CSCI 136: Data Structures and Advanced Programming

Lecture 32

Heaps

Instructor: Dan Barowy

Williams

Your to-dos

1. Read before Wed: Review readings from Bailey.
2. Lab 10 (partner lab), due Tuesday 5/10 by 10pm.

Announcements

1. Senior thesis presentations in Wege auditorium:
   a. Monday, May 16, 10am-noon
   b. Monday, May 16, 1:30-3pm
2. Ward prize presentations for best class project in Wege auditorium:
   Tuesday, May 17, 2:30-4pm
Announcements

1. **Final exam**: Sunday, May 22, 9:30am in TPL 205.
2. Note that all of the practice quiz solutions are on the course website.

Announcements

1. **Student course surveys**, in class, **Wednesday, 5/11**.
   a. Please bring laptop/tablet to fill out survey.
2. **Final exam review session**, in class, **Friday 5/13**.

Practice Activity (+ cookies)

Activity: connectedness

```java
boolean connected():
How might I compute this using fundamental ops?
(adjacent, vertices, incident, degree, neighbors)
```

(note that graph is undirected)
Idea: breadth-first counting

Idea:
(suppose we know $|G|$)

```java
boolean isConnected(Vertex start)
1. let count = 0
2. let Q be an empty queue
3. enqueue start
4. while Q not empty
   a. dequeue v
   b. count v
   c. mark v as visited
   d. put v's unmarked neighbors in Q
5. if count = # of vertices in graph, return true else false
```

Algorithm: connectedness
initialize algorithm

![Algorithm diagram](image)
Algorithm: connectedness

mark \( v \)

count \( 1 \)

\( \mathcal{Q} \)

Algorithm: connectedness

enqueue unmarked neighbors

count \( 1 \)

\( \mathcal{Q} \)

dequeue \( v \)

count \( 1 \)

\( \mathcal{Q} \)

Algorithm: connectedness

count \( v \)

count \( 2 \)

\( \mathcal{Q} \)
Algorithm: connectedness
mark v

Algorithm: connectedness
tenque unmarked neighbors

count 2
Q

Algorithm: connectedness
dequeue v
count 2
deque v
Q

Algorithm: connectedness
count 3
Algorithm: connectedness
mark v

Algorithm: connectedness
tenqueue unmarked neighbors

Algorithm: connectedness
dequeue v

Algorithm: connectedness
count v
Algorithm: connectedness

mark v

count
Q

enqueue unmarked neighbors

count
Q

Algorithm: connectedness
dequeue v

count
Q

count v

count
Q
Algorithm: connectedness

mark v

A priority queue is an abstract data type that returns the elements in priority order. Under priority ordering, an element e with a higher priority (an integer) is returned before all elements L having lower priority, even if that e was enqueued after all L. When any two elements have equal priority, they are returned in first-in, first-out order (i.e., in the order in which they were enqueued).
I will refer here to the **maximum** priority. But you could also refer to **minimum** priority. All that matters is that you order your data with respect to some **extremum**.
Priority Queue

blue letters: enqueue

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Ordinary letter    Blue letter

Priority Queue: Operations

**insert**: inserts an element with a given priority value. Ensures that the next element of the queue is in priority order. Like *enqueue*.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Priority Queue: Operations

**find-max**: returns the next element with a highest priority value. Like *peek*, does not modify the queue.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Priority Queue: Operations

**extract**: removes and returns the next element with a maximum priority value. Like **dequeue**.

Priority Queue

How to implement?

**Vector**:  
- **find-max**: $O(1)$  
- **insert**: $O(n)$  
- **extract**: $O(n)$

**Heap**:  
- **find-max**: $O(1)$  
- **insert**: $O(\log n)$  
- **extract**: $O(\log n)$

Priority Queue

Is it **necessary** to keep the **entire queue** in sorted order?

Operations:

- **find-max**
- **insert**
- **extract**

Heaps
Max Heap

A max heap is a tree-based data structure that returns its elements in priority order. A heap maintains the max heap property: for any given node $n$, if $p$ is a parent node of $n$, then the key of $p$ is $\geq$ to the key of $n$.

A max heap is a tree whose root is the maximum element and whose subtrees are, themselves, heaps.

Is this a binary search tree?

No. Nodes do not obey binary search property.

(Binary) max heap

Max heap property: for any given node $n$, if $p$ is a parent node of $n$, then the key of $p$ is $\geq$ the key of $n$.

Insertion

A binary heap is usually implemented as an always-complete binary tree.
Suppose we want to insert a new node, 78.

First, insert the new node at the first available position in the tree that maintains completeness.

Next, compare the new node with its parent.

If the max heap property is violated, swap.
Insertion

Continue swapping the new node with parents until the max heap property is satisfied.

Insertion

Continue swapping the new node with parents until the max heap property is satisfied (parent ≥ node or no parents remain).

Insertion

The swapping procedure performed on insert is often referred to as heap-up or percolate-up.

Find-max

To find the maximum element in a max heap, simply return the root.
To remove and return the maximum element in a max heap, first perform `find-max`.

Temporarily store the max element.

Replace the root with the last element in the complete tree.
Extract the root with its children. Swap the root with the largest element.

\[ \begin{array}{c}
\text{Extract} \\
23 \\
3 \\
1 \\
0 \\
-1 \\
42 \\
78
\end{array} \]

23 \geq 42? No.

Extract

\[ \begin{array}{c}
\text{Extract} \\
42 \\
3 \\
1 \\
0 \\
-1 \\
23
\end{array} \]

23 \geq 42? No.

Continue swapping until the max heap property is satisfied (parent \geq node or no parents remain).

\[ \begin{array}{c}
\text{Extract} \\
42 \\
3 \\
1 \\
0 \\
-1 \\
23 \\
78
\end{array} \]

23 \geq -1? Yes.

Return the saved maximum element.
The swapping procedure performed on extract is often referred to as heap-down or percolate-down.

Recap & Next Class

Today:

- Priority queues
- Heaps

Next class:

- Dijkstra’s algorithm