Admins

Topics to be discussed:
1. Class Status (Tutorial, Lab, CS2010 Account Issues?)
2. Plan for Help Session?

This slide will be updated on the lecture day itself
Class Status

See Steven’s private IVLE tutorial and lab section for Tutorial and Lab status

See Steven’s private IVLE leaderboard section to see if your CS2010 account is in order

• You should be able to see your VisuAlgo Online Quiz 0 scores (most people will score 1% or near 1%, after practice)

• Remember that you will use CS2010 account to check your class score in private IVLE leaderboard section, login to VisuAlgo Online Quiz, and login to Mooshak for all Pses (PS1 will be out soon)
Outline

What are you going to learn in this lecture?

• Motivation: Abstract Data Type: **PriorityQueue**
• With major help from [VisuAlgo Binary Heap Visualization](#)  
  – **Binary Heap** data structure and it’s operations  
  – Building Heap from a set of \(n\) numbers in \(O(n)\)  
  – **Heap Sort** in \(O(n \log n)\)
• CS2010 PS1 Overview: “Scheduling Deliveries, v2015”

Reference in CP3 book: Page 43-47 + 148-150
Imagine that you are the Air Traffic Controller:

- You have scheduled the next **aircraft X** to land in the **next 3 minutes**, and **aircraft Y** to land in the **next 6 minutes**
- Both have enough fuel for at least the next **15 minutes** and both are just **2 minutes** away from your airport
The next two slides are hidden...

Attend the lecture to figure out
Abstract Data Type: PriorityQueue

Important Basic Operations:

• Enqueue(x)
  – Put a new item x in the priority queue PQ (in some order)

• y \leftarrow Dequeue()
  – Return an item y that has the highest priority (key) in the PQ
  – If there are more than one item with highest priority, return the one that is inserted first (FIFO)

Note: We can always define highest priority = higher number or it’s opposite: highest priority = lower number
A Few Points To Remember

Data Structure (DS) is...

• A way to **store** and **organize data** in order to support efficient insertions, searches, deletions, queries, and/or updates

Most data structures have propert(ies)

• Each operation on that data structure has to **maintain** that propert(ies)
PriorityQueue Implementation (1)

(Circular) Array-Based Implementation (Strategy 1)

- Property: The content of array is always in correct order
- Enqueue(x)
  - Find the correct insertion point, O(n) – recall insertion sort
- y ← Dequeue()
  - Return the front-most item which has the highest priority, O(1)

<table>
<thead>
<tr>
<th>Index</th>
<th>0 (front)</th>
<th>1 (back)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Aircraft X*</td>
<td>Aircraft Y*</td>
</tr>
</tbody>
</table>

We do not need to close the gap, just advance the front pointer, O(1)
PriorityQueue Implementation (2)

(Circular) Array-Based Implementation (Strategy 2)

- Property: dequeue() operation returns the correct item
- Enqueue(x)
  - Put the new item at the back of the queue, O(1)
- y ← Dequeue()
  - Scan the whole queue, return first item with highest priority, O(n)

<table>
<thead>
<tr>
<th>Index</th>
<th>0 (front)</th>
<th>1 (back)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Aircraft X*</td>
<td>Aircraft Y*</td>
</tr>
</tbody>
</table>

We may need to close the gap if this operation causes it, also O(n)
PriorityQueue Implementation (3)

If we just stop at CS1020 knowledge level:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Enqueue</th>
<th>Dequeue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circular-Array-Based PQ (1)</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Circular-Array-Based PQ (2)</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Can we do better?</td>
<td>$O(?)$</td>
<td>$O(?)$</td>
</tr>
</tbody>
</table>

If $n$ is large, our queries are slow...
INTRODUCING BINARY HEAP DATA STRUCTURE

http://visualgo.net/heap.html
Complete Binary Tree

Introducing a few concepts:

• **Complete** Binary Tree
  – Binary tree in which every level, *except possibly the last*,
    is completely filled, and all nodes are as far left as possible

• Important Q:
  If you have a complete binary tree of $N$ items,
  what will be the **height of it**?
  – Height = number of levels - 1 =
    max edges from root to deepest leaf
The Height of a Complete Binary Tree of N Items is...

1. O(N)
2. O(sqrt(N))
3. O(log N)
4. O(1)

Memorize this answer! We will need that for nearly all time complexity analysis of binary heap operations
Storing a Complete Binary Tree

Q: Why not 0-based?

As a 1-based compact array: $A[1..\text{size}(A)]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NIL</td>
<td>90</td>
<td>19</td>
<td>36</td>
<td>17</td>
<td>3</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Navigation operations:

- $\text{parent}(i) = \lfloor i/2 \rfloor$, except for $i = 1$ (root)
- $\text{left}(i) = 2 \times i$, No left child when: $\text{left}(i) > \text{heapsize}$
- $\text{right}(i) = 2 \times i + 1$, No right child when: $\text{right}(i) > \text{heapsize}$
Binary Heap Property

**Binary Heap property** (except root)

- $A[parent(i)] \geq A[i]$ (**Max Heap**)
- $A[parent(i)] \leq A[i]$ (**Min Heap**)

Without loss of generality, we will use **(Binary Max) Heap** for all examples in this lecture and we ensure that the numbers are distinct.

Q: Can we write Binary Max Heap property as:
$A[i] \geq A[left(i)]$
$\&\&$
$A[i] \geq A[right(i)]$

?
The largest element in a **Binary Max Heap** is stored at...

1. One of the leaves
2. One of the internal vertices
3. Can be anywhere in the heap
4. The root
Insertion to an Existing B Max Heap

The most appropriate insertion point into an existing Binary Max Heap is at $A[\text{heapsize}]$

• Q: Why?
  A: ______________________________

• But Binary Max Heap property can still be violated?
  – No problem, we use $\text{ShiftUp}(i)$ to fix the heap property
Insert(v) – Pseudo Code

Insert(v)

heapsize = heapsize+1;  // extend, O(1)
A[heapsize] = v  // insert at the back, O(1)
ShiftUp(heapsize)  // fix the heap property
   // in O(?)

// Preliminary analysis:
// Insert(v) depends on ShiftUp(i)
ShiftUp – Pseudo Code

This name is **not unique**, the alternative names are: ShiftUp/BubbleUp/IncreaseKey/etc

```
ShiftUp(i)
  while i > 1 and A[parent(i)] < A[i] // don't swap
    swap(A[i], A[parent(i)])
    i = parent(i)
// Analysis: ShiftUp() runs in ______
```

“not root”  “violates max heap property”
Binary Heap: Insert(v)

Ask VisuAlgo to perform various insert operations on the sample Binary (Max) Heap

In the screen shot below, we show the first step of Insert(26)
Deleting Max Element (1)

The max element of a Binary Max Heap is at the root

• But simply taking the root out from a Binary Max Heap will disconnect the complete binary tree 😞
  – We do not want that...

Q: Which node is the best candidate to replace the root yet still maintain the complete binary tree property?
Deleting Max Element (2)

• A: The __________________________ leaf
  – Which is the last element in the compact array
• But the heap property can still be violated?
  – No problem, this time we call ShiftDown(1)
ExtractMax - Pseudocode

ExtractMax()

maxV $\leftarrow$ A[1] \hspace{1em} // O(1)
A[1] $\leftarrow$ A[heapsize] \hspace{1em} // O(1)
heapsize = heapsize-1 \hspace{1em} // O(1)
ShiftDown(1) \hspace{1em} // O(?)
return maxV

// Preliminary analysis:
// ExtractMax() depends on ShiftDown()
ShiftDown – Pseudo Code

ShiftDown(i)
    while i <= heapsize
        maxV ← A[i]; max_id ← i;
        if left(i) <= heapsize and maxV < A[left(i)]
            maxV ← A[left(i)]; max_id ← left(i)
        if right(i) <= heapsize and maxV < A[right(i)]
            maxV ← A[right(i)]; max_id ← right(i)
        // be careful with the implementation
        if (max_id != i)
            swap(A[i], A[max_id])
            i ← max_id;
    else
        break; // Analysis: ShiftDown() runs in ______
Binary Heap: ExtractMax()

Ask VisuAlgo to perform various ExtractMax() operations on the sample Binary (Max) Heap

In the screenshot below, we show the first step of ExtractMax() from the sample Binary (Max) Heap.
PriorityQueue Implementation (4)

Now, with knowledge of non linear DS from CS2010:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Enqueue</th>
<th>Dequeue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array-Based PQ (1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Array-Based PQ (2)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Binary-Heap (actually uses array too)</td>
<td>Insert(key)  O(log n)</td>
<td>ExtractMax() O(log n)</td>
</tr>
</tbody>
</table>

Summary so far:
Heap data structure is an efficient data structure -- $O(\log n)$ enqueue/dequeue operations -- to implement ADT priority queue where the ‘key’ represent the ‘priority’ of each item
Next Items:
• Building Binary Max Heap from an ordinary Array, the $O(n \log n)$ version
• And the faster $O(n)$ version
• Heap Sort, $O(n \log n)$
• Java Implementation of Binary Max Heap
• PS1 overview and introduction of one more Binary Max Heap operation: UpdateKey that has been purposely left out from this lecture
Review: We have seen MergeSort in CS1020. It can sort $n$ items in...

1. $O(n^2)$
2. $O(n \log n)$
3. $O(n)$
4. $O(\log n)$
HeapSort Pseudo Code

With a max heap, we can do sorting too 😊
- Just call ExtractMax() \( n \) times
- If we do not have a max heap yet, simply build one!

HeapSort(array)
    BuildHeap(array) // \( O(?) \)
    \( n \leftarrow \) size(array)
    for \( i \) from 1 to \( n \) // \( O(n) \)
        \( A[n-i+1] \leftarrow \) ExtractMax() // \( O(\log n) \)
    return \( A \)

// Preliminary analysis:
// HeapSort runs in \( O(?) + n \log n) \)
BuildHeap, $O(n \log n)$ Version

BuildHeapSlow(array) // naïve version

n $\leftarrow$ size(array)

A[0] $\leftarrow$ 0 // dummy entry

for $i = 1$ to $n$ // $O(n)$

Insert(array[$i-1$]) // $O(\log n)$

// Analysis: This clearly runs in $O(n \log n)$
// So HeapSort in previous slide is $O(n \log n)$ 😊

Can we do better?
Build Binary Heap in $O(n \log n)$

Ask VisuAlgo to build Binary (Max) Heap from an array in $O(n \log n)$ time by inserting each number one by one.

In the screen shot below, the **partial state** of the $O(n \log n)$ Build Heap of the sample Binary (Max) Heap.
BuildHeap, the Faster One

BuildHeap(array)

heapsize ← size(array)
A[0] ← 0 // dummy entry
for i = 1 to heapsize // copy the content O(n)
  A[i] ← array[i-1]
for i = parent(heapsize) down to 1 // O(n/2)
  ShiftDown(i) // O(log n)

// Analysis: Is this also O(n log n) ??
// No... soon, we will see that this is just O(n)
Build Binary Heap in O(n)

Ask VisuAlgo to build Binary (Max) Heap from an array in O(n) time by calling ShiftDown strategically.

In the screen shot below, the *partial state* of the O(n) Build Heap of the sample Binary (Max) Heap.
**BuildHeap() Analysis... (1)**

Recall: How many levels (height) are there in a complete binary tree (heap) of size \( n \)?

Recall: What is the cost to run \( \text{shiftDown}(i) \)?

Q: How many nodes are there at height \( h \) of a full binary tree?

- \( h = 3 \)
- \( h = 2 \)
- \( h = 1 \)
- \( h = 0 \)
BuildHeap() Analysis... (2)

Cost of BuildHeap() is thus:

\[\sum_{h=0}^{\lfloor \lg (n) \rfloor} \frac{n}{2^{h+1}} O(h) = \sum_{h=0}^{\lfloor \lg (n) \rfloor} \left( \frac{n}{2^{h+1}} \right) \cdot c \cdot h = O \left( n \sum_{h=0}^{\lfloor \lg (n) \rfloor} \frac{h}{2^h} \right) = O \left( n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) = O(2n) = O(n)\]

\[
\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} = 2 \quad \text{for} \quad x = 1/2
\]

\[
\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 \\
2^0 & \quad 2^1 & \quad 2^2 & \quad 2^3 & \quad 2^4 & \quad 1 & \quad 2 & \quad 4 & \quad 8 & \quad 16 \\
0 & + & 0.5 & + & 0.5 & + & 0.375 & + & 0.25 & + & 0.15625 & + & 0.09375 & + \ldots & < & 2
\end{align*}
\]
HeapSort Analysis

HeapSort(array)

BuildHeap(array)  // The best we can do is _______
n ← size(array)
for i from 1 to n  // O(n)
    A[n-i+1] ← ExtractMax()  // O(log n)
return A

// Analysis: Thus HeapSort runs in O(____________)
// Do you notice that we do not need extra array
// like merge sort to perform sorting?
// Thus heap sort is more memory friendly.
// This is called "in-place sorting"
// But HeapSort is not "cache friendly"
Binary Heap: HeapSort()

Ask VisuAlgo to run HeapSort() on the sample Binary (Max) Heap

In the screen shot below, the *partial state* of the O(n log n) HeapSort() of the sample Binary (Max) Heap
Java Implementation

Priority Queue ADT

Heap Class (Java file given, you *can use* it for PS1)

- ShiftUp(i)
- Insert(v)
- ShiftDown(i)
- ExtractMax()
- BuildHeapSlow(array) and BuildHeap(array)
- HeapSort()

In OOP Style 😊
Scheduling Deliveries, v2015 (PS1)

This happens in the delivery suite (or surgery room for Caesarean section) of a hospital.
PS1, the task

Given a list of ("insanely" many) pregnant women, prioritize the one who will give birth sooner over the one who will give birth later...

• Open on Wed, 19 Aug 2015, 11.45am, right after this lecture
• Clearly involving *some kind* of PriorityQueue 😊

PS1 Subtask A should be very easy
PS1 Subtask B may need Lab Demo 01 on Week 03
PS1 Subtask C is the challenge
• Introducing **UpdateKey** operation of a PriorityQueue
End of Lecture Quiz 😊

After Lecture 02, I will set a random test mode @ VisuAlgo to see if you understand Binary Heap

Go to:

http://visualgo.net/test.html

Use your CS2010 account to try the 5 Binary Heap questions (medium difficulty, 5 minutes)

Meanwhile, train first 😊

http://visualgo.net/training.html
Summary

In this lecture, we have looked at:

- Heap DS and its application as efficient PriorityQueue
- Storing heap as a compact array and its operations
  - Remember how we always try to maintain complete binary tree and heap property in all our operations!
- Building a heap from a set of numbers in $O(n)$ time
- Simple application of Heap DS: $O(n \log n)$ HeapSort

We will use PriorityQueue in the 2nd part of CS2010

- If some concepts are still unclear, ask your personal tutor: http://visualgo.net/heap.html