Universal Hashing

Universal Hashing

Dictionary Data Type

Dictionary. Given a universe U of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in S is efficient.

Dictionary interface.

- `Create()`: Initialize a dictionary with $S = \phi$.
- `Insert(u)`: Add element $u \in U$ to S.
- `Delete(u)`: Delete u from S, if u is currently in S.
- `Lookup(u)`: Determine whether u is in S.

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

Applications. File systems, databases, Google, compilers, checksums P2P networks, associative arrays, cryptography, web caching, etc.

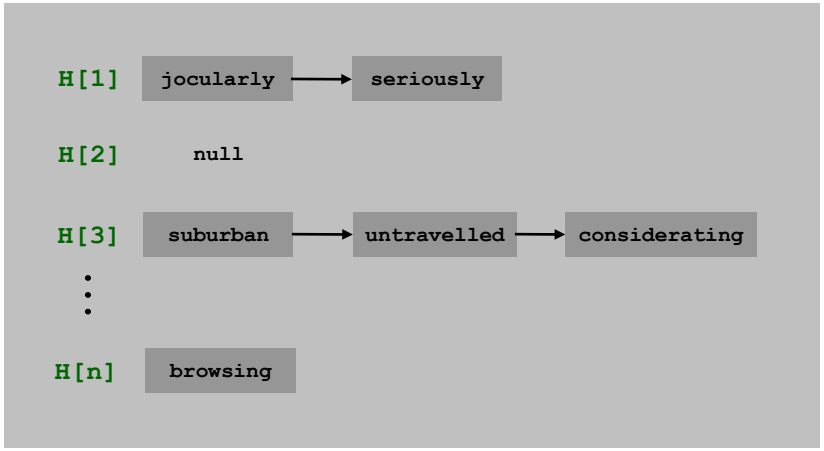
Hashing

Hash function. $h : U \rightarrow \{ 0, 1, \ldots, n-1 \}$.

Hashing. Create an array H of size n. When processing element u, access array element H[h(u)].

Collision. When $h(u) = h(v)$ but $u \neq v$.

- A collision is expected after $\Theta(\sqrt{n})$ random insertions. This phenomenon is known as the "birthday paradox."
- Separate chaining: H[i] stores linked list of elements u with $h(u) = i$.



Ad Hoc Hash Function

Ad hoc hash function.

```
int h(String s, int n) {
   int hash = 0;
   for (int i = 0; i < s.length(); i++)
      hash = (31 * hash) + s[i];
   return hash % n;
}
```

hash function ala Java string library

Deterministic hashing. If $|U| \geq n^2$, then for any fixed hash function h, there is a subset $S \subseteq U$ of n elements that all hash to same slot. Thus, $\Theta(n)$ time per search in worst-case.

Q. But isn't ad hoc hash function good enough in practice?

Idealistic hash function

Idealistic hash function. Maps m elements uniformly at random to n hash slots.

- Running time depends on length of chains.
- Average length of chain = α = m / n.
- Choose n $\approx$ m $\Rightarrow$ on average O(1) per insert, lookup, or delete.

Theorem. If we map each element u.a.r to n hash slots, then for any k and any distinct numbers $x_1,\ldots,x_k$ and any numbers $y_1,\ldots,y_k$, we have

$$\Pr(h(x_1)=y_1,\ h(x_2)=y_2,\ldots,h(x_k)=y_k)=1/n^k$$

And moreover, for any u and v we have Pr(h(u)=h(v))=1/n

There is a challenge (will be explained next) to reach this goal.

adversary knows the randomized algorithm you're using,
but doesn't know random choices that the algorithm makes

Challange

Challenge. Since we may see several occurrence of j, we should remember where j is mapped upon the arrival of the first occurrence of j.

Two methods to resolve this:

- We have to maintain all h(j) in a data structure which of course needs a linear data structure, or

- we use an explicit formula for the hash function. It seems we have a fix function and each input is uniquely mapped to an element of [n]

Challange

Offline fashion. Instead of mapping j u.a.r. to an element in [n] in the online fashion, we can do all randomness in the offline fashion,i.e., at the beginning, we specify h(j) u.a.t. instead of waiting to receive the first j and specify h(j).

Example. for planning a single-elimination knockout tournament over n teams, we can construct a tree with n leaves and put a random permutation of teams in the leaves or in each round the opponent of each team is specified u.a.r. The result is the same meaning that the probability that two teams meet each other is the same in both methods.

- This method is equivalent to select a function h: from U to [n] u.a.r. from all functions from U to [n] (#such functions = $n^{|U|}$) .
- If we do that, we don't have an explicit formula for h and we need a data structure to maintain h.

K-universal hash Family and Universal Hash Family

Approach. A practical solution is to put some functions (not all functions) with explicit formula into a set H (called a family of hash functions) and select one of them u.a.r. at the beginning (offline fashion).

Remark. Even if a adversary knows your hash family (before running your program), he can not detect which hash function is used by your program when it starts running.

K-universal hash family: H is a k-universal hash family if for any k distinct number $x_1,\ldots,x_k$ we have

$$\Pr(h(x_1)= h(x_2)=\ldots=h(x_k)) \le 1/n^{k-1}$$

Strongly K-universal hash family: H is a strongly k-universal hash family if for any k distinct number $x_1,\ldots,x_k$ and any k numbers $y_1,\ldots,y_k$ we have

$$\Pr(h(x_1)=y_1,\ h(x_2)=y_2,\ldots,h(x_k)=y_k) = 1/n^k$$

2-Universal hash family: H is a 2-universal hash family if for any distinct number u and v we have

$$\Pr(h(u)=h(v)) \le 1/n$$

K-universal hash Family and Universal Hash Family

- For most applications, having a 2-universal hashing family suffices.
- How to show the family H is 2-universal: for any distinct u and v, count the number of hash functions h that h(u) = h(v). Divide this number by |H|. If it always (for any distinct u and v) is equal or less than 1/n, H is universal.
- Note that any strongly 2-universal hash family is a 2-universal family but the reverse is not necessarily true.

Universal Hashing

.Ex. U = { a, b, c, d, e, f }, n = 2.

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1

$H = \{h_1, h_2\}$
$\Pr_{h \in H}[h(a) = h(b)] = 1/2$
$\Pr_{h \in H}[h(a) = h(c)] = 1$
$\Pr_{h \in H}[h(a) = h(d)] = 0$
. . .

not 2-universal

	a	b	c	d	e	f
$h_1(x)$	0	1	0	1	0	1
$h_2(x)$	0	0	0	1	1	1
$h_3(x)$	0	0	1	0	1	1
$h_4(x)$	1	0	0	1	1	0

$H = \{h_1, h_2, h_3, h_4\}$
$\Pr_{h \in H}[h(a) = h(b)] = 1/2$
$\Pr_{h \in H}[h(a) = h(c)] = 1/2$
$\Pr_{h \in H}[h(a) = h(d)] = 1/2$
$\Pr_{h \in H}[h(a) = h(e)] = 1/2$
$\Pr_{h \in H}[h(a) = h(f)] = 0$
. . .

2-universal

Universal Hashing

Universal hashing property. Let H be a 2-universal class of hash functions; let $h \in H$ be chosen uniformly at random from H; and let $u \in U$. For any subset $S \subseteq U$ of size at most n, the expected number of items in S that collide with u is at most 1.

Pf. For any element $s \in S$, define indicator random variable $X_s = 1$ if h(s) = h(u) and 0 otherwise. Let X be a random variable counting the total number of collisions with u.

$$E_{h \text{ in } H}[X] = E\left[\sum X_s\right] = \sum E[X_s] = \sum \Pr[X_s = 1] \leq \sum \frac{1}{n} = |S|\frac{1}{n} \leq 1$$

linearity of expectation

X_s is a 0-1 random variable

universal (assumes $u \notin S$)

Designing a Universal Family of Hash Functions

Theorem. [Chebyshev 1850] There exists a prime between n and 2n.

Modulus. Choose a prime number $p \approx n$. ← no need for randomness here

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

Hash function. Let A = set of all r-digit, base-p integers. For each $a = (a_1, a_2, \ldots, a_r)$ where $0 \le a_i < p$, define

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

Hash function family. $H = \{ h_a : a \in A \}$.

Designing a Universal Class of Hash Functions

Theorem. $H = \{ h_a : a \in A \}$ is a universal class of hash functions.

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

- 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

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

- 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 p is prime, $a_j z = m \bmod p$ has at most one solution among p possibilities. ← see lemma on next slide
- Thus $\Pr[h_a(x) = h_a(y)] = 1/p \leq 1/n$. ▪

Number Theory Facts

Fact. Let p be prime, and let $z \neq 0 \bmod p$. Then $\alpha z = m \bmod p$ has at most one solution $0 \leq \alpha < p$.

Pf.

- Suppose α and β are two different solutions.
- Then $(\alpha - \beta)z = 0 \bmod p$; hence $(\alpha - \beta)z$ is divisible by p.
- Since $z \neq 0 \bmod p$, we know that z is not divisible by p; it follows that $(\alpha - \beta)$ is divisible by p.
- This implies $\alpha = \beta$. ▪

Bonus fact. Can replace "at most one" with "exactly one" in above fact.
Pf idea. Euclid's algorithm.

References

References

- Section 13.3 of the text book "Probability and Computing" by Mitzenmacher and Upfal
- The orginal slides were prepared by Kevin Wayne. The slides are distributed by Pearson Addison-Wesley.