CSCI 136
Data Structures &
Advanced Programming

Bill Jannen

Lecture 26

April 19, 2017

Administrative Details

- Super Lexicon lab today
  - May work with a partner
  - But must work ***with*** your partner
    - Attend same lab section
    - "Pair program" in the lab (or elsewhere)
- Posted hints to get you started
- Tools to help you test
  - Main.java
  - small.txt, small2.txt, ospd2.txt

Last Time

- Huffman Codes (AN ANTARCTIC PENGUIN)

- Briefly talked about how to represent a tree using an array (or vector/list)

Today's Outline

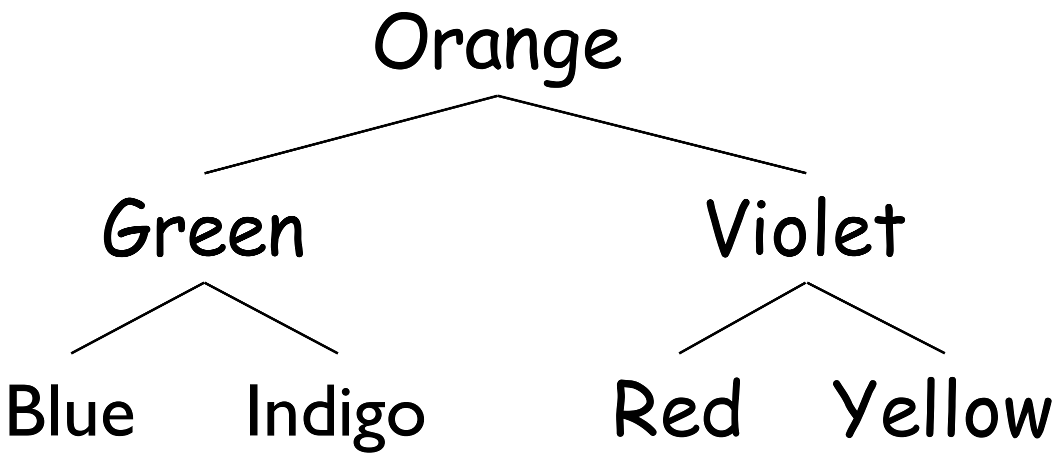
- Finish binary-trees-as-arrays discussion
- Discuss priority queues
- (maybe) Introduce heaps

Using Arrays to Store Trees

- Implicitly encode tree structure using indexes:
  - Consider a **full** tree
  - Index nodes as in level-order traversal

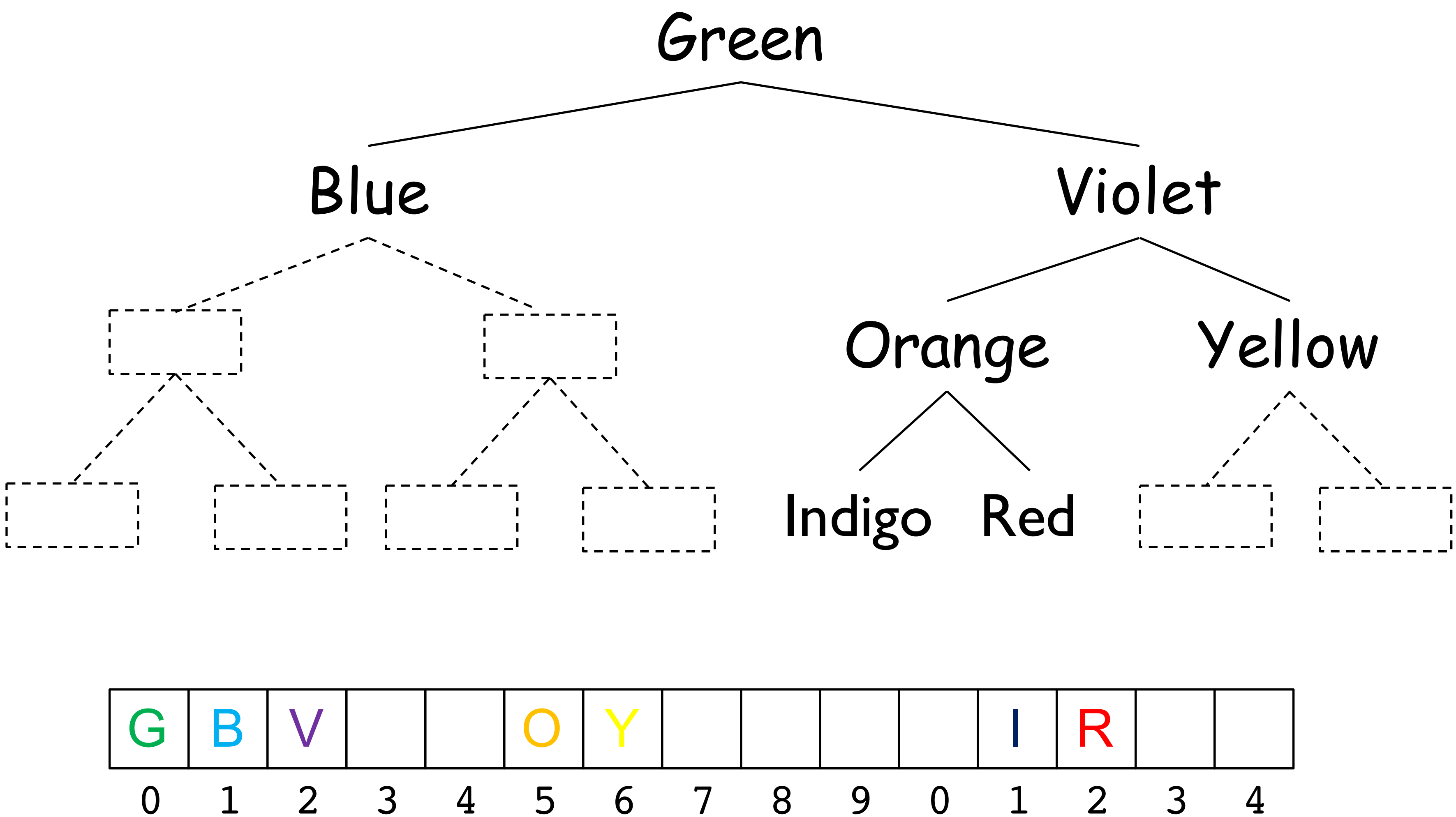
- Instead of pointers, use math to walk the tree
  - Children of node i are at 2i+1 and 2i+2
  - Parent of node j is at (j-1)/2

Example



O	G	V	B	I	R	Y
0	1	2	3	4	5	6

Same Contents, Different Tree



Cost of Imbalance

- Possible nodes in level *i* of a binary tree?
  - 2^i

- For a tree with *n* elements…

	Height	**Total Array Elements**
Full Tree:	$\log_2(n)$	n
“Degenerate” Tree:	n	$2^{n+1}-1$

ArrayTree Tradeoffs

- Why are ArrayTrees good?
  - Save space for links (no “slots” needed)
    - Relationships between values are implicitly stored (index + math)
  - Works well for complete trees
    - “A *complete* binary tree of height h is a full binary tree with 0 or more of the rightmost leaves of level h removed”
- Why bad?
  - Could waste a lot of space (sparse trees)
  - Height of n requires $2^{n+1}-1$ array slots even if only O(n) elements

Open Question: What Does it Mean to be "Fair"?

- How are people "served" in:
  - Cafeterias

A Queue

Multiple Queues

  - Airplanes
  - Emergency room

Priority Queue

Priority Queues

- Name is misleading
- PQs are a bit like normal queues, except they are **not FIFO**
- Always dequeue object with **highest priority** regardless of when it was enqueued
- Data can be received/inserted in any order, but it is always returned/removed in same order (according to priority)

Priority Queues vs. Ordered Structures

- Like ordered structures (i.e., OrderedVectors and OrderedLists), PQs *appear* to keep data in order
  - What did we gain from ordered structures?
    - Search cost
  - What is the cost of maintaining order?
    - Insert cost

- Unlike ordered structures, PQs allow the user only to remove its "smallest/best" element
  - Can't search, no random access

Priority Queues vs. Linear Structures

- PQs are also similar to Linear structures (i.e., stacks and queues):
  - values are added to the structure one at a time
  - may be inspected or removed one at a time
- Unlike Linear structures, not LIFO or FIFO
  - Always removed the minimum value (i.e., value with highest priority)

Priority Queue Uses

- Priority queues are used for:
  - Scheduling processes in an operating system
    - Priority is function of time lost + process priority
  - Order services on server
    - low priority tasks shouldn't interfere with high priority tasks
      - Backup, virus scanning, certain updates
  - Medical waiting room
  - Huffman codes - order by tree size/weight

  - To generally rank choices that are generated out of order

PQ Interface

```
public interface PriorityQueue<E extends Comparable<E>> {
   public E getFirst();
   public E remove();
   public void add(E value);
   public boolean isEmpty();
   public int size();
   public void clear();
}
```

Non-destructive

Do not specify location, priority

Things to Note about PQ Interface

- Unlike previous structures, we do not extend any other interfaces
- PriorityQueue methods *consume* Comparable parameters and *return* Comparable values
- Possibilities besides using Comparables?
  - Comparators

Implementing PQs

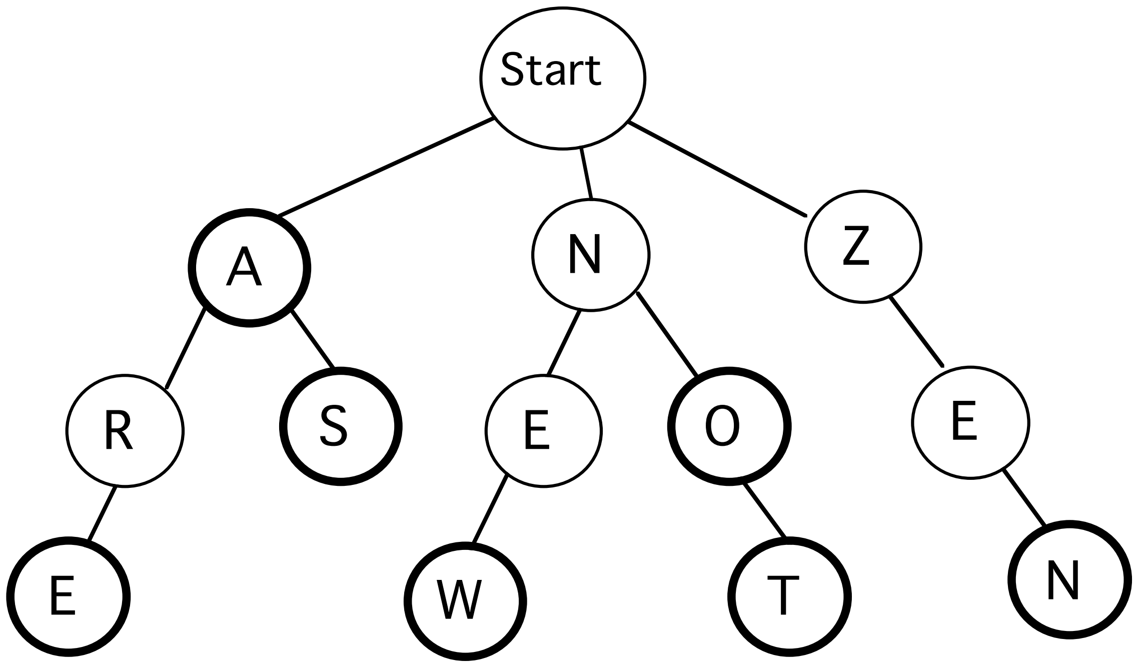
- Queue?
  - Wouldn't work so well because we can't insert and remove in the "right" way (i.e., keeping things ordered)
- OrderedVector?
  - Keep ordered vector of objects
  - O(n) to add/remove from vector
  - Details in book…
  - Can we do better than O(n)?
- Heap?
  - Partially ordered binary tree

Heap

- A heap is a **complete** binary tree where:
  - Root holds smallest (highest priority) value
  - Left and right subtrees are also heaps (this is important!)
- Any path from root to leaf is in descending order
- Invariant for nodes
  - node.value() <= node.left.value()
  - node.value() <=node.right.value()
- Several valid heaps for same data set (no unique representation)

Says nothing about sibling relationships!

Tries



Nodes:

- letter
- isWord

What are the words represented in this trie?

Leaf node: isWord must be true

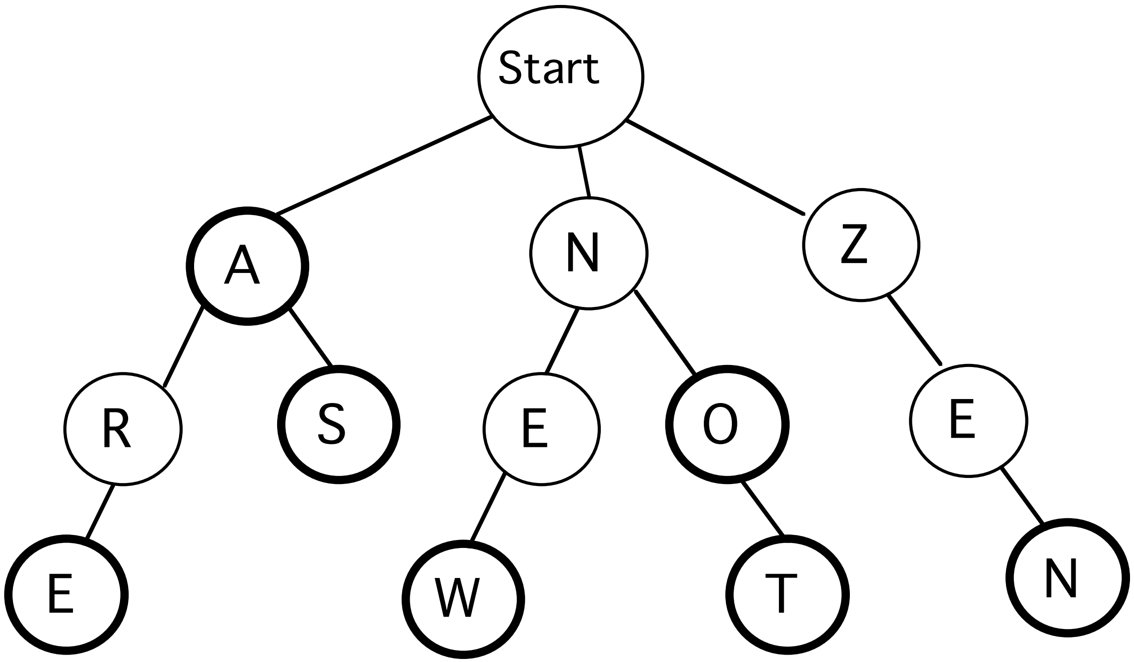
Representing Tries

- Not a binary tree… how to store children?
  - Options: an array of characters, a Vector, an OrderedStructure
    - Maximum number of children for any node?
  - If you have to scan 26 elements to find a child, how does this affect the Big-O cost of walking from root to leaf?
  - Why might it still be important to keep the children sorted?

Regular Expressions (Sort of…)

- The '*' wildcard character matches any sequence of zero or more characters.
- The '?' wildcard character matches either zero or one character

Regular Expressions (Sort of...)



What word(s) match **`*T`** ?

What word(s) match **`*E*`** ?

What word(s) match **`?S`** ?

Sets

- Store unique elements (ignore duplicates)
- Useful for checking membership *quickly*
- Giving the data structures we have covered, what would be an appropriate choice?
  - In reality, probably use *hashing*