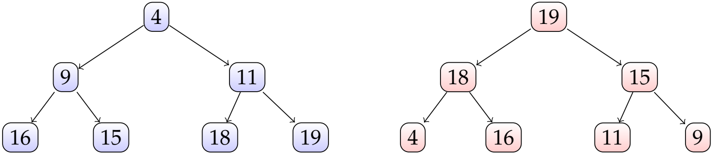
CSCI 2270: Data Structures

Lecture 25: Priority Queues and Heaps

Ashutosh Trivedi



Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Priority Queues

Heaps

Priority Queues

1. A priority queue is an abstract data type (ADT) similar to queues, where the elements have a priority field attached to them.
2. In a priority queue, the elements with high priority are served before the elements with low priority.
3. In addition, in some implementations, the order amongst the elements with equal priority follows the order they were enqueued.
4. Stacks and queues may be modeled as priority queues, where in a stack the priority of each inserted element is monotonically increasing, while in a queue the priority of each inserted element is monotonically decreasing.
5. Applications: Job-scheduling in operating systems, load-balancing problems, emergency-room patient priority, Dijkstra's algorithm to find shortest path, best-first search algorithms in graphs, Prim's minimum spanning tree algorithm.

Priority Queue Implementations

- Key operations:
  - Adding an element (push)
  - Deleting an Element (pop).
- Implementation as an unsorted array:
  - push: $O(1)$
  - pop: $O(n)$
- Implementation as a sorted array:
  - push: $O(n)$
  - pop: $O(1)$
- Similar complexities for sorted and unsorted linked-list implementations.
- Can we do better?

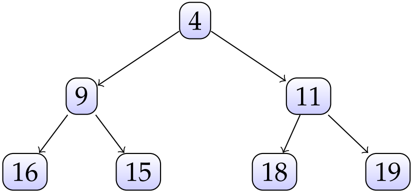
Priority Queue Implementations

- Key operations:
  - Adding an element (push)
  - Deleting an Element (pop).
- Implementation as an unsorted array:
  - push: $O(1)$
  - pop: $O(n)$
- Implementation as a sorted array:
  - push: $O(n)$
  - pop: $O(1)$
- Similar complexities for sorted and unsorted linked-list implementations.
- Can we do better? Binary Heaps!

Priority Queues

Heaps

Binary Heaps: Min Heap

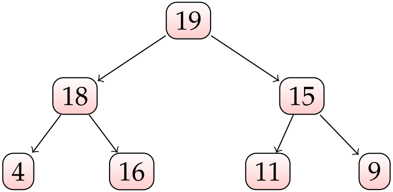


Min Heap Property:

- A complete binary-tree, i.e. difference in height between two branches is at most 1.
- If x is a node and y is its (either left or right) child then

$$x.priority \leq y.priority.$$

Binary Heaps: Max Heap

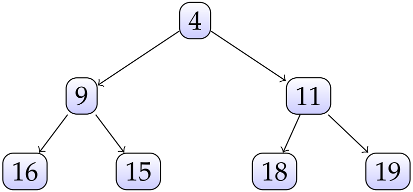


Max Heap Property:

- A complete binary-tree, i.e. difference in height between two branches is at most 1.
- If x is a node and y is its (either left or right) child then

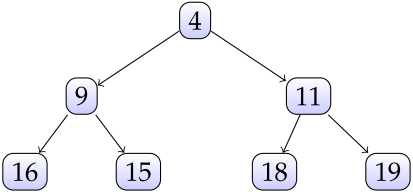
$$x.priority \geq y.priority.$$

Implementing a Heap as an array



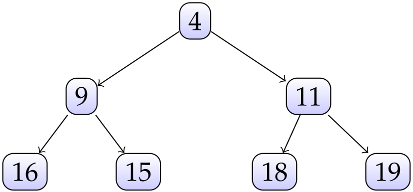
0	1	2	3	4	5	6
4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Implementing a Heap as an array



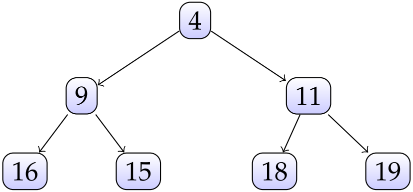
0	1	2	3	4	5	6
4	9	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Implementing a Heap as an array



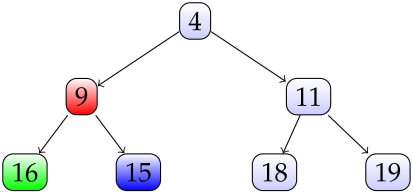
0	1	2	3	4	5	6
4	9	11	$\perp$	$\perp$	$\perp$	$\perp$

Implementing a Heap as an array



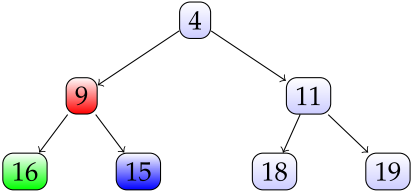
0	1	2	3	4	5	6
4	9	11	16	15	18	19

Implementing a Heap as an array



0	1	2	3	4	5	6
4	9	11	16	15	18	19

Implementing a Heap as an array



0	1	2	3	4	5	6
4	9	11	16	15	18	19

$$
\begin{aligned}
\mathit{leftChild}(i) &= 2*i+1 \\
\mathit{rightChild}(i) &= 2*i+2 \\
\mathit{parent}(i) &= \mathit{floor}((i-1)/2)
\end{aligned}
$$

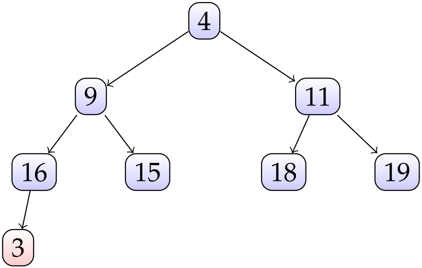
Min Heap: Abstract DataType

```
class MinHeap {
private:
  int* heap;
  int capacity;
  int currentSize;

public:
  MinHeap();
  MinHeap(int s);
  ~MinHeap();

  void push(int value);
  int pop();
  int peek();
  void printHeap();

private:
  void minHeapify(int index);
  int parent(int index) {return (index-1)/2;}
  int leftChild(int index) {return 2*index+1;}
  int rightChild(int index) {return 2*index+2;}
  void swap(int &x, int &y) {int z = x; x = y; y = z;}
};
```

Max Heap: Abstract DataType

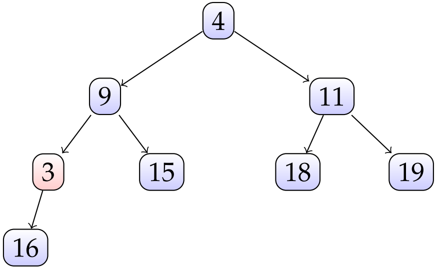
```
class MaxHeap {
private:
  int* heap;
  int capacity;
  int currentSize;

public:
  MaxHeap();
  MaxHeap(int s);
  ~MaxHeap();

  void push(int value);
  int pop();
  int peek();
  void printHeap();

private:
  void maxHeapify(int index);
  int parent(int index) {return (index-1)/2;}
  int leftChild(int index) {return 2*index+1;}
  int rightChild(int index) {return 2*index+2;}
  void swap(int &x, int &y) {int z = x; x = y; y = z;}
};
```

Push: Inserting an element to a Min Heap



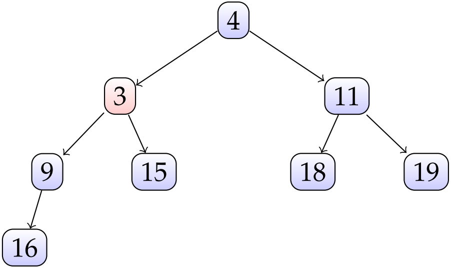
0	1	2	3	4	5	6	7
4	9	11	16	15	18	19	3

Push: Inserting an element to a Min Heap



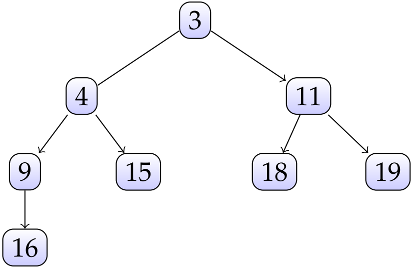
0	1	2	3	4	5	6	7
4	9	11	3	15	18	19	16

Push: Inserting an element to a Min Heap



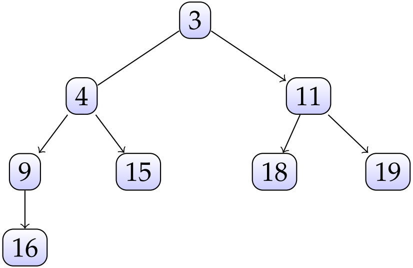
0	1	2	3	4	5	6	7
4	3	11	9	15	18	19	16

Push: Inserting an element to a Min Heap



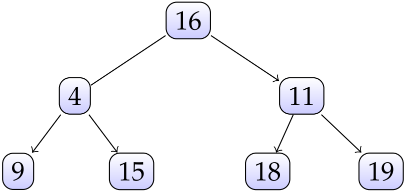
0	1	2	3	4	5	6	7
3	4	11	9	15	18	19	16

Pop: Deleting maximal element from a Min Heap



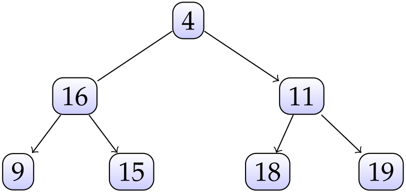
0	1	2	3	4	5	6	7
3	4	11	9	15	18	19	16

Pop: Deleting maximal element from a Min Heap



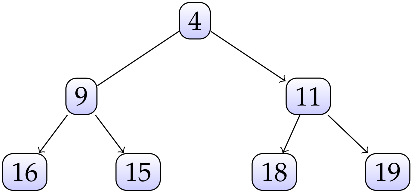
0	1	2	3	4	5	6	7
16	4	11	9	15	18	19	$\perp$

Pop: Deleting maximal element from a Min Heap



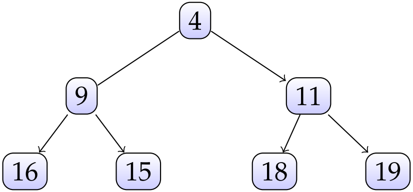
0	1	2	3	4	5	6	7
4	16	11	9	15	18	19	$\perp$

Pop: Deleting maximal element from a Min Heap



0	1	2	3	4	5	6	7
4	9	11	16	15	18	19	$\perp$

Min Heap: Complexity



0	1	2	3	4	5	6	7
4	9	11	16	15	18	19	$\perp$

- Push: $O(log(n))$
- Pop: $O(log(n))$