Priority Queue

Yu-Tai Ching
Department of Computer Science
National Chiao Tung University

Min Priority Queue

- Return an element with minimum priority
- Insert an element with arbitrary priority
- Delete an element with minimum Priority

Max priority queue is similar.

Extensions

- "Meldable single-ended priority queue", to merge two priority queues.
- data structures used are leftist tree and binomial heap.
- Further extension,
  1. delete an arbitrary element (location in the data structure is known)
  2. to decrease (increase) the key of a element.
- data structures are Fibonacci heap and pairing heaps.

Double-Ended Priority Queue

- DEPQ a data structure that supports the operations
  - Return the element with the minimum Priority.
  - Return the element with the maximum Priority.
  - Insert an element with arbitrary priority.
  - Delete an Element with the minimum priority.
  - Delete an element with the maximum priority.
- DEPQ is a min and max priority queue rolled into one structure

An Application of DEPQ

- Implement a network buffer.
- The buffer holds packets that are waiting for their turn to send out over a network link.
- maximum priority should be sent first,
- if the buffer is full, and we have to insert an element, the element with the minimum key has to be deleted.

Leftist Tree

- There are height biased (HBLT) and weight biased (WBLT) leftist trees.
- HBLTs were first invented, and are generally referred to leftist tree.

Leftist Tree

- An efficient implementation of meldable priority queue.
- Suppose there are n nodes in the priority queues needed to be melded, Leftist tree supports logarithmic merge time.
- It also supposrt insert arbitrary and delete minimum in logarithmic time.
- To define the leftist tree, we define the extended binary tree.

Extended Binary Tree

- A binary tree,
- all empty binary subtrees have been replaced by square nodes. Figures 9.1 and 9.2.
- Square nodes are the external nodes.
- original circular nodes are the internal nodes.

Leftist Tree

- x a node in an extended binary tree,
- $LeftChild(x)$ and $RightChild(x)$ denote the left and right children of the internal node x.
- $Shortest(x)$: the length of a shortest path from x to an external node.

$Shortest(x)$ satisfies the recurrence

$$Shortest(x) = \begin{cases} 0, & \text{if } x \text{ is an external node} \\ 1+ & \min\{Shortest(LeftChild(x)), \\ & Shortest(RightChild(x))\}, \text{ otherwise.} \end{cases} \tag{1}$$

Figure 9.2, the $Shortest(x)$ of nodes are written down.

Definition: A leftist tree is a binary tree such that if it is not empty, then

$$Shortest(LeftChild(x)) \geq Shortest(RightChild(x))$$

for every internal node x.
Figure 9.2 (a) is not a leftist tree, but 9.2 (b) is.

Lemma 9.1: Let r be the root of the leftist tree that has n internal nodes.

1. $n \geq 2^{shortest(r)} - 1$
2. The right most root to external node path is the shortest root to external node path. Its length is $Shortest(r) \leq \log_2(n+1)$.

Proof By the definition of the leftist tree, the path from r to the right most external node is the shortest. Thus the total number of nodes n,
$n \geq \sum_{i=1}^{shortest(r)} 2^{i-1} = 2^{shortest(r)} - 1$.

Definition: A *min leftist tree* (*max leftist tree*) is a leftist tree in which the key value in each node is no larger (smaller) than the key values in its children.
A min (max) leftist tree is a leftist tree that is also a min (max) tree.

Figure 9.3

Insert Arbitrary and Delete Min

- These operations are based on melding two min leftist tree.
- Insert x into T
  1. Create a min leftist tree containing a single element x,
  2. meld the two min leftist tree
- Delete min
  1. Meld the min leftist trees $root->leftChile$ and $root->rightChild$,
  2. delete the root.

Meld Operation

- Suppose that two min leftist trees, T_1 and T_2 are to be merged, and root of T_1 has smaller key value.
- Create a new binary tree T containing all elements in both T_1 and T_2 by
  1. root of T is root of T_1, the smaller one,
  2. Left subtree is the left subtree of T_1,
  3. The right subtree is obtained by melding the right subtree of T_1 and T_2 (note that this is a recursive process).
- Note that T has the property that it is a min tree,
- Next, interchange the left subtree and the right subtree if necessary, from bottom to top.
- Figure 9.3.

Weight-Biased Leftist Trees

- A leftist tree by considering the number of nodes in a subtree.
- define the weight $w(x)$ of a node x, the number of internal nodes in the subtree with root x.
- $w(x) = 0$ if x is an external node.
- x is an internal node: $w(x)$ is 1 more than the sum of the weights of its children.

Definition: A binary tree is a weight-biased tree (WBLT) iff at every internal node the w value of left child is greater than or equal to the w value of the right child.

Lemma 9.2: Let x be any internal node of a weight-biase leftist tree. The length, $rightmost(x)$, of the rightmost path from x to an external node satisfies

$$rightmost(x) \leq \log_2(w(x) + 1).$$

Insert arbitrary and delete maximum are similar to the Height-Biased Leftist Tree.

Double-Ended Priority Queue- DEPQ

- Represented using a symmetric min-max heap (SMMH).
- SMMH: a complete binary tree.
- Each node other than the root has exactly one element.
- root is empty, thus there are $n + 1$ nodes for an n elements DEPQ.

- N: any node of SMMH, $elements(N)$ denotes the elements in the subtree rooted at N (but not including N), if $elements(N) \neq \emptyset$
  1. The left child of N has the minimum element in $elements(N)$,
  2. the right child of N (if any) has the maximum elements in $elements(N)$.
- Figure 9.25, let N be the node with value 80, $elements(N)$ = $\{6, 14, 30, 40\}$,
  - Left Child is the smallest, 6
  - Right Child is the largets, 40.

Facts about SMMH

1. The element in each node is less than or equal to that in its right sibling (if any)
2. For every node N that has a grandparent, the element in the left child of the grandparent is less than or equal to that in N.
3. For every node N that has a grandparent, the element in the right child of the grandparent is greater than or equal to that in N.

Implemented using a 1-D array.

Insert Arbitrary into an SMMH

- Insert the new node at the end of the array.
- If the new node is the right child of its parent, verify Fact 1. Swap if needed.
- bubble-up according the Facts 2, and 3.

Figure 9.26, insert 2, then insert 50.

Deletion from an SMMH

- Similar to that a standard min heap or maximum heap.
- Discuss the delete min operation, delete maximum is the same.
  - delete min ($h[2]$) and move the last one to this position.
  - check Fact 1.
  - check Facts 2 and 3, swap with the appropriate one.
  - do this recursively.