Advanced topics on Algorithms

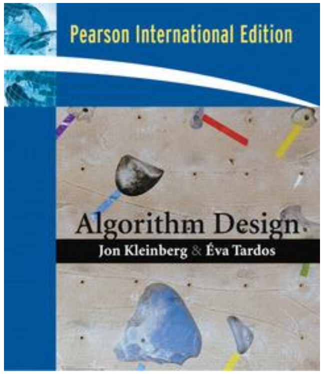
Luciano Gualà
www.mat.uniroma2.it/~guala/

Algorithms for Big Data
Episode I

Hash tables

A randomized implementation of dictionaries

reference
(Chapter 13.6)



+

Design and Analysis of Algorithms
(MIT opencourseware)
Lecture 8

https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/lecture-8-randomization-universal-perfect-hashing/

The dictionary problem:

Given a universe U of possible elements, maintain a subset $S \subseteq U$ subject to the following operations:

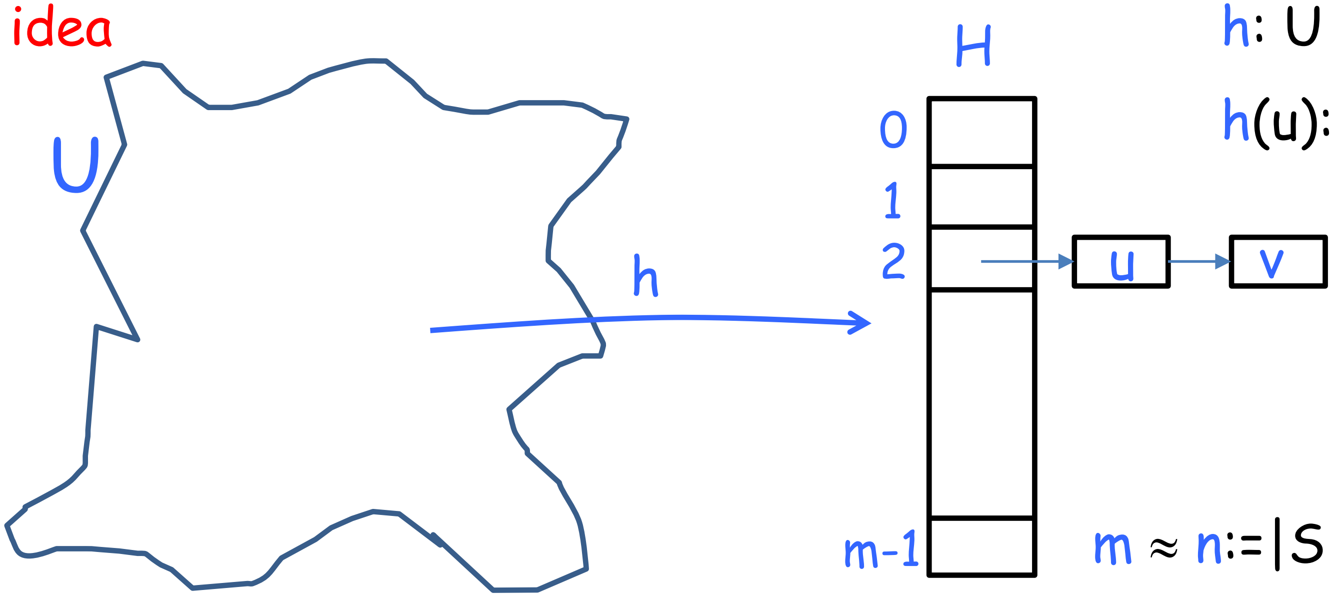
- make-dictionary(): Initialize an empty dictionary.
- insert(u): Add element $u \in U$ to S.
- delete(u): Delete u from S, if u is currently in S.
- look-up(u): Determine whether u is in S.

Challenge: Universe U can be extremely large so defining an array of size $|U|$ is infeasible.

A solution: balanced (e.g. AVL) trees
- $O(|S|)$ space
- $O(\log |S|)$ time per operation

hash tables:
- $O(|S|)$ space
- $O(1)$ expected time per operation

idea



h: U ⟶ {0,1,...,m-1}

h(u): slot where u is put

$m \approx n := |S|$

collision: when h(u) = h(v) but u ≠ v.

H[i]: linked list of all elements that h maps to slot i
(hashing with chaining)

Insert/Delete/Lookup of u:

- compute h(u)
- insert/delete/search u by scanning list H[h(u)]

goal: find a function h that "spreads out" elements

choosing a good hash function

obs: for any deterministic hash function one can find a set S where all elements of S are mapped to the same slot

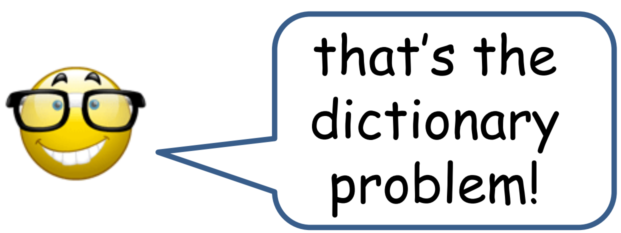
⇒ Θ(n) time per operation

idea: use randomization

obvious approach: for each u, choose h(u) uniformly at random

look-up(u): ...where did we put u?

we have to maintain the set of pairs $\{(u,h(u)): u \in S\}$



universal hashing

A family $\mathcal{H}$ of hash functions is universal if

for each distinct $u,v \in U$ $\qquad \Pr_{h \in \mathcal{H}}(h(u)=h(v)) \leq 1/m$

Theorem

Let $\mathcal{H}$ be a family of universal hash functions. Let $S \subseteq U$ of n elements. Let $u \in S$. Pick a random function h from $\mathcal{H}$, and let X be the random variable counting the number of elements of S mapped to h(u). Then $E[X] \leq 1+n/m$

proof

for each $s \in S$ $\quad X_s$ r. v. = $\begin{cases} 1 & \text{if } h(s)=h(u) \\ 0 & \text{otherwise} \end{cases}$ $\qquad X = \sum_{s \in S} X_s$

$$E[X] = E\left[\sum_{s \in S} X_s\right] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr(h(s)=h(u))$$

$$= 1 + \sum_{s \in S \setminus \{u\}} \Pr(h(s)=h(u)) \leq 1 + n/m$$

■

notice: $m=\Theta(n)$ ⇒ expected O(1) time per operation

designing a universal family of hash functions

always exists [Chebyshev 1850]

Table size: choose m as a prime number such that $n \le m \le 2n$

Integer encoding: Identify each element $x \in U$ with a base-m integer of r digits: $x = (x_1, x_2, \ldots, x_r)$.

Hash function:
given $a \in U$, $a = (a_1, a_2, \ldots, a_r)$

$$h_a(x) = \left[\sum_{i=1}^{r} a_i x_i \right] \bmod m$$

hash function family: $\mathcal{H} = \{h_a : a \in U\}$

word RAM model:

- manipulating O(1) machine words takes O(1) time
- every object of interest fits in a machine word

⇒

- storing $h_a(x)$ requires just storing a single value, a (1 machine word)
- computing $h_a(x)$ takes O(1) time

Theorem

$\mathcal{H} = \{h_a : a \in U\}$ is universal

proof

Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. We need to show that $\Pr[h_a(x) = h_a(y)] \le 1/m$.

since $x \neq y$, there exists an integer j such that $x_j \neq y_j$.

we have $h_a(x) = h_a(y)$ iff

$$\underbrace{a_j\,(y_j - x_j)}_{z} = \underbrace{\sum_{i \neq j} a_i(x_i - y_i)}_{\alpha} \mod m$$

we can assume a was chosen uniformly at random by first selecting all coordinates a_i where $i \neq j$, then selecting a_j at random. Thus, we can assume a_i is fixed for all coordinates $i \neq j$.

since m is prime & $z \neq 0$, z has a multiplicative inverse z^{-1}, i.e. $z\,z^{-1} = 1 \mod m$

Theorem

$\mathcal{H} = \{h_a : a \in U\}$ is universal

proof

Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. We need to show that $\Pr[h_a(x) = h_a(y)] \leq 1/m$.

since $x \neq y$, there exists an integer j such that $x_j \neq y_j$.

we have $h_a(x) = h_a(y)$ iff

$$a_j = z^{-1} \underbrace{\sum_{i \neq j} a_i(x_i - y_i)}_{\alpha} \mod m$$

we can assume a was chosen uniformly at random by first selecting all coordinates a_i where $i \neq j$, then selecting a_j at random. Thus, we can assume a_i is fixed for all coordinates $i \neq j$.

since m is prime & $z \neq 0$, z has a multiplicative inverse z^{-1}, i.e. $z z^{-1} = 1 \mod m$

⇒ $\Pr[h_a(x) = h_a(y)] \leq 1/m$.

■

another universal hash family

choose a prime $p \geq |U|$ (once)

Hash function:

given $a,b \in U$,

$$h_{ab}(x) = [(ax+b) \bmod p] \bmod m$$

hash function family: $\mathcal{H} = \{h_{ab} : a,b \in U\}$

how to (dynamically) choose the table size

notice: S changes over time and we want to use $O(|S|)$ space

parameters:

- n: # of elements currently in the table, i.e. $n=|S|$;
- N: virtual size of the table
- m: actual size of the table (a prime number between N and 2N)

doubling/halving technique:

- init $n=N=1$;
- whenever $n>N$:
  - $N:=2N$
  - choose a new m
  - re-hash all items (in $O(n)$ time)
- whenever $n<N/4$:
  - $N:=N/2$
  - choose a new m
  - re-hash all items (in $O(n)$ time)

⇒ $O(1)$ amortized time per insertion/deletion

perfect hashing

optimal static dictionary

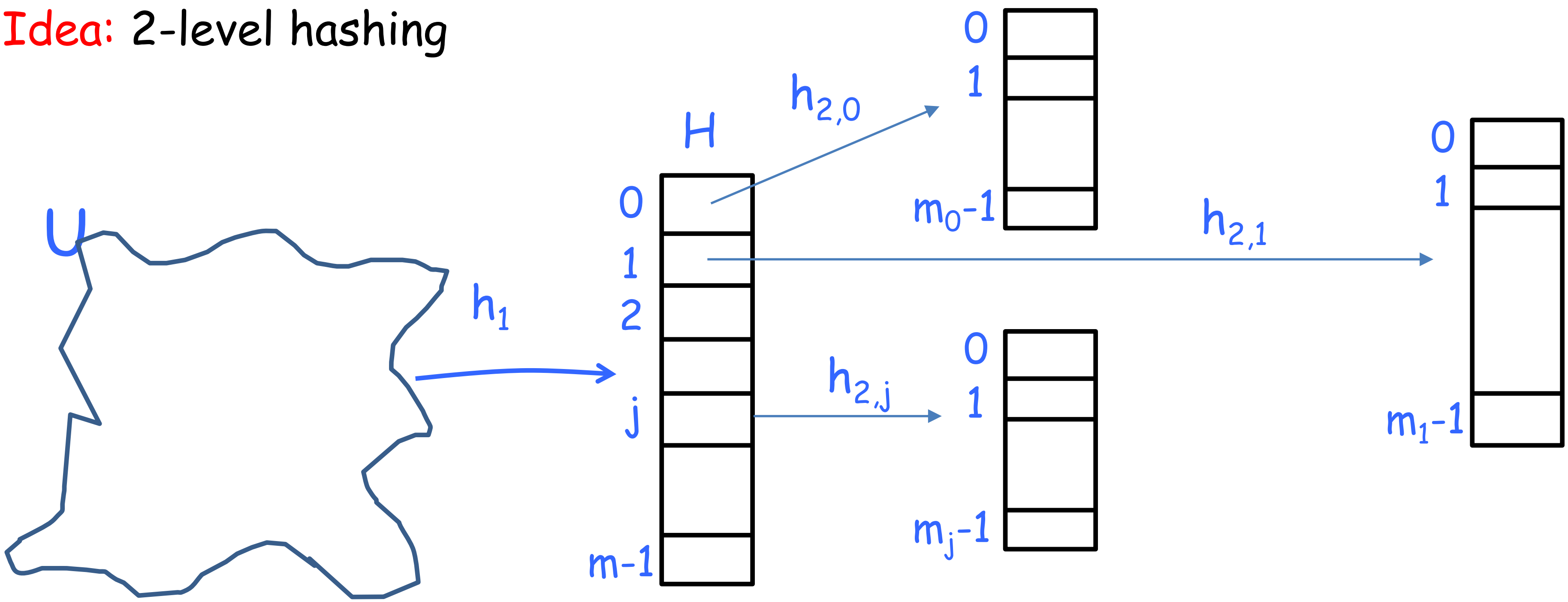
The static dictionary problem:

given a set S of n elements (keys), build a data structure supporting search operations.

Perfect Hashing:

- $O(1)$ worst-case time per search
- space $O(n)$
- build time: almost linear with high probability

Idea: 2-level hashing



$m = \Theta(n)$

Building the dictionary

Step 1:

- pick h_1: $U \longrightarrow \{0,1,\ldots,m-1\}$ u.a.r. from a universal hash family, with $m=\Theta(n)$ (e.g. nearby prime)
- hash all items with chaining using h_1

Step 2:

for each $j \in \{0,1,\ldots,m-1\}$

- n_j: # of elements mapped to j by h_1
- pick $h_{2,j}$: $U \longrightarrow \{0,1,\ldots, m_j -1\}$ u.a.r. from a universal hash family, with $n_j^2 \leq m_j \leq O(n_j^2)$
- replace linked list for slot j with a hash table of size m_j using $h_{2,j}$.

Building the dictionary

Step 1:

- pick $h_1 : U \longrightarrow \{0,1,\ldots,m-1\}$ u.a.r. from a universal hash family, with $m=\Theta(n)$ (e.g. nearby prime)
- hash all items with chaining using h_1

Step 1.5: if $\sum_{j=0}^{m-1} n_j^2 > c\,n$ for some c (chosen later) redo Step 1

Step 2:

for each $j \in \{0,1,\ldots,m-1\}$

- n_j: # of elements mapped to j by h_1
- pick $h_{2,j} : U \longrightarrow \{0,1,\ldots, m_j -1\}$ u.a.r. from a universal hash family, with $n_j^2 \leq m_j \leq O(n_j^2)$
- replace linked list for slot j with a hash table of size m_j using $h_{2,j}$.

Step 2.5:

while $h_{2,j}(u) = h_{2,j}(v)$ for some $u \neq v$ with $h_1(u)=h_1(v)$

- repick $h_{2,j}$ and re-hash all those n_j elements

$\Rightarrow$ no collision at second level & linear size

Building time

Step 1&2 take O(n) time

Step 2.5

$$\Pr_{h_{2,j}} \{ h_{2,j}(u) = h_{2,j}(v), \text{ for some } u \neq v \} \leq$$

$$\sum_{\substack{u,v \in S \\ u \neq v \\ \text{with } h_1(u) = h_1(v)}} \underbrace{\Pr\{ h_{2,j}(u) = h_{2,j}(v) \}}_{\leq 1/n_j^2} \leq \tfrac{1}{2} n_j (n_j - 1) \; 1/n_j^2 < 1/2$$

for each j:

- $E[\# \text{ trials}] \leq 2$
- O(log n) trials w.h.p.
- each trial takes $O(n_j)$ time

time for Step 2.5:

$$\sum_j (\# \text{ trials for } j) \, O(n_j) \Rightarrow$$

O(n log n) with high probability

Building time (Step 1.5)

Idea: we show that $E\left[\sum_{j=0}^{m-1} n_j^2\right] = \Theta(n)$ and then we use Markov's inequality

$$X_{u,v} \text{ r. v.} = \begin{cases} 1 & \text{if } h_1(u)=h_1(v) \\ 0 & \text{otherwise} \end{cases} \qquad \sum_{j=0}^{m-1} n_j^2 = \sum_{u\in S}\sum_{v\in S} X_{u,v}$$

$$E\left[\sum_{j=0}^{m-1} n_j^2\right] = \sum_{u\in S}\sum_{v\in S} E[X_{u,v}] = \sum_{u\in S}\sum_{v\in S} \Pr\{h_1(u)=h_1(v)\} \le n + n^2/m \le 2n$$

$$\Pr\left\{\sum_{j=0}^{m-1} n_j^2 > c\,n\right\} \le \frac{E\left[\sum_{j=0}^{m-1} n_j^2\right]}{c\,n} \le \frac{2n}{c\,n} \le 1/2$$

by suitably choosing c

- E[# trials]≤2
- O(log n) trials w.h.p.
- each trial takes O(n)

⇒ **O(n log n) with high probability**