Heaps

- Heaps
- Properties
- Deletion, Insertion, Construction
- Implementation of the Heap
- Implementation of Priority Queue using a Heap
- An application: HeapSort

Heaps (Min-heap)

Complete binary tree that stores a collection of keys (or key-element pairs) at its internal nodes and that satisfies the additional property:

key(parent) ≤ key(child)

REMEMBER: complete binary tree
all levels are full, except the last one, which is left-filled

Max-heap

key(parent) ≥ key(child)

We store the keys in the internal nodes only

After adding the leaves the resulting tree is full

Height of a Heap

- Theorem: A heap storing \( n \) keys has height \( \Theta (\log n) \)
  - Let \( h \) be the height of a heap storing \( n \) keys
  - Since there are \( 2^i \) keys at depth \( i \), \( 0 \leq i \leq h - 2 \) and at least one key at depth \( h - 1 \), we have \( n \geq 1 + 2 + 4 + \ldots + 2^{h-2} + 1 \)
  - Thus, \( n \geq 2^h - 1 \), i.e., \( h \leq \log n + 1 \)

Notice that ...

- We could use a heap to implement a priority queue
- We store a (key, element) item at each internal node

RemoveMin():

\[ \rightarrow \text{Remove the root} \]
\[ \rightarrow \text{Re-arrange the heap!} \]
Removal From a Heap

- The removal of the top key leaves a hole
- We need to fix the heap
- First, replace the hole with the last key in the heap
- Then, begin Downheap

Downheap

- Downheap compares the parent with the smallest child. If the child is smaller, it switches the two.

Downheap Continues

Downheap Continues

End of Downheap

- Downheap terminates when the key is greater than the keys of both its children or the bottom of the heap is reached.
  - (total #swaps) ≤ (h - 1), which is O(log n)

Heap Insertion

The key to insert is 6
Heap Insertion
Add the key in the next available position in the heap.

Now begin Upheap.

Upheap
• Swap parent-child keys out of order

Upheap Continues

End of Upheap
• Upheap terminates when new key is greater than the key of its parent or the top of the heap is reached
• (total #swaps) (h - 1), which is O(log n)

Heap Construction
We could insert the items one at the time with a sequence of Heap Insertions:

\[ \sum_{k=1}^{n} \log k = O(n \log n) \]

But we can do better ….

Bottom-up Heap Construction
• We can construct a heap storing n given keys using a bottom-up construction
Construction of a Heap

Idea: Recursively re-arrange each sub-tree in the heap starting with the leaves

Example 1 (Max-Heap)

Example 1

Example 2 (min-heap)

Example 2

--- keys given one at a time ---

--- keys already in the tree ---

I am now drawing the leaves anymore here

This is not a heap!

Example 1

Finally: 2 \leftrightarrow 10

2 \leftrightarrow 8
Analysis of Heap Construction

Number of swaps

- 3 swaps
- 2 swaps
- 1 swap
- 0 swaps

Let L be the max level

At level i the number of swaps is ≤ L - i for each node

At level i there are ≤ 2^i nodes

Total: ≤ Σ(L - i)·2^i

Calculating O(Σ(L - i)·2^i)

Let j = L - i, then i = L - j and Σ(L - i)·2^i = Σ j·2^i = 2^j·Σ j·2^i

Consider Σ j·2^i:

Σ j·2^i = 1/2 + 2·1/4 + 3·1/8 + 4·1/16 + ...
= 1/2 + 1/4 + 1/8 + 1/16 + ... < 1 + 1/2 + 1/4 + 1/8 + 1/16 + ... = 1

So 2^L·Σ j·2^i ≤ 2^L

So, the number of swaps is ≤ O(n)
Implementing a Heap with an Array

A heap can be nicely represented by a vector (array-based), where the node at rank $i$ has
- left child at rank $2i$
and
- right child at rank $2i + 1$

The leaves do not need to be explicitly stored.

Reminder ....

<table>
<thead>
<tr>
<th>Left child of $T[i]$</th>
<th>Right child of $T[i]$</th>
<th>Parent of $T[i]$</th>
<th>The Root</th>
<th>Leaf? $T[i]$</th>
</tr>
</thead>
</table>

$n = 11$

Implementation of a Priority Queue with a Heap

Application: Sorting

Heap Sort

Construct initial heap $O(n)$

Remove root $O(1)$

Re-arrange $O(\log n)$

Remove root $O(1)$

Re-arrange $O(\log (n-1))$

...$O(1)$

...$O(\log (n-1))$
When there are \( i \) nodes left in the PQ: \( \left\lfloor \log i \right\rfloor \)

\[ \text{TOT} = \sum_{i=1}^{i} \left\lfloor \log i \right\rfloor \]

\[ = (n + 1)q - 2^{q-1} + 2 \]

where \( q = \left\lfloor \log (n+1) \right\rfloor \)

\[ \Rightarrow O(n \log n) \]