

Lecture 5: Hashing

Hashing is not only useful in designing efficient data structures for searching, but is also useful in designing data streaming algorithms and in derandomization.

An important concept that we will study is the notion of k -wise independent random variables.

We will see applications in data streaming in the next lecture.

Hash Functions

A primary motivation of designing a hash function is to obtain an efficient data structure for searching.

We would like to achieve $O(1)$ search time using the RAM (random access machine) model.

In the RAM model, we assume that we can access an arbitrary position in an array in $O(1)$ time.

We also assume that the word size is large enough and it takes $O(1)$ time for a word operation.

Consider the scenario that we would like to store n elements (keys) from the set $\{0, 1, \dots, M-1\}$.

An obvious approach is to use an array A of m elements, initially $A[i]=0$ for all i , and when a key is inserted, we set $A[i]=1$, and this supports searching in $O(1)$ time in the RAM model.

But, of course, this approach requires too much space, when $m \gg n$.

For example, consider the scenario that M is the set of all IP-addresses and n is the number of IP addresses visiting Waterloo per day.

Ideally, we would like to use only an array of size $O(n)$ and still supports searching in $O(1)$ time.

A hash function is used to map the elements of the big universe to the locations in the small table.

A hash table is a data structure that consists of the following components:

- a table T with n cells indexed by $N = \{0, 1, \dots, n-1\}$, each cell can store $O(\log m)$ bits.
- a hash function $h: M \rightarrow N$.

Ideally, we want the hash function to map different keys into different locations.

But, of course, by the pigeonhole principle, this is impossible to achieve if we do not know the keys in advance.

We say there is a collision if $x \neq y$ but $h(x) = h(y)$.

Instead, what we can hope for is to have a family of hash functions, so that the number of collisions is small with high probability, if we pick a random hash function from the family.

We assume the keys coming are independent from the hash function that we choose (i.e. there is no

"adversary" who knows our internal randomness and choose a bad set of keys for our hash functions). We do not assume that we know the keys in advance . (Actually, even if we know the keys in advance, it is non-trivial to design a good hash function ; see perfect hashing later.)

A natural idea is to consider the hash family being the set of all functions from M to N , and we just pick a random function $h: M \rightarrow N$ as our hash function.

But the problem is that it takes too much space to store a random hash function .

We need to find a tradeoff between time/space efficiency and the randomness achieved.

k-wise independence

For a set of n (fully) independent random variables, they satisfy $\Pr(\bigcap_{i=1}^n X_i = x_i) = \prod_{i=1}^n \Pr(X_i = x_i)$.

k -wise independence is a weaker notion where it only requires the above conditions to hold for any subset of (up to) k variables, rather than all possible subsets of variables.

Definition A set of random variables X_1, X_2, \dots, X_n is k -wise independent if for any subset $I \subseteq [1, n]$ with $|I| \leq k$ and for any values $x_i, i \in I$, it holds that $\Pr(\bigcap_{i \in I} (X_i = x_i)) = \prod_{i \in I} \Pr(X_i = x_i)$.

The special case when $k=2$ is called pairwise independence.

We see two examples why pairwise independent random variables are easier to be generated.

Example 1 Given b random bits X_1, \dots, X_b , we can generate $2^b - 1$ pairwise independent bits as follows:

Enumerate the $2^b - 1$ non-empty subsets of $\{1, 2, \dots, b\}$.

For each such subset S , define $Y_S = \bigoplus_{i \in S} X_i$, where \oplus denotes the mod-2 operation.

Then, it is easy to see that each bit Y_S is random by the principle of deferred decisions.

(The principle of deferred decision says that we can fix a bit x_i with $i \in S$, and think of the process of choosing a random string x_1, \dots, x_b as first deciding the value of other bits and then deciding the value of the i -th bit. By fixing all other bits, since $x_i = 1$ or 0 with probability $1/2$, Y_S is still equally likely to be 0 or 1 .)

More formally, this principle is just saying that $\Pr(Y_S = 1 \mid \text{all bits except } x_i) = \frac{1}{2}$ for all possible values of the remaining bits implies that $\Pr(Y_S = 1) = \frac{1}{2}$, and it follows from conditional probability.)

Consider two subsets S_1 and S_2 . We would like to show that the two bits Y_{S_1} and Y_{S_2} are independent.

We assume without loss of generality that $S_1 - S_2 \neq \emptyset$. Let $x_i \in S_1 - S_2$.

By the principle of deferred decision on x_i , we can argue that $\Pr(Y_{S_1} = c \mid Y_{S_2} = d) = \frac{1}{2}$ for any $c, d \in \{0, 1\}$.

Therefore, $\Pr(Y_{S_1}=c \cap Y_{S_2}=d) = \Pr(Y_{S_1}=c \mid Y_{S_2}=d) \Pr(Y_{S_2}=d) = \frac{1}{2} \cdot \frac{1}{2} = \Pr(Y_{S_1}=c) \cdot \Pr(Y_{S_2}=d)$,

which proves the pairwise independence of Y_S .

Are the variables Y_S 3-wise independent?

Example 2 Given two independent uniformly random variables X_1, X_2 over $\{0, 1, \dots, p-1\}$, we can generate p pairwise independent random variables by setting $Y_i = (X_1 + iX_2) \bmod p$ for $i=0, 1, \dots, p-1$, where p is a prime.

Again, by the principle of deferred decisions, Y_i is uniformly random, i.e. $\Pr(Y_i=a) = \frac{1}{p}$ for any $0 \leq a \leq p-1$.

For pairwise independence, we want to show that $\Pr(Y_i=a \cap Y_j=b) = \Pr(Y_i=a) \cdot \Pr(Y_j=b) = \frac{1}{p^2}$.

The event $Y_i=a$ and $Y_j=b$ is equivalent to $(X_1 + iX_2) \bmod p = a$ and $(X_1 + jX_2) \bmod p = b$.

These two equations on two variables have a unique solution (because p is prime and so multiplicative inverse exist)

$$X_2 = (b-a)(j-i)^{-1} \bmod p \quad \text{and} \quad X_1 = (a - i(b-a)(j-i)^{-1}) \bmod p.$$

Since there are p^2 choices for X_1 and X_2 , and there is only one choice for $Y_i=a$ and $Y_j=b$,

this implies that $\Pr(Y_i=a \text{ and } Y_j=b) = \frac{1}{p^2}$, proving the pairwise independence.

Intuitively, this construction can be thought of as generating a random line over a field, i.e. if

we only know one point Y_i of the line then another point Y_j is still random, but once we know two points of the line then we can figure out all other points of the line.

Note that in both examples, we used significantly fewer random bits to generate many random variables.

Chebyshev's inequality for sum of pairwise independent random variables

We cannot apply Chernoff bounds for pairwise independent random variables.

An important remark is that Chebyshev's inequality still applies for pairwise independent random variables.

This is because for pairwise independent variables X_1, X_2, \dots, X_n , we have $E[X_i X_j] = E[X_i] E[X_j] \quad \forall i \neq j$.

This implies that $\text{Cov}(X_i, X_j) = 0 \quad \forall i \neq j$, and so $\text{Var}[\sum_{i=1}^n X_i] = \sum_{i=1}^n \text{Var}[X_i]$ for pairwise independent variables.

Applying Chebyshev's inequality gives $\Pr(|X - E[X]| \geq a) \leq \text{Var}[X] / a^2 = \frac{\sum_{i=1}^n \text{Var}[X_i]}{a^2}$ for $X = \sum_{i=1}^n X_i$.

This will be important in analysis of data streaming algorithms.

Universal Hash Functions

Definition Let U be a universe with $|U| \geq n$ and let $V = \{0, 1, \dots, n-1\}$. A family of hash functions

\mathcal{H} from U to V is said to be k-universal if, for any distinct elements x_1, x_2, \dots, x_k and a

hash function h chosen uniformly random from \mathcal{H} , we have $\Pr_{h \in \mathcal{H}}(h(x_1) = h(x_2) = \dots = h(x_k)) \leq \frac{1}{n^{k-1}}$.

\mathcal{H} is said to be strongly k-universal if for any values $y_1, y_2, \dots, y_k \in \{0, 1, \dots, n-1\}$ and a random

hash function h from \mathcal{H} , we have $\Pr_{h \in \mathcal{H}} (h(x_1)=y_1, h(x_2)=y_2, \dots, h(x_k)=y_k) = \frac{1}{n^k}$ for distinct x_i .

We can think of \mathcal{H} is strongly k -universal if the random variables $h(0), h(1), \dots, h(|U|-1)$ are k -wise independent, when h is chosen uniformly random from \mathcal{H} .

With this connection, it is not surprising that the constructions for generating k -wise independent random variables can be used to construct universal hash functions.

We will focus on 2-universal and strongly 2-universal hash families.

2-universal and strongly 2-universal families of hash functions

We begin with a construction where the size of the universe and the size of the table are the same.

Let $U=V = \{0, 1, \dots, p-1\}$ where p is a prime number.

Let $h_{a,b}(x) := (ax+b) \bmod p$.

Let $\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b \leq p-1\}$.

Claim \mathcal{H} is strongly 2-universal.

Proof We need to prove that $\Pr_{a,b} ((h_{a,b}(x_1)=y_1) \cap (h_{a,b}(x_2)=y_2)) = \frac{1}{p^2}$ for any y_1, y_2 and any $x_1 \neq x_2$.

Satisfying $h_{a,b}(x_1)=y_1$ and $h_{a,b}(x_2)=y_2$ means $(ax_1+b) \bmod p = y_1$ and $(ax_2+b) \bmod p = y_2$.

Given x_1, x_2, y_1, y_2 , these are two linear equations with two variables, and there is a unique solution:

$$a = (y_2 - y_1)(x_2 - x_1)^{-1} \bmod p \quad \text{and} \quad b = (y_1 - ax_1) \bmod p.$$

Hence, there is only one choice of a, b out of p^2 possibilities satisfying the conditions, proving the claim. \square

The above construction only defines a strongly 2-universal hash family when $|U|=|V|$.

In applications, we usually want to define a mapping from a large universe to a small table.

There is an easy way to extend the above construction to this setting.

Let $U = \{0, 1, \dots, p^k-1\}$ and $V = \{0, 1, \dots, p-1\}$ for some positive integer k and some prime p .

Interpret each element $u \in U$ as a vector $\vec{u} = (u_0, \dots, u_{k-1})$ where $0 \leq u_i \leq p-1$ and $\sum_{i=0}^{k-1} u_i p^i = u$.

In other words, interpret u as a " p -ary" number where u_0 is the least significant "digit"

and u_{k-1} is the most significant "digit".

For any vector $\vec{a} = (a_0, \dots, a_{k-1})$ where $0 \leq a_i \leq p-1$ for every $0 \leq i \leq k-1$, and any $0 \leq b \leq p-1$,

$$\text{let } h_{\vec{a},b}(\vec{u}) = \left(\sum_{i=0}^{k-1} a_i u_i + b \right) \bmod p.$$

Let $\mathcal{H} = \{h_{\vec{a},b} \mid 0 \leq a_i \leq p-1 \text{ for every } 0 \leq i \leq k-1, \text{ and } 0 \leq b \leq p-1\}$.

Claim: \mathcal{H} is strongly 2-universal.

Proof: We need to prove that $\Pr_{h \in \mathcal{H}} (h_{\vec{a},b}(\vec{u}) = y \text{ and } h_{\vec{a},b}(\vec{w}) = z) = \frac{1}{p^2}$ for any y, z and $\vec{u} \neq \vec{w}$.

Assume without loss of generality that $u_0 \neq w_0$. Then these conditions are equivalent to

$$a_0 u_0 + b = (y - \sum_{j=1}^{k-1} a_j u_j) \pmod p \quad \text{and} \quad a_0 w_0 + b = (z - \sum_{j=1}^{k-1} a_j w_j) \pmod p.$$

Fixing $a_1, \dots, a_{k-1}, \vec{u}, \vec{w}, y, z$, this is again two linear equations with two variables having a unique solution.

Hence, for every a_1, \dots, a_{k-1} , there is exactly one choice of (a_0, b) out of p^2 possibilities satisfying the conditions.

Therefore, $\Pr_{h \in \mathcal{H}} (h_{\vec{a},b}(\vec{u}) = y \text{ and } h_{\vec{a},b}(\vec{w}) = z) = 1/p^2$, proving the claim. \square

Finally, the above construction can be "truncated" to any table size and is still 2-universal.

$$\text{Let } h_{a,b}(x) = ((ax+b) \pmod p) \pmod n \quad \text{and} \quad \mathcal{H} = \{ h_{a,b} \mid 0 \leq a \leq p-1 \text{ and } 0 \leq b \leq p-1 \}.$$

Claim \mathcal{H} is 2-universal (but not strongly 2-universal).

Proof Exercise. See Lemma 13.6 of MO. \square

Note that there is a prime number between m and $2m$ for any integer m by Bertrand's postulate (wiki).

and so the above family works for any m and n by choosing a prime p such that $m \leq p \leq 2m$.

Also, we can define hash families for other fields, e.g. fields with a power of two elements.

k-universal families of hash functions

The idea is similar. Instead of generating a random line (a degree one polynomial), we can generate a random degree k polynomial.

By polynomial interpolation, we know that given any k distinct points x_1, x_2, \dots, x_k and k values y_1, y_2, \dots, y_k , there is a unique degree k polynomial p with $p(x_i) = y_i$ for $1 \leq i \leq k$.

This means that if we pick a random degree k polynomial, the outputs are k -wise independent.

More precisely, we can construct a k -universal hash family as follows.

Pick random $a_0, a_1, \dots, a_{k-1} \in \{0, 1, \dots, p-1\}$. Let $h_{\vec{a}}(x) = ((a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \dots + a_1x + a_0) \pmod p) \pmod n$.

Then it can be shown that $\mathcal{H} = \{ h_{\vec{a}} \mid 0 \leq a_i \leq p-1 \text{ for } 0 \leq i \leq k-1 \}$ is a k -universal hash family.

Hashing using 2-universal families

Let $|U| = m$ and $|V| = n$. Let p be a prime number such that $m \leq p \leq 2m$.

Using a hash family $\mathcal{H} = \{ h_{a,b}(x) \mid 0 \leq a, b \leq p-1 \}$ where $h_{a,b}(x) = ((ax+b) \pmod p) \pmod n$,

we only need to choose a, b to select a hash function.

So, we can store this hash function using only 2 cells (recall that each cell can store $\log_2 m$ bits).

Also, the hash value can be evaluated very quickly, using only $O(1)$ operations on $\log_2(m)$ -bit words.

Therefore, these hash functions satisfy the small space requirement and also the fast evaluation requirement.

Can they provide the same guarantees as random hash functions? Yes and no.

Expected search time: The expected search time of chain hashing can still be guaranteed.

Lemma Assume m elements in S are hashed into an n -bin hash table by using a random hash function from a 2-universal family. For an arbitrary element x , let X be the number of items at bin $h(x)$.

$$\text{Then, } E[X] \leq \begin{cases} \frac{m}{n} & \text{if } x \notin S \\ 1 + \frac{m-1}{n} & \text{if } x \in S \end{cases}$$

Proof Let $X_i = 1$ if the i -th element in S is in the same bin as x and 0 otherwise.

Since the hash function is chosen from a 2-universal family, it follows that $\Pr(X_i = 1) = 1/n$.

$$\text{Therefore } E[X] = \sum_{i=1}^m E[X_i] = \frac{m}{n} \text{ if } x \notin S \text{ and } E[X] = 1 + \frac{m-1}{n} \text{ if } x \in S. \quad \square$$

The above statement is simple but it should be read carefully where we used the 2-universal property.

Consider the stupid hashing scheme where we choose a random bin and put all the balls there.

It is still true that the expected number of balls in a bin is $\frac{m}{n}$, but the search time would be m always.

Maximum load: However we cannot guarantee that the maximum loaded bin has $O(\log n / \log \log n)$ balls.

We can still use the 2-universal property to give a non-trivial bound.

Let $X_{ij} = 1$ if item i and item j are mapped to the same bin, and 0 otherwise.

Let $X = \sum_{i,j} X_{ij}$ be the number of collision pairs

$$\text{Then } E[X] = \sum_{i,j} E[X_{ij}] = \sum_{i,j} \Pr(h(x_i) = h(x_j)) \underset{\substack{\uparrow \\ \text{by 2-universality}}}{\leq} \sum_{i,j} \frac{1}{n} = \binom{m}{2} \cdot \frac{1}{n} \leq \frac{m^2}{2n}$$

By Markov's inequality, $\Pr(X \geq m^2/n) \leq 1/2$, or equivalently $\Pr(X < m^2/n) \geq 1/2$.

Suppose the maximum load is Y . Then there are at least $\binom{Y}{2}$ collision pairs.

Therefore, with probability $\geq 1/2$, we have $\binom{Y}{2} < \frac{m^2}{n}$, which implies that $Y \leq m \sqrt{\frac{2}{n}}$.

When $m=n$, this implies that the maximum load is $\sqrt{2n}$ with probability $1/2$.

Remark: To guarantee that the maximum load is $O(\log n / \log \log n)$ with high probability,

we can use a $\Omega(\log n / \log \log n)$ -universal hash family (why?).

But then the evaluation time is $\Omega(\log n / \log \log n)$, which is not a good tradeoff.

References and further reading

Material in this lecture is from chapter 13 of MU.

In practice, people don't use k -universal hash family for large k because of the computational overhead.

Instead, they just use some simple hash family and observe that it works well.

A particularly simple way is called tabulation hashing, which only involves table lookups and XORs.

It is not 4-wise independent, but can be proved to have nice properties of random functions, e.g.

maximum load is $O(\log n / \log \log n)$.

There are also other beautiful hashing techniques such as cuckoo hashing.

k -wise independent variables are used for derandomization, to turn randomized algorithms into

deterministic, by doing brute force search in the sample space of size $O(n^k)$.

For some applications, an even weaker notion called "almost k -wise independence" is enough,

and it has smaller sample space of size $O(2^k \cdot n)$.

This has become a standard tool in derandomizing "fixed parameter algorithms".