Hashing

COMP 215 Lecture 18
Hashing

• Quick reminder for hashing...
  – $\lg n$ work for binary search.
  – $O(1)$ work to retrieve an element from an array, if we know the index.
  – Why not just use keys as indices?
    • Two keys: .032 and 1,000,00.234.

• The idea behind hashing-
  – Remap keys so that they are uniformly distributed in the range 1..$n$ where $n$ is the number of keys.
  – Use the remapped (hashed) keys to index an array.
Hashing

• A simple hashing scheme:
  - \( h(key) = key \% n \)
  - If we have 100 keys that are decimal numbers, this hash function just returns the last two digits.
  - Works fine if there is the last two digits are drawn from a uniform distribution.
  - Not always a good assumption.

• Getting hashing to work requires:
  - A good hashing function.
  - A method for handling collisions. (open and closed hashing)
Hashing Analysis

• If we have a hash function that distributes our $n$ keys uniformly in $m$ buckets.
  – Expected number of keys/bucket is?
  – Expected time for a failed search?
  – Expected time for a successful search?

• Worst case:
  – Number of keys in a bucket?
  – Failed search?
  – Successful search is?
Hashing Analysis

• If we have a hash function that distributes our $n$ keys uniformly in $m$ buckets.
  - Expected number of keys/bucket is $n/m$.
  - Expected time for a failed search $n/m$.
  - Expected time for a successful search $(n/m +1)/2$.

• Worst case:
  - Number of keys in a bucket?
  - failed search is $n$.
  - successful search is $n$. 
Hashing and Binary Search

- Worst case for hashing is much worse than worst case for binary search.
- Average case for hashing is (arguably) not that much better.
- Why not just use binary search?
- The probability of the worst case situation occurring for hashing is
  \[ n \times \left( \frac{1}{n} \right)^n \]
- If \( n = 100 \), this is \( 10^{-198} \).
Hashing and Binary Search

• What is the probability that hashing will be less efficient than binary search?

• The probability that any given set of \( k \) keys will end up in a given bucket is: \( \left( \frac{1}{m} \right)^k \)

• Probability that any particular bucket will contain at least \( k \) keys is: \( \binom{n}{k} \left( \frac{1}{m} \right)^k \)

• Probability that some bucket will contain \( k \) keys is:

\[
m \times \binom{n}{k} \left( \frac{1}{m} \right)^k = \binom{n}{k} \left( \frac{1}{m} \right)^{k-1}
\]
Hashing and Binary Search

- We can then plug in $\lg n$ for $k$ to compute the probability that hashing will make as many comparisons as binary search for a given input size.
  - $n = 128, \ p = .021$
  - $n = 65,536, \ p = 3.1 \times 10^{-9}$

- All of this assumes that keys are uniformly distributed.
- The real danger is that the hash function will not uniformly distribute the keys.