# Hashing

- Hashing Recap
- Dictionaries
- Perfect Hashing
- String Hashing

Philip Bille

# Hashing

- Hashing Recap
- Dictionaries
- Perfect Hashing
- String Hashing

## Hashing Recap

- Hash function idea.
  - Want a random, crazy, chaotic function that maps a large universe to a small range. The function should distribute the items "evenly."
- Hash function.
  - Let H be a family of functions mapping a universe U to {0, ..., m-1}.
  - A hash function h is a function chosen randomly from H.
  - Typically $m \ll |U|$.
- Goals.
  - Low collision probability: for any x≠y, we want Pr(h(x) = h(y)) to be small.
  - Fast evaluation.
  - Small space.

## Hashing Recap

- Universal hashing.
  - Let H be a family of functions mapping a universe U to {0, ..., m-1}.
  - H is universal if for any x ≠ y in U and h chosen uniformly at random from H

$$\Pr(h(x) = h(y)) \leq \frac{1}{m}.$$

- Examples.
  - Multiply-mod-prime.
    - $h_{a,b}(x) = ax + b \mod p$ with $H = \{h_{a,b} \mid a \in \{1, \ldots, p-1\}, b \in \{0, \ldots, p-1\}\}$.
  - Multiply-shift.
    - $h_a(x) = (ax \bmod 2^k) \gg (k - l)$ with $H = \{h_a \mid$ a is an odd integer in $\{1, \ldots, 2^k - 1\}\}$

# Hashing

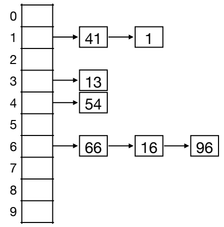
- Hashing Recap
- Dictionaries
- Perfect Hashing
- String Hashing

## Dictionaries

- Dictionary problem. Maintain a dynamic set of integers $S \subseteq U$ subject to following operations
  - LOOKUP(x): return true if $x \in S$ and false otherwise.
  - INSERT(x): set $S = S \cup \{x\}$
  - DELETE(x): set $S = S \setminus \{x\}$
- Satellite information. Information associated with each integer.
- Applications. Lots of practical applications and key component in other algorithms and data structures.
- Challenge. Can we get a compact data structure with fast operations.

## Chained Hashing

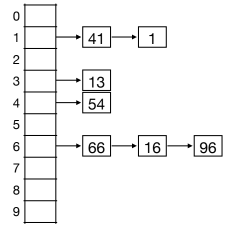
- Chained hashing.
  - Choose universal hash function h from U to {0, ..,m-1}, where $m = \Theta(n)$.
  - Initialize an array A[0, ..., m-1].
  - A[i] stores a linked list containing the keys in S whose hash value is i.




- Space. $O(m + n) = O(n)$

## Chained Hashing

- Operations.
  - LOOKUP(x): Compute h(x). Scan A[h(x)]. Return true if x is in list and false otherwise.
  - INSERT(x): Compute h(x). Scan A[h(x)]. Add x to the front of list if it is not there already.
  - DELETE(x): Compute h(x). Scan A[h(x)]. Remove x from list if it is there.




- Time. $O(1 + |A[h(x)]|)$

## Chained Hashing

- What is the expected length of A[h(x)]?
- Let $I_y = \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}$
- $E\left(|A[h(x)]|\right) = E\left(\sum_{y \in S} I_y\right) = \sum_{y \in S} E\left(I_y\right) = 1 + \sum_{y \in S \setminus \{x\}} \Pr\left(h(x) = h(y)\right) \leq 1 + (n-1) \cdot \frac{1}{m} = O(1)$
- Theorem. We can solve the dictionary problem in O(n) space and constant expected time per operation.

## Hashing

- Hashing
- Dictionaries
- Perfect Hashing
- String Hashing

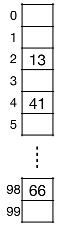
## Static Dictionaries and Perfect Hashing

- Static dictionary problem. Given a set $S \subseteq U = \{0,..,u-1\}$ of size n for preprocessing support the following operation
  - LOOKUP(x): return true if $x \in S$ and false otherwise.
- Challenge. Can we do better than (dynamic) dictionary solution?
- Perfect Hashing. A perfect hash function for S is a collision-free hash function on S.
  - Perfect hash function in O(n) space and O(1) evaluation time $\Longrightarrow$ solution with O(n) space and O(1) worst-case lookup time.
  - Do perfect hash functions with O(n) space and O(1) evaluation time exist for any set S?

## Static Dictionaries and Perfect Hashing

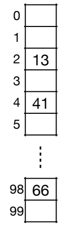
- Goal. Perfect hashing in linear space and constant worst-case time.
- Solution in 3 steps.
  - Solution 1. Collision-free but with too much space.
  - Solution 2. Many collisions but linear space.
  - Solution 3: FKS scheme [Fredman, Komlós, Szemerédi 1984]. Two-level solution. Combines solution 1 and 2.

## Solution 1: Collision-Free, Quadratic Space




- Data structure.
  - Array A of size $n^2$.
  - Universal hash function mapping U to $\{0, ..., n^2-1\}$. Choose randomly during preprocessing until collision-free on S. Store each $x \in S$ at position $A[h(x)]$.
- Space. $O(n^2)$.

## Solution 1: Collision-Free, Quadratic Space




- Queries.
  - LOOKUP(x): Check A[h(x)].
- Time. O(1).
- Preprocessing time?

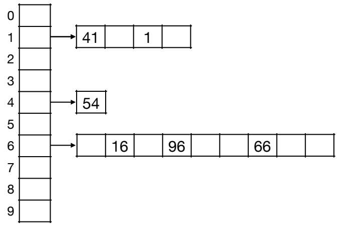
## Solution 1: Collision-Free, Quadratic Space

- Analysis.
- Let $I_{x,y} = \begin{cases} 1 & \text{if } h(y) = h(x) \\ 0 & \text{if } h(y) \neq h(x) \end{cases}$
- Let C = total number of collisions on S.
- $E(C) = E\left(\sum_{x,y \in S, x \neq y} I_{x,y}\right) = \sum_{x,y \in S, x \neq y} E\left(I_{x,y}\right) = \sum_{x,y \in S, x \neq y} \Pr\left(h(x) = h(y)\right) \leq \binom{n}{2}\frac{1}{n^2} < \frac{1}{2}$
- $\Longrightarrow$ With probability 1/2 we get perfect hashing function. If not perfect try again.
- $\Longrightarrow$ Expected number of trials before we get a perfect hash function is O(1).
- Theorem. We can solve the static dictionary problem in
  - $O(n^2)$ space and $O(n^2)$ expected time preprocessing time.
  - O(1) worst-case query time.

## Solution 2: Many Collisions, Linear Space.

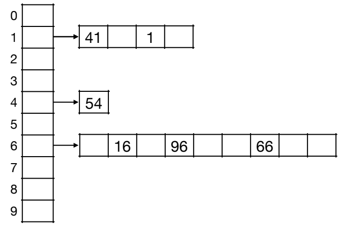
- As solution 1 but with an array of length n. What is the expected number of collisions?
- $E(C) = E\left(\sum_{x,y \in S, x \neq y} I_{x,y}\right) = \sum_{x,y \in S, x \neq y} E\left(I_{x,y}\right) = \sum_{x,y \in S, x \neq y} \Pr\left(h(x) = h(y)\right) \leq \binom{n}{2}\frac{1}{n} < \frac{n}{2}$

## Solution 3: FKS-Scheme.




- Data structure. Two-level solution.
  - At level 1 use solution with many collisions and linear space.
  - Resolve each collisions at level 1 with collision-free solution at level 2.
- Space?

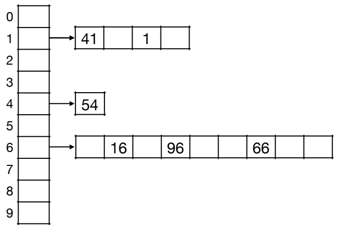
## Solution 3: FKS-Scheme.




- Queries.
  - LOOKUP(x): Check level 1 to find the correct level 2 dictionary. Lookup in level 2 dictionary.
- Time. O(1).

## Solution 3: FKS-Scheme.

- Space analysis. What is the the total size of level 1 and level 2 hash tables?
  - Let $S_i = \{x \in S \mid h(x) = i\}$
  - Let C = total number of collisions on level 1.
  - $C = \sum \binom{|S_i|}{2}$ by construction.
  - C = O(n) by solution 2.




- Space.

$a^2 = a + 2\binom{a}{2}$

$$O\left(n + \sum_i |S_i|^2\right) = O\left(n + \sum_i \left(|S_i| + 2\binom{|S_i|}{2}\right)\right)$$

- $$= O\left(n + \sum_i |S_i| + 2\sum_i \binom{|S_i|}{2}\right) = O(n + n + 2n) = O(n)$$

## Static Dictionaries and Perfect Hashing

- FKS scheme.
  - O(n) space and O(n) expected preprocessing time.
  - Lookups with two evaluations of a universal hash function.
- Theorem. We can solve the static dictionary problem for a set S of size n in
  - O(n) space and O(n) expected preprocessing time.
  - O(1) worst-case time per lookup.
- Multilevel data structures.
  - FKS is example of multilevel data structure technique. Combine different solutions for same problem to get an improved solution.

# Hashing

- Hashing Recap
- Dictionaries
- Perfect Hashing
- String Hashing

## String Hashing

- Define hash function on strings.
- Goals.
  - Low collision probability.
  - Fast evaluation.
  - Small space.
  - Fast string manipulation.

## String Hashing

- Karp-Rabin Fingerprint.
  - Let S be a string of length n. We view characters as digits and S as an integer.
  - Let p is a prime number. Pick uniformly at random integer z ∈{0, ..., p-1}.
  - The Karp-Rabin fingerprint of S is

$$\phi_{p,z}(S) = S[1]z^{n-1} + S[2]z^{n-2} + \cdots + S[n-1]z^1 + S[n] \mod p$$
$$= \left(\sum_{i=1}^{n} S[i] \cdot z^{n-i}\right) \mod p$$

- The fingerprint of S is the polynomial over the field $Z_p$ evaluated at the random integer z.

## String Hashing

- Theorem. (Collision probability) Let S and T be distinct strings of length s, and let p be a prime. For a random z ∈{0, ..., p-1}:

$$\Pr(\phi_{p,z}(S) = \phi_{p,z}(T)) \le \frac{s}{p}$$

- Proof.

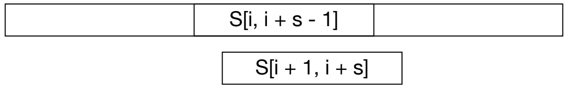
$$\Pr\left(\phi_{p,z}(S) = \phi_{p,z}(T)\right) = \Pr\left(\sum_{i=1}^{s} S[i] \cdot z^{s-i} = \sum_{i=1}^{s} T[i] \cdot z^{s-i} \mod p\right)$$
$$= \Pr\left(\sum_{i=1}^{s} (S[i] - T[i]) \cdot z^{s-i} = 0 \mod p\right)$$

  $\sum_{i=1}^{s} (S[i] - T[i]) \cdot z^{s-i}$ is a non-zero polynomial over $Z_p$ of degree s-1.

  - ⇒ It has at most s-1 roots ⇒ The probability that our random z is one of those is at most (s-1)/p < s/p.

## String Hashing

- Consider substrings of S of length s.




- Fingerprint computation. We can compute $\phi_{p,z}(S[i, i+s-1])$ in O(s) time.
  - Proof. See exercises.
- Rolling property. $\phi_{p,z}(S[i+1, i+s]) = (\phi_{p,z}(S[i, i+s-1]) - S[i]z^{s-1})z + S[i+s] \mod p$
  - Proof. See exercises.
- ⇒ We can compute $\phi_{p,z}(S[i+1, i+s])$ from $\phi_{p,z}(S[i, i+s-1])$ in constant time.

## String Hashing

- String matching. Given strings S and P, determine if P is a substring in S.

S = yabbadabbado

P = abba

- What solutions do we know? |P| = m, |S| = n.
  - Brute force comparison: O(nm) time
  - Knuth-Morris-Pratt algorithm [KMP1977]: O(n + m) time.

## String Hashing

S = yabbadabbado

P = abba

- Karp-Rabin Algorithm.
  - Pick $p \geq m^2$.
  - Compute $\phi(P)$.
  - Compute and compare $\phi(S[i, i+m-1])$ with $\phi(P)$ for all i.
  - If fingerprints match, verify using brute-force comparison. Return “yes!” if we match.
- Time.
  - Let F be the number of collisions, i.e., S[i, i + m - 1] ≠ P but $\phi(S[i, i+m-1]) = \phi(P)$.
  - ⇒ O(n + m + Fm).

## String Hashing

S = yabbadabbado

P = abba

- Expected number of collisions.
  - The probability of collision at a single substring is m/p ≤ 1/m.
  - ⇒ Expected number of collision on all n-m+1 substrings ≤ (n-m+1)/m < n/m.
- ⇒ Expected time is O(n + m + mn/m) = O(n + m).

## String Hashing

- Theorem. We can solve the string matching problem in O(n + m) time expected time.

- String matching with Karp-Rabin fingerprints.
  - Simple, practical, fast.
  - More techniques ⇒ Fast reporting, small space, real-time, streaming, etc.

## Hashing

- Hashing
- Dictionaries
- Perfect Hashing
- String Hashing