Chapter 23
Merging Priority Queues

The Skew Heap

- Skew heap – heap-ordered binary tree without a balancing condition.
- With these, there is no guarantee that the depth of the tree is logarithmic.
- It supports all operations in logarithmic amortized time.
- It is somewhat like a splay tree.

Merging

- Many operations with heap-ordered trees can be done using merging.
- Operations:
  - Insert – create a one-node tree containing $x$ and merge that tree into the priority queue.
  - Find minimum – return the item at the root of the priority queue.
  - Delete minimum – delete the root and merge its left and right subtrees.
  - Decrease the value of a node – assume that $p$ points to the node in the priority queue. Lower the value of $p$’s key. Detach $p$ from its parent, which yields two priority queues. Merge the two resulting priority queues.
- Thus, we need only see how merging priority queues is implemented.

Simplistic Merging of Heap-Ordered Trees

- Assume we have two heap-ordered trees, $H_1$ and $H_2$, that need to be merged.
- If either tree is empty, the other tree is the merged tree.
- Otherwise, compare the roots.
- Recursively merge the tree with the larger root into the right subtree of the tree with the smaller root.
- See Figure 1.

![Figure 1 Simplistic merge of heap-ordered trees](image)

- The practical effect of the above operation is in fact an ordered arrangement consisting only of a single right path.
- Thus the operations can be linear.

The Skew Heap – A Simple Modification
- We can make a simple modification to the merge operation and get better results.
- Prior to the completion of a merge, we swap the left and right children for every node in the resulting right path of the temporary tree.
- Consider the example in Figure 2.

\[
\begin{array}{c}
\text{3} & \text{8} \quad + \quad \text{4} & \text{5} \\
\text{6} \\
\end{array}
\longrightarrow
\begin{array}{c}
\text{2} & \text{4} \quad + \quad \text{3} & \text{1}
\quad \quad \quad \quad \\
\text{8} & \text{9} & \text{7} & \text{6}
\end{array}
\]

Figure 2 Merging a skew heap

- When a merge is performed in this way, the heap-ordered tree is also called a skew heap.
- Let’s consider this operation from a recursive point of view. Let \( L \) be the tree with the smaller root and \( R \) be the other tree.
  1. If one tree is empty, the other is the merged result.
  2. Otherwise, let \( Temp \) by the right subtree of \( L \).
  3. Make \( L \)'s left subtree its new right subtree.
  4. Make the result of the recursive merge of \( Temp \) and \( R \) the new left subtree of \( L \).
- The result of child swapping is that the length of the right path will not be unduly large all the time.
- The amortized time needed to merge two skew heaps is \( O(\log n) \).

The Pairing Heap

- The pairing heap is a structurally unconstrained heap-ordered \( M \)-ary tree for which all operations, except deletion, take constant worst-case time.
- Deletion could take linear worst-case time.
- Consider the pairing heap shown in Figure 3.

\[
\begin{array}{c}
\text{6} & \text{3} & \text{4} \\
\text{10} & \text{13} & \text{11} & \text{15} & \text{12} & \text{17} & \text{19}
\quad \quad \quad \quad \quad \\
\text{16} & \text{18}
\end{array}
\]

Figure 3 Abstract pairing heap

- The actual implementation, using a left child/right sibling representation is shown in Figure 4.
- Constant time operations on a pairing heap
  - Merging
    - Make the heap with the larger root the new first child of the heap with the smaller root
  - Insertion
    - A special case of merge
  - Decrease key
    - Decrease the value of the requested node
    - Detach the adjusted node from its parent and merge the two pairing heaps that result
- Deletion
  - Remove the root of the tree creating a collection of heaps
  - If there are \( c \) children of the root, combining these heaps requires \( c - 1 \) merges
  - Consequently, this operation can take \( O(n) \) time.
  - The order of the merging is important.
  - A two-pass merge has been proposed
    - First scan – merges pairs of children from left to right
    - Second scan – merge the rightmost tree that remains from the first scan with the current merged result.
    - Suppose we have children \( c_1 \) through \( c_8 \).  
      - The first pass merges \( c_1 \) and \( c_2 \), \( c_3 \) and \( c_4 \), \( c_5 \), and \( c_6 \), and \( c_7 \) and \( c_8 \).
      - The result is \( d_1 \), \( d_2 \), \( d_3 \), and \( d_4 \).
      - The second pass merges \( d_3 \) and \( d_4 \); \( d_2 \) is then merged with the result, and \( d_1 \) is then merged with the result of that merge.
  - See Figure 5 after deleting 2.
Dijkstra’s Shortest Weighted Path Algorithm

- Find the shortest path (measured by total cost) from a designated vertex $S$ to every vertex. All edge costs are nonnegative
- From current node, set the cost of adjacent nodes to that of current node plus the path weight. If the node has not been visited, set the cost.
- Next node visited is the one with the least cost associated with it.
- Figure 6 shows an example.
Figure 6  Example of Dijkstra's algorithm