5.1 Hash table

Definition 1. Given each data with a key in $U = \{0, 1, \ldots, t-1\}$. A **Hash table** is capable of performing the following three operations:

1. `Insert` k: Insert a data with key k.
2. `Delete` k: Delete a data with key k.
3. `Search` k: Search a data with key k.

And each of them cost $\mathcal{O}(1)$ time.

The simplest way is to allocate an array with size $|U|$, and directly store the data with key k in the k-th slot.

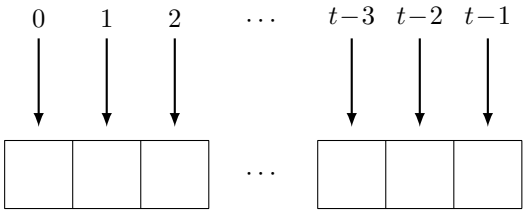


Figure 1: The simplest hash table

But then, the space complexity is $\mathcal{O}(U)$, which is usually intractable (e.g., if U is the set of ascii string with length no more than 10, then $U \approx \mathcal{O}(128)^{10}$).

Thus we shall pick a **hash function** $h :: U \to \{0, 1, \ldots, m-1\}$ with m be a reasonable size of an array, and store the data with key k in the $h(k)$-th slot.

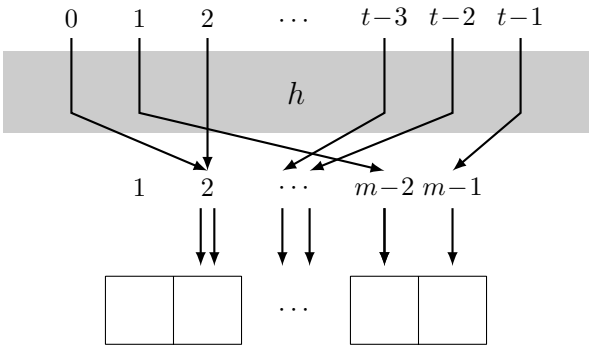


Figure 2: Hash table with a reasonable hash function

But since $m < t$, h could not be injective, so there would be $k_1 \neq k_2$ such that $h(k_1) = h(k_2)$.

Definition 2. If $h(k_1) = h(k_2)$ but $k_1 \neq k_2$, then a **collision** occurs.

▶ Collision resolution by chaining

There are several ways to resolve the collision problem. "Collision resolution by chaining" is one of them.

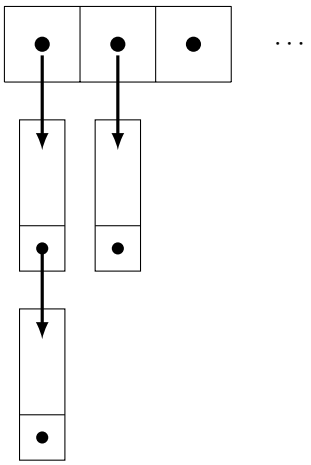


Figure 3: An intuition of the chaining method

Instead of using an array, we use an array of link lists to store the data. For each operations (`Insert`/`Delete`/`Search`) with key k, we first get the link list in the k-th slot of the array, and do the correspond operation on the link list.

When analysing the performance below, we shall assume that the chaining resolution is used.

Now we define the load factor, which plays an important role in the performance of the hash table.

Definition 3. We define the **load factor** to be $\alpha \triangleq n/m$, where n is the number of elements and m is the number of table slots.

Definition 4. A **uniform hashing function** is a hash function such that

- $\Pr(h(k) = x) = 1/m$ for each $0 \leq x < m$.
- $h(K_1), h(K_2), \ldots, h(K_q)$ are independent if $K_1, \ldots, K_q$ are independent.

A uniform hashing function is an ideal hash function, which has the following property.

Theorem 1. *Assume uniform hashing, then the expected time for each operation is* $\mathcal{O}(\alpha)$.

▶ Expected single operation worst-case time complexity

Let n be the number of datas to store, and m be the size of the hash table. If $m = \mathcal{O}(n)$, by theorem 1, we know that the expected average time complexity of each operation is $\mathcal{O}(n/m) = \mathcal{O}(1)$.

Then we ask, what is the expected *worst-case* time complexity?

Consider the following analogous problem. We put n balls in n bins uniformly at random and independently, what is the expected number of balls in the most loaded bin?

The answer turns out to be $\mathcal{O}(\log n/\log\log n)$. Sadly, the proof is quite complicated, so we shall prove a weaker form.

Theorem 2. *The event "The number of balls in all bins are lower then $4\log n/\log\log n$" has probability at least $1-1/n$. (i.e., the answer is $\mathcal{O}(\log n/\log\log n)$ in high probability as $n\to\infty$.)*

Proof. Let $k = 4\log n/\log\log n$ and $[S]^k \triangleq \{\, A \subseteq S \,:\, |A| = k \,\}$. We have

$$\begin{aligned}
\Pr\{\,\text{Bin } i \text{ has} \geq k \text{ balls}\,\} &\overset{\text{(a)}}{\leq} \sum_{A\in[S]^k} \Pr\{\,\text{Every ball in } A \text{ is in Bin } i\,\} \\
&\overset{\text{(b)}}{=} \binom{n}{k}\left(\frac{1}{n}\right)^k \\
&= \frac{n!}{k!\,(n-k)!}\,\frac{1}{n^k} \\
&= \frac{1}{k!}\,\frac{n!}{(n-k)!\,n^k} \overset{\text{(c)}}{\leq} \frac{1}{k!} \overset{\text{(d)}}{\leq} \frac{1}{n^2}
\end{aligned}$$

□

Where (a) holds by the uniform bound, and (b) is because there are $\binom{n}{k}$ such subset A, and each corresponded event has probability $(1/n)^k$. (c) holds by the fact that $n(n-1)\cdots(n-k+1) \leq n^k$.

To prove (d), we need the help from Stirling's approximation formula.

Theorem 3 (Stirling's approximation formula)**.**

$$k! = \sqrt{2\pi k}\left(\frac{k}{\mathrm{e}}\right)^k\left(1+\mathcal{O}\left(\frac{1}{k}\right)\right)$$

Also, for each k,

$$k! \geq \left(\frac{k}{\mathrm{e}}\right)^k$$

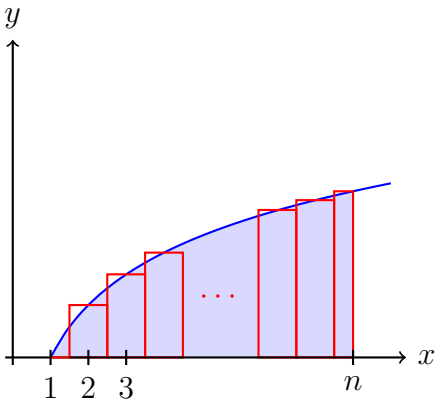
Then we have

$$\begin{aligned}
\frac{1}{k!} &\overset{\text{(a)}}{\leq} \left(\frac{\mathrm{e}}{k}\right)^k \overset{\text{(b)}}{\leq} \left(\frac{\log\log n}{\log n}\right)^k \\
&= \exp\Big(\big(\log(\log\log n/\log n)\big)\cdot k\Big) \\
&= \exp\left(\big(\log\log\log n - \log\log n\big)\cdot\frac{4\log n}{\log\log n}\right) \\
&\overset{\text{(c)}}{\leq} \frac{1}{n^2}
\end{aligned}$$

Where the Stirling's approximation formula is used in (a). Since $k = 4\log n/\log\log n$ and $\mathrm{e} \leq 4$, (b) holds. Finally, (c) holds due to the fact that $\log\log\log n/\log\log n \leq \sup_x \log(x)/x = 1/\mathrm{e}$ so $4(\log\log\log n - \log\log n)/\log\log n \leq 4(1/\mathrm{e}-1) \leq -2$.

▶ Intuition of the Stirling's approximation formula

We could approximate the integral $\int_1^n \log x \, \mathrm{d}x$ by $1/2 \log 1 + \log 2 + \cdots + 1/2 \log n$. See the figure below.



Then we have [1]

$$\log 2 + \cdots + \log(n-1) + \frac{1}{2} \log n \approx \int_1^n \log x \, \mathrm{d}x = n \log n - n + 1$$

Thus

$$\log(n!) = \log 2 + \cdots + \log n \approx n \log n - n + \frac{1}{2} \log n + 1,$$

and hence

$$n! \approx \mathrm{e} \sqrt{n} \left(\frac{n}{e} \right)^n$$

The error in the estimation of the integral cause the final multiplier in the formula to be $\sqrt{2\pi}$ instead of e.

5.2 2-Universal Family of Hash Functions

▶ Examples of hash function

First we give some examples of hash function.

Example 1.

1. $h(k) = ak \pmod m$, where $a \in \mathbb{N}$ and m is a prime.
2. $h(k) = \lfloor m(ka \mod 1) \rfloor$, where $a \in \mathbb{R}$, $m \in \mathbb{N}$, and $(x \mod 1) \triangleq x - \lfloor x \rfloor$.

These two hash functions map the keys to $\{ 0, 1, \ldots, m-1 \}$.

[1] Notice that we have no guarantee on how close the approximation is here.

Definition 5. Let H be a family of hash functions $\{ h_1, \ldots, h_k \}$, where each hash function has the signature $h_i :: U \to \{ 0, 1, \ldots, m-1 \}$.

H is 2-**universal** if for any two elements $x, y \in U$ and $x \neq y$, $\Pr\{ h(x) = h(y) \} \leq 1/m$, where h is uniformly choosed in random.

Another equivalent definition is the number of h_i satisfying $h_i(x) = h_i(y)$ is no more than k/m for each $x \neq y$.

We now give an example of a 2-universal family of hash functions.

Example 2. Given x, y, take the binary representations $x = x_0 x_1 \cdots x_r$ and $y = y_0 y_1 \cdots y_r$.

Let $a = \langle a_0, \ldots, a_r \rangle$ with each $a_i \in \{ 0, 1, \ldots, m-1 \}$. For each a, define

$$h_a(x) = (a_0 x_0 + a_1 x_1 + \cdots + a_r x_r) \mod m.$$

We claim that this is a universal family of hash functions with $|H| = m^{r+1}$.

Proof. Since $x \neq y$, without lost of generality, assume $x_0 = 0$ and $y_0 = 1$. Then it is easy to see that $f_k \triangleq h_{\langle k, a_1, \ldots, a_r \rangle}$ satisfies $f_i(x) = f_j(x)$ and $f_i(y) \neq f_j(y)$ for any $0 \leq i, j < m$ and $i \neq j$. So $f_k(y)$ iterate through $0, 1, \ldots, m-1$. Thus there are exactly 1 hash function f satisfies $f(x) = f(y)$ among these m hash functions. Hence $\Pr\{ h(x) = h(y) \} \leq 1/m$. □

Notice that if the key k of each data satisfies $0 \leq k < n$, then taking $r = \mathcal{O}(\log n)$ is sufficient.

▶ Another randomized max-cut algorithm

Let $G = (V, E)$ be the problem instance, where $V = \{ v_0, \ldots, v_{n-1} \}$ and $H = \{ h_i \}$ be a 2-universal family of hash functions with each $h_i :: \{ 0, \ldots, n \} \to \{ 0, 1 \}$. Pick a hash function h randomly from H uniformly. We give a cut $C = (S, T)$, $S \sqcup T = V$ by $S = \{ v_i \mid h(i) = 0 \}$ and $T = \{ v_i \mid h(i) = 1 \}$.

For each edge (u, v) in the cut of the optimal solution C_{opt}, since $u \neq v$, $\Pr\{ h(u) \neq h(v) \} \geq 1/2$, so $\Pr\{ (u, v) \in C \} \geq 1/2$, thus $\mathbf{E}|C| \geq \mathbf{E}|C_{\text{opt}}|/2$, which proves this algorithm is a 2-approximation algorithm.

Notice that we only need $\mathcal{O}(\log n)$ random bits to choose h, and is less then the original algorithm, which required $\mathcal{O}(n)$ random bits.

5.3 Perfect hashing

Definition 6. A **perfect hash function** is a hash function with no collisions.

Obviously this is not achievable in general, so we shall assume that we have prior knowledge on the keys which would be stored in the hash table, and thus we could find a suitable hash function before all the operations come.

Example 3. We give an example of creating a perfect hash function of n keys. The procedure is listed in the following:

1. Choose a hash function h from a 2-universal family of hash functions, which maps the key to the set $\{ 0, \ldots, n-1 \}$. Collisions are likely to happen here. Let m_i be the number of keys that is mapped to i.

2. Redo step 1. until $m_1^2 + m_2^2 + \cdots + m_n^2 \leq 4n$.

3. For each $0 \leq i < n$, let $K_i \triangleq \{ x \mid h(x) = i \}$, then $|K_i| = m_i$.

   (1) Choose a hash function h_i from another 2-universal family of hash functions which maps K_i to the set $\{ 0, \ldots, m_i^2 - 1 \}$.

   (2) Similarly, redo (1) until no collision happen

The final hashing value of a key x is $(h(x), h_{h(x)}(x))$, which could be stored in an array of size $m_1^2 + \cdots + m_n^2 = \mathcal{O}(n)$.

We shall then prove that the algorithm halts in reasonable time.

Proof. In step 1., let $C \triangleq \#\{ h(x) = h(y) \mid x < y \}$ be the number of collisions. Then we know that

$$\mathbf{E}\, C \overset{\text{(a)}}{\leq} \sum_{x<y} \Pr\{ h(x) = h(y) \} \overset{\text{(b)}}{\leq} \frac{n(n-1)}{2} \frac{1}{n} < \frac{n-1}{2}$$

Where (a) is due to the union bound. Since h is choosen from a 2-univeral family of hash functions, $\Pr\{ h(x) = h(y) \} = 1/n$, and there are $n(n-1)/2$ pairs (x, y), (b) holds.

Notice that

$$C = \sum_{i=1}^{n} \frac{m_i(m_i - 1)}{2} = \frac{1}{2} \sum m_i^2 - \frac{n}{2}$$

Hence

$$\mathbf{E} \sum m_i^2 \leq 2 \left(\frac{n-1}{2} + \frac{n}{2} \right) \leq 2n$$

Recall the Markov's inequality:

Theorem 4. *If X is a random variable with $\Pr\{ X \geq 0 \} = 1$, then $\Pr\{ X \geq \alpha\, \mathbf{E}\, X \} \leq 1/\alpha$.*

Thus by Markov's inequality, we have $\Pr\{ \sum m_i^2 > 4n \} \leq 1/2$. Hence in average, we only need to redo twice in step 1.

In step 3-(1), let C be the number of collisions in j-th table, which has m_j elements and m_j^2 slots. Then

$$\mathbf{E}\, C \leq \frac{m_j(m_j - 1)}{2} \frac{1}{m_j^2} < \frac{1}{2}$$

So again by Markov's inequality, $\Pr\{ \text{A collision occurs} \} \leq 1/2$. Thus in average, we only need to redo twice for each table. □