CSE 373

MAY 15TH – ITERATORS
ASSORTED MINUTIAE

• HW4 feedback delayed
  • HW5 code now due Friday
• Extra assignment out tomorrow
• Final Exam – Tue Jun 6, 2:30-4:20 SMI 120
  • If you cannot make this exam, I need to know by the end of the week
  • Make up exam will be offered either late on the last Friday or during the day Saturday
EXTRA ASSIGNMENT

- Out Wednesday
- AVL implementation
  - Will replace lowest grade, for any HW assignment part
  - Up to 50 points possible, so can earn up to 25 points of EC
    - 25 points for correct implementation of AVL
    - 10 points for implementing a BFSIterator
    - 15 points for writing an AVL test suite
EXTRA ASSIGNMENT

• Alternatively, you may complete the write-up assignment
  • Write up will be a 3-5 page write up about splay trees
  • I expect consideration of runtime, memory, design and implementation as well as an understanding of “amortized analysis”
  • Points for this write up are capped at 25 points
EXTRA ASSIGNMENT

• You may only complete one of the two assignments

• Due: Friday June 2\textsuperscript{nd} (the last day of class) at midnight

• No late submissions will be allowed
  • The assignment will close at 12:30 am, but any submissions turned in after midnight will be accepted at my discretion – 12:01 is likely to be okay, but 12:29 will not be.
ITERATORS

• An iterator is a Java object that goes over collection of data
  • Supports two functions
    • boolean hasNext(): returns true if the iterator has another object
    • E next(): returns the next object from the data structure
      • “E” is a Java generic and it represents whatever data is actually in the data structure.
ITERATORS

• What is “next”?
  • Depends on how we want to iterate through the graph
  • Does not have to be a complete traversal
  • Examples:
    • BFSIterator
    • PathIterator
    • DuplicatelIterator
    • SortedIterator
ITERATORS

• These may have different runtimes, depending on how long it takes to find the next object

• Example, let’s consider an iterator which finds all people with the same first name in an unsorted linked list.
  • Suppose that the data is a First Name, Last Name object
  • What does the iterator need to keep track of?
    • Which element it’s on
    • What first name it’s looking for
ITERATORS

• **What happens at each call of next()?**
  - Think of the iterator as a cursor
  - Right now, the cursor can only move forward (since it’s a singly-linked list)
  - Go forward, checking each node until you find the next object with the searching name
  - Since the iterator is an object itself, it keeps track of all this information in between calls—*separate from the data structure!*
  - But, the iterator can access private nodes of the data structure and provide new orderings for the client
  - Linked lists are somewhat simple, but what about a more general case?
ITERATORS

• Graph iterators
  • How do we implement a BFS iterator?
    • Need to maintain the queue
    • Keep track of visited nodes
    • At each call of “next()” we return the next item in the queue and process its children
  • Same approach as the traversal, but iterators can terminate early and do not have to traverse the whole graph
ITERATORS

• What about a path iterator?

• Given two connected vertices in a graph, provide an iterator that returns all vertices (including start and end) on the shortest path between them.

• How do we do this?
  • Dijkstra’s! Do we need to run the whole algorithm at once? Yes! You don’t know what the first edge is until you know the whole path.
  • The iterator then just returns the path one vertex at a time.
ITERATORS

• Why iterators?
  • Iterators allow chained related finds within a data set
  • FindNextPrime requires keeping track of which primes have been returned AND of finding which numbers are prime
  • Moves through data in some well known an organized manner
  • How would a BFSIterator be useful for testing AVL trees?
TESTING

• AVL Trees
  • What is an AVL Tree?
    • Dictionary
    • Binary Search Tree
    • Balanced
  • Just using insert() and find(), how can you tell the difference between an AVLTree and a BSTree?
    • Insert sorted data and time the difference
    • What if you had a BFSIterator?
TESTING

• If you can provide the BFSIterator, you can verify that the AVL tree has the correct balanced shape.

• If you do the extra credit for testing AVL trees, an iterator is a great tool for verifying the shape.

• Anything that returns a BFS traversal of the tree, however, can help observe differences.

• This is not pure black box testing, it requires the DS to support the iterator to allow the testing.
ITERATORS

• How do we analyze these?
• These may have different runtimes, depending on how long it takes to find the next object
• Example, let’s consider a SortedIterator over an unsorted array.
  • What are some approaches that we might use here?
    • We could just sort the array
      • Does all the work in advance
    • Traverse the whole array to find the next element
ITERATORS

• How might this be problematic?
  • Just keeping track of the “current element”

• Consider this example
  • On first call, we iterate and find the lowest element (-3)
  • On the second call, remembering that our last was -3, we iterate to find the lowest element again and find (1)
  • What happens on later calls?
    • Either it skips over 1, because it thinks it’s done it before, or it repeats 1 because it doesn’t know how many one’s it has found

| 5 | 1 | 7 | 9 | 4 | 1 | 8 | -3 |
ITERATORS

• Solutions?
  • Sorting the list in advance is still an option
  • Iterators have a benefit of partial work
  • We can keep track of which elements we have seen
  • Must build a collection of returned items within the iterator
  • Or we can use a way to indicate that we may still be searching for duplicates

• Not trivial to implement

• What is the runtime of each call to next? O(N)
SORTING

• What then is the total time to return the complete set of sorted data?
  • N pieces of data at $O(N)$ times
  • $O(N^2)$

• This is one of the slow sorts
  • This method is called Selection sort
  • We search through the whole list and “select” the next smallest element
SELECTION SORT

• What are some benefits of this sorting technique?
  • Can be interrupted (don’t need to sort the whole array to get the first element)
  • Doesn’t need to mutate the original array (if the array has some other sorted order)
  • Preserves the other order if it does exist
    • This is called a stable sort
SELECTION SORT

• What are some downsides of this sort?
  • $O(N^2)$
  • Must look through all elements each time
  • If done as an iterator, it requires extra memory in order to implement
  • If we don’t care about the original array, can we perform a selection sort without extra memory?
    • If a sort only needs a constant amount of memory to operate, it is called an \textit{in-place sort}
  • How do we perform an in-place selection sort?
    • Swap the lowest item with the element at the beginning of the array
SELECTION SORT

- Swapping Selection
  - We iterate through the list to find the lowest element
  - When we find it (-3), which do we swap with?
  - When we go to find the next element, what do we do?
  - Must search through the entire array, even though the next element is in the correct place at the start
  - Which 1 do we select? The first one otherwise the sort is not stable

-3 1 7 9 4 1 8 5
NEXT CLASS

• “Cool” graph problems
• New Analysis Trick
  • Recurrences