CMSC 420 – 0201 – Fall 2019
Lecture 10

Hashing – Basics and Hash Functions

Hashing

Recap

- So far, we have discussed a variety of structures for dictionaries:
  - insert
  - delete
  - find
- We have assumed a comparison-based model (e.g, `x < p.key`)
- Finding a key among n elements requires $O(\log n)$ time, cannot hope for better

- Hashing: if we abandon comparisons, we can find keys in $O(1)$ expected time!

Hashing

Intuition

- We store the n keys in a table containing m entries
- We assume that the table size m is at least a constant factor larger than n
  - E.g., $m > c\,n$, where $c = 1.25$
- We scatter the keys throughout the table using a pseudo-random hash function
  - $h(x) \in [0 \ldots m-1]$
  - Store x at entry $h(x)$ in the table
- Sometimes different keys will map to the same location, $x \neq y$, but $h(x) = h(y)$
- This is a collision, and we will need strategies for resolving them (next time)
- If the number of keys colliding with x is small ($O(1)$), then we can access x in $O(1)$ time.

Hash Functions

Desirable properties

- A good hash function h should:
  - Should be efficiently computable (constant time)
  - Should produce few collisions
    - Use every bit of the input key
    - Break up (scatter) naturally occurring clusters of keys

- For example, keys “temp1”, “temp2”, and “temp3” should *not* be stored in consecutive entries of the hash table

Hash Functions

Popular Hash Functions

- Some popular functions:
  - Division Hashing: $h(x) = x \bmod m$ (Simple, but not very strong)
  - Multiplicative Hashing: $h(x) = (a \cdot x) \bmod m$ or $h(x) = ((ax) \bmod p) \bmod m$, where a and p are large primes
  - Linear Hashing: $h(x) = a \cdot x + b \bmod m$ or $h(x) = ((ax + b) \bmod p) \bmod m$, where a, b and p are large primes
- Why mod with both p and m?
  - m is often a power of 2, and so $x \bmod m$ is just the lower-order bits of x
  - Taking mod p is much more “random”. Then do “mod m” to reduce to table size.

Hash Functions

Polynomial Hashing - Finer Points

- Polynomial Hashing:
  - Compute a polynomial function of the key. Convenient when the key is a sequence of numbers (e.g., a character string)
  - Let: $x = (c_0, c_1, c_2, c_3, \ldots,)$, and let p be a suitable prime
  - Then: $h(x) = (c_0 + c_1 p + c_2 p^2 + c_3 p^3 + \cdots) \bmod m$
- Computing polynomial functions efficiently - Horner's rule
  - $(c_0 + c_1 p + c_2 p^2 + c_3 p^3 + \cdots) = c_0 + p(c_1 + p(c_2 + p(c_3 + \cdots)))$

Hash Functions

Polynomial Hashing - Finer Points

- Computing polynomial functions efficiently - Horner's rule
  - $(c_0 + c_1\, p + c_2 p^2 + c_3 p^3 + \cdots) = c_0 + p(c_1 + p(c_2 + p(c_3 + \cdots)))$

```
public int hash(String c, int m) {      // polynomial hash of a string
    final int P = 37;                   // replace this with whatever you like
    int hashValue = 0;
    for (int i = c.length()-1; i >= 0; i--) {  // Horner's rule
        hashValue = P * hashValue + Character.getNumericValue(c.charAt(i));
    }
    return hashValue % m;               // take the final result mod m
}
```

Randomization and Universal Hashing

- Assuming the keys are not known in advance, no hashing function is "perfect" – collisions are inevitable
- But randomness can help
- Intuition: By selecting the hash function randomly, it will be good (in expectation) for any given pair of keys

Randomization and Universal Hashing

- Universal Hashing:
  - A "bag" of possible hash functions H
  - Select one function h from the bag at random
  - The system is universal if, for any x, y, the probability that $h(x) = h(y)$ for a randomly chosen function h is $\frac{1}{m}$
- Carter & Wegman (1977): There exist universal hash functions
  - Pick a large prime p (larger than any possible key)
  - Pick a at random from $\{1, 2, \ldots, p-1\}$
  - Pick b at random from $\{0, 1, 2, \ldots, p-1\}$
  - Hash function: $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$

Randomization and Universal Hashing

- Theorem: Consider any two integers x and y, where $0 \leq y < x < p$. Let $h_{a,b}$ be a random hash function described in the previous slide. Then the probability that $h_{a,b}(x) = h_{a,b}(y)$ is at most $1/m$.
- Proof: (See full lecture notes)

Summary

- Hashing –
  - Basic concept
  - Hash functions
- Stay tuned –
  - Collision resolution methods