Lecture 26: El Grandé Finalé

✦ Agenda for the final class:
  ➤ A Taste of Amortization (Chapter 11)
    ✦ Aggregate method
    ✦ Potential Method
    ✦ Covered in Section 11.2 in the textbook
  ➤ Final Review
    ✦ Summary of what you’ve learned in this course

A Midterm Problem from Section BA/BB

✦ Consider the class **BigNum** with the method:
  ```
  void addOne(); /* add one to the big number */
  ```

✦ Implements a big number as a list of K binary digits
  Each list cell contains a 0 or a 1
  E.g. K = 10
  E.g. 4 is represented as the list: 0000000100
  E.g. 7 is represented as the list: 0000000111

✦ What does the run time for a given `addOne()` operation depend on?

✦ Run time depends on number of bits affected by the operation
A Midterm Problem from Section BA/BB

✦ Example:
   If the number is 0010101111
   After \texttt{addOne()}, number becomes: 0010110000
   Run time \( T = 5 \) because five bits were changed

✦ Question:
   Start with the number 0
   Consider a sequence of \( N \) \texttt{addOne()} operations
   What is the amortized running time as a function of \( N \)?

Tackling the \textbf{BigNum}: Naïve Strategy

✦ Question:
   Start with the number 0
   Consider a sequence of \( N \) \texttt{addOne()} operations
   What is the amortized running time as a function of \( N \)?

✦ Worst case:
   ✦ \( K \) binary digits of which \( K-1 \) could be 1’s
   ✦ \texttt{addOne()} flips all \( K-1 \) bits and changes \( K \)th bit to a 1
   ✦ Worst case run time = \( K \)
   ✦ For \( N \) operations, \textbf{Amortized run time} = \( O(KN) \)
   ✦ Amortized run time per operation = \( O(KN)/N = O(K) \)
   ✦ Is this a good bound?
Tackling the **BigNum**: A Better Strategy

- Amortized run time of $O(Nk)$ is not a “tight” bound
  - Worst case occurs very rarely (not at all if $N < k$)
  - Worst case assumes every bit changes at every operation
- Can get a better bound by looking at how many times each bit can change during $N$ `addOne()` operations

```
0000000000 
0000000001 
0000000010 
0000000011 
0000000100 
0000000101 
0000000110 
0000000111 
0000001000 
```

0th bit changes for every operation
1st bit changes for every 2nd operation
2nd bit changes for every 4th operation
3rd bit changes for every 8th operation
….

Every bit (change) counts

- For a sequence of $N$ `addOne()` operations
  - 0th bit changes $N$ times
  - 1st bit changes $N/2$ times
  - 2nd bit changes $N/4$ times
  - 3rd bit changes $N/8$ times
  - $i$th bit changes $N/2^i$ times
  - How big can $i$ get for $N$ `addOne()` operations?
    - It takes $\log N$ bits to represent the value $N$, max $i = \log N$
- Total number of bit changes for $N$ `addOne()` operations =

$$
\sum_{i=0}^{\log N} N/2^i < N \sum_{i=0}^{\infty} 1/2^i = 2N
$$
Amortized Analysis: The Aggregate Method

- **Amortized Run Time** \( T(N) \) for a sequence of \( N \) `addOne()` operations = \( O(\text{total number of bit changes}) = O(N) \)
  - Much better than the naïve bound on \( O(KN) \)
- **Amortized Run Time per operation** = \( O(N)/N = O(1) \)
  - Much better than the naïve bound of \( O(K) \)
- This is the **aggregate method** for amortized analysis
  - Basic Idea:
    - Calculate best possible big-oh bound \( T(N) \) on total run time of \( N \) operations (using a brute-force method)
    - Amortized run time per operation = \( T(N)/N \)

Amortized Analysis: The Potential Method

- Inspired by the concept of “potential energy” in physics
- Initial data structure \( D_0 \) on which \( N \) operations are performed
- We get \( D_i \) after applying \( i \)th operation to \( D_{i-1} \) at a cost of \( c_i \)
  - \( c_i \) is the run time of the \( i \)th operation
  - We do not know \( c_i \) but would like to put an upper bound on it
- Suppose we can come up with a “potential function” \( \Phi \) that maps each \( D_i \) to a real number \( \Phi(D_i) \)
- Our goal: Use “potential function” \( \Phi \) to put an upper bound on the total amortized cost of \( N \) operations
Amortized Analysis: The Potential Method

Define the amortized cost of $i$th operation as:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Total amortized cost for $N$ operations:

$$\sum_{i=1}^{N} \hat{c}_i = \sum_{i=1}^{N} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{N} c_i + \Phi(D_N) - \Phi(D_0)$$

What is the relation between this cost and the total run time for $N$ operations?

Amortized Analysis: The Potential Method

Total amortized cost for $N$ operations:

$$\sum_{i=1}^{N} \hat{c}_i = \sum_{i=1}^{N} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{N} c_i + \Phi(D_N) - \Phi(D_0)$$

Then, if $\Phi(D_N) \geq \Phi(D_0)$ (or better, if $\Phi(D_i) \geq \Phi(D_0)$ for all $i$), then total run time for $N$ operations:

$$T(N) = \sum_{i=1}^{N} c_i = O \left( \sum_{i=1}^{N} \hat{c}_i \right)$$
Back to **BigNum**: The Potential Method

- What should the “potential function” \( \Phi \) be on the List \( D \) of binary digits?
  - Choice of \( \Phi \) is somewhat of an art
  - Many choices may exist
  - But all should obey \( \Phi(D_i) \geq \Phi(D_0) \) for all \( i \)
  - Some may give better bounds than others

Think of what changes after each \texttt{addOne()} operation

- What about \( \Phi(D_i) = n_i = \text{number of 1's in the List after } i \text{th operation?} \)
  - \( \Phi(D_i) \geq \Phi(D_0) \) for all \( i \)
  - \( \Phi(D_i) \) changes after every operation

\textbf{BigNum}: Grinding out its Potential

- \( \Phi(D_i) = n_i = \text{number of 1's in the List after } i \text{th operation} \)
  - \( \Phi(D_i) \geq \Phi(D_0) \) for all \( i \)

Suppose \( i \text{th operation changes } x_i \text{'s to 0's} \)

- Cost \( c_i \) of \( i \text{th operation} = x_i + 1 \) (to change last 0 to 1)

- \( \Phi(D_i) = \text{Number of 1's after the } i \text{th operation} = n_{i-1} - x_i + 1 \)

Amortized cost for \( i \text{th operation} = \)

\[
\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})
\]

\[
= (x_i + 1) + (n_{i-1} - x_i + 1) - n_{i-1} = 2
\]

- Total amortized run time = \( O\left(\sum_{i=1}^{N} \hat{c}_i\right) = O(2N) = O(N) \)

R. Rao, CSE 326
Other Applications

✦ Binomial Queues:
  ➤ Starting from an empty queue, buildBinomialQueue takes $O(N)$ rather than $O(N \log N)$ to insert $N$ nodes
  ➤ Analysis very similar to that for BigNum
  ➤ Read Section 11.2 for the final

✦ Splay Trees
  ➤ Result: Starting from an empty tree, $M$ consecutive tree operations take $O(M \log N)$ time
  ➤ Amortized run time per operation = $O(\log N)$
  ➤ Uses the potential function $\Phi(T) = \text{sum over all nodes } x \text{ in } T \text{ of } \log(\text{number of descendants of } x)$
  ➤ Complicated analysis in Section 11.5 which you don’t need to know for the final

Final Review

(“We’ve covered way too much in this course…

What do I really need to know?”)
Final Review: What you need to know

✦ Basic Math
   ➤ Logs, exponents, summation of series
   ➤ Proof by induction

✦ Asymptotic Analysis
   ➤ Big-oh, little-oh, Theta and Omega
   ➤ Know the definitions and how to show f(N) is big-oh/little-oh/Theta/Omega of g(N)
   ➤ How to estimate Running Time of code fragments
     ✤ E.g. nested “for” loops

✦ Recurrence Relations
   ➤ Deriving recurrence relation for run time of a recursive function
   ➤ Solving recurrence relations by expansion to get run time

\[ \sum_{i=1}^{N} i = \frac{N(N+1)}{2} \]

\[ \sum_{i=0}^{N} A_i = \frac{A^{N+1} - 1}{A-1} \]

What you need to know: Stacks, trees, …

✦ Lists, Stacks, Queues
   ➤ Brush up on ADT operations – Insert/Delete, Push/Pop etc.
   ➤ Array versus pointer implementations of each data structure
   ➤ Header nodes, circular, doubly linked lists

✦ Trees
   ➤ Definitions/Terminology: root, parent, child, height, depth etc.
   ➤ Relationship between depth and size of tree
     ✤ Depth can be between O(log N) and O(N) for N nodes
What you need to know: BSTs

- **Binary Search Trees**
  - How to do Find, Insert, Delete
  - Bad worst case performance – could take up to O(N) time
  - AVL trees
    - Balance factor is +1, 0, -1
    - Know single and double rotations to keep tree balanced
    - All operations are O(log N) worst case time
  - Splay trees – good amortized performance
    - A single operation may take O(N) time but in a sequence of operations, average time per operation is O(log N)
    - Every Find, Insert, Delete causes accessed node to be moved to the root
    - Know how to zig-zig, zig-zag, etc. to “bubble” node to top
  - B-trees: Know basic idea behind Insert/Delete

WYNTK: Priority Queues and Hashing

- **Priority Queues**
  - Binary Heaps: Insert/DeleteMin, Percolate up/down
    - Array implementation
    - BuildHeap takes only O(N) time (used in heapsort)
  - Binomial Queues: Forest of binomial trees with heap order
    - Merge is fast – O(log N) time
    - Insert and DeleteMin based on Merge

- **Hashing**
  - Hash functions based on the mod function
  - Collision resolution strategies
    - Chaining, Linear and Quadratic probing, Double Hashing
  - Load factor of a hash table
WYNTK: Sorting

✦ **Sorting Algorithms**: Know run times and how they work
  ➔ Elementary sorting algorithms and their run time
    ✦ Bubble sort, Selection sort, Insertion sort
  ➔ Shellsort – based on several passes of Insertion sort
    ✦ Increment Sequence
  ➔ Heapsort – based on binary heaps (max-heaps)
    ✦ BuildHeap and repeated DeleteMax’s
  ➔ Mergesort – recursive divide-and-conquer, uses extra array
  ➔ Quicksort – recursive divide-and-conquer, Partition in-place
    ✦ fastest in practice, but $O(N^2)$ worst case time
    ✦ Pivot selection – median-of-three works best
  ➔ Know which of these are stable and in-place
  ➔ Lower bound on sorting, bucket sort, and radix sort

WYNTK: Disjoint Sets and Graphs

✦ **Disjoint Sets and Union-Find**
  ➔ Up-trees and their array-based implementation
  ➔ Know how Union-by-size and Path compression work
  ➔ No need to know run time analysis – just know the result:
    ✦ Sequence of $M$ operations with Union-by-size and P.C. is $\Theta(M \alpha(M,N))$ – basically $\Theta(1)$ amortized time per op

✦ **Graph Algorithms**
  ➔ Adjacency matrix versus adjacency list representation of graphs
  ➔ Know how to Topological sort in $O(|V| + |E|)$ time using a queue
  ➔ Breadth First Search (BFS) for unweighted shortest path
WYNTK: Graph Algorithms

✦ Graph Algorithms (cont.)
  ➤ Dijkstra’s shortest path algorithm – greed works!
    ♦ Know how a priority queue can speed up the algorithm
  ➤ Depth First Search (DFS)
  ➤ Minimum Spanning Trees: Know the 2 greedy algorithms
    ♦ Prim’s algorithm – similar to Dijkstra’s algorithm
    ♦ Kruskal’s algorithm
      ● Know how it uses a priority queue and Union/Find
    ♦ Euler versus Hamiltonian circuits – difference in run times
    ♦ Know what P, NP, and NP-completeness mean
      ● How one problem can be “reduced” to another (e.g. input to HC can be transformed into input for TSP)

WYNTK: Algorithm Design Techniques

✦ Greedy Algorithms
  ➤ Bin Packing
✦ Divide & Conquer
  ➤ Solving various types of recurrence relations for \( T(N) \)
✦ Dynamic Programming (Memoizing)
  ➤ DP-Fibonacci
  ➤ Go over other examples in text
✦ Randomized Data Structures and Algorithms
  ➤ Average run time over all inputs vs. Expected run time for one input
  ➤ Treaps
  ➤ Primality Testing
✦ Backtracking and Game Trees
WYNTK: Amortized Analysis

✦ Know the basic concept
  ➤ Amortized run time per operation over a sequence of N operations

✦ Two Techniques
  ➤ Aggregate Method: Compute directly the total time for N operations and divide by N
  ➤ Potential Method: Find a “potential function” that can be used to place an upper bound on total run time
  ➤ E.g. Binary counter, binomial queues (see textbook)

WYNTK about the Final

✦ Details:
  ➤ Covers Chapters 1-10, 11.2, 12.5 in the textbook
    ◆ Emphasis on Chapters 7-10, Sec. 11.2, and 12.5
    ◆ Emphasis on material covered in lecture slides
  ➤ Closed book, closed notes except:
    ◆ You may bring one 8 ½” x 11” sheet of notes
  ➤ Time: 1 hour and 50 minutes
  ➤ When: 8:30-10:20 a.m., Thursday, March 20 in class
  ➤ Sample questions are on class website
  ➤ Final will contain space for answers; no bluebooks
  ➤ Bring pens/sharpened pencils (and sharpened minds!)
  ➤ No, the final won’t be NP-complete (it will be in P)
No class on Friday!

**Final Exam:**
Where: This room
When: 8:30-10:20 a.m., Thursday, March 20

**To Do:**
Go over practice final and problems on web site
Prepare, prepare, prepare…

N’joy da spring break!