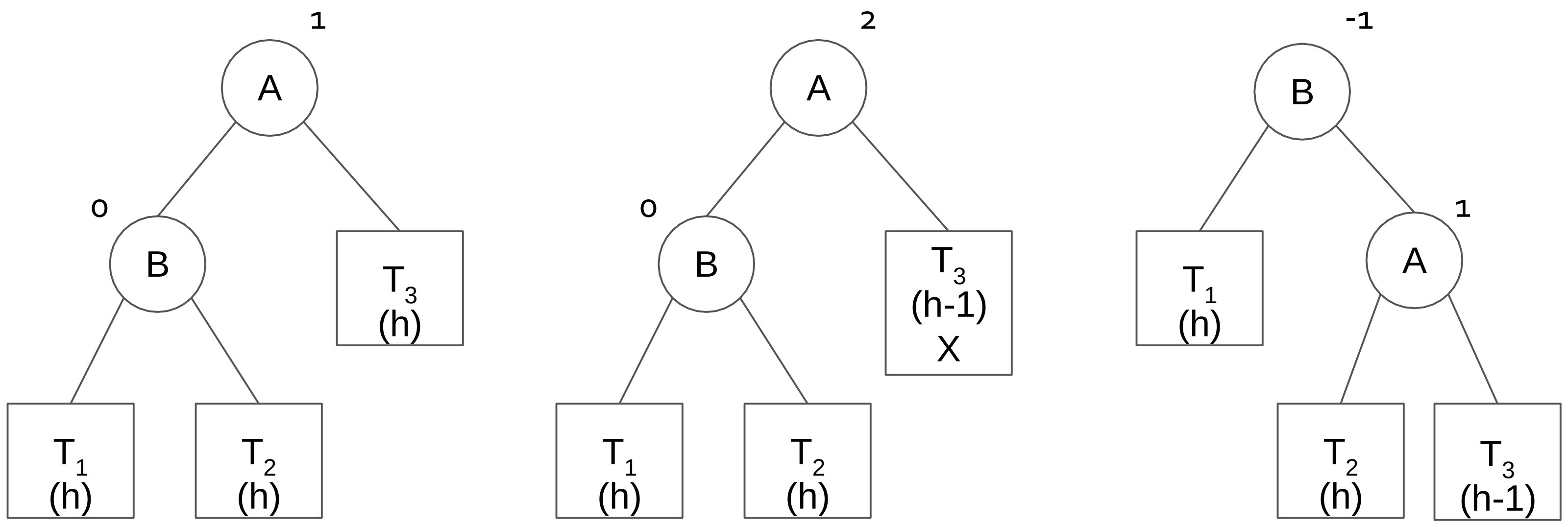
- An AVL tree is a self-balancing binary search tree.
- Deleting a value will be same as binary search tree.
- After deleting a value, balance factors of every nodes will be re-calculated.
- If resulting AVL tree becomes imbalanced then rebalancing is done by one or more tree rotations.

Imbalance Types

- Imbalance incurred by deletion is classified into the types R0, R1, R-1, L0, L1, and L-1
  1. R0: imbalance in right subtree and opposite side balance factor 0.
  2. R1: imbalance in right subtree and opposite side balance factor 1.
  3. R-1: imbalance in right subtree and opposite side balance factor -1.
- L0,L1,L-1 will be same and mirror of R0,R-1,R1

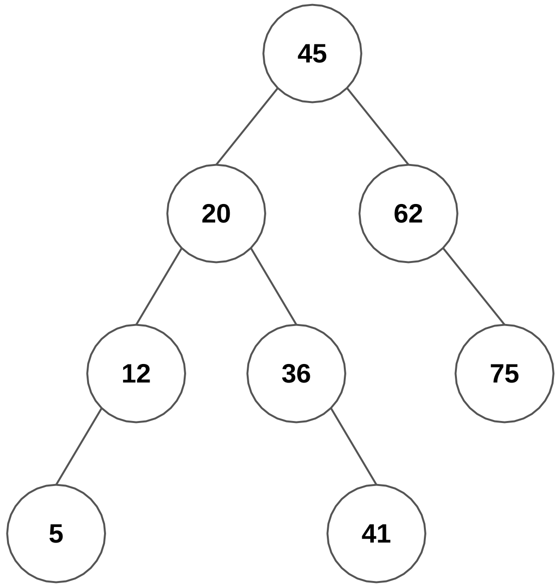
Ro Imbalance

- Ro: imbalance in right subtree and opposite side balance factor 0. Perform right rotation.



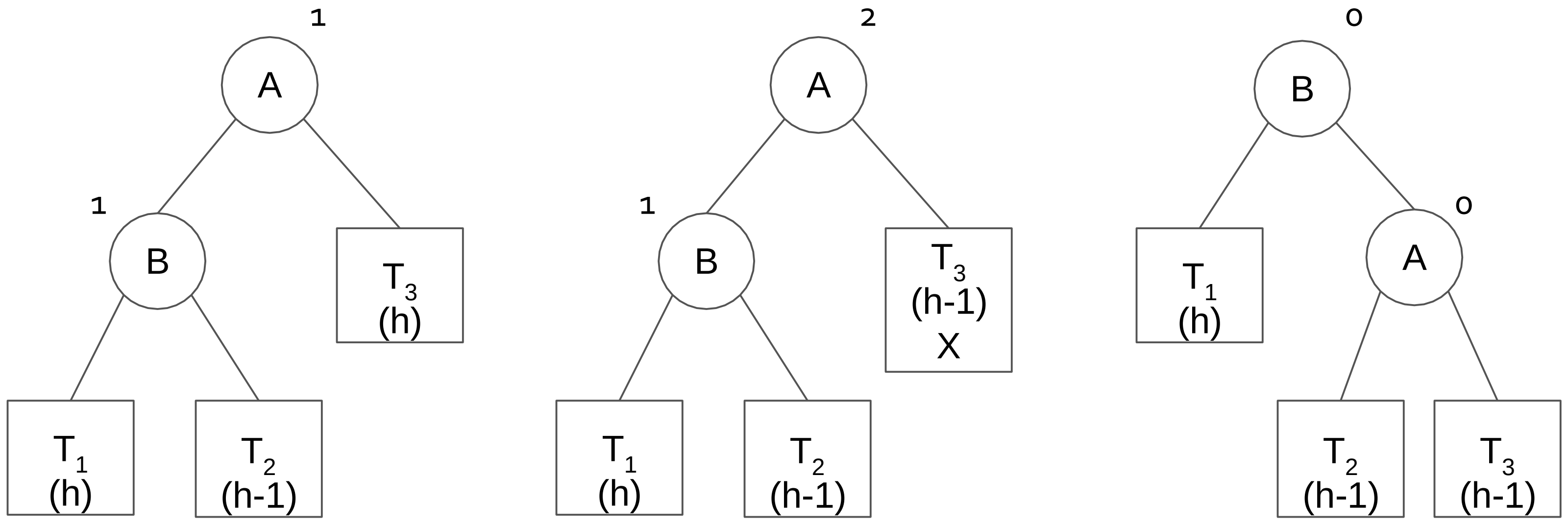
Ro Imbalance Example

- Delete: 75



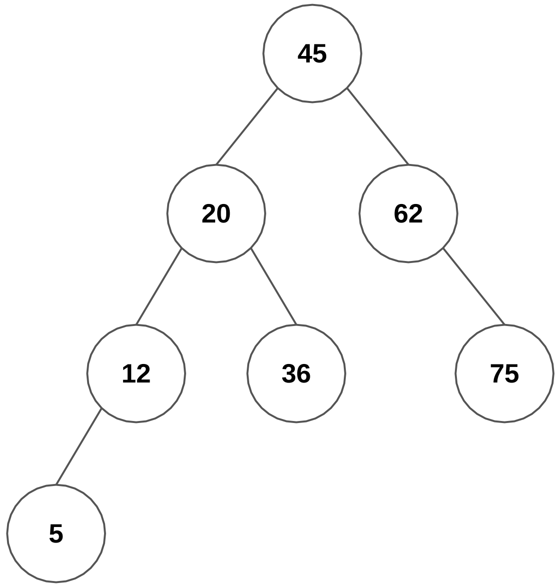
R1 Imbalance

- R1: imbalance in right subtree and opposite side balance factor 1. Perform right rotation.



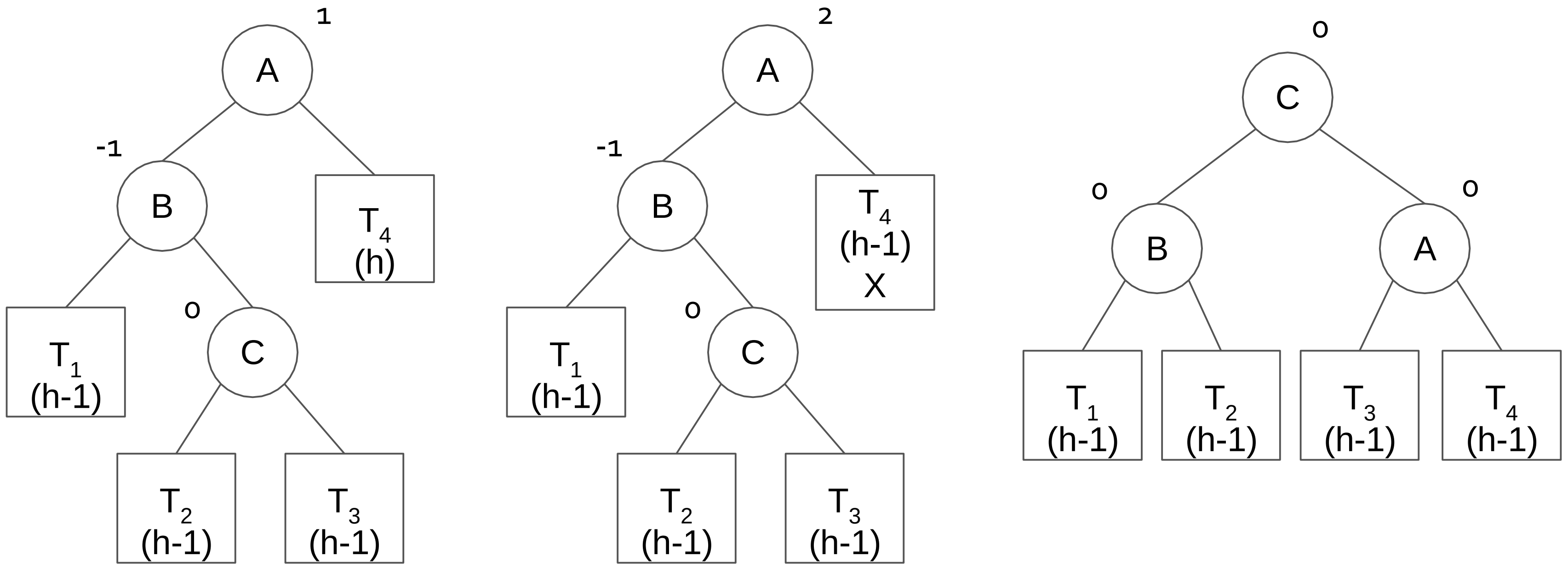
R1 Imbalance Example

- Delete: 75



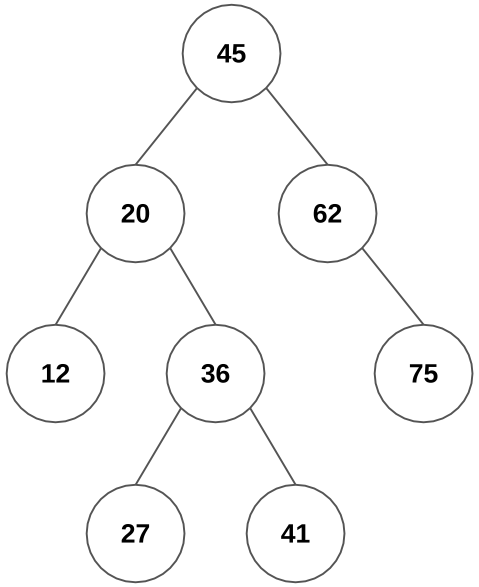
R-1 Imbalance

- R-1: imbalance in right subtree and opposite side balance factor -1. Perform right rotation.



R-1 Imbalance Example

- Delete: 75



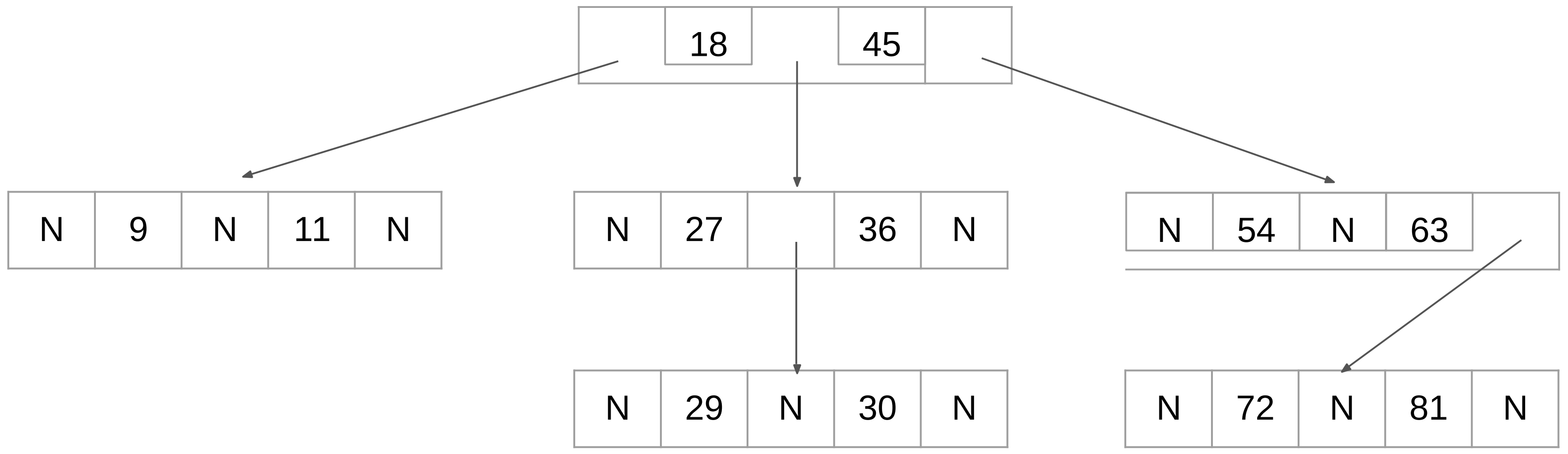
AVL Tree

- AVL Tree Insert and Delete Exercise.

M-way Search Tree

- A binary search tree has one value in each node and two subtrees.
- This notion easily generalizes to an M-way search tree, which has (M-1) values per node and M subtrees.
- M is called the degree of the tree.
- A binary search tree, therefore, has degree 2.

M-way Search Tree



M-way Search Tree

- In fact, it is not necessary for every node to contain exactly (M-1) values and have exactly M subtrees.
- In an M-way subtree a node can have anywhere from 1 to (M-1) values, and the number of (non-empty) subtrees.
- M is thus a fixed upper limit on how much data can be stored in a node.

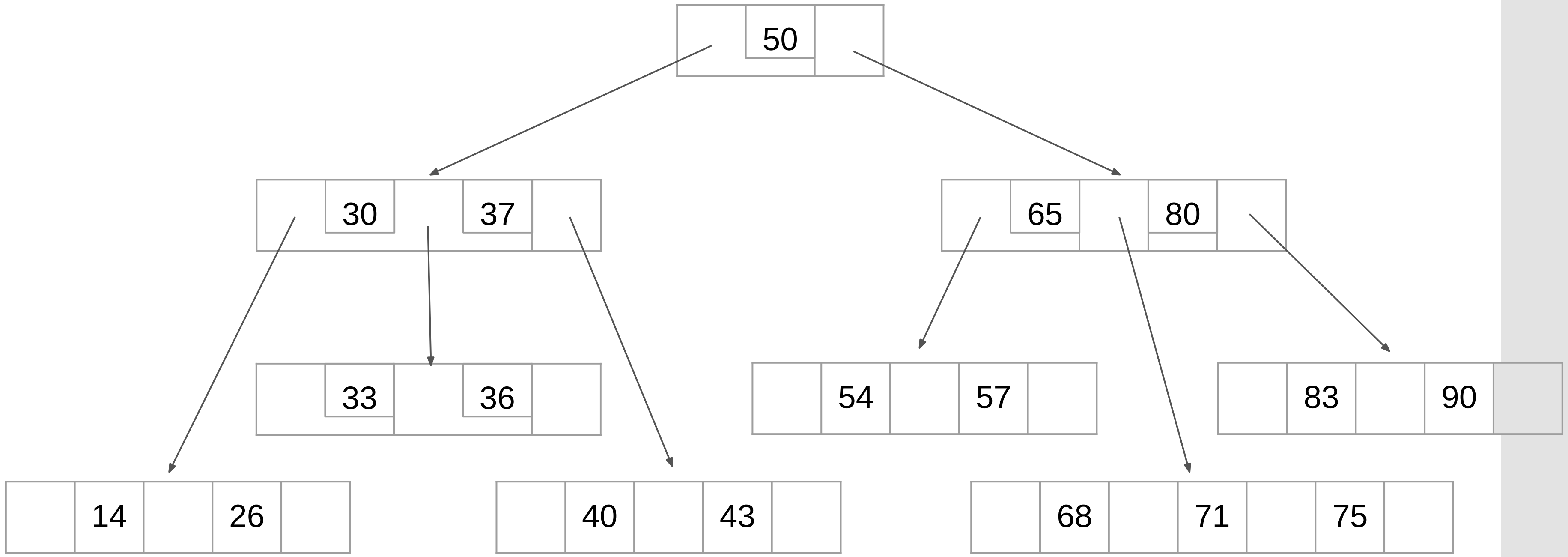
B-trees

- B-tree is a specialized M-way search tree widely used for disk access.
- The B-tree is optimized for systems that read and write large blocks of data.
- It is commonly used in databases and file systems.
- An M-way B-tree will have:
  - Maximum (M-1) values per node and M subtrees.
  - Minimum (M-1)/2 values per node and M/2 subtrees (except root node).

B-trees

- The root can have 1 to M-1 keys
- All the nodes (except root) have between [(M-1)/2] and M-1 keys
- All leaves are at the same depth
- If a node has t no. of children then it must have (t-1) no. of keys
- Keys of a node are stored in ascending order.

B-tree



B-tree: Insert Operation

- All insertions done at a leaf node.
- To insert a new element, search the tree to find the leaf node where the new element should be added.
- Insert the new element into that node with the following steps:
  - If the node contains fewer than the maximum legal number of elements, then there is room for the new element. Insert the new element in the node, keeping the node's elements ordered.
  - Otherwise the node is full, evenly split it into two nodes so:
    - A single median is chosen from among the leaf's elements and the new element.
    - Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value.

B-tree: Insert Operation

- Order: 5, Insert:

B-tree: Insert Operation

- Order: 5, Insert:

B-tree: Delete Operation

- Delete operations are performed same as search trees.
- Choose a node to delete value.
  - Internal node: replace with InOrder Pre/Succ. and delete that InOrder Pre/Succ.
  - Leaf node: contains more than minimum of key values(m/2), then delete it.
  - Leaf node: contains less than minimum of keys, choose from left or right sibling.
    - If left sibling has more than minimum then choose largest key.
    - If right sibling has more than minimum then choose smallest key.
    - Else choose from parent and balance tree.

B-tree: Delete Operation

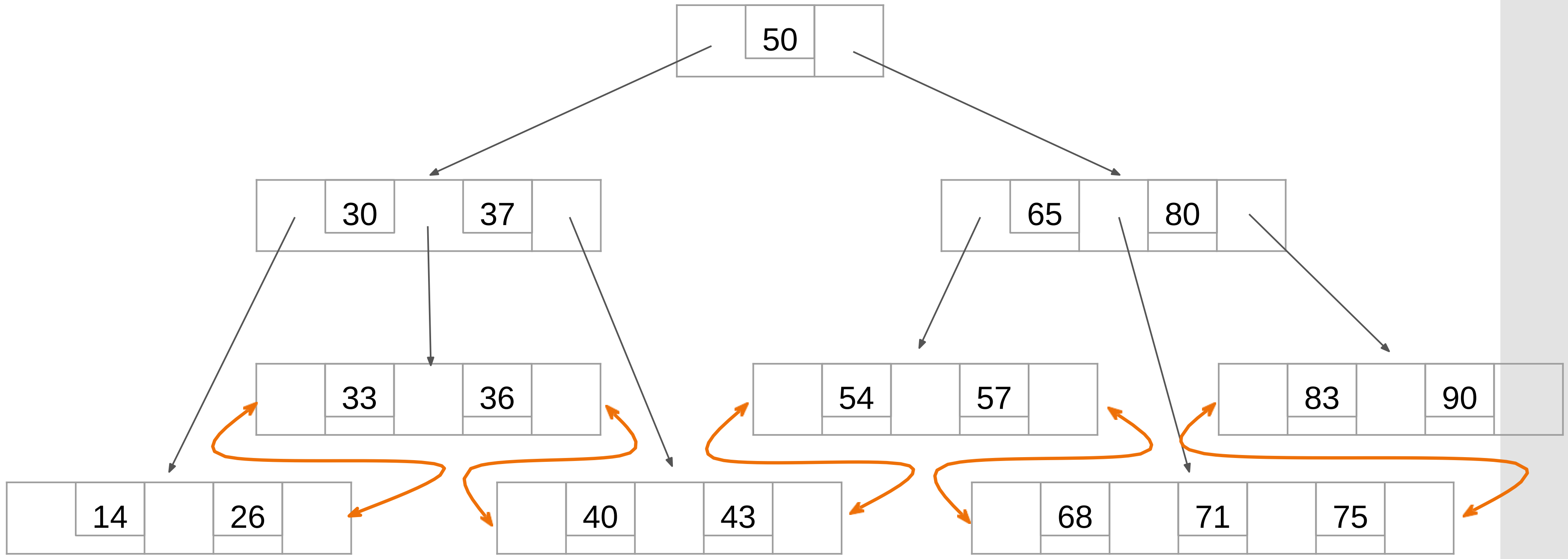
- Order: 5, Delete:

B-tree: Delete Operation

- Order: 5, Delete:

B+ Tree

- Similar to B trees, with a few differences.
- Leaf nodes are linked to each other.



B+ Tree Insertion

- Insert at bottom level.
- If **leaf page overflows**, split page and **copy** middle element to next index page.
- If **index page overflows**, split page and **move** middle element to next index page.

B+ Tree Insertion

- Order: 4, Insert:

B+ Tree Insertion

- Order: 4, Insert:

B+ Tree Deletion

- Delete from bottom level.
- If leaf page underflows, bring data from its sibling and update the index page value.
- If none of the siblings have enough data to provide then merge the page with any of the siblings and remove middle index value from index page.

B+ Tree Deletion

- Order: 4, Delete:

B+ Tree Deletion

- Order: 4, Delete:

2-3 Tree

- A 2–3 tree is a B-tree of order 3
- Each interior node has either two or three children.
- Nodes with 2 children are called 2 nodes.
- This will have 1 data value and 2 children.

- Nodes with 3 children are called 3 nodes.
- This will have 2 data value and 3 children.

2-3 Tree: Properties

- Every non-leaf is a 2-node or a 3-node. A 2-node contains one data item and has two children. A 3-node contains two data items and has 3 children.
- All leaves are at the same level (the bottom level)
- All data are kept in sorted order
- Every leaf node will contain 1 or 2 fields.

2-3 Tree: Insert and Delete Example

Weight - Balanced Tree

- A Weight-Balanced Tree is a binary tree which is balanced based on the knowledge of the probabilities of searching for each individual node.
- Within each sub-tree, the node with highest weight appears at the root thereby resulting in more efficient searching performance.
- The nodes which are most likely to be searched/accessed have the lowest search time.
- Ex. : Huffman Tree.

Huffman code

- Huffman coding is a form of statistical coding.
- It is a lossless data compression algorithm.
- Not all characters occur with the same frequency! Yet all characters are allocated the same amount of space.
  - 1 char = 1 byte, be it a or x
- Any savings in tailoring codes to frequency of character?
  - Code word lengths are no longer fixed like ASCII.
  - Code word lengths vary and will be shorter for the more frequently used characters.

Huffman code

- The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.
- The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.
- Let a,b,c and d are four characters and their corresponding variable length codes be 00, 01, 0 and 1.
- Ambiguity? because code assigned to c is the prefix of codes assigned to a and b

Huffman code

- The Basic Algorithm

1. Scan text to be compressed and tally occurrence of all characters.
2. Sort or prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.

Huffman code Example 1

- The cost of message is measured in bits.
- Example
- Message = BCCABBDDAECCBBAEDDCC
- How to send?
- ASCII Code
- ASCII code takes how many bits?
- 8 bit
- Length of the message is 20 bits

Char	ASCII	Bit
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101

8 bit * 20 char = 160 bits required

Huffman code

- Can we use fixed size code?
- 8 bit is actually required for 128 characters but in the example we use only 5 characters.
- Observe the following,
- For 1 bit 0 or 1
- For 2 bit 0 0
- 0 1
- 1 0
- 1 1
- For 3 bit 0 0 0 required
- So the table will be

Huffman code Example 1

- Fixed size code

BCCABBDDAECCBBAEDDCC

Char	Frequency /Count	Code
A	3	000
B	5	001
C	6	010
D	4	011
E	2	100

3 bit * 20 char = 60 bits required

To decode the table is needed.

Characters, 5*8 bit =
Code, 3* 5 char =

Message =
Table =
Total =

30 to 40% reduced. Sounds good?

Huffman code Example 1

- Variable size code
- Message = BCCABBDDAECCBBAEDDCC
- Optimal merge pattern i.e get own variable size pattern
- Arrange alphabets in increasing order

Char	Frequency /Count	Code	
A	3		3*3=9
B	5		5*2=10
C	6		6*2=12
D	4		4*2=8
E	2		2*3=6
	20 Char		45 bits

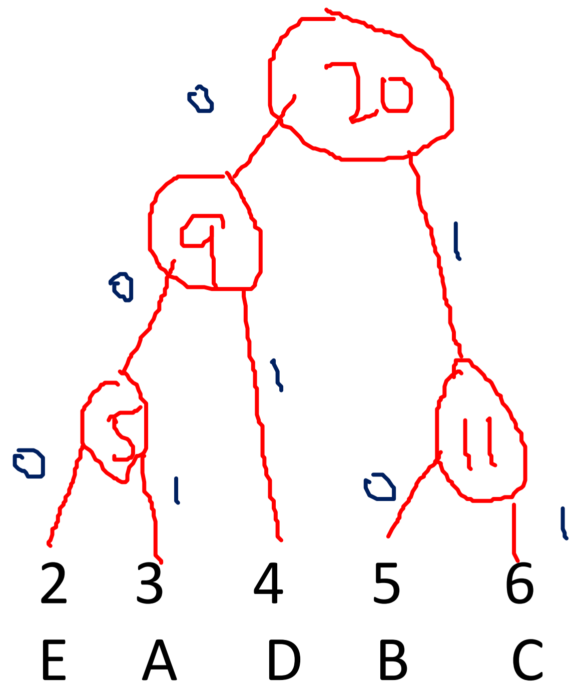
2	3	4	5	6
E	A	D	B	C

Now merge two smallest and make one.
After merge, mark left edges as 0 and right as 1

Msg = 45 bits
Table = 5 char * 8bit=40
45+40+12bit code = 97 bits

Huffman code

- To calculate bits from tree

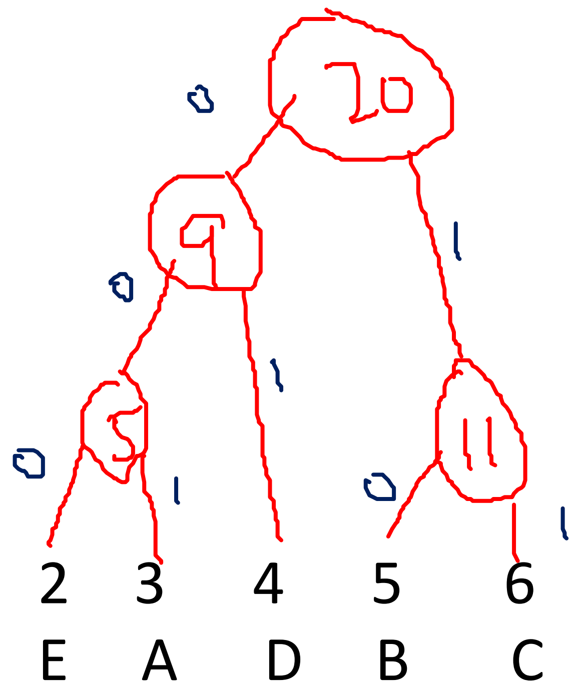


di * fi

Huffman code

- Decoding

- Message = BCCABBDDAECCBBAEDDCC



Huffman code

- Example 2
- Consider the following short text:

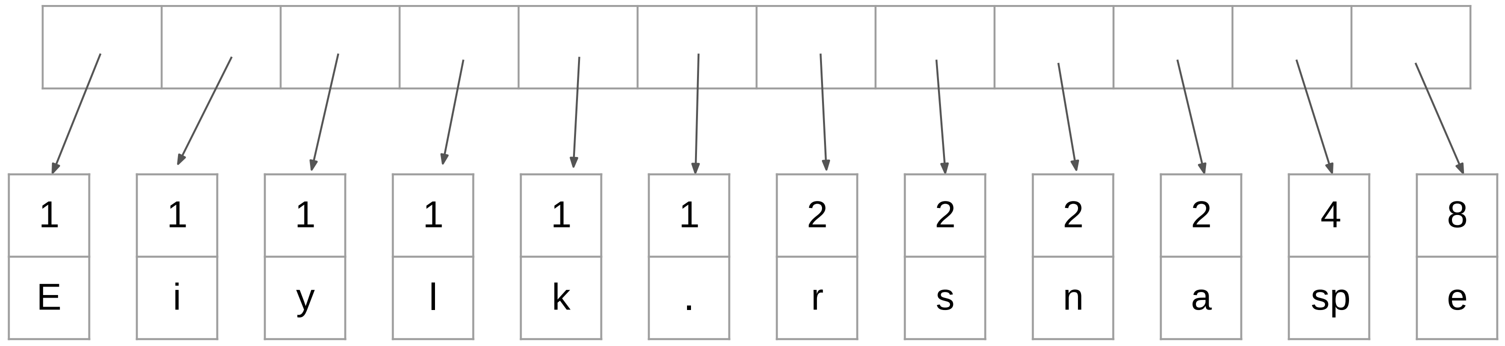
  Eerie eyes seen near lake.
- What characters are present?
- What is the frequency of each character in the text?

E	e	r	i	sp	y	s	n	a	l	k	.

Huffman Tree Example 2

E	e	r	i	sp	y	s	n	a	l	k	.
1	8	2	1	4	1	2	2	2	1	1	1

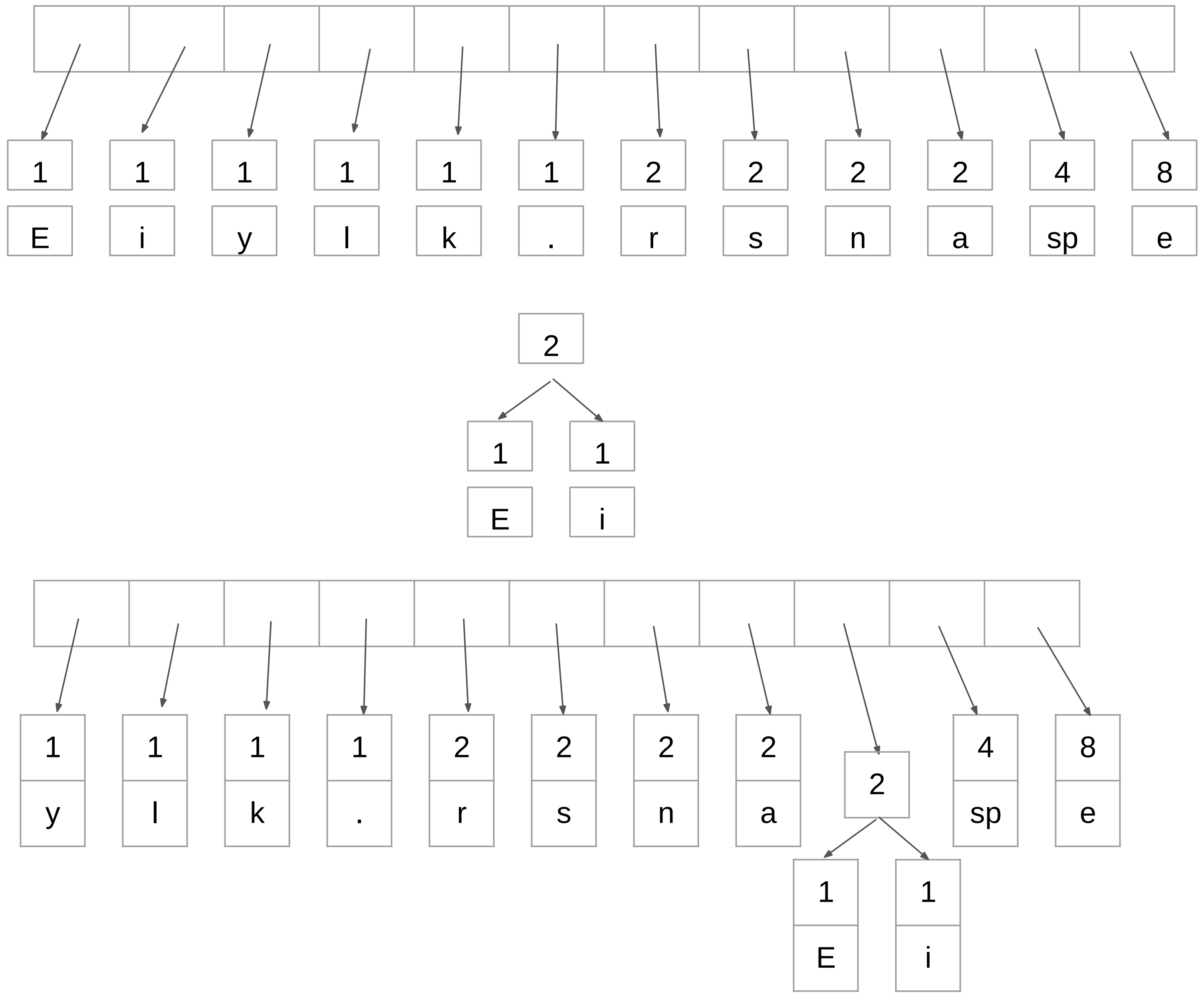
- Create binary tree nodes with character and frequency of each character
- Place nodes in a priority queue
- The lower the occurrence, the higher the priority in the queue



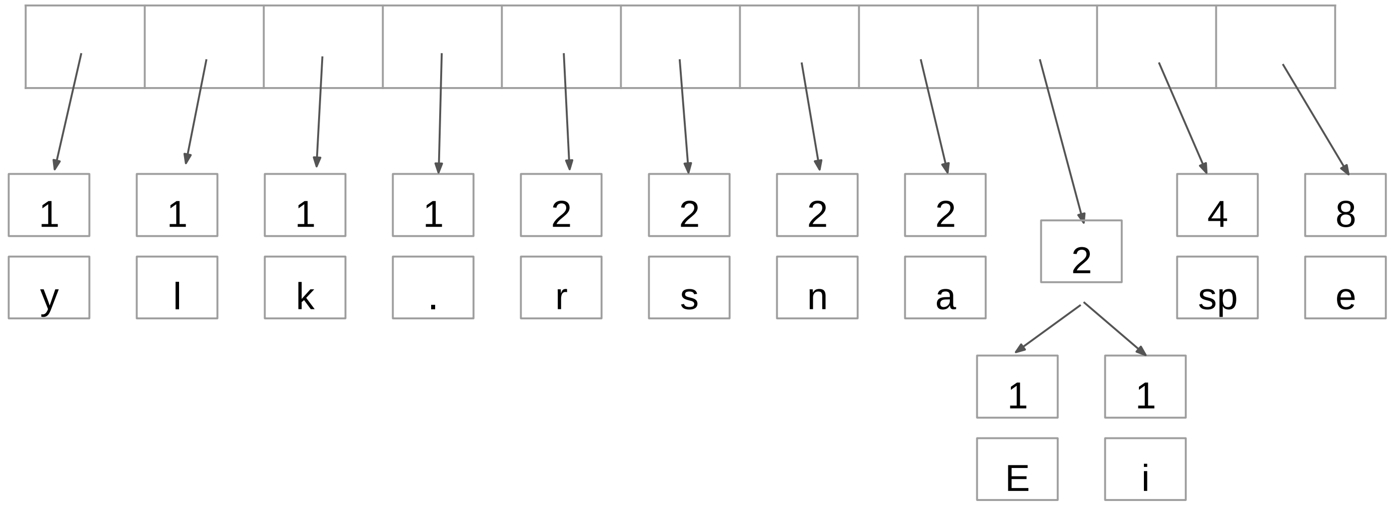
Huffman Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue

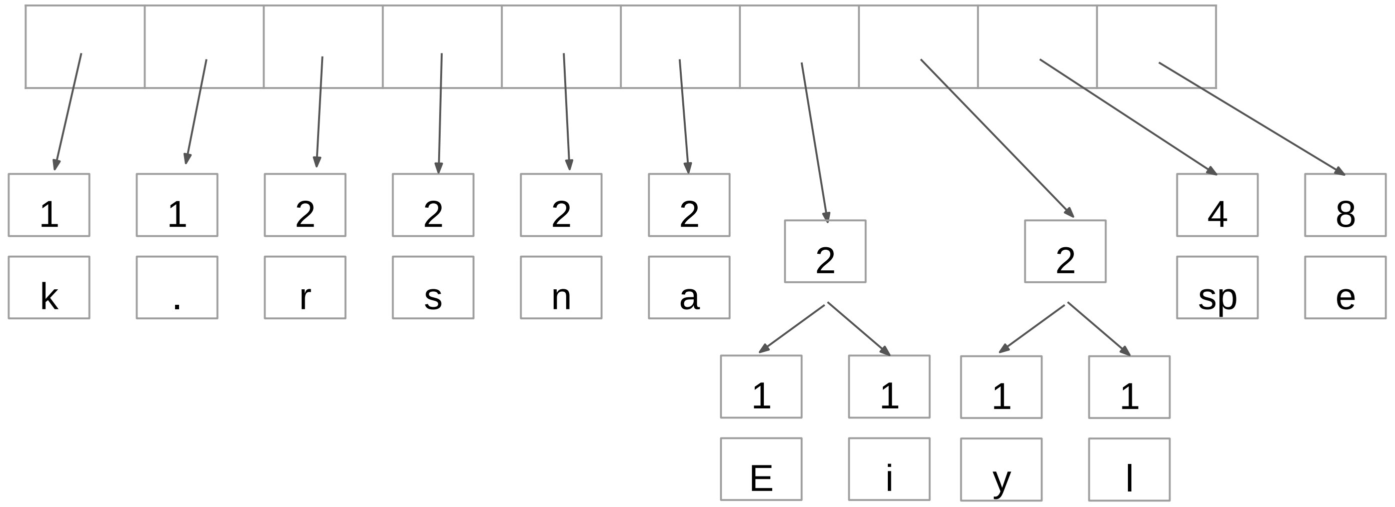
Huffman Tree



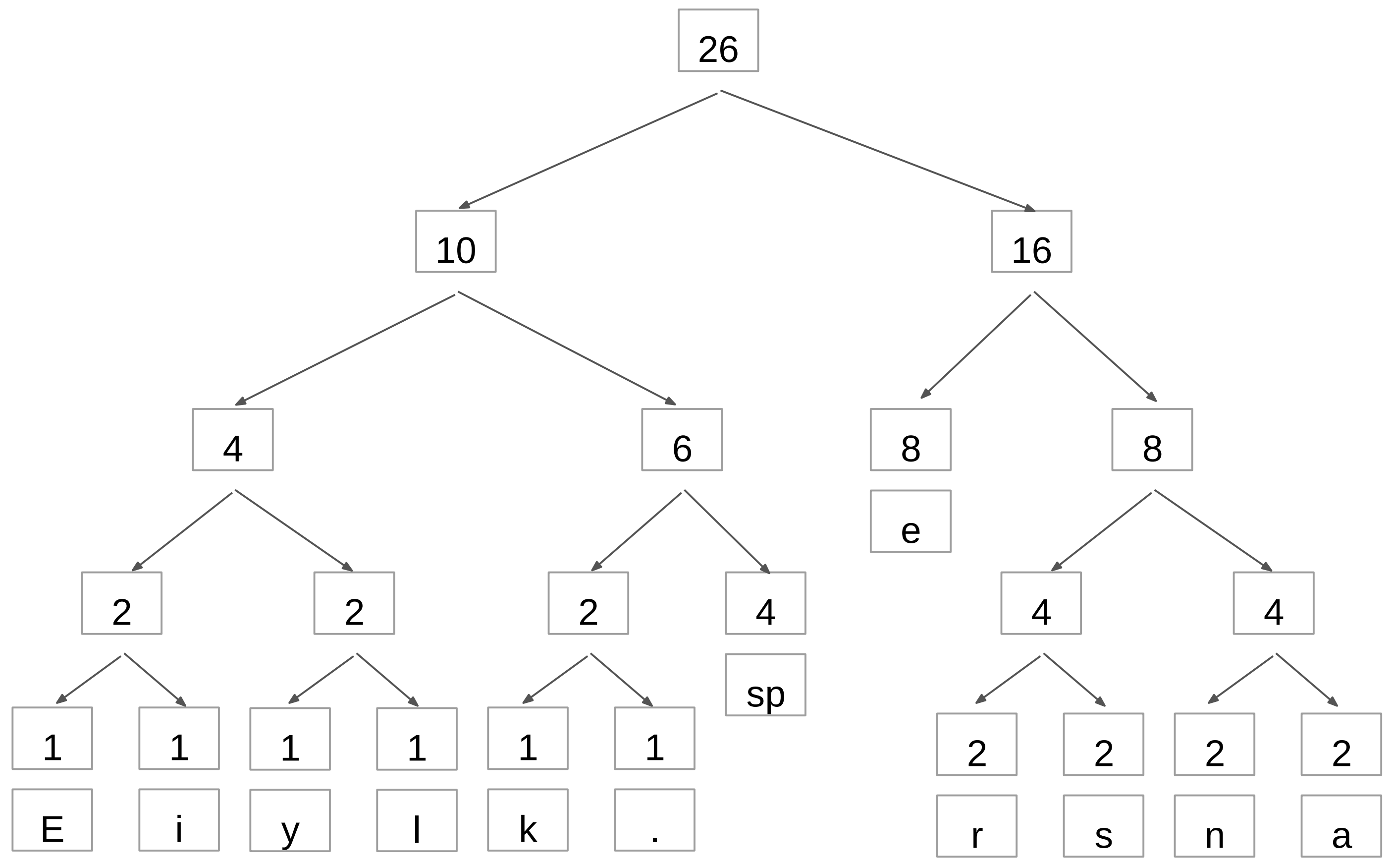
Huffman Tree



Huffman Tree



Complete Huffman Tree

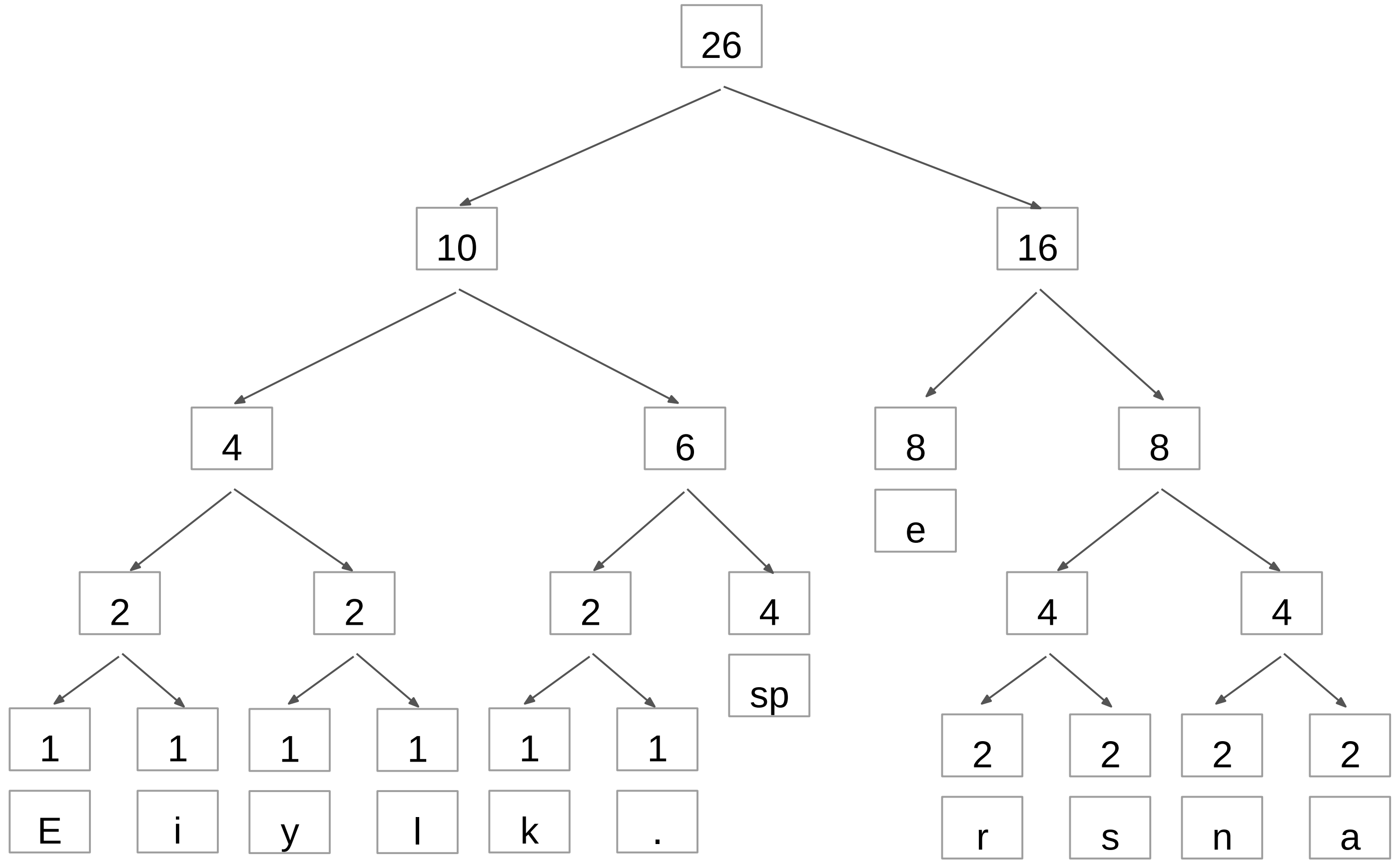


Huffman code

- This tree contains the new code words for each character.
- Frequency of root node should equal to the number of characters in text.
- *Eerie eyes seen near lake. ≈ 26 characters*

- Perform a traversal of the tree to obtain new code words
- Going left is a 0
- Going right is a 1
- Code word is only completed when a leaf node is reached.

Huffman code



Char	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
sp	011
e	10
r	1100
s	1101
n	1110
a	1111

Huffman code

- Encode:

  Eerie eyes seen near lake.

- Rescan text and encode file using new code words

0000101100000110011100010101101101001111101011111100011001111110100100101

Char	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
sp	011
e	10
r	1100
s	1101
n	1110
a	1111

Huffman code

- Encoded Text:

0000101100000110011100010101101101001111101011
1111000110011111110100100101

- Have we made things any better?
- 73 bits in encoded text.
- ASCII would take 8 * 26 = 208 bits.

Huffman code

- Huffman coding is a technique used to compress files for transmission.
- Uses statistical coding.
  - more frequently used symbols have shorter code words.
- Works well for text and fax transmissions.
- An application that uses several data structures.

Till Now

- **Trees**
- Graphs

- Tree definitions and their concepts
- Representation of binary tree
- Binary tree traversal
- Binary search trees
- General trees vs binary trees
- Threaded binary tree
- Applications of Trees
- Balanced tree and its mechanism
- Height and Weight Balanced Trees

Up Next

- Trees
- **Graphs**

- Graphs and their understanding
- Matrix representations of a given graph
- Depth First Search (DFS)
- Breadth First Search (BFS)
- Minimum Spanning Trees Algorithms (Prims, Kruskal, Dijkstra)
- Path Matrix
- Warshall's Algorithm

Thank You.