1 Heaps

Heaps are data structures that make it easy to find the element with the most extreme value in a collection of elements. A Min-Heap prioritizes the element with the smallest value, while a Max-Heap prioritizes the element with the largest value. Because of this property, heaps are often used to implement priority queues.

You can find more about heaps by reading pages 151–169 in CLRS.

1.1 Representing a Heap

While a heap can be represented as a regular tree, it is often more efficient to store a binary heap as an array. We call the first element in the heap element 1. Now, given an element $i$, we can find its left and right children with a little arithmetic:

Exercise.
- $\text{Parent}(i) =$
- $\text{Left}(i) =$
- $\text{Right}(i) =$

The completeness requirement makes sure this representation of heaps is compact.
1.2 Heap operations

1.2.1 Max-Heapify

Max-Heapify($H, N$): Given that the children of the node $N$ in the Max-Heap $H$ are each the root of a Max-Heap, rearranges the tree rooted at $N$ to be a Max-Heap.

\[
\text{Max-Heapify}(H, N):
\]

Require: \text{Left}(N), \text{Right}(N) are each the root of a Max-Heap

\[
(l, r) \leftarrow (\text{Left}(N), \text{Right}(N))
\]

if exists($l$) and $H[l] > H[N]$ then

\[
largest \leftarrow l
\]

else

\[
largest \leftarrow N
\]

end if

if exists($r$) and $H[r] > H[largest]$ then

\[
largest \leftarrow r
\]

end if

if $largest \neq N$ then

\[
\text{SWAP}(H[N], H[largest])
\]

Max-Heapify($H, largest$)

end if

Ensure: $N$ is the root of a Max-Heap

Exercise.

- Run Max-Heapify with $N = 1$ on

\[
H = [14, 16, 10, 8, 7, 9, 6, 2, 4, 1]
\]

- What is Max-Heapify’s run-time?
1.2.2 Build-Heap

**Build-Heap**(\(A\)): Given an unordered array, makes it into a max-heap.

**BUILD-HEAP**(\(A\)):

**Require:** \(A\) is an array.

\[
\text{for } i = \lceil \text{length}(A)/2 \rceil \text{ downto } 1 \text{ do} \\
\text{MAX-HEAPIFY}(A, i) \\
\text{end for}
\]

Exercise.

- *Run** BUILD-HEAP on \(A = [2, 1, 4, 3, 6, 5]\)
- *Running time (loose upper bound):*
- *Running time (tight upper bound):*
1.2.3 Extract-Max

**Extract-Max** \( H \): Remove the element with the largest value from the heap.

**Extract-Max** \( H \):

**Require:** \( H \) is a non-empty Max-Heap

\[
\begin{align*}
\text{max} & \leftarrow H[\text{root}] \\
H[\text{root}] & \leftarrow H[\text{Size}(H)] \text{ (last element of the heap)} \\
\text{Size}(H) & \leftarrow 1 \\
\text{Max-Heapify}(H, \text{root}) \\
\text{return} & \text{ max}
\end{align*}
\]

Exercise.

- Run **Extract-Max** on \( H = [6, 3, 5, 2, 1, 4] \).
- What is **Extract-Max**’s run time?
1.2.4 Insert

Insert($H, v$): Add the value $v$ to the heap $H$.

**Insert($H, v$):**

**Require:** $H$ is a MAX-HEAP, $v$ is a new value.

$\text{Size}(H) += 1$

$H[\text{Size}(H)] \leftarrow v$ \{Set $v$ to be in the next empty slot.\}

$N \leftarrow \text{Size}(H)$ \{Keep track of the node currently containing $v$.\}

**while** $N$ is not the root and $H[\text{Parent}(N)] < H[N]$ **do**

\quad \text{Swap}(H[\text{Parent}(N)], H[N])

\quad $N \leftarrow \text{Parent}(N)$

**end while**

Exercise.

- Run Insert($H, v$) with $v = 8$ and $H = [6, 3, 5, 2, 1, 4]$

- What is Insert’s runtime?
2 Graph Problems

Exercise. Answer T/F for the following problems:

(a) We know that the node with the highest post-order belongs to a source SCC. Then the node with the lowest post-order always belongs to a sink SCC.

(b) Suppose two vertices $u$ and $v$ in a directed graph satisfy $\text{pre}(u) < \text{post}(u) < \text{pre}(v) < \text{post}(v)$, then there can be no edge in either direction between $u$ and $v$.

(c) In a DFS of a directed graph $G$, the set of vertices reachable from the vertex with lowest post-order is a strongly-connected component of $G$.

(d) In a DFS of a directed graph $G$, the set of vertices reachable from the vertex with highest post-order is a strongly-connected component of $G$.

(e) If a DFS has a cross edge, the graph is not strongly connected.