Trees - II

Overview

- Balanced Tree
  - AVL Trees
  - Rotations
- Array representation of BST
- Almost Complete Binary Trees
- Heaps and its applications

Balanced Trees

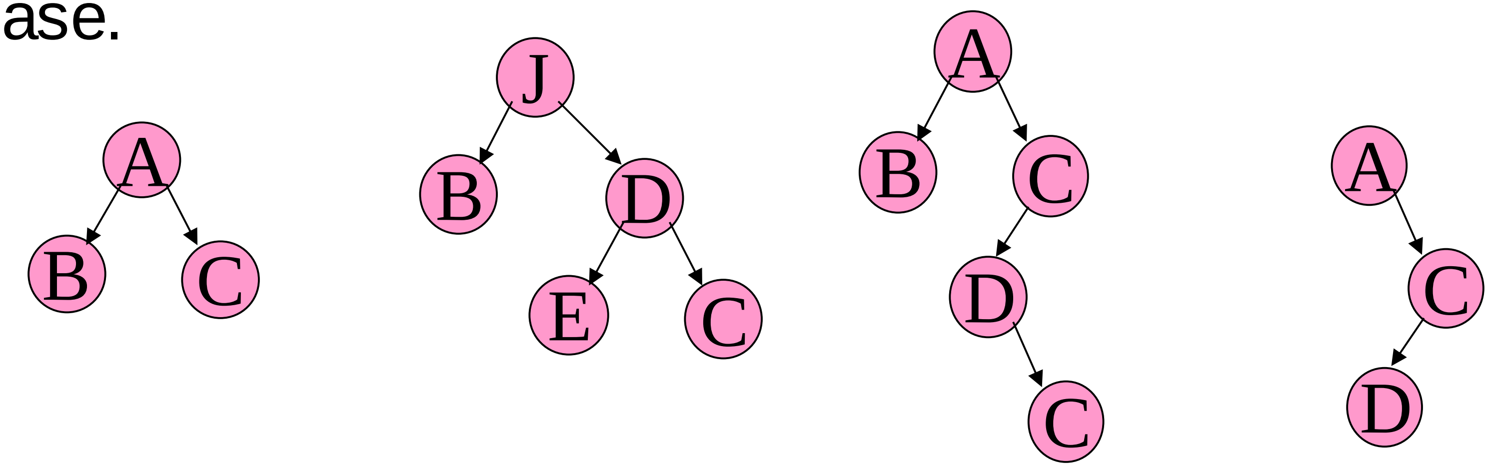
- Depth of average BST is 1.4 times that of completely balanced tree.
- Hence, no major concern about degenerate cases.
- However, over a number of insertions/deletions, tree tends to degenerate.

Balanced Trees

- Also, we want to avoid worst cases - too expensive.
- Hence the interest in keeping the tree reasonably balanced.
- Height Balanced Trees - e.g. AVL trees
- Perfectly balanced trees
  – A height balanced tree where all leaves are at level h(height of the BST) or h –1

AVL Trees

- Height of RST and LST, differ by atmost 1, at all nodes.
- Balance factor = Ht. LST – Ht. RST
- Ensures worst case height of 1.44 log N.
- Thus, about 40% overhead, even in the worst case.

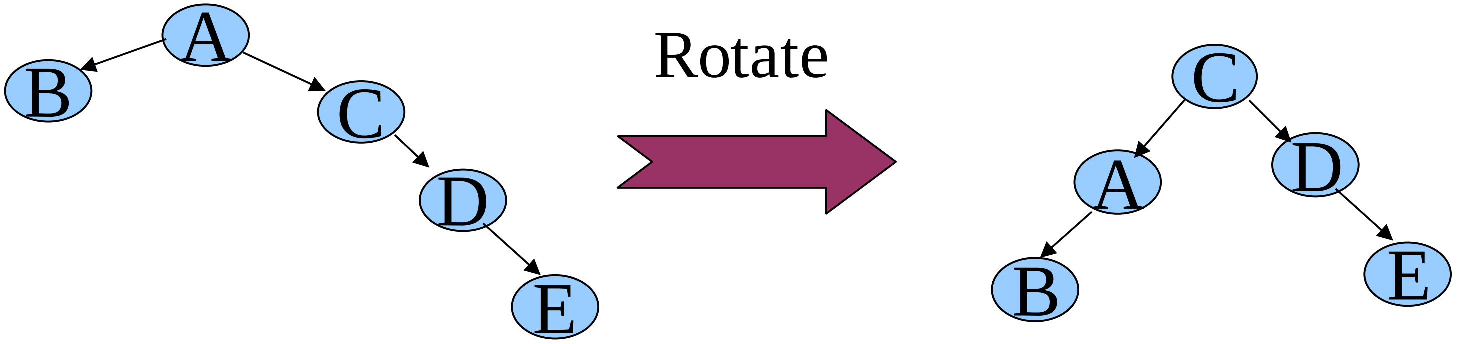


AVL Trees

- Construction done as per BST. When the balance gets violated, some corrective steps are performed.
- Normally, an additional field keeping the balance information is maintained in the nodes.

Rotations

- If balance at a node becomes +2 or -2, AVL criteria at the node is violated.
- Shift the root of the subtree one unit to the heavier side and re-arrange the nodes.
- Will make the new balance zero, and total height remains same as the height before the arrival of new node

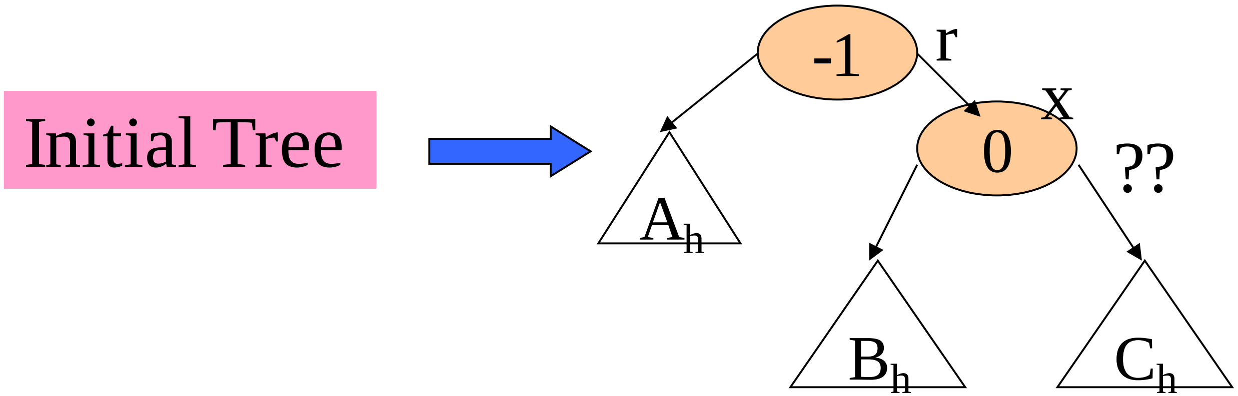


Rotations

- Thus, the parent does not see a height change and hence no balance change
  - Adjustments are confined to a small part of the tree
- Rearranging should also preserve the search tree property.

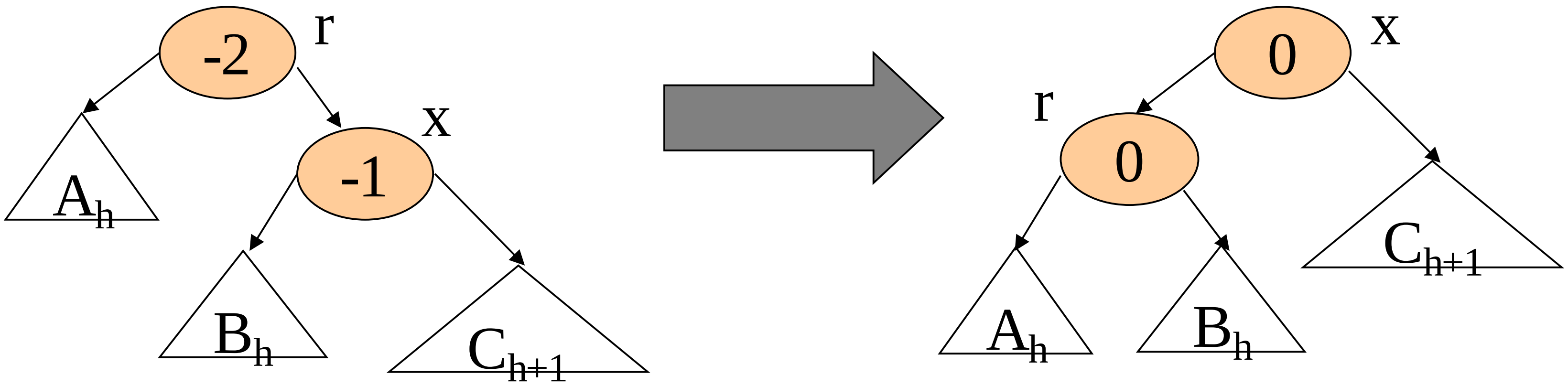
Restoring Balance

- Consider node r which was at balance -1, with an insertion on the right, making its balance -2.
- r must have had a right child, x with new balance of +1 or -1 (why not zero?).
- Two cases: x becomes +1, x becomes -1



Restoring Balance: x at -1

- Insertion in C
  - change the root of the subtree to x, make r its LST.

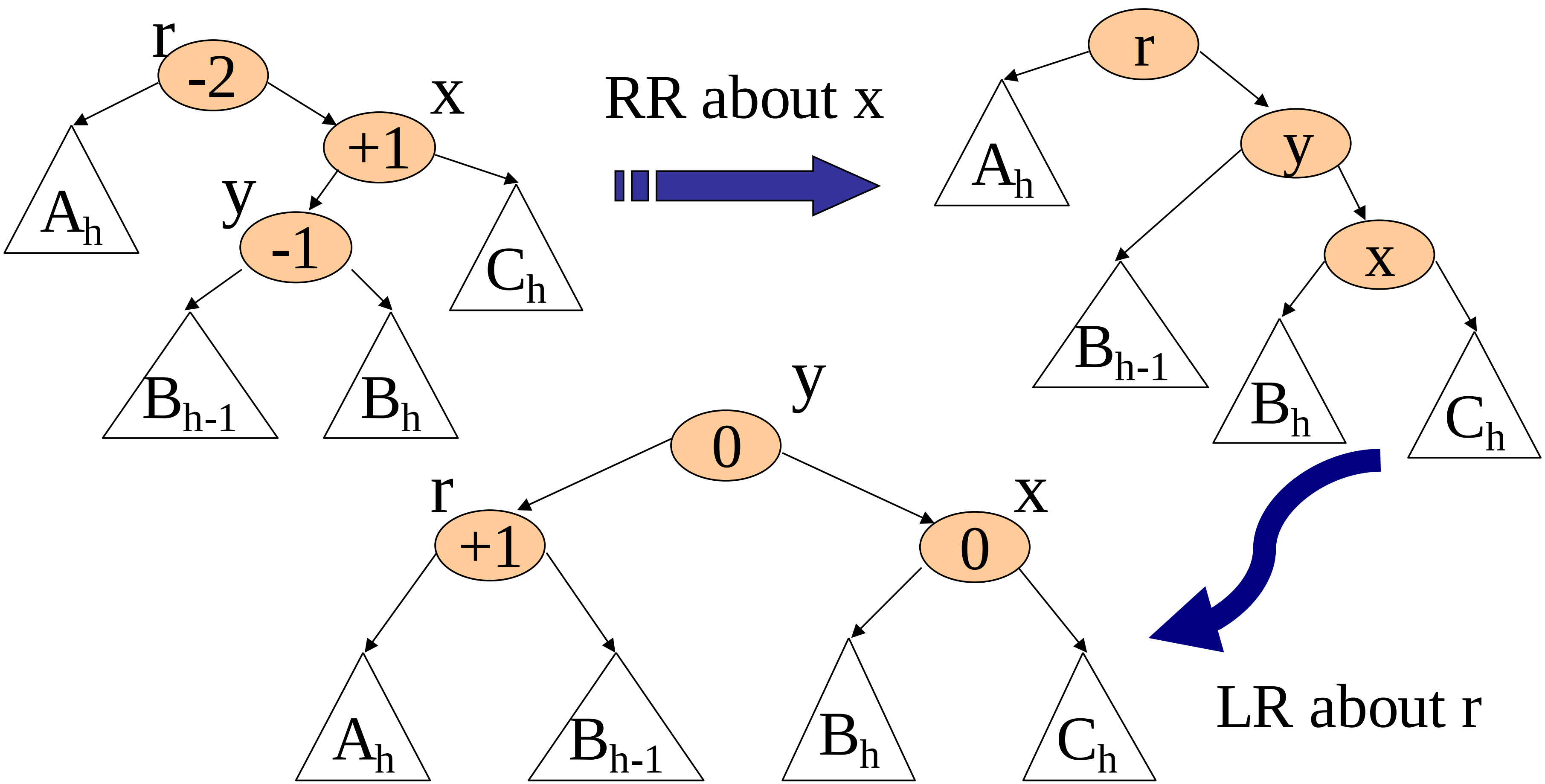


- Height of subtree unaffected (= h+2)

Restoring Balance: x at +1

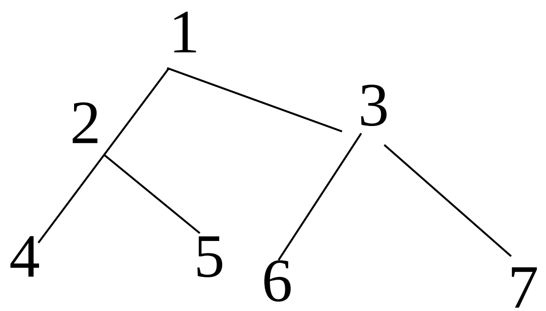
- Simple rotation will not help.
- Examine x's left child, y.
- Perform one rotation on the x-y line to make y as the new root of the RST of r.
- Now RST of r is right heavy.
- Apply the earlier method.

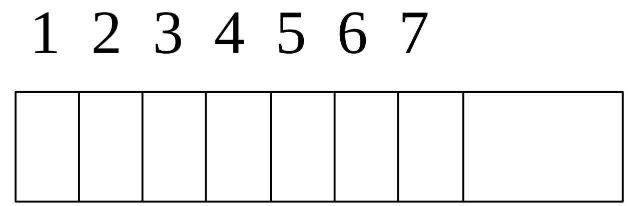
Restoring Balance: x at +1



Array Representation

- Consider a complete binary tree (CBT) and number; its nodes as follows





- View the numbers as indices of an array.
- left-child of node at p will be at 2*p.
- right-child of node at p will be at 2*p+1.
- A node does not have to keep references to its children.

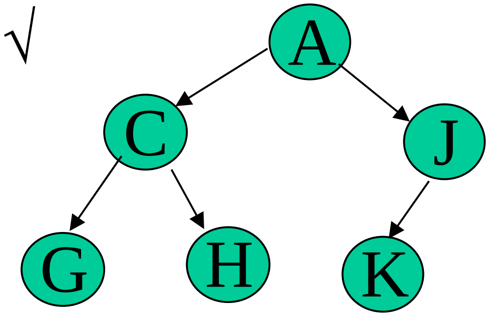
Array Representation

- For arbitrary trees, this results in many vacant spaces.
- Example: a degenerate binary tree of n nodes requires an array of 2^n elements.
- But for CBT or Almost CBT, this can be efficient.
- Also allows us to view a normal array as a binary tree!

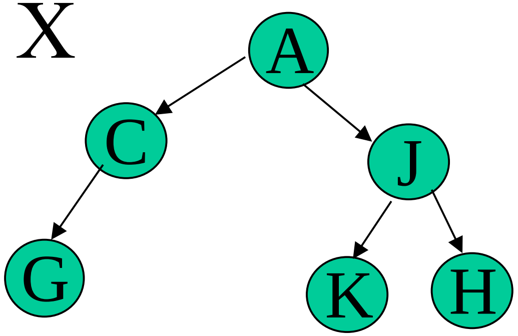
ACBT

- All leafs at lowest and next-to-lowest levels only.
- All except the lowest level is full.
- No gaps except at the end of a level.
- A perfectly balanced tree with leaves at the last level all in the leftmost position.
- A tree which can be represented without any vacant space in the array representation.

√



X



Heaps

- ACBT + a value constraint
- value in node >= value in left-child and >= value in right-child
- No relation between left-child and right-child values.
- No connection with the Heap-memory in Operating Systems.
- Applications
  - Sorting, Priority Queues

Creating a Heap

```
int heap = new int[n]; // n element heap
heap[ii] ≥ heap[2*ii+1], for 0≤ ii ≤(n-1)/2
heap[ii] ≥ heap[2*ii+2], for 0≤ ii ≤(n-2)/2
```

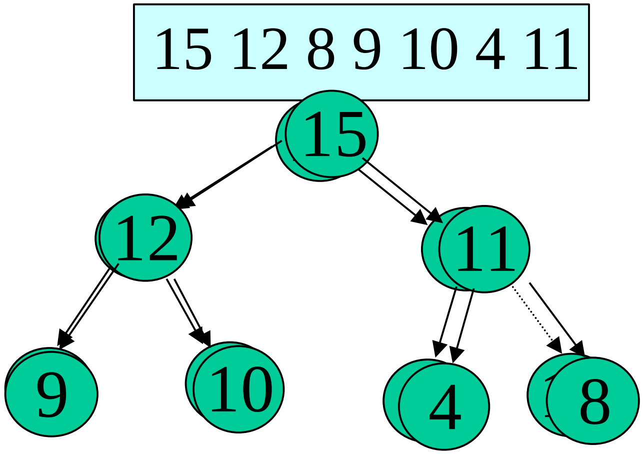
```
heapEnqueue(int el){

Insert el at the end of the heap

while(el is not in the root  and
el > parent(el))

 swap el with parent;

}
```



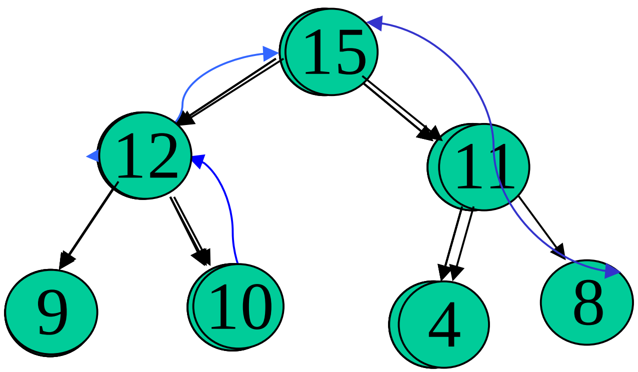
N log(N) complexity

Priority Queue Using Heap

- Descending priority queue: Always select maximum among the available items.
- Heap ensures this at the root.
- After removing the max, readjust the heap: O(log N) algorithm available.
- Insertion can be done as before.

Priority Queue Using Heap

```
heapDequeue()
1.   swap the last element
     with the root element.
2.   P = the root
3.   while(p != leaf &&
       p <any of its children)
       Swap p with larger child;
```



Priority Queue

- PQ with unsorted array gives easy insertion, but retrieval is O(N).
- PQ with sorted array gives easy retrieval, but insertion is O(N).
- PQ with Heap gives both at log N.
- No extra space requirement.

Summary

- Balanced BST – AVL Trees.
- Rotations at a node in BST.
- Almost complete binary trees and heap.