

Skew Heap

Tobias Nipkow

May 6, 2022

Abstract

Skew heaps are an amazingly simple and lightweight implementation of priority queues. They were invented by Sleator and Tarjan [1] and have logarithmic amortized complexity. This entry provides executable and verified functional skew heaps.

The amortized complexity of skew heaps is analyzed in the AFP entry [Amortized Complexity](#).

Contents

1	Skew Heap	1
1.1	Merge	1

1 Skew Heap

theory *Skew-Heap*

imports

HOL-Library.Tree-Multiset

HOL-Library.Pattern-Aliases

HOL-Data-Structures.Heaps

begin

unbundle *pattern-aliases*

Skew heaps [1] are possibly the simplest functional priority queues that have logarithmic (albeit amortized) complexity.

The implementation below could be generalized to separate the elements from their priorities.

1.1 Merge

function *merge* :: (*a*::*linorder*) *tree* \Rightarrow *a tree* \Rightarrow *a tree* **where**

merge Leaf t = t |

merge t Leaf = t |

merge (Node l1 a1 r1 =: t1) (Node l2 a2 r2 =: t2) =

```

    (if a1 ≤ a2 then Node (merge t2 r1) a1 l1
     else Node (merge t1 r2) a2 l2)
⟨proof⟩
termination
⟨proof⟩

```

```

lemma merge-code: merge t1 t2 =
  (case t1 of
   Leaf ⇒ t2 |
   Node l1 a1 r1 ⇒ (case t2 of
    Leaf ⇒ t1 |
    Node l2 a2 r2 ⇒
      (if a1 ≤ a2
       then Node (merge t2 r1) a1 l1
       else Node (merge t1 r2) a2 l2)))
⟨proof⟩

```

An alternative version that always walks to the Leaf of both heaps:

```

function merge2 :: ('a::linorder) tree ⇒ 'a tree ⇒ 'a tree where
merge2 Leaf Leaf = Leaf |
merge2 Leaf (Node l2 a2 r2) = Node (merge2 r2 Leaf) a2 l2 |
merge2 (Node l1 a1 r1) Leaf = Node (merge2 r1 Leaf) a1 l1 |
merge2 (Node l1 a1 r1) (Node l2 a2 r2) =
  (if a1 ≤ a2
   then Node (merge2 (Node l2 a2 r2) r1) a1 l1
   else Node (merge2 (Node l1 a1 r1) r2) a2 l2)
⟨proof⟩
termination
⟨proof⟩

```

```

lemma size-merge: size(merge t1 t2) = size t1 + size t2
⟨proof⟩

```

```

lemma size-merge2: size(merge2 t1 t2) = size t1 + size t2
⟨proof⟩

```

```

lemma mset-merge: mset-tree (merge t1 t2) = mset-tree t1 + mset-tree t2
⟨proof⟩

```

```

lemma set-merge: set-tree (merge t1 t2) = set-tree t1 ∪ set-tree t2
⟨proof⟩

```

```

lemma heap-merge:
  [ heap t1; heap t2 ] ⇒ heap (merge t1 t2)
⟨proof⟩

```

```

interpretation skew-heap: Heap-Merge
where merge = merge
⟨proof⟩

```

end

References

- [1] D. D. Sleator and R. E. Tarjan. Self-adjusting heaps. *SIAM J. Comput.*, 15(1):52–69, 1986.