CSE 326
Lecture 18

Leftist Heaps and Skew Heaps

Winter Quarter 1999

Mergeable Priority Queue

- Mergeable priority queues allow for the efficient implementation of the operation Merge(H_1,H_2) which merges two priority queues into a single priority queue.
- Normally, Merge(H_1,H_2) returns a new priority queue destroying H_1 and H_2 in the process.
- Applications

  1. Specific algorithms like one for the optimal binary search tree.

  2. Managing work queues in a distributed system.

  3. N-body Simulation for easily maintaining the sum of all keys in the in a numerically stable way.

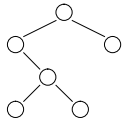
Leftist Trees

- A leftist tree is a binary tree with two properties.

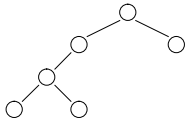
  1. Heap property - key at a node is less than or equal to the keys of its children.

  2. Leftist property - For each node the null path length of the left subtree is at least as large as the null path length of the right subtree.
- Null path length (nlp) - The null path length of a node (subtree) is the shortest path to a node without two children. A null has null path length = -1.
- Shortest path to a null is alway on the right path.

not a leftist tree



leftist tree



The Right Path of a Leftist Tree is Short

- In a leftist tree if the right path has length at least r then the leftist tree has at least $2^r - 1$ nodes.
- Proof is by induction on r.

  Basis: r = 1. The tree must have at least 1 node, the root, and $2^1 - 1 = 1$.

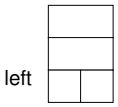
  Inductive Step: Assume the statement is true for all r' < r and r > 1. The right subtree has a right path of at least r-1 nodes, so by the induction hypothesis, it has at least $2^{r-1} - 1$ nodes. The left subtree must also have a right path of length at least r - 1. Otherwise, there is a shorter path to a null by first going left. By the inductive hypothesis it has at least $2^{r-1} - 1$ nodes. Thus the total number of nodes is at least

  $$2^{r-1} - 1 + 2^{r-1} - 1 + 1 = 2^r - 1.$$
- Thus a leftist tree with n nodes must have a right path of length $\leq \log n$.

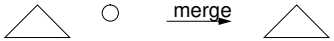
Data Structure for Leftist Trees

- Assume that a node of a leftist tree contains the following

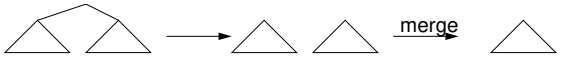
npl = null path length



- The basic operation is the merging of two leftist trees. O(log n)
- Insert by merging a single node with a leftist tree. O(log n)

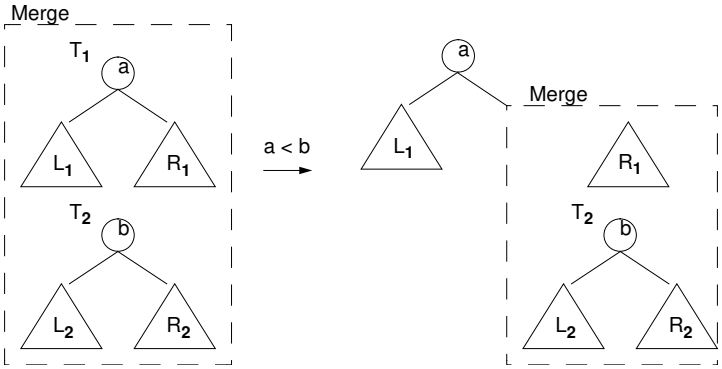
insert  merge

- Delete_min by merging the left and right subtrees of the root. O(log n)

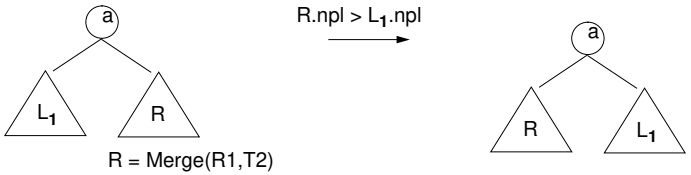
delete-min  merge

Merging Two Leftist Trees

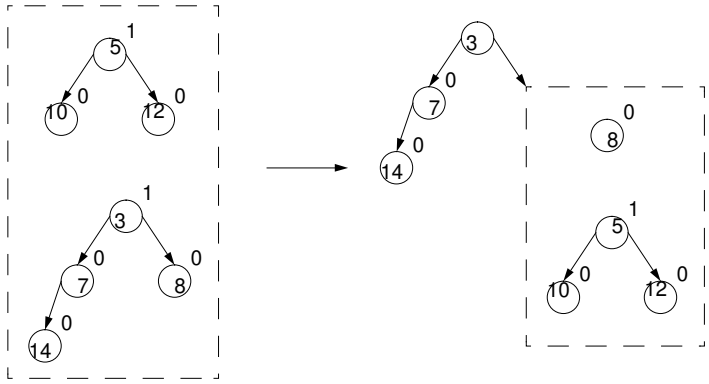
- Merge(T_1,T_2) returns a new leftist tree assuming T_1 and T_2 are two distinct leftist trees. Without loss of generality we can assume root of T_1 has the smaller key of the two roots.



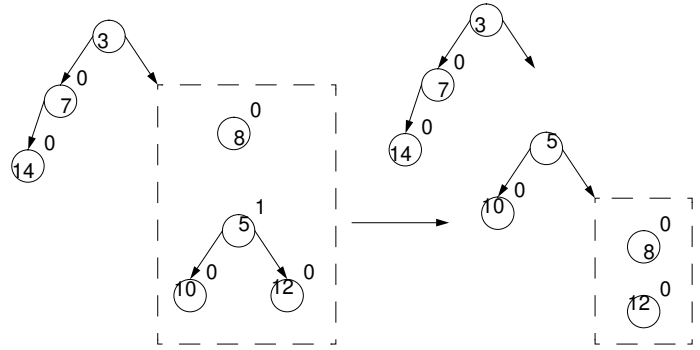
Merge Continued



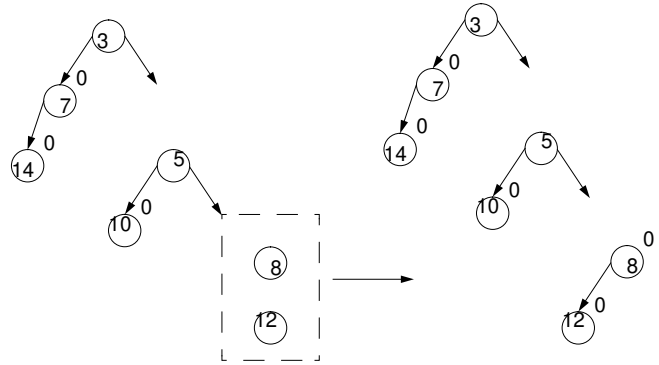
Example



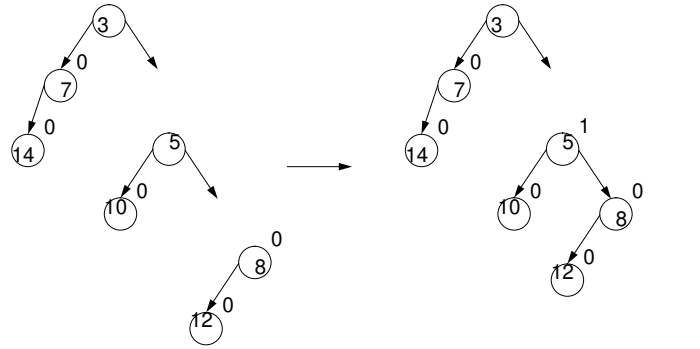
Example Continued



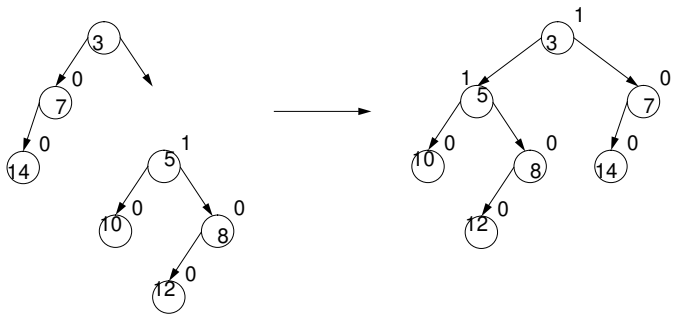
Example Continued



Example Continued



Example Continued



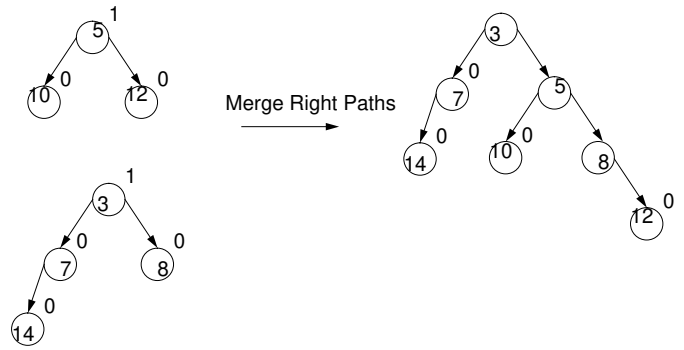
Iterative Leftist Tree Merging

- Two passes to merge T_1 and T_2.

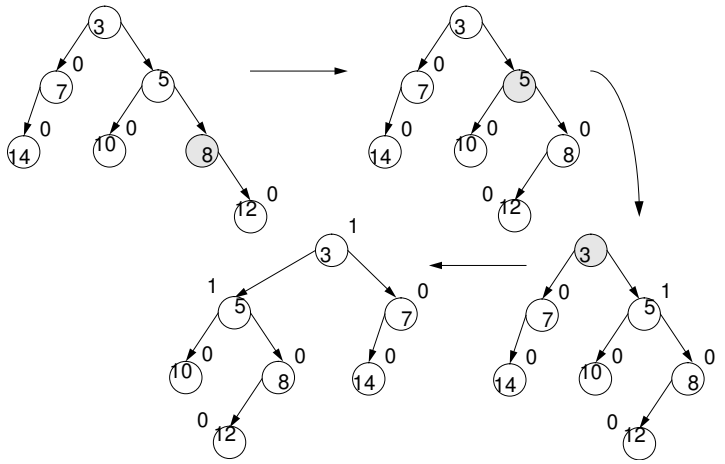
  Downward pass: merge the right paths of T_1 and T_2 forming a new tree T.

  Upward pass: Return along the right path of T computing the null path lengths and interchanging the left and right subtrees to maintain the leftist property.

Iterative Leftist Tree

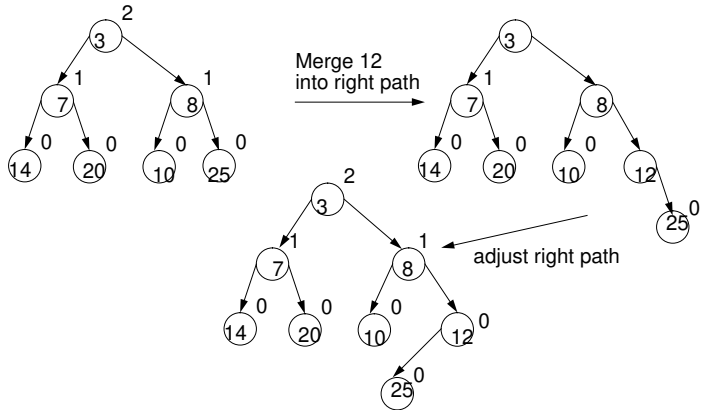


Upwards Pass

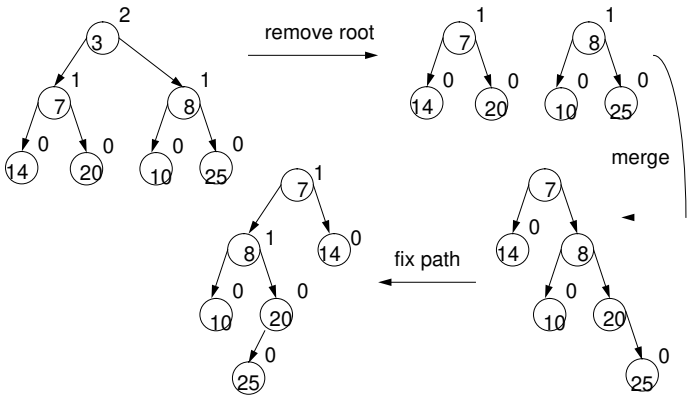


Insertion in leftist trees

- Insert 12



Delete_min

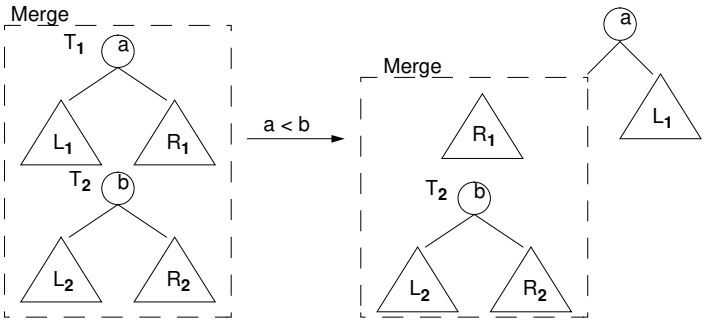


Skew Heaps

- Self adjusting version of leftist heaps (priority queues)
- Amortized time to merge, insertion, and delete_min is O(log n)
- Worst case time is O(n) for these operations.
- Merging

  Always switch children on merging except for the final step.
- Top down method - one pass

Skew Heap



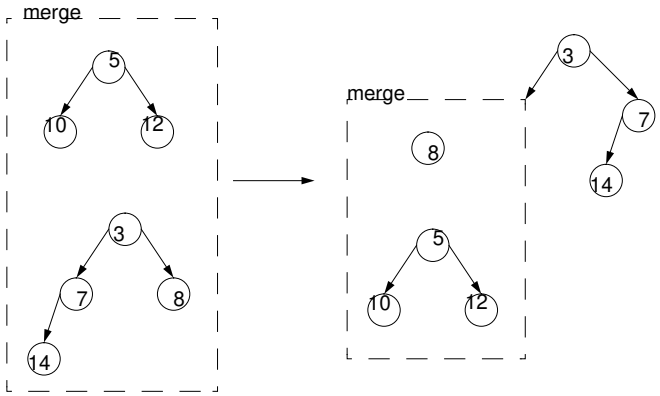
Skew Heap Code

- merge(S, T: tree pointer): tree pointer {

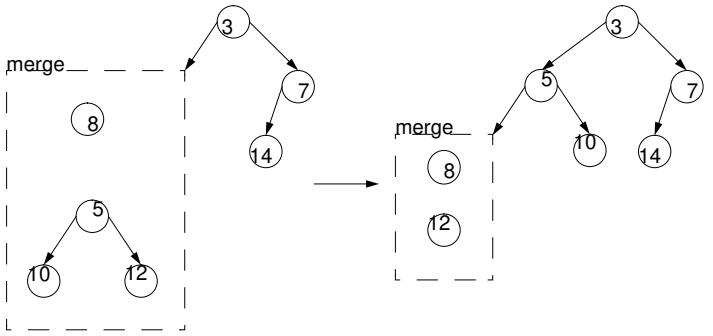
  temp: local tree pointer;

```
case
  S=null: return(T);
  T=null: return(S);
  S.key < T.key: {
       temp := S.right;
       S.right := S.left;
       S.left := merge(temp,T);
        return(S) }
  T.key ≤ S.key: {
       temp := T.right;
       T.right := T.left;
       T.left := merge(temp,S);
       return(T) }
endcase }
```

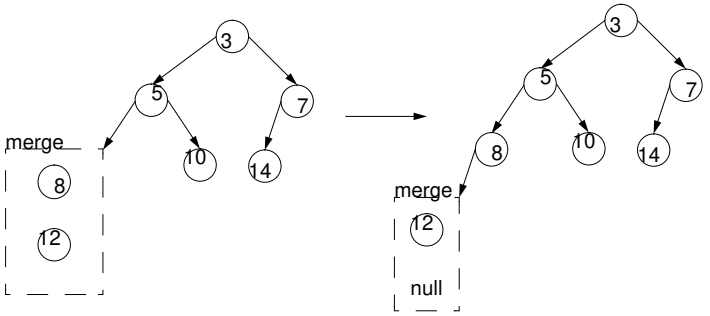
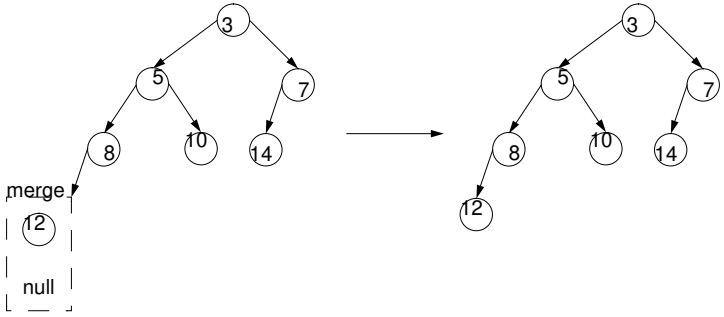
Skew Heap Example



Example Continued



Example Continued



Example Continued