Lecture 02/15/17

Lecturer: Xiaodi Wu

Reading: Chap 6.3
Heap Example: min-heap only keys

The diagram represents a min-heap with the following structure:

- **Root (4)**
  - **Left child (5)**
    - **Left grandchild (15)**
      - **Left great-grandchild (16)**
  - **Right child (6)**
    - **Left grandchild (9)**
      - **Left great-grandchild (12)**
    - **Right grandchild (7)**
      - **Right great-grandchild (8)**
    - **Right great-grandchild (20)**
      - **Right great-grandgreatchild (24)**
Heap Example: min-heap only keys

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8, 22, 24]
Heap: Bottom-Up Build

Building a Heap of $n$ key-element pairs

- The first part of the heap sort.
Heap: Bottom-Up Build

Building a Heap of $n$ key-element pairs

- The first part of the heap sort.
- Approach 1: insert $n$ key-element pairs one by one. $O(n \log n)$
Building a Heap of $n$ key-element pairs

- The first part of the heap sort.
- Approach 1: insert $n$ key-element pairs one by one. $O(n \log n)$

\[ \sum_{i=1}^{n} \log(i) \in O(n \log n) \]
Heap: Bottom-Up Build

Building a Heap of $n$ key-element pairs

- The first part of the heap sort.
- Approach 1: insert $n$ key-element pairs one by one. $O(n \log n)$

$$
\sum_{i=1}^{n} \log(i) \in O(n \log n), \Omega(n \log n)?
$$
Heap: Bottom-Up Build

Building a Heap of $n$ key-element pairs

▶ The first part of the heap sort.
▶ Approach 1: insert $n$ key-element pairs one by one. $O(n \log n)$

$$\sum_{i=1}^{n} \log(i) \in O(n \log n), \Omega(n \log n)$$

▶ Can we improve the efficiency if $n$ key-element pairs have already been stored in the array $A[1 \cdots n]$?
Heap: Bottom-Up Build

Building a Heap of \( n \) key-element pairs

- The first part of the heap sort.
- Approach 1: insert \( n \) key-element pairs one by one. \( O(n \log n) \)

\[
\sum_{i=1}^{n} \log(i) \in O(n \log n), \Omega(n \log n)\
\]

- Can we improve the efficiency if \( n \) key-element pairs have already been stored in the array \( A[1 \cdots n] \)?
- Use the array-based implementation, and use the bottom-up build of heaps, \( O(n) \)!
Heap: Bottom-Up Build

Building a Heap of $n$ key-element pairs

- The first part of the heap sort.
- Approach 1: insert $n$ key-element pairs one by one. $O(n \log n)$

\[ \sum_{i=1}^{n} \log(i) \in O(n \log n), \Omega(n \log n) \]

- Can we improve the efficiency if $n$ key-element pairs have already been stored in the array $A[1 \cdots n]$?
- Use the array-based implementation, and use the bottom-up build of heaps, $O(n)!$ optimal? $\Omega(n)$?
Heap: Bottom-Up Build

Building a Heap of $n$ key-element pairs

- The first part of the heap sort.
- Approach 1: insert $n$ key-element pairs one by one. $O(n \log n)$

$$\sum_{i=1}^{n} \log(i) \in O(n \log n), \Omega(n \log n)?$$

- Can we improve the efficiency if $n$ key-element pairs have already been stored in the array $A[1 \cdots n]$?
- Use the array-based implementation, and use the bottom-up build of heaps, $O(n)!$ optimal? $\Omega(n)$?
- Imply any improvement of the heap sort?
Algorithm Build-Min-Heap(A)
Input: an $n$-element array $A$.
Output: a valid min-heap stored in $A$

A.heap-size = A.length; i.e., $n$
for $i = \lfloor A.heap-size/2 \rfloor$ downto 1 do
   Min-Heapify(A,i)
end for
Heap Example: only keys

[14, 9, 8, 25, 5, 11, 27, 16, 15, 4, 12, 6, 7, 23, 20]
Heap Example: only keys

[14, 9, 8, 25, 5, 11, 27, 16, 15, 4, 12, 6, 7, 23, 20]
Heap Example: only keys

\[ [14, 9, 8, 25, 5, 11, \textcolor{red}{20}, 16, 15, 4, 12, 6, 7, 23, \textcolor{red}{27}] \]
Heap Example: only keys

[14, 9, 8, 25, 5, 11, 20, 16, 15, 4, 12, 6, 7, 23, 27]
Heap Example: only keys

[14, 9, 8, 15, 4, 6, 20, 16, 25, 5, 12, 11, 7, 23, 27]
Heap Example: only keys

[14, 9, 8, 15, 4, 6, 20, 16, 25, 5, 12, 11, 7, 23, 27]
Heap Example: only keys

[14, 4, 6, 15, 5, 7, 20, 16, 25, 9, 12, 11, 8, 23, 27]
Heap Example: only keys

[14, 4, 6, 15, 5, 7, 20, 16, 25, 9, 12, 11, 8, 23, 27]
Heap Example: only keys

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8, 23, 27]
Heapify: Correctness & Efficiency

Correctness

- Prove by the loop invariant: Each node $i + 1, i + 2, \ldots, n$ is the root of a (sub)-min-heap.
Heapify: Correctness & Efficiency

Correctness

- Prove by the loop invariant: Each node $i + 1, i + 2, \ldots, n$ is the root of a (sub)-min-heap.

- (Ex 6.1-7) In an $n$-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$. 

Efficiency

- What is the worst case complexity?

- What is the worst case for each level?

- On Level $i$, $2^i$ nodes. Each node could down-heap bubbling from level $i$ to the external nodes: $O(h - i)$.

- Thus, the total running time is $O\left(\sum_{i=0}^{h} 2^i (h - i)\right) = O\left(\log(n) \sum_{i=0}^{h} 2^i (\log(n) - i)\right)$.
Heapify: Correctness & Efficiency

Correctness

▶ Prove by the loop invariant: Each node \(i + 1, i + 2, \cdots, n\) is the root of a (sub)-min-heap.

▶ (Ex 6.1-7) In an \(n\)-element heap, the leaves are the nodes indexed by \(\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n\).

Efficiency

▶ What is the worst case complexity?
Heapify: Correctness & Efficiency

Correctness

► Prove by the loop invariant: **Each node** $i + 1, i + 2, \cdots, n$ **is the root of a (sub)- min-heap.**
► (Ex 6.1-7) In an $n$-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n$.

Efficiency

► What is the worst case complexity?
► What is the worst case for each level?
Heapify: Correctness & Efficiency

Correctness

- Prove by the loop invariant: **Each node** \( i + 1, i + 2, \cdots, n \) **is the root of a (sub)-min-heap.**

- (Ex 6.1-7) In an \( n \)-element heap, the leaves are the nodes indexed by \( \lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n. \)

Efficiency

- What is the worst case complexity?
- What is the worst case for each level?
- On Level \( i \), \( 2^i \) nodes. Each node could down-heap bubbling from level \( i \) to the external nodes: \( O(h - i) \).
Heapify: Correctness & Efficiency

Correctness

- Prove by the loop invariant: Each node \( i + 1, i + 2, \cdots, n \) is the root of a (sub)-min-heap.
- (Ex 6.1-7) In an \( n \)-element heap, the leaves are the nodes indexed by \( \lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n \).

Efficiency

- What is the worst case complexity?
- What is the worst case for each level?
- On Level \( i \), \( 2^i \) nodes. Each node could down-heap bubbling from level \( i \) to the external nodes: \( O(h - i) \).
- Thus, the total running time is

\[
O\left( \sum_{i=0}^{h} 2^i (h - i) \right) = O\left( \sum_{i=0}^{\log(n)} 2^i (\log(n) - i) \right)
\]
Efficiency Cont’d

\[
\log(n) \sum_{i=0}^{\log(n)} 2^i (\log(n) - i) = \sum_{i=0}^{\log(n)} 2^{\log(n) - i} i
\]

\[
= n \sum_{i=0}^{\log(n)} \frac{i}{2^i}
\]

\[
\leq n \times 2 = 2n
\]

The last inequality comes from the bonus problem in assignment 1.
Minqueue v.s. Priority Queue

Similarity

- Queue.
- Find Min: Minqueue $O(1)$ vs Priority Queue $O(1)$. 
Minqueue v.s. Priority Queue

Similarity

- Queue.
- Find Min: Minqueue $O(1)$ vs Priority Queue $O(1)$.

Difference

- Minqueue: does not support removeMin().
Minqueue v.s. Priority Queue

Similarity

- Queue.
- Find Min: Minqueue $O(1)$ vs Priority Queue $O(1)$.

Difference

- Minqueue: does not support removeMin().
- Minqueue cannot be directly useful for sorting.
Minqueue v.s. Priority Queue

Similarity

- Queue.
- Find Min: Minqueue $O(1)$ vs Priority Queue $O(1)$.

Difference

- Minqueue: does not support removeMin().
- Minqueue cannot be directly useful for sorting.
- Essential tradeoff?