Final Review

Thu May 10, 2018

Project 4

- We will allow submissions for grade improvements.
  - Through May 29th, midnight.
- Weight the same (40%), whether or not you submit a 4th project.
  - ELMS takes care of weighting of grade average automatically!
- Tough-ish project to understand, but actual coding small to medium
  - Smaller than 1st project, bigger than 2nd / 3rd
- Talk about it today.
- Grades will be submitted to Testudo as soon as we can next week, and should be the same as on ELMS.
  - If you submit through May 29th and that improves your ELMS letter grades, shoot me an e-mail and we will update your Testudo grade.

Final

- Final is **cumulative** with $\approx 60\%$ of credit on strings and spatial data structures.
- Percentage of grade coming from the final is 20%.
- All the **skills** that you have developed through homeworks and previous exams are relevant.
- Format of the final: similar to previous exams, homeworks.
  - Duration: 2 hours.
  - BRB1101, Monday at 4pm.
  - 1146 AVW: Coffee Tasting event at 2.
    - So you can be well awake at 4.

Off the final

- Implementation of **2-3-4** trees as Red-Black Trees: only mentioned it in passing; won't test it.
- Why **splay trees** have amortized complexity over m operations equal to $\mathcal{O}(m \cdot \log_2 N)$, where N is the maximum number of nodes they ever had. ONLY THE REASON WHY THEY HAVE THIS COMPLEXITY IS NOT REQUIRED: YOU WILL **STILL** HAVE TO KNOW THAT IT IS THAT WAY.
- **Hard** deletion in Red-Black Trees: Didn't cover it, won't test for it.
- **De La Briandais tries**: if we have something that involves them, we will remind you of exactly what they look like.
- **Quadratic / Double hashing**: If used, we will re-define them for you.

Will be given

- All formulae
  - B+-trees: relating fan-out p and height h, spatial cost formula.
  - Expectation of #probes for search hits / misses *when hash function assumed perfectly uniform.*
- ASCII Table

Skills for success

1. Intro stuff
   - Array-backed lists (Vector, ArrayList) and how various resizing policies might affect amortized insertion *or deletion* complexity. (Resizing can and does work both ways!)
   - Worst-case complexity of an operation if given a simple data structure like a list, singly linked or doubly linked.
2. Balanced Binary Trees:
   - Initially empty tree, add Comparable elements. Show the final tree. Count “probes”, i.e memory accesses.
   - Insert or delete elements into / from a non-empty tree. Show the final tree.
   - Complexity metrics / worst-case height, average height, amortized complexity of operations.

Skills for success

3. B-Trees
   - Know how to **use** the formulae for B+-trees (since those will be given)
   - Insertion / Deletion into a ready-made B-tree, with provided p and $hops$.
   - Build a B-Tree **from scratch** given some keys to insert.
4. Hashing
   - What makes a good hash function?
     - ✓ **Fast to compute**
     - ✓ **Approximately uniform distribution of keys**
     - ✓ **Equality of objects implies equality of hash code**
   - Collision Resolution
     - Separate Chaining
     - Linear Probing
     - Other methods I might explain on the spot like Quadratic or Double hashing: you will need to be able to adapt what you already know!
     - Math analysis of probes: **Average** worst-case number of probes, worst-case number of probes. Knuth's formula.
     - Questions like the last two ones of problem 1, midterm 2 (amortized complexity of inserting some elements into a hash table of a given load factor alpha.

Skills for success

5. Strings

- **Encodings**: What's a variable vs non-variable length encoding? Benefits? Drawbacks? Examples of both?
- **Tries**: Everything. The result about how search in tries is independent of n. Ordinary vs compressed (Patricia) tries: which nodes are compressible? Can I draw a compressed trie and show how it's updated after insertions / deletions?
- **Huffman**: Why do we do it? How does it work? How does it guarantee the "prefix property"? Can you run Huffman encoding on paper? Can you show us the final trie that Huffman will produce?
- **LZW**: Pretty much the same stuff, except for the fact that general case codewords **don't** satisfy the prefix property.
  - Can you run it on paper?
  - Can you tell me how the **encoder** encodes the string?
  - If you play the role of the **decoder**, which string was encoded and sent?

Skills for success

6. Spatial data structures
   - Given a spatial decomposition, can I find the KD-Tree?
   - Do I remember the algorithms for insertion and deletion in KD-Trees?
   - Characteristics of KD-Trees; they are sensitive to insertion order, which can lead to arbitrarily imbalanced KD-Trees. They loop through dimensions level by level, however their fan-out is always 2 (they are binary trees).
   - Do I remember that a Point QuadTree is **different** from a Point-Region (P-R) Quadtree?
     - The “point” quadtree is the one that draws four axes per point and has horrible deletion. Probably the first spatial data structure introduced, late 70s by Fink, deletion improved by Samet early 80s. Sensitive to insertion order.
     - The P-R QuadTree is the one that subdivides the space all the time. So those are **insensitive** to insertion order.