Lecture 10/31/16

Lecturer: Xiaodi Wu

Reading: Chapter 2.4, [CLRS] Chap 6
Heap Example: only keys

```plaintext
[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8, 22, 24]
```
Heap Example: 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. $O(n \log n)$
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. $O(n \log n)$

$$\sum_{i=1}^{n} \log(i) \in O(n \log n)$$
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 \( A[0 \cdots n - 1] \)?
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 $A[0 \cdot \cdot n - 1]$?
- Use the array-based implementation, and use the bottom-up build of heaps, $O(n)$!
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. $O(n \log n)$

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

- Can we improve the efficiency if $n$ key-element pairs have already been stored in the array $A[0 \cdots n - 1]$?
- Use the array-based implementation, and use the bottom-up build of heaps, $O(n)!$ optimal? $\Omega(n)?$
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. $O(n \log n)$

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

- Can we improve the efficiency if $n$ key-element pairs have already been stored in the array $A[0 \cdots n - 1]$?
- 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 BottomUpHeapify(A)
Input: an $n$-element array $A$.
Output: a valid heap stored in $A$
Note: array-based implementation of binary trees
Bottom-Up Heapify

**Algorithm** BottomUpHeapify(A)
Input: an $n$-element array $A$.
Output: a valid heap stored in $A$
Note: array-based implementation of binary trees
if $A$ is empty then
    return an empty heap (a single external node)
end if
Algorithm BottomUpHeapify($A$)
Input: an $n$-element array $A$.
Output: a valid heap stored in $A$
Note: array-based implementation of binary trees
if $A$ is empty then
    return an empty heap (a single external node)
end if
Let $u$ be the root of the subtree $A$. Let $k$ be its key.
Let $A_L$, $A_R$ be the left-subtree and the right-subtree of $u$ respectively.
**Algorithm** BottomUpHeapify(A)
Input: an $n$-element array $A$.
Output: a valid heap stored in $A$
Note: array-based implementation of binary trees

- **if** $A$ is empty **then**
  - **return** an empty heap (a single external node)
  **end if**

Let $u$ be the root of the subtree $A$. Let $k$ be its key.
Let $A_L$, $A_R$ be the left-subtree and the right-subtree of $u$ respectively.

- $T_L \leftarrow \text{BottomUpHeapify}(A_L)$.
- $T_R \leftarrow \text{BottomUpHeapify}(A_R)$.

Create Binary Tree with root $u$ and $T_L$ the left-subtree, $T_R$ the right-subtree.
Algorithm BottomUpHeapify(A)
Input: an $n$-element array $A$.
Output: a valid heap stored in $A$
Note: array-based implementation of binary trees
if $A$ is empty then
    return an empty heap (a single external node)
end if
Let $u$ be the root of the subtree $A$. Let $k$ be its key.
Let $A_L$, $A_R$ be the left-subtree and the right-subtree of $u$ respectively.
$T_L \leftarrow$ BottomUpHeapify($A_L$).
$T_R \leftarrow$ BottomUpHeapify($A_R$).
Create Binary Tree with root $u$ and $T_L$ the left-subtree, $T_R$ the right-subtree.
Down-Heap Bubbling on $u$ if necessary.
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

$[14, 9, 8, 15, 4, 6, 20, 16, 25, 5, 12, 11, 7, 23, 27]$
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 induction.

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 \(O(\sum_{i=0}^{h-1} 2^i (h-i)) = O\left(\log(n) \sum_{i=0}^{h-1} 2^i (\log(n)-i)\right)\).
Heapify: Correctness & Efficiency

Correctness

- Prove by induction.
- Both subtrees are valid heap. So only need to down-heap bubbling the root. Remember the removeMin() case.
Heapify: Correctness & Efficiency

Correctness

- Prove by induction.
- Both subtrees are valid heap. So only need to down-heap bubbling the root. Remember the removeMin() case.

Efficiency

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

Correctness

- Prove by induction.
- Both subtrees are valid heap. So only need to down-heap bubbling the root. Remember the removeMin() case.

Efficiency

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

Correctness

- Prove by induction.
- Both subtrees are valid heap. So only need to down-heap bubbling the root. Remember the removeMin() case.

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 induction.
- Both subtrees are valid heap. So only need to down-heap bubbling the root. Remember the removeMin() case.

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

$$O \left( \sum_{i=0}^{h} 2^i (h - i) \right) = O \left( \sum_{i=0}^{\log(n)} 2^i (\log(n) - i) \right)$$
\[
\sum_{i=0}^{\log(n)} 2^i (\log(n) - i) = \sum_{i=0}^{\log(n)} 2^{\log(n)-i} i
\]
\[
= n \sum_{i=0}^{\log(n)} \frac{i}{2^i}
\leq n \times 2 = 2n
\]

The last inequality comes from the bonus problem in assignment 1.
\[
\sum_{i=0}^{\log(n)} 2^i (\log(n) - i) = \sum_{i=0}^{\log(n)} 2^{\log(n) - i} i
\]
\[
= n \sum_{i=0}^{\log(n)} \frac{i}{2^i}
\]
\[
\leq n \times 2 = 2n
\]

The last inequality comes from the bonus problem in assignment 1. **Remark:** the textbook uses another (visualized) approach of proving the complexity.
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?