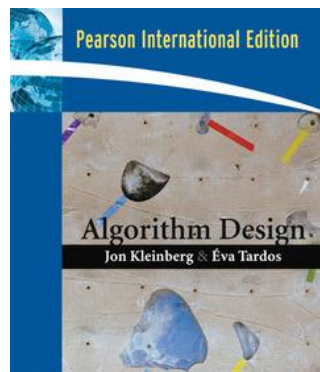


Hash tables

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.

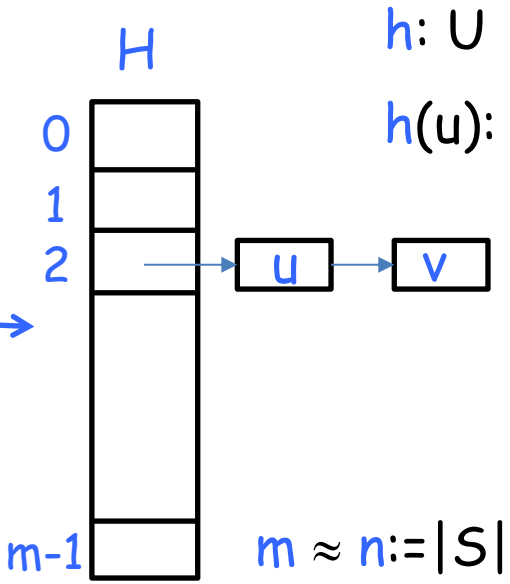
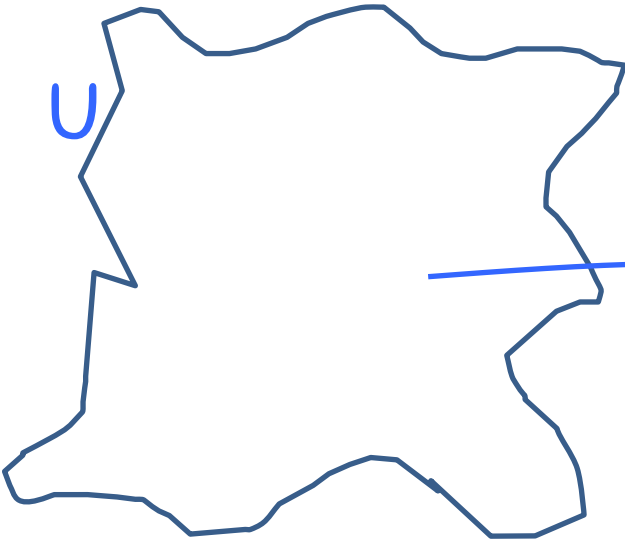
A solution: balanced (e.g. AVL) trees

- $O(|S|)$ space
- $O(\log |S|)$ time per operation

hash tables:

- $O(|S|)$ space
- $O(1)$ expected amortized time per operation

idea



$$h: U \longrightarrow \{0,1,\dots,m-1\}$$

$h(u)$: slot where u is put

collision: when $h(u) = h(v)$ but $u \neq v$.

$H[i]$: 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

➔ $\Theta(n)$ 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 $\{(u, h(u)): u \in S\}$



that's the
dictionary
problem!



maybe I
can use a
hash table

universal hashing

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

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

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

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

$$\begin{aligned} E[X] &= E\left[\sum_{s \in S} X_s\right] = \sum_{s \in S} E[X_s] = \sum_{s \in S} \Pr(h(s)=h(u)) \\ &= 1 + \sum_{s \in S \setminus \{u\}} \Pr(h(s)=h(u)) \leq 1 + n/m \end{aligned}$$



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

designing a universal family of hash functions

always exists
[Chebyshev 1850]

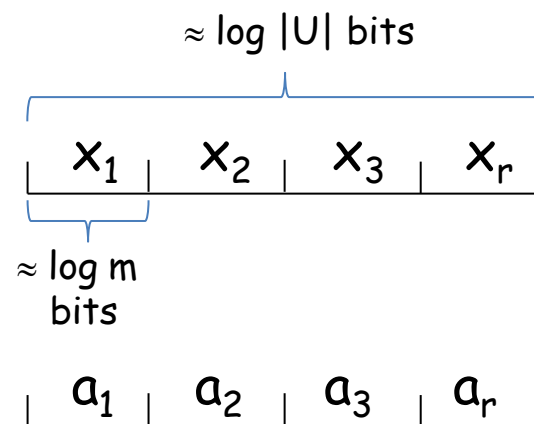
Table size: choose m as a prime number such that $n \leq m \leq 2n$

Integer encoding: Identify each element $x \in U$ with a base- m integer of r digits: $x = (x_1, x_2, \dots, x_r)$, $x_i \in \{0, 1, \dots, m-1\}$.

Hash function:

given $a \in U$, $a = (a_1, a_2, \dots, a_r)$

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



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 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 $x = (x_1, x_2, \dots, x_r)$ and $y = (y_1, y_2, \dots, y_r)$ 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

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

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. $z z^{-1} = 1 \pmod m$

Theorem

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

proof

Let $x = (x_1, x_2, \dots, x_r)$ and $y = (y_1, y_2, \dots, y_r)$ 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

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

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. $z z^{-1} = 1 \pmod m$

 $\Pr[h_a(x) = h_a(y)] \leq 1/m$.



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

What's next?

randomization: an amazing algorithmic tool

- hash functions/tables
- sampling
- sketches
- ...

it allows to solve efficiently many practical important problems:

counting distinct elements in a stream:

given a stream of m element x_1, x_2, \dots, x_m where each $x_i \in U$, return the number d of distinct elements

simple solution: $O(d)$ space or $O(|U|)$ space

probabilistic counters: $O(\log d)$ space

finding similar items

Given N items, find pairs of them whose similarity is above a give threshold

main challenge: N is huge and a $\Theta(N^2)$ -time solution is infeasible

LSH technique: solves the problem in $O(N \text{ polylog } N)$ time and space

randomization: an amazing algorithmic tool

- hash functions/tables
- sampling
- sketches
- ...

it allows to solve efficiently many practical important problems:

counting dist

given a stream
number d of c

simple solution

probabilistic

if interested:

in the

Algorithms for Big Data
(prof. Clementi)

finding similar items

Given N items, find pairs of them whose similarity is above a give threshold

main challenge: N is huge and a $\Theta(N^2)$ -time solution is infeasible

LSH technique: solves the problem in $O(N \text{ polylog } N)$ time and space