Lecture 5: Data Structures II

COMP526: Efficient Algorithms

Will Rosenbaum
University of Liverpool

Updated: October 17, 2024

Announcements

1. Second Quiz Open, due Friday
   - Similar format to before
   - Covers asymptotic (Big-O) notation
   - Quiz is **closed resource**
     - No books, notes, internet, etc.
     - Do not discuss until after submission deadline (Friday night, after midnight)
2. CampusWire
   - Use for discussion of material, questions about lectures, etc
   - Public comments for matters related to module content & administration
   - https://campuswire.com/p/GBB00CD7A, Code: 4796
3. Attendance Code:

Meeting Goals

- Introduce Programming Assignment 1: Prefix Reversal Sorting
- Discuss the Queue ADT and implementations
- Introduce the Priority Queue ADT
- Introduce the heap data structure

Programming Assignment 1

Non-Standard Sorting

Fundamental Task: sorting a list of elements from smallest to largest

7	1	2	5	3	4	8	6

$\longmapsto$

1	2	3	4	5	6	7	8

Typical basic (unit cost) operations:

- **compare** two elements to see which is larger
- **swap** two elements in the array

Non-standard sorting models:

- natural in contexts other than sorting arrays
- e.g., sorting physical objects with physical constraints
- compare and swap may not be elementary operations

Credit: Andy Goldsworthy

Sorting with Prefix Reversals

Basic Operation: Prefix Reversal

- Reverse the elements up to index i in a list/array
- For example

$$\begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 3 & 4 & 5 \\ \hline \end{array} \xmapsto{2} \begin{array}{|c|c|c|c|c|}\hline 3 & 2 & 1 & 4 & 5 \\ \hline \end{array}$$

- Natural operation for
  - DNA
  - stacks of physical objects

Basic Algorithmic Question: Given an array a of length n, what is the fewest number of **prefix reversal** operations necessary to sort a?

Your Task

Input: an array (`list`) *a* of numbers between 1 and *n*.
Output: the array *p* of *prefix reversals* that when applied to *a* will result in a sorted array.
Goal: sort each array *a* using the fewest possible prefix reversals.
Example: Sort `[4, 1, 3, 2]`

PollEverywhere Question

Starting from the array `[4, 3, 5, 6, 1, 2]` what is the resulting array after performing the following prefix reversals? `[2, 5, 3]`

`pollev.com/comp526`

More Specifically

Array Structures for input

1. **random permutation** a is uniformly random shuffling of numbers from 1 to n
2. **tritonic** for $0 < a < b < n$ the values of a are *increasing* from indices 0 to a, *decreasing from* indices a to b, then *increasing* from indices b to $n-1$.
3. **binary** a's values are all 0 or 1
4. **ternary** a's values are all 0, 1, or 2

For each structure you will define a function that generates a prefix reversal sequence that sorts arrays with the given structure.

Scoring:

- your program must correctly sort all arrays
- points for minimizing the number of prefix reversals over all challenge arrays

Suggestion: it is possible to sort any array of length n with fewer than $2n$ prefix reversals

- start by implementing a simple baseline procedure to sort all arrays

Queues

From Last Time

- Stack ADT
  - **linked list** implementation
  - **array** implementation
- Amortized analysis

The Queue ADT

Queues, Intuitively

Goal: to store a *collection* of elements

- elements arranged as in a queue at Tesco
- new people enter the **back** of the queue
- only the person at the **front** of the queue can be removed (serviced)

First In, First Out (FIFO) priority

Tons of Applications!

- Scheduling
- Messaging
- …

Queues, Formally

- S is the state of the queue, initially $S = \varnothing$
- $S.\textsc{Enqueue}(x) : S \mapsto xS$
- $S.\textsc{Front}()$: returns x_{n-1} where $S = x_0 x_1 \cdots x_{n-1}$
- $S.\textsc{Dequeue}() : Sx \mapsto S$, returns x
- $S.\textsc{Empty}()$ returns $\textsc{True} \iff S = \varnothing$

List Backed Queues

Idea

- Store each element in a Node
- Store references to Node:
  - head at the front of the queue
  - tail at the back of the queue

Issues:

- Similar to linked list stack implementation
  - Locality of reference
  - Node memory overhead

```
class ListQueue
    Node head
    Node tail
    procedure Enqueue(x)
        n ← new Node
        n.data ← x
        tail.next ← n
        tail ← n
    end procedure
    procedure Dequeue
        n ← head
        head ← n.next
        return n.data
    end procedure
end class
```

Array Backed Queues

Idea:

- Store elements in the stack in an array
- Maintain indices of head and tail

What is the problem here?

- What happens if we repeatedly call
  - ENQUEUE(1)
  - DEQUEUE()
  - ENQUEUE(1)
  - DEQUEUE()
  - …

Ignores resizing/checking if full

```
class ArrayQueue
    a ← new array, size n
    head, tail ← 0
    procedure Enqueue(x)
        a[tail] ← x
        tail ← tail + 1
    end procedure
    procedure Dequque
        head ← head + 1
        return a[head − 1]
    end procedure
end class
```

Priority Queues

The (Min) Priority Queue ADT

Priority Queues, Intuitively

Goal: to store a *collection* of elements

- Each element x has an associated *priority*, $p(x)$
- New elements **inserted** with prescribed priorities
- Can access/remove element with the *minimum* priority in the collection

Applications

- efficient sorting
- implementing "greedy" algorithms
- resource management

Priority Queues, Formally

- S is the state of the queue, initially $S = \varnothing$
- $S.\textsc{Insert}(x, p(x)) : S = x_0 x_1 \cdots x_i x_{i+1} \cdots x_{n-1} \mapsto x_0 x_1 \cdots x_i \, x \, x_{i+1} \cdots x_{n-1}$
  - where $p(x_i) \leq p(x) < p(x_{i+1})$
- $S.\textsc{Min}() :$ returns x_0 where $S = x_0 x_1 \cdots x_{n-1}$
- $S.\textsc{RemoveMin}() : xS \mapsto S$, returns x

Naive Priority Queue Implementations

Array backed implementation

- Store element/priority pairs, sorted by priority
- MIN and REMOVEMIN can be implemented in $O(1)$ time
- INSERT is $\Theta(n)$ worst-case
  - must **shift** elements around x

Linked list backed implementation

- Store element/priority pairs, sorted by priority
- MIN and REMOVEMIN can be implemented in $O(1)$ time
- INSERT is $\Theta(n)$ worst-case
  - must **find** location to insert x

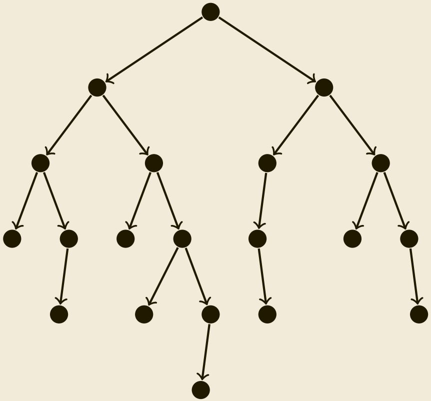
Question. Can we perform *all* operations in $o(n)$ time?

Heaps

Binary Trees

A (rooted) **binary tree** consists of

- A set V of vertices
- A distinguished vertex called the **root**
- Each vertex has (possibly empty):
  - left child
  - right child
- Non-root vertices have a parent
- All vertices are descendants of the root

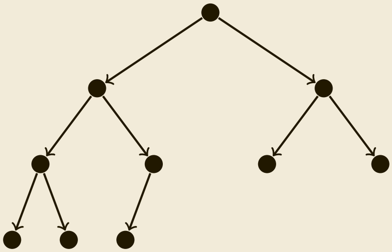


- Vertices without children are **leaves**
- The **height** is the maximal distance from root to a leaf

Complete Binary Trees

A **complete binary tree** of height h is a binary tree in which:

- All vertices up to depth $h-2$ have exactly two children
- At most one vertex at depth $h-1$ has one child
  - if v has children, then vertices to the left of v have two children



Properties of Complete Binary Trees

Proposition

Suppose T is a complete binary tree of height h. Then the number n of vertices of T satisfies $2^h \leq n \leq 2^{h+1} - 1$.

PollEverywhere Question

If a complete binary tree T has n vertices, what is its height?

1. $h = \Theta(1)$
2. $h = \Theta(\log n)$
3. $h = \Theta(\sqrt{n})$
4. $h = \Theta(n)$

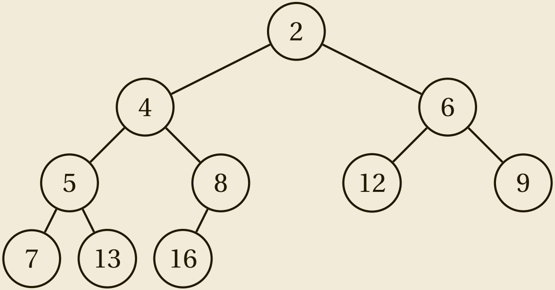
pollev.com/comp526

Min Heaps

Definition

A **heap** is a complete binary tree T with the following properties:

- each vertex (node) has an associated value from an *ordered set*
  - for each pair of values p and q we can *compare* $p < q$
- the value associated with each vertex v is *smaller* than the values associated with its children



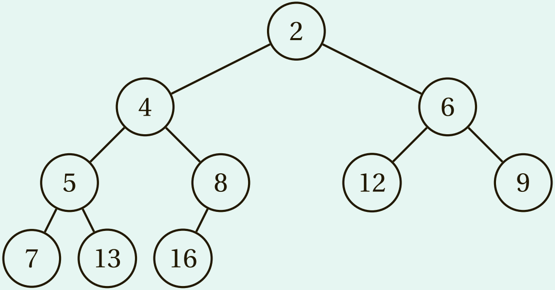
Inserting Into a Heap

Question

Given a heap T, how can we *efficiently* **insert a new a value** into T and maintain the heap properties?

Example

How to insert the value 3?



"Bubble Up" Insert Procedure

procedure INSERT(p)
  $v \leftarrow$ new vertex storing p
  $u \leftarrow$ first vtx with < 2 children
  add v as u's child
  PARENT$(v) \leftarrow u$
  while value$(v) <$ value(u) and $u \neq \perp$ **do**
    SWAP(value(v), value(u))
    $v \leftarrow u$
    $u \leftarrow$ PARENT(v)
  end while
end procedure

PollEverywhere Question

What is the running time of INSERT if T has n vertices?

1. $\Theta(1)$

3. $\Theta(\sqrt{n})$
4. $\Theta(n)$

pollev.com/comp526

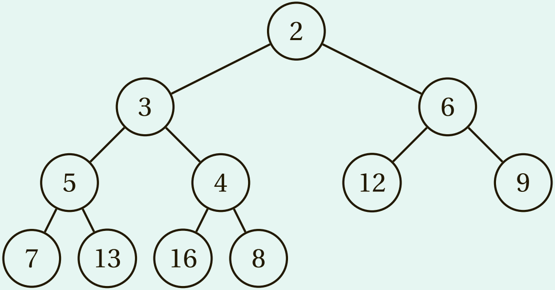
Removing Min

Question

Given a heap T, how can we *efficiently* **remove the minimum value** from T and maintain the heap properties?

Example

How to remove 2?



“Trickle Down” Remove Min Procedure

```
procedure RemoveMin
    v ← tree root
    w ← “last” vertex in tree
    value(v) ← value(w)
    remove w from tree
    u ← smaller of v’s children
    while value(v) ≥ value(u) and u ≠ ⊥ do
        Swap(value(v), value(u))
        v ← u
        u ← v’s smaller child
    end while
end procedure
```

PollEverywhere Question

What is the running time of RemoveMin if T has n vertices?

1. $\Theta(1)$
2. $\Theta(\log n)$
3. $\Theta(\sqrt{n})$
4. $\Theta(n)$

pollev.com/comp526

Heap Data Structures?

Question

What *elementary* data structures can we use to represent heaps?

- Our tree representation was somewhat vague…

Natural Choice: Tree of Nodes

- Have a NODE data structure where NODE stores:
  - data (value)
  - reference to PARENT
  - references to LEFTCHILD and RIGHTCHILD
- Similar drawbacks to linked lists: data overhead, locality of reference

Less Obvious Choice: Arrays!

- **For next time:** think about how you could represent a heap using an **array** and minimal additional overhead!

Heap Priority Queues

So far we heaps only store values from a ordered set

- A priority queue needs two data fields:
  1. a value x
  2. a priority $p(x)$
- The *priorities* are from an ordered set
- To implement a priority queue with a heap
  - each vertex v stores (refers to) x and $p(x)$
  - the value value(v) is the priority $p(x)$

The payoff

- INSERT in time $\Theta(\log n)$
- REMOVEMIN in time $\Theta(\log n)$
- MIN in time $\Theta(1)$

Next Time: More Trees!

- Searching Sorted Arrays
- Binary Search Trees
- Balanced Binary Trees

Scratch Notes