# Introduction to Algorithms

Meng-Tsung Tsai

10/22/2019

# Announcements

Written Assignment 2 is due by Oct 31, 15:40. at https://e3.nctu.me

Programming Assignment 2 was extended, and is due by Nov 5, 23:59. at https://oj.nctu.me

Quiz 1 will be held in class on Oct 24.

Scope: slides 01 - 09, assignments, and their generalizations.

# About Quiz 1

Asymptotic Bounds: was1-p1, was1-p2, was2-p1.

Basic DP: was2-p2, pas2-p1.

Reduction: was1-p5, was2-p6.

------

Don't forget the "I don't know" policy.

You may bring two cheating sheets in A4 size.

# Dynamic Sets

# Dynamic Sets

Input: a sequence of insert(T, x), delete(T, x), and search(T, x) operations. Let n denote the number of operations and let U = {0, 1, 2, ..., m-1} denote the universe of keys, i.e. x in [0, m-1].

Output: for each search(x), answer "Yes" if x is currently in T, or "No" otherwise.

# Reading Assignment

Binary Search Trees can support insertions and deletions, each in O(height) time. I2A pp. 294

Red-Black Trees is an implementation of Binary Search Trees while always keeping the height of n-node trees O(log n). I2A pp. 308

| n keys | search cost | insertion cost | deletion cost |
|---|---|---|---|
| BST | O(height) | O(height) | O(height) |
| RBT | O(log n) | O(log n) | O(log n) |

# Reading Assignment

Binary Search Trees can support insertions and deletions, each in O(height) time. I2A pp. 294

Red-Black Trees is an implementation of Binary Search Trees while always keeping the height of n-node trees O(log n). I2A pp. 308

| n keys | search cost | insertion cost | deletion cost |
|---|---|---|---|
| BST | O(height) | O(height) | O(height) |
| RBT | O(log n) | O(log n) | O(log n) |

BST is an implementation of dynamic sets.

# Hash Tables

# Direct-Address Tables

If the universe U is small, then one may use the identity function

$$hash_I(x) = x$$

as a hash function to access a hash table T of size |U|.

--- Example ---

```
                  // T[0..m-1] = {0};
insert(T, 11);    // T[hash_I(11)] ++;
insert(T, 3);     // T[hash_I(3)] ++;
delete(T, 11);    // T[hash_I(11)] --;
search(T, 3);     // print T[hash_I(3)] > 0;
...
...
search(T, 21);    // print T[hash_I(21)] > 0;
```

# Direct-Address Tables

If the universe U is small, then one may use the identity function

$$hash_I(x) = x$$

as a hash function to access a hash table T of size |U|.

--- Downside ---

If U is the set of all 32-bit integers, then T has size 4G bytes.

# Hash Tables

If one can afford an array of s entries, then one may use

$$\text{hash}_M(x) = x \bmod s$$

as a hash function to access a hash table T of size p.

--- Issues ---

Say p = 13, and process insert(T, 2) and insert(T, 15). Then we have

$$\text{hash}_M(2) = \text{hash}_M(15).$$

Two different keys access the same table entry, i.e. a *collision*.

# Hash Tables

If one can afford an array of s entries, then one may use

$$hash_M(x) = x \bmod s$$

as a hash function to access a hash table T of size p.

--- Resolve Collisions by Chaining ---

Each table entry is replaced with a linked list L.

search: a linear scan in $O(|L|)$ time.

insertion: add to the front in $O(1)$ time.

deletion: a linear scan in $O(|L|)$ time followed by re-wiring in $O(1)$ time.

# Random Hash Functions

# A Uniformly-Random Function

Given n keys, and a table of n entries.

Suppose one can hash every key uniformly at random to a table entry, then a longest chain has length O(log n) w.h.p.

# The Power of Two Choices

Given n keys, and a table of n entries.

Suppose one can hash every key uniformly at random to two table entries and add the newly inserted keys into a shorter list, then a longest chain has length loglog n+O(1) w.h.p.

# The Power of d Choices

Given n keys, and a table of n entries.

Suppose one can hash every key uniformly at random to k table entries and add the newly inserted keys into a shortest list, then a longest chain has length (loglog n)/(log d)+O(1) w.h.p.

# Universal Hashing Family

$H = \{h_{ab}(x) = ((ax + b \bmod p) \bmod s)\}$ for some prime $p > |U| > s$.

Given n keys, pick a random hash function from H, then the expected length of a longest chain has length O(1).

# Comparison

| n keys | search cost | insertion cost | deletion cost | space |
|---|---|---|---|---|
| BST | O(height) | O(height) | O(height) | O(n) |
| RBT | O(log n) | O(log n) | O(log n) | O(n) |
| $hash_I$ | O(1) | O(1) | O(1) | O(\|U\|) |
| $hash_M$ | O(n) | O(1) | O(n) | O(n) |
| $hash_{U(0, n-1)}$ | O(log n) w.h.p. | O(1) | O(log n) w.h.p. | O(n) |
| $hash_{U(0, n-1)}$ + d choices | O(dloglog n/log d) w.h.p. | O(d) | O(dloglog n/log d) w.h.p. | O(n) |
| $hash_{universal}$ | expected O(1) | expected O(1) | expected O(1) | O(n) |

# Exercise

Pick up the notion of expection, varianece, Union bound, Markov inequality, Chebyshev inequality, Chernoff bound, independence, k-wise independence in a probability course.

We will cover the proof in the lecture of probabilitic data structures at the end of this semester.