CS 420: Advanced Algorithm Design and Analysis
Spring 2015 – Lecture 5

Department of Computer Science
University of British Columbia

January 20, 2015

Announcements

Assignments...

- Asst2....(due next Thursday)
- remark on email consultation

Readings...

- material on x-fast and y-fast tries [on line]
- material on hashing [Kleinberg, 13.6; Cormen+, chap 11; Erickson, chapt 12]
- material on closest-pair problem [Kleinberg]

Looking ahead...

Our goal, in the next few lectures is to understand how we might circumvent this lower bound, by *stepping outside the abstract comparison-based model*. We will consider:

- exploiting assumptions about the structure/size of the key space $\mathcal{U}$
- exploiting assumptions about the distribution of keys in S
- exploiting assumptions about the pattern of successive queries
- (if time permits) other issues: randomization, error tolerance...

Last class...

Continue exploiting assumptions about the structure/size of the key space $\mathcal{U}$

- inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m - 1\}$ (cont.)
  - simple augmentation of direct access tables to facilitate predecessor/successor queries
    - using a tree-directory with marked nodes; simple optimizations
  - finding the predecessor and successor keys using auxiliary structures (*x-fast tries*)
    - perform binary search on access paths to find lowest marked ancestor
  - handling updates efficiently (*y-fast tries*)
    - key ideas: (i) *partition* keys in S into subsets of size about $\lg m$, (ii) represent each subset as a standard balanced binary search tree, and (iii) store one *representative* from each subset in an x-fast trie.

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots m - 1\}$

But...what about the space requirements?!

Today...

Stepping away from the most general (comparison-based) dictionary model...different possibilities

- inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots u - 1\}$ (cont.)
  - overcoming space concerns with previous structures
    - hashing (the role of randomization)
    - universal hashing
    - perfect hashing

Inputs are drawn from a restricted *universe* $\mathcal{U} = \{0, 1, \ldots u - 1\}$

But...what about the space requirements!

- How can we exploit *direct access* but reduce space?
  - build *hash tables!*
- What do we give up?
  - essentially *nothing*...

Hashing review

Basic definitions

- key *universe* $\mathcal{U} = \{0, 1, \ldots u - 1\}$
- set $S \subset \mathcal{U}$ of size $|S| = n$
- map keys in S to a table $T[0 : m - 1]$, with *hash function* $h : \mathcal{U} \to \{0, 1, \ldots, m - 1\}$

Resolving collisions (since $u >> m$)

- chaining
- open addressing

Why does it work?

- randomization!

On randomness in the design and analysis of hashing

Sources of (assumed) randomness

- Assume randomness resides in the set S; elements chosen *at random* from $\mathcal{U}$
  - all sets S of size n are equally likely
  - it suffices to choose h to be any *uniform* mapping (u/m elements of $\mathcal{U}$ map to each index in $\{0, \ldots, m-1\}$)
- Assume h behaves like a *random mapping*
  - appears "patternless"
  - good behaviour (few collisions) may be supported empirically
  - problem: *every* fixed mapping behaves poorly on some sets
- Desired property: *simple uniform hashing assumption*
  - If $x \neq y$ then $\Pr[h(x) = h(y)] = 1/m$

On randomness in the design and analysis of hashing

Sources of (assumed) randomness (cont.)

- Choose h *randomly* from the set of all m^u possible hash functions
  - all mappings are equally likely
  - problem: how do we describe h?
  - essentially the only way is to describe its value on all inputs explicitly ($\Theta(u \lg m)$ bits)
- Choose h *randomly* from some smaller *universal* set $\mathcal{H}$ of hash functions
  - $\mathcal{H}$ is *universal* if for all $x, y \in \mathcal{U}$, $|\{h \in \mathcal{H} \text{ s.t. } h(x) = h(y)\}| \leq |\mathcal{H}|/m$
  - note: this is essentially the best we could hope for (by counting)
  - note: use of randomization here is like quicksort and the hiring algorithm: random choice makes all inputs behave the same (in expectation); adversary cannot choose a bad input.

Properties of universal families of hash functions

Suppose h is chosen uniformly at random from a universal family $\mathcal{H}$. Then, with expectation (over the choice of h), but independent of the choice of S:

- simple uniform hashing assumption is satisfied
  - $E[|\{x \in (S \setminus k) \text{ s.t. } h(x) = h(k)\}|] \leq n/m$

Properties of universal families of hash functions

Suppose h is chosen uniformly at random from a universal family $\mathcal{H}$. Then, with expectation (over the choice of h), but independent of the choice of S:

- Elements of S are evenly spread, in expectation
  - expected cost of insert/delete/member is $O(1 + n/m)$
  - Note: the expected length of the *longest* collision list when $m = n$ is $\Theta(\lg n/\lg\lg n)$, so worst case search is not that much better than a binary search tree!

Properties of universal families of hash functions

Expected *total* number of collisions is $\binom{n}{2}/m$

- So, if we choose $m = n^2$ then with probability $\geq 1/2$ there will be *no collisions*: *near-perfect* hashing
- Furthermore, if we choose $m = n$, the expected total number of collisions is $\Theta(n)$
  - So if n_i items map to $T[i]$, it follows that $E[\sum_i n_i^2] = O(n)$.

construction of *perfect* hash functions

A two-level hash scheme:

- *level 1*: Use universal hashing with table size m equal to n (the size of S)
    - Suppose n_i elements map to $T[i]$
    - with $O(1)$ expected trials we can guarantee that $\sum_i n_i^2 = O(n)$; (recall $E[\sum_i n_i^2] = O(n)$)
- *level 2*: resolve first level collisions by using a (near-perfect) secondary hash table (of size n_i^2) associated with table slot $T[i]$
    - each secondary table is formed with $O(1)$ expected trials
    - total space for secondary tables is $O(n)$

construction of *perfect* hash functions (cont.)

A *dynamic* two-level hash scheme:

- how can we deal with insertions/deletions?
- design for expansion
  - make first-level and second-level tables twice as large as we need
  - rebuild when n doubles, or desired properties no longer hold
  - amortize cost of expansion

Properties of universal families of hash functions

Compact universal families $\mathcal{H}$ exist and are efficient to construct.

- Kleinberg&Tardos (section 13.6) describe one construction based on modular arithmetic

Next time...

Compact universal families $\mathcal{H}$ exist and are efficient to construct

- Kleinberg&Tardos (section 13.6) describe one construction based on modular arithmetic

Applications of universal hashing (cont.)

- finding the closest pair of points in a point set: Kleinberg&Tardos (section 13.7)

Other geometric (higher-dimensional) search problems

- two-dimensional dictionaries?