Class Meeting #14

COS 226 — Spring 2018

Mark Braverman

Midterm overview

- Multiple choice questions:
  - Variable difficulty
  - Hardest questions:
    - compareTo;
    - loneliest using standard BST
    - Other questions: 70-95% answered correctly
- Median priority queue:
  - Considered a ‘B-level’ question.
  - Most students got the high-level idea.
  - Implementation details vary

Midterm overview

- Loneliest element question.
  - Considered an ‘A level’ question
  - About 25% got 7/10 or more
  - Median/average around 4/10
  - Hard to see the complete solution right away – start collecting the pieces (e.g. the idea of using two search trees).

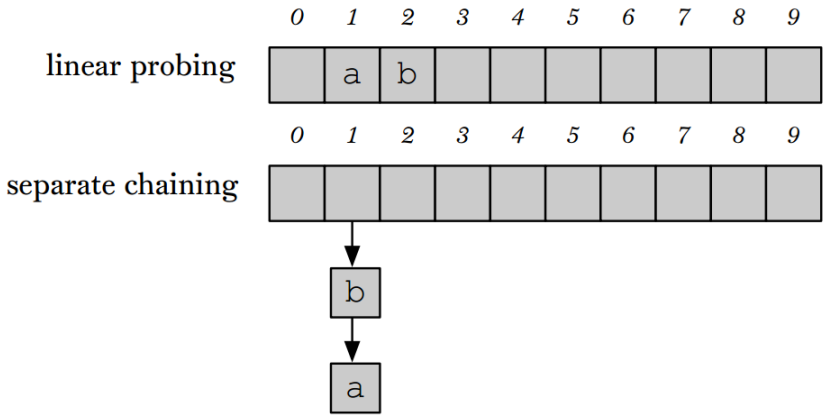
Strategies for test #2

- Multiple choice part: quizerra, lectures, readings
- Design part, preparation:
  - Old exams
  - Not reading the checklist right away
  - During assignments, code mindfully
- Design part, test taking strategies:
  - Meta: understand which problems to tackle first.
  - Start with high-level structure.
  - Write out an explanation of the roles of the major pieces. These should lead to rules to guide implementation.
  - Implementation (as needed).

General advice

- When making long-term decisions, think long-term.
- When making short-term decisions, think short-term.

Hash tables



$h(a) = 1$ [inserted first]

$h(b) = 1$ [inserted second]

Hash tables

True/false:

A. With *separate chaining*, if the hash table is smaller than the number of distinct keys to insert, the table will overflow.

B. With *linear probing*, if the hash table is smaller than the number of distinct keys to insert, the table will explode.

C. Recall that a hash table is a symbol table which associates some data to a key (and we call entry the key-data pair). Then, the best definition of a collision in a hash table is when:

(i) two entries are identical except for their keys;

(ii) two entries with different data have the exact same key;

(iii) two entries with different keys have the exact same hash value;

(iv) two entries with the exact same key have different hash values.

D. Choosing a bad hash function can result in

(i) too many collisions;

(ii) slow performance;

(iii) no expected constant runtime guarantees;

(iv) too much clustering when using linear probing.

F. Which of the following scenarios leads to a linear running time for a random search hit in a linear probing hash table containing N keys?

(i) All keys hash to different indices.

(ii) All keys hash to the same index.

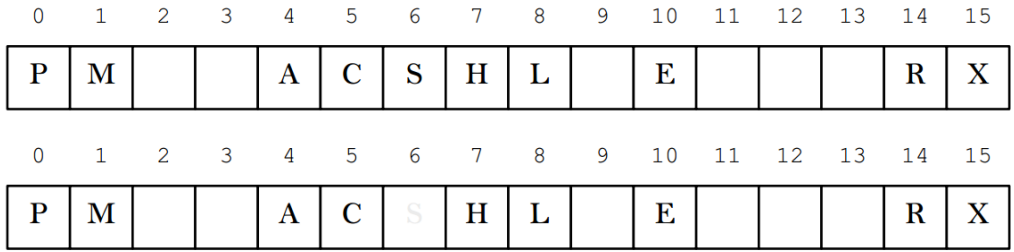
(iii) All keys hash to an even-numbered index.

(iv) The table has size larger than N^2.

True/false:

G. With separate chaining for collision resolution, it is unnecessary to resize the hash table to obtain good efficiency.

H. Deletions in linear probing are easier to implement than in separate chaining.



$h(\mathrm{S}) = 6$
$h(\mathrm{H}) = 6$

removing S this way won't work because when looking for H, the hash table will think that it is not in the table

Kd-tree assignment

Timing demo

Timing question

(J. Lumbroso's slides)

How many get() calls can a BST implementation perform per second for a BST that contains 1,000 random keys?

You can assume that the keys are integers.

When timing do not include the time to build the BST. (Algorithms textbook 3.2)

Timing question

```
BST<Integer, Integer> testTree = new BST<Integer, Integer>();

// insert N random values
for(int i = 0; i < N; i++)
    testTree.put(i, StdRandom.uniform(1, 1000000));

// time number of gets
// (make enough operations to last 1 second at least)
Stopwatch timer1 = new Stopwatch();
int numGets = 0;

for(int j = 0; j < 100; j++) {
    for(int i = 0; i < N; i++) {
        testTree.get(i);
        numGets += 1;
    }
}

double timing = timer1.elapsedTime();

StdOut.println(numGets/timing);
```