به نام خدا

ساختمان داده

آرش شفیعی

درهم‌سازی

درهم‌سازی

- الگوریتم‌ها ممکن است نیاز داشته باشند عملیات متعددی بر روی مجموعه‌های پویا انجام دهند. بسیاری از الگوریتم‌ها نیاز دارند تنها عملیات درج، حذف و جستجو انجام دهند. مجموعه‌ای پویا که این عملیات را پشتیبانی کند دیکشنری [1] نامیده می‌شود.
- برای مثال در یک کامپایلر جدولی به نام جدول علائم وجود دارد که متغیرها و ویژگی‌های آنها را ذخیره می‌کند. برای ذخیره‌سازی متغیرهای یک برنامه و ویژگی‌های آن متغیرها از دیکشنری استفاده می‌کنیم.

[1] dictionary

درهم‌سازی

- جدول درهم‌سازی [1] یک ساختار داده کارا برای پیاده‌سازی دیکشنری است.
- اگرچه پیچیدگی زمانی جستجوی یک عنصر در جدول درهم‌سازی می‌تواند در بدترین حالت مانند جستجو در یک لیست پیوندی یعنی $\Theta(n)$ باشد، اما با تعدادی پیش‌فرض می‌توان جستجوی یک عنصر در جدول درهم‌سازی را در زمان $O(1)$ انجام داد.
- نوع داده‌ای دیکشنری در زبان پایتون توسط جداول درهم‌سازی پیاده‌سازی شده است.
- درواقع جدول درهم‌سازی مفهوم آرایه‌ها را تعمیم می‌دهد تا جستجو در مجموعه پویا همانند دسترسی به یک عنصر در آرایه در زمان $O(1)$ انجام شود.
- ایده اصلی جدول درهم‌سازی این است که اندیس یک مکان در آرایه، که حاوی یک مقدار کلید است، از طریق مقدار کلید آن با استفاده از روشی که توضیح خواهیم داد، محاسبه می‌شود.

[1] hash table

جداول آدرس‌دهی مستقیم

- آدرس‌دهی مستقیم [1] یک روش ساده است که وقتی مجموعه مرجع کلیدها (U) کوچک باشد می‌تواند مورد استفاده قرار بگیرد.

[1] direct addressing

جداول آدرس‌دهی مستقیم

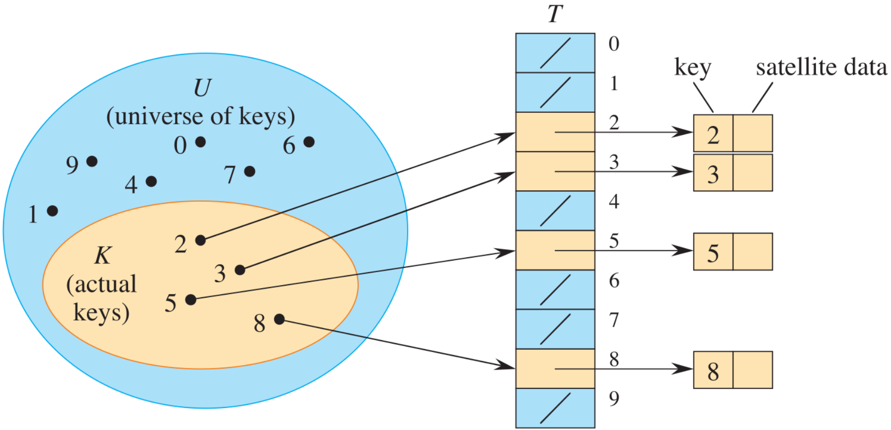
- فرض کنید برنامه‌ای می‌خواهد از یک مجموعهٔ پویا استفاده کند که کلیدهای عناصر آن یکتا بوده و زیر مجموعه‌ای از مجموعهٔ مرجع [1] $U = \{0, 1, \cdots, m-1\}$ هستند به طوری که m عدد بسیار بزرگی نیست.

- برای ذخیره‌سازی این مجموعه پویا، می‌توانیم از یک آرایه استفاده کنیم که به آن جدول آدرس‌دهی مستقیم گفته می‌شود. این آرایه را با $T[0 : m-1]$ نمایش می‌دهیم. هریک از مکان‌های آن (که به آن اسلات [2] نیز گفته می‌شود) به یکی از کلیدها در مجموعهٔ مرجع U اختصاص دارد.

[1] universal set (universe)
[2] slot

جداول آدرس‌دهی مستقیم

- شکل زیر این روش را نمایش می‌دهد. مکان k به عنصری اشاره می‌کند که مقدار کلید آن k است. اگر مجموعه هیچ عنصری با کلید k نداشته باشد، آنگاه خواهیم داشت $T[k] = NIL$.



جداول آدرس‌دهی مستقیم

– همه توابع جستجو Direct-Address-Search و درج Direct-Address-Insert و حذف Direct-Address-Delete در زیر نشان داده شده‌اند که هرکدام در زمان O(1) انجام می‌شوند.

Algorithm Direct Address Search

```
   function DIRECT-ADDRESS-SEARCH(T,k)
1: return T[k]
```

Algorithm Direct Address Insert

```
   function DIRECT-ADDRESS-INSERT(T,x)
1: T[x.key] = x
```

جداول آدرس‌دهی مستقیم

Algorithm Direct Address Delete

```
    function DIRECT-ADDRESS-DELETE(T,x)
T[x.key] = NIL
```

جداول درهم‌سازی

- اگر مجموعهٔ مرجع U بسیار بزرگ یا نامحدود باشد، ذخیره‌سازی جدول T به اندازه $|U|$ غیر عملی یا حتی غیرممکن است.
- علاوه بر این، مجموعه k از کلیدهایی که واقعا ذخیره شده‌اند ممکن است نسبت به مجموعهٔ مرجع U بسیار کوچک باشد و در نتیجه بیشتر فضایی که به T تخصیص داده شده است ممکن است هدر رود.
- وقتی مجموعه k یعنی کلیدهای ذخیره شده در دیکشنری بسیار کوچک‌تر از مجموعه مرجع U یعنی تمام کلیدهای ممکن باشد، جداول درهم‌سازی کمک می‌کنند که فضای بسیار کوچک‌تری نسبت به جدول آدرس‌دهی مستقیم اشغال شود.
- پیچیدگی حافظه با استفاده از جداول درهم‌سازی به $\Theta(|k|)$ کاهش پیدا می‌کند و پیچیدگی زمانی جستجو همچنان $O(1)$ باقی می‌ماند.

جداول درهم‌سازی

- در آدرس‌دهی مستقیم یک عنصر با کلید k در مکان k ذخیره می‌شود، اما در جداول درهم‌سازی از یک تابع درهم‌سازی h [1] استفاده می‌کنیم تا مکان کلید k را محاسبه کنیم و بنابراین کلید k در مکان $h(k)$ قرار می‌گیرد. تابع درهم‌سازی h مجموعه مرجع U را به مکان‌های جدول درهم‌سازی $T[0 : m-1]$ نگاشت می‌کند.

$$h : U \rightarrow \{0, 1, \cdots, m-1\}$$

- در اینجا m اندازه جدول درهم‌سازی است که معمولا بسیار کوچک‌تر از اندازهٔ U است.

- می‌گوییم عنصر با کلید k به مکان $h(k)$ نگاشت می‌شود [2] .

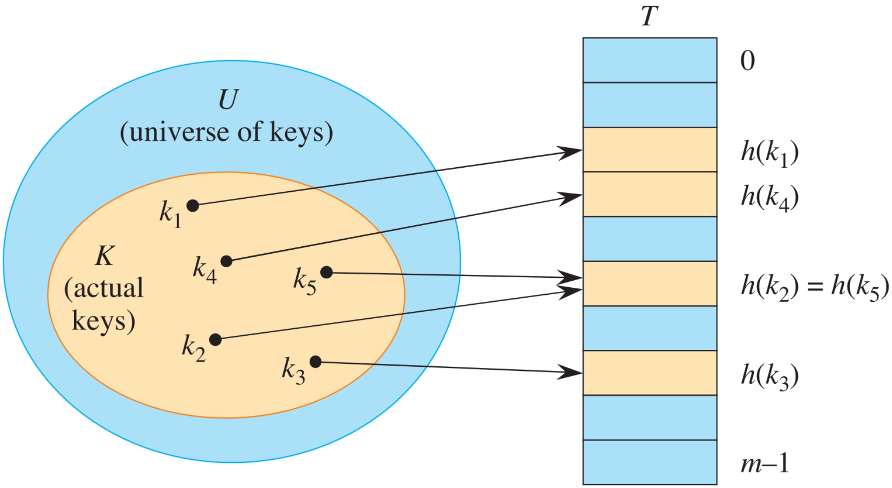
- همچنین می‌گوییم $h(k)$ مقدار نگاشت شده یا مقدار هش [3] برای کلید k است.

---

[1] hash function
[2] k hashes to slot h(k)
[3] hash value

جداول درهم‌سازی

- ایده اصلی جدول درهم‌سازی در زیر نشان داده شده است.



جداول درهم‌سازی

- تابع درهم‌سازی محدوده اندیس‌های آرایه را کاهش می‌دهد و بنابراین اندازه آرایه مورد نیاز برای ذخیره کاهش پیدا می‌کند، در نتیجه به جای استفاده از جدولی به اندازه $|U|$ از جدولی به اندازه m استفاده می‌کنیم.
- یک مثال بسیار ساده از تابع درهم‌سازی تابع $h(k) = k \mod m$ است.

جداول درهم‌سازی

- مشکلی که ممکن است در فرایند محاسبه تابع درهم‌سازی به وجود آید این است که دو کلید به یک مکان حافظه نگاشت شوند. به چنین وضعیتی برخورد یا تصادم [1] گفته می‌شود.

- روش‌های کارایی برای حل مشکل تصادم وجود دارد.

- در شرایط ایده‌آل هیچ تصادمی نباید رخ دهد و تلاش می‌کنیم تابع درهم‌سازی را به نحوی انتخاب کنیم که تصادم حداقل شود.

[1] collision

جداول درهم‌سازی

- تابع درهم‌سازی باید قطعی [1] باشد به طوری که به ازای ورودی k همیشه خروجی $h(k)$ را تولید کند.
- از آنجایی که $|U| > m$ بنابراین امکان تصادم وجود دارد. بنابراین اگرچه یک تابع درهم‌سازی خوب تعداد تصادم‌ها را کاهش می‌دهد، اما همچنان امکان تصادم وجود دارد و باید روشی برای برطرف کردن تصادم به کار ببریم.
- در ادامه در مورد توابع درهم‌سازی و همچنین روش‌های برخورد با تصادم صحبت خواهیم کرد.

[1] deterministic

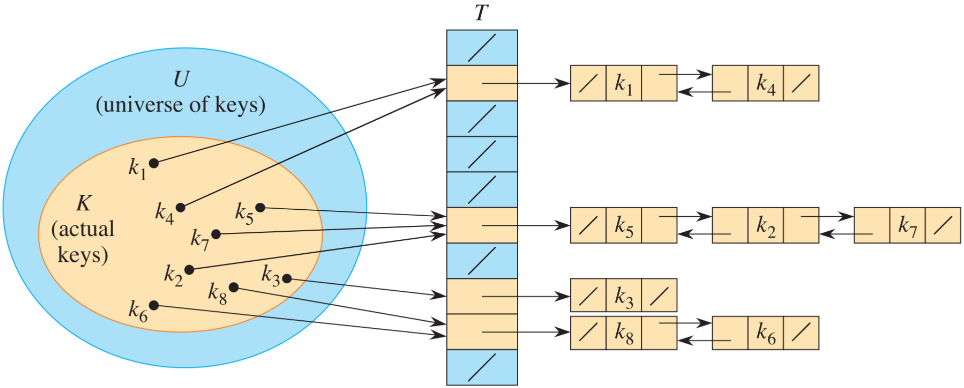
جداول درهم‌سازی

- درهم‌سازی مستقل یکنواخت [1] : یک تابع درهم‌سازی ایده‌آل h به ازای هر ورودی k در دامنه U خروجی $h(k)$ را مستقلا به طور یکنواخت از محدوده $\{0, 1, \cdots, m-1\}$ انتخاب می‌کند.
- مستقل بدین معناست که نگاشت یک کلید به کلیدهای دیگر بستگی ندارد و یکنواخت بدین معناست که یک کلید با احتمال یکسان می‌تواند به هر یک از خانه‌های جدول درهم‌سازی نگاشت شود.
- وقتی مقدار $h(k)$ انتخاب شد، هر فراخوانی بعدی h با ورودی k همان مقدار $h(k)$ را تولید می‌کند.
- تابعی که چنین ویژگی هایی داشته باشد را یک تابع درهم‌سازی مستقل یکنواخت [2] می‌نامیم.

[1] independent uniform hashing
[2] independent uniform hash function

جداول درهم‌سازی

– رفع تصادم با روش زنجیرسازی : شکل زیر ایدهٔ اصلی روش زنجیرسازی [1] برای رفع تصادم را نشان می‌دهد.



[1] chaining

جداول درهم‌سازی

- هر مکان غیرتهی در جدول به یک لیست پیوندی اشاره می‌کند و همه عناصری که به یک مکان نگاشت می‌شوند وارد لیست پیوندی آن مکان می‌شوند.
- مکان j یک اشاره‌گر به ابتدای یک لیست پیوندی دارد که همه عناصری که به مکان j نگاشت می‌شوند را نگهداری می‌کند. اگر هیچ عنصری به مکان j نگاشت نشده باشد، مکان j تهی است و مقدار NIL را نگهداری می‌کند.
- اگر از چنین روشی برای رفع تصادم استفاده کنیم عملیات دیکشنری به صورت زیر پیاده‌سازی خواهند شد.

Algorithm Chained Hash Insert

```
   function CHAINED-HASH-INSERT(T,x)
1: List-Prepend (T[h(x.key)],x)
```

جداول درهم‌سازی

Algorithm Chained Hash Search

```
   function CHAINED-HASH-SEARCH(T,k)
1: return List-Search (T[h(k)],k)
```

Algorithm Chained Hash Delete

```
   function CHAINED-HASH-DELETE(T,x)
1: List-Delete (T[h(x.key)],x)
```

جداول درهم‌سازی

- زمان اجرای عملیات درج در بدترین حالت $O(1)$ است. عملیات درج سریع است زیرا فرض می‌کنیم عنصر x که می‌خواهیم درج کنیم در جدول وجود ندارد. بدون وجود این فرض، قبل از درج باید جستجو کنیم که آیا x.key در لیست پیوندی وجود دارد یا خیر.
- زمان اجرای عملیات جستجو در بدترین حالت متناسب با طول لیست است که این زمان اجرا را دقیق‌تر مورد بررسی قرار خواهیم داد.
- زمان اجرای عملیات حذف $O(1)$ است اگر لیست‌های پیوندی دو طرفه باشند. اگر لیست یک طرفه باشد، برای پیدا کردن عنصر قبلی عنصری که می‌خواهیم حذف کنیم، باید لیست را پیمایش کنیم. از آنجایی که تابع حذف یک اشاره‌گر x به عنصر مورد نظر دریافت می‌کند و نه مقدار کلید آن را، بنابراین نیازی به جستجو نیست.

جداول درهم‌سازی

- تحلیل درهم‌سازی با روش زنجیرسازی : به ازای جدول درهم‌سازی T شامل m مکان که n عنصر را ذخیره می‌کند، ضریب بار [1] α برای T را به صورت n/m تعریف می‌کنیم که تعداد میانگین عناصر ذخیره‌شده در یک زنجیره است. ضریب بار α می‌تواند کمتر، مساوی یا بیشتر از یک باشد.
- در بدترین حالت همه n کلید در یک مکان جدول ذخیره می‌شوند که درواقع لیستی به طول n می‌سازند. زمان اجرا در بدترین حالت برای جستجو $\Theta(n)$ است به علاوه زمان مورد نیاز برای محاسبه تابع درهم‌سازی.
- کارایی درهم‌سازی در حالت میانگین بستگی به این دارد که تابع درهم‌سازی h چگونه مجموعه n کلید را در m مکان توزیع می‌کند.

[1] load factor

جداول درهم‌سازی

- فرض کنید به ازای $j = 0, 1, \cdots, m-1$ طول لیست $T[j]$ برابر با n_j باشد. بنابراین خواهیم داشت $n = n_0 + n_1 + \cdots + n_{m-1}$ و مقدار مورد انتظار [1] برای n_j برابر با $E[n_j] = \alpha = n/m$ است.
- فرض می‌کنیم زمان $O(1)$ برای محاسبه مقدار هش $h(k)$ کافی است و بنابراین زمان لازم برای جستجوی یک مقدار با کلید k به طول $n_{h(k)}$ از لیست $T[h(k)]$ بستگی دارد.

[1] expected value

جداول درهم‌سازی

- قضیه : در یک جدول درهم‌سازی که تصادم با زنجیرسازی رفع می‌شود، یک جستجوی ناموفق به طور متوسط در زمان $\Theta(1+\alpha)$ انجام می‌شود.
- اثبات : با فرض بر این‌که درهم‌سازی مستقل و یکنواخت است، هر کلید k که هنوز در جدول ذخیره نشده است، می‌تواند با احتمال یکنواخت در هر یک از m مکان جدول ذخیره شود. زمان مورد نیاز برای یافتن کلیدی که در جدول وجود ندارد یعنی برای جستجوی ناموفق کلید k برابر با زمان مورد انتظار آن $E[n_{h(k)}] = \alpha$ است.
- بنابراین تعداد عناصر بررسی شده در یک جستجوی ناموفق برابر با α است و کل زمان مورد نیاز (شامل زمان محاسبه $h(k)$) برابر با $\Theta(1+\alpha)$ است.

جداول درهم‌سازی

- بنابراین پیچیدگی زمانی جستجو در جدول درهم‌سازی که در آن تصادم با زنجیرسازی رفع می‌شود، در بدترین حالت برابر با $\Theta(1+\alpha)$ است.
- این تحلیل بدین معناست که اگر تعداد عناصر در جدول متناسب با تعداد مکان‌های جدول درهم‌سازی باشد، داریم $n = O(m)$ و $\alpha = n/m = O(m)/m = O(1)$.
- بنابراین جستجو به طور میانگین در زمان ثابت انجام می‌شود. از آنجایی که درج در بدترین حالت در زمان $O(1)$ و حذف در بدترین حالت در زمان $O(1)$ انجام می‌شود، وقتی لیست دو طرفه باشد، بنابراین همه عملیات دیکشنری به طور متوسط در زمان $O(1)$ انجام می‌شوند.
- تحلیل ارائه شده تنها وقتی معتبر است که درهم‌سازی یکنواخت و مستقل باشد. یکنواخت بودن بدین معناست که کلیدها با احتمال برابر در هریک از m مکان نگاشت می‌شود و مستقل بودن بدین معناست که هر دو کلید متمایز با احتمال $1/m$ برخورد می‌کنند و وابستگی بین کلیدها در محاسبه مقدار درهم‌سازی وجود ندارد.

توابع درهم‌سازی

- برای اینکه درهم‌سازی خوبی داشته باشیم نیاز به یک تابع درهم‌سازی خوب داریم.
- یک تابع درهم‌سازی خوب باید به طورکارا قابل محاسبه باشد، و کلیدها را به طور یکنواخت در جدول درهم‌سازی توزیع کند.
- یک تابع درهم‌سازی خوب بر اساس فرض درهم‌سازی یکنواخت مستقل عمل می‌کند بدین معنا که درآن هرکلید با احتمال یکسان به هرکدام از m مکان جدول نگاشت می‌شود مستقل از این که بقیه کلیدها به چه مکانی نگاشت شده‌اند.
- اگر بدانیم کلیدها اعداد تصادفی حقیقی k هستند که به طور یکنواخت و مستقل در محدودهٔ $0 \leqslant k < 1$ توزیع شده‌اند، آنگاه تابع درهم‌سازی $h(k) = \lfloor km \rfloor$ یک تابع یکنواخت و مستقل خواهد بود.
- اما معمولا دسترسی به توزیع احتمالی کلیدهای ورودی نداریم.

توابع درهم‌سازی

- در عمل، یک تابع درهم‌سازی برای نوع داده‌های زیر طراحی می‌شود:

(۱) کلیدهای که اعداد صحیح غیر منفی کوچک هستند و در w بیت جای می‌گیرند. معمولا مقدار w برابر با ۳۲ یا ۶۴ است. (اگر کلید ۳۲ بیتی باشد، مقدار آن می‌تواند بین ۰ تا ۴۲۹۴۹۶۷۲۹۵ باشد.)

(۲) کلیدهایی که یک وکتور کوچک از اعداد صحیح غیرمنفی هستند که هرکدام اندازه محدودی دارند. برای مثال، هر عنصر وکتور می‌تواند ۸ بیت باشد. یک وکتور را در این حالت یک رشته نیز می‌نامیم.

درهم‌سازی ایستا

- درهم‌سازی ایستا [1] از یک تابع درهم‌سازی ثابت استفاده می‌کند.
- در اینجا در مورد سه روش استاندارد برای درهم‌سازی ایستا صحبت می‌کنیم که روش تقسیم، ضرب، و ضرب‌ـانتقال نام دارند.
- در عمل به جای درهم‌سازی ایستا از درهم‌سازی تصادفی استفاده می‌شود که کارایی بالاتری دارد، اما درهم‌سازی ایستا پایه‌ای برای یادگیری مفاهیم درهم‌سازی است.

[1] static hashing

درهم‌سازی ایستا

- روش تقسیم : در روش تقسیم [1] برای ساختن یک تابع درهم‌سازی کلید k را به یکی از m مکان، با محاسبه باقیمانده k بر m نگاشت می‌کنیم. بنابراین تابع درهم‌سازی برابر است با $h(k) = k \mod m$.
- برای مثال، اگر اندازه جدول درهم‌سازی $m = 12$ باشد و کلید مورد نظر $k = 100$ باشد، آنگاه $h(k) = 4$ است. از آنجایی که تنها عملیات مورد نیاز تقسیم است، روش تقسیم روش نسبتا سریعی است.
- روش تقسیم وقتی m یک عدد اول باشد که به اعداد توان ۲ نزدیک نباشد خوب عمل می‌کند. اما هیچ تضمینی وجود ندارد که این روش درحالت میانگین کارایی خوبی داشته باشد.

[1] division method

درهم‌سازی ایستا

- روش ضرب : روش ضرب [1] برای ساخت تابع درهم‌سازی در دو گام عمل می‌کند. ابتدا، مقدار کلید k را در یک ثابت A در محدوده $0 < A < 1$ ضرب می‌کند و قسمت اعشاری kA را استخراج می‌کند. سپس مقدار به دست آمده را m ضرب می‌کند و کف آن را به عنوان نتیجه محاسبه می‌کند. بنابراین تابع درهم‌سازی برابر است با

$$h(k) = \lfloor m(kA \mod 1) \rfloor = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$$

- در اینجا $kA \mod 1$ به معنای قسمت اعشاری kA است یعنی $kA - \lfloor kA \rfloor$.
- در عمل، روش ضرب وقتی m توانی از عدد ۲ است یعنی $m = 2^l$ (به ازای عدد صحیح l) به طوری‌که $l \leqslant w$ و w تعداد بیت‌های یک کلمه در ماشین است، خوب عمل می‌کند.

[1] multiplication method

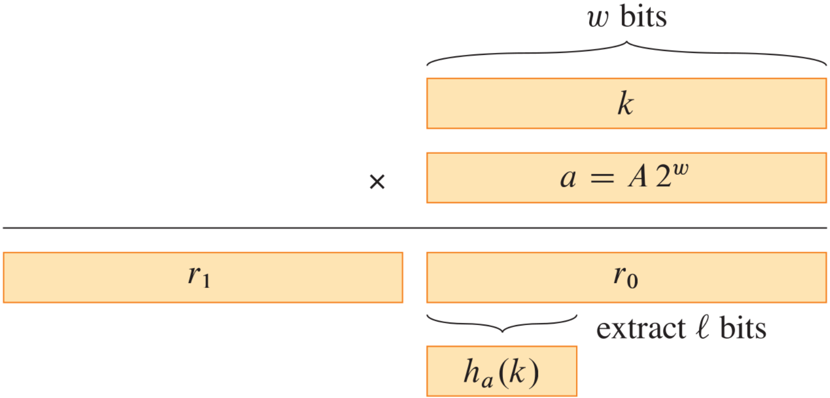
درهم‌سازی ایستا

- روش ضرب-انتقال [1] : فرض می‌کنیم کلید k در یک فضای w بیتی جای می‌گیرد. عدد w بیتی $a = A2^w$ را انتخاب می‌کنیم، به طوری که $0 < A < 1$ و $0 < a < 2^w$.
- ابتدا k را در عدد صحیح w بیتی a ضرب می‌کنیم. نتیجه مقدار $2w$ بیتی $r_1 2^w + r_0$ است جایی که r_1 مقدار w بیت مرتبه بالا و r_0 مقدار w بیت مرتبه پایین نتیجه ضرب است.
- مقدار l بیتی هش تشکیل شده است از l بیت پر ارزش r_0.
- از آنجایی که از r_1 چشم‌پوشی می‌شود، تابع درهم‌سازی می‌تواند برروی یک کامپیوتر به گونه‌ای پیاده‌سازی شود که باقیمانده یک حاصلضرب بر 2^w را محاسبه کند.
- تابع درهم‌سازی در اینجا به صورت $h_a(k) = (ka \mod 2^w) \gg (w - l)$ محاسبه می‌شود.
- از آنجایی که حاصلضرب ka از دو عدد w بیتی، میزان $2w$ فضا اشغال می‌کند، باقیمانده ضرب بر 2^w تعداد w بیت پر ارزش را حذف می‌کند (r_1) و w بیت کم ارزش را به دست می‌دهد (r_0) .

[1] multiply-shift method

درهم‌سازی ایستا

- این روش در شکل زیر نشان داده شده است.



درهم‌سازی ایستا

- عملگر $\gg$ یک انتقال به چپ به اندازه $w-l$ بیت انجام می‌دهد، که معادل است با تقسیم عدد بر 2^{w-l} و محاسبه کف نتیجه.
- تابع درهم‌سازی h_a می‌تواند با سه دستور پیاده‌سازی شود، ضرب، تفریق و انتقال.
- به عنوان مثال، فرض کنید $k = 123456$ و $l = 14$ و $m = 2^{14} = 16384$ و $w = 32$.
- همچنین فرض کنید $a = 2654435769$. در این صورت $ka = 327706022297664 = (76300 \times 2^{32}) + 17612864$ و بنابراین $r_1 = 76300$ و $r_0 = 17612864$.
- اگر ۱۴ بیت پر ارزش r_0 را استخراج کنیم خواهیم داشت $h_a(k) = 67$.

درهم‌سازی ایستا

- اگرچه روش ضرب-انتقال روش سریعی است اما تضمین نمی‌کند که کارایی خوبی در حالت میانگین داشته باشد.
- از روش ضرب-انتقال در درهم‌سازی تصادفی استفاده می‌شود که کارایی مطلوبی دارد وقتی a به طور تصادفی از بین اعداد فرد انتخاب شود.

درهم‌سازی تصادفی

- در درهم‌سازی ایستا می‌توانیم کلیدها را به گونه‌ای انتخاب کنیم که همه آنها به یک مکان جدول نگاشت شوند. در این حالت پیچیدگی زمانی جستجو $\Theta(n)$ خواهد بود. هرگونه درهم‌سازی ایستا دارای این نقطه ضعف است.
- یک روش برای مقابله با این مشکل استفاده از درهم‌سازی تصادفی [1] است.
- یکی از حالات خاص این روش درهم‌سازی سراسری [2] است که کارایی خوبی دارد وقتی تصادم‌ها با زنجیرسازی مدیریت می‌شوند.
- برای استفاده از درهم‌سازی تصادفی، در ابتدای اجرای برنامه تابع درهم‌سازی را به صورت تصادفی از میان خانواده‌ای از توابع درهم‌سازی انتخاب می‌کنیم. از آنجایی که تابع درهم‌سازی را به صورت تصادفی انتخاب می‌کنیم، الگوریتم در هر اجرا رفتار متفاوتی دارد.

---
[1] random hashing
[2] universal hashing

درهم‌سازی تصادفی

- فرض کنید H یک خانواده محدود از توابع درهم‌سازی باشد که یک مجموعه مرجع U از کلیدها را به بازهٔ $\{0, 1, \cdots, m-1\}$ نگاشت می‌کند.
- به این خانواده از توابع یک خانواده سراسری گفته می‌شود، اگر با انتخاب یک تابع درهم‌سازی به صورت تصادفی از مجموعه H ، احتمال تصادم بین دو کلید متمایز k_1 و k_2 بیشتر از $1/m$ نباشد.
- به عبارت دیگر با انتخاب یک تابع درهم‌سازی به صورت تصادفی از مجموعه H ، احتمال تصادم بین دو کلید متمایز k_1 و k_2 بیشتر از احتمال تصادم $1/m$ وقتی که $h(k_1)$ و $h(k_2)$ به صورت تصادفی و مستقل از مجموعه $\{0, 1, \cdots, m-1\}$ انتخاب شده باشند نیست.

درهم‌سازی تصادفی

- در اینجا دو روش برای طراحی یک خانواده سراسری از توابع درهم‌سازی معرفی می‌کنیم.
- روش اول : عدد اول p که به اندازه کافی بزرگ است را انتخاب می‌کنیم به طوری که هر کلید k در محدودهٔ 0 تا $p-1$ قرار بگیرد.
- فرض کنید $\mathbb{Z}_p$ مجموعه $\{0, 1, 2, \cdots, p-1\}$ باشد و $\mathbb{Z}_p^*$ مجموعه $\{1, 2, \cdots, p-1\}$.
- از آنجایی که اندازه مجموعه مرجع کلیدها بزرگ‌تر از تعداد مکان‌ها در جدول درهم‌سازی است، داریم $p > m$.

درهم‌سازی تصادفی

- به ازای هر $a \in \mathbb{Z}_p^*$ و $b \in \mathbb{Z}_p$ ، تابع h_{ab} را به صورت یک تبدیل خطی و یک کاهش با استفاده عملگر باقیمانده بر p و سپس باقیمانده بر m تعریف می‌کنیم.

$$h_{ab}(k) = ((ak + b) \mod p) \mod m$$

- برای مثال اگر $p = 17$ و $m = 6$ باشد، داریم

$$\begin{aligned} h_{3,4}(8) &= ((3 \times 8 + 4) \mod 17) \mod 6 \\ &= (28 \mod 17) \mod 6 \\ &= 11 \mod 6 \\ &= 5 \end{aligned}$$

درهم‌سازی تصادفی

- به ازای p و m ، خانواده همه توابع درهم‌سازی به صورت زیر است.
$$H_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p\}$$
- یک ویژگی این خانواده از توابع درهم‌سازی این است که اندازه m می‌تواند هر مقدار دلخواهی باشد و نیازی نیست اول باشد.
- از آنجایی که می‌توانیم تعداد $p-1$ مقدار برای a و p مقدار برای b انتخاب کنیم، خانواده H_{pm} شامل $p(p-1)$ تابع درهم‌سازی است.
- می‌توان اثبات کرد H_{pm} یک خانواده سراسری است.

درهم‌سازی تصادفی

- روش دوم : از این روش در عمل بیشتر استفاده می‌شود زیرا کارایی بالایی دارد.
- فرض کنید H یک خانواده از توابع درهم‌سازی ضرب-انتقال صورت زیر باشد.

$H = \{h_a :$ فرد است a و $1 \leqslant a < m$ و یک تابع درهم‌سازی ضرب انتقال است $h_a\}$

- تابع درهم‌سازی ضرب-انتقال را قبلا به صورت زیر تعریف کردیم.

$$h_a(k) = (ka \mod 2^w) \gg (w - l)$$

- می‌توان اثبات کرد توابع خانواده H یک خانواده سراسری هستند.

درهم‌سازی تصادفی

- درهم‌سازی ورودی‌های بزرگ مانند وکتورها یا رشته‌ها:
- برخی مواقع ورودی تابع درهم‌سازی آنقدر بزرگ است که به سادگی نمی‌توان یک عدد اول p پیدا کرد که در ۶۴ بیت جای بگیرد. برای مثال ممکن است بخواهیم یک رشته را به عنوان کلید در نظر بگیریم.
- یک روش درهم‌سازی برای چنین ورودی‌هایی تعمیم روش‌های شرح داده شده است. برای این کار می‌توانیم از توابع رمزنگاری استفاده کنیم.

- یک تابع رمزنگاری یک ورودی با اندازه دلخواه را دریافت می‌کند و یک خروجی با طول ثابت تولید می‌کند.
- برای مثال تابع استاندارد SHA-256 یک خروجی ۲۵۶ بیتی (۳۲ بایتی) تولید می‌کند.
- توابع رمزنگاری به قدری مورد استفاده‌اند که برخی از پردازنده‌ها توابع رمزنگاری را به صورت سخت‌افزاری پیاده‌سازی می‌کنند و بنابراین سرعت اجرای بالایی دارند.
- می‌توانیم تابع درهم‌سازی برای یک کلید با طول دلخواه را به صورت زیر تعریف کنیم.

$$h(k) = \text{SHA-256}(k) \mod m$$

- برای تعریف یک خانواده از توابع درهم‌سازی می‌توانیم یک رشته a به ابتدای ورودی الحاق کنیم.

$$h_a(k) = \text{SHA-256}(a \| k) \mod m$$

آدرس‌دهی باز

- آدرس‌دهی باز [1] روشی برای رفع تصادم است که برخلاف زنجیرسازی از فضایی خارج از جدول درهم‌سازی استفاده نمی‌کند و همه عناصر را در خود جدول درهم‌سازی ذخیره می‌کند.

- در نتیجه ضریب بار α هیچگاه نمی‌تواند بیشتر از ۱ شود.

[1] open addressing

آدرس‌دهی باز

- در این روش تصادم به صورت زیر مدیریت می‌شود : وقتی می‌خواهیم یک عنصر در جدول وارد کنیم، این عنصر جدید در صورت امکان در مکانی که به عنوان انتخاب اول آن محاسبه می‌شود قرار می‌گیرد. اگر اولین انتخاب اشغال شده بود، مکانی به عنوان دومین انتخاب ذخیره‌سازی کلید محاسبه می‌شود و در صورتی که آن مکان نیز اشغال شده بود، این روند ادامه پیدا می‌کند تا جایی که یک مکان خالی پیدا شود.
- برای جستجوی یک عنصر مکان‌های جدول به ترتیب انتخاب‌ها بررسی می‌شوند تا جایی که یا عنصر یافته شود و یا به یک خانه خالی برخورد کنیم که در این صورت پیام جستجوی ناموفق صادر می‌کنیم.
- آدرس‌دهی باز از هیچ اشاره‌گر و فضای اضافی استفاده نمی‌کند، در نتیجه می‌توان از جدول بزرگ‌تری برای داده‌های آن استفاده کرد.

آدرس‌دهی باز

- برای درج در جدول توسط آدرس‌دهی باز، به ترتیب همهٔ انتخاب‌ها برای مکان کلید مورد نظر را وارسی [1] می‌کنیم تا وقتی که یک فضای خالی در جدول پیدا شود.

- تابع درهم‌سازی به ازای یک کلید باید شماره انتخاب یا شماره وارسی [2] را به عنوان ورودی دوم دریافت کند. بنابراین دامنه و برد تابع درهم‌سازی به صورت زیر خواهد بود.

$$h : U \times \{0, 1, \cdots, m-1\} \to \{0, 1, \cdots, m-1\}$$

- در آدرس‌دهی باز نیاز داریم به ازای هر کلید k ، یک دنباله انتخاب‌ها یا دنباله وارسی [3] به صورت $\langle h(k,0), h(k,1), \cdots, h(k,m-1) \rangle$ تولید کنیم تا همه مکان‌های جدول مورد بررسی قرار بگیرند.

---

[1] probe
[2] probe number
[3] probe sequence

آدرس‌دهی باز

- تابع Hash–Insert جدول T و کلید k را دریافت می‌کند و مکان ذخیره آن را باز می‌گرداند و در صورتی که جدول پر باشد پیام خطا صادر می‌کند.

Algorithm Hash Insert

```
function HASH-INSERT(T,k)
i = 0
repeat
   q = h(k,i)
   if T[q] == NIL then
      T[q] = k
      return q
   else  i = i + 1
until i == m
error "hash table overflow"
```

آدرس‌دهی باز

- الگوریتم جستجوی کلید k نیز دنباله‌ای از مکان‌های جدول را بررسی می‌کند تا وقتی که کلید k یافته شود. وقتی به یک مکان خالی می‌رسیم جستجو با پیامی مبنی بر عدم موفقیت و بازگرداندن NIL به پایان می‌رسد.

آدرس‌دهی باز

- تابع Hash-Search جدول T و کلید k را دریافت می‌کند و در صورتی که مکان q شامل کلید k پیدا شد مکان q را باز می‌گرداند و در صورتی که k در جدول وجود نداشت مقدار NIL را باز می‌گرداند.

Algorithm Hash Search

```
function HASH-SEARCH(T,k)
i = 0
repeat
    q = h(k,i)
    if T[q] == k then
        return q
    i = i + 1
until T[q] == NIL or i == m
return NIL
```

آدرس‌دهی باز

- حذف کردن از جدول درهم‌سازی با آدس‌دهی باز کمی پیچیده‌تر است. وقتی یک کلید را از مکان q حذف می‌کنیم ممکن است به اشتباه آن مکان به عنوان مکان خالی در نظر گرفته شود. اگر مکان حذف شده را با NIL علامتگذاری کنیم، ممکن است نتوانیم کلید k را که مکان q برای آن وارسی شده بوده، در حالی که مکان اصلی آن اشغال بوده است، پیدا کنیم.
- برای حل این مشکل، وقتی یک کلید را در جدول حذف می‌کنیم به جای علامتگذاری آن با مقدار NIL ، مقدار Deleted را در مکان حذف شده قرار می‌دهیم.

آدرس‌دهی باز

- در هنگام درج در جدول، تابع Hash-Insert مکانی که با Deleted علامتگذاری شده است را به عنوان مکان خالی در نظر می‌گیرد.
- در هنگام جستجو، وقتی تابع Hash-Search به مکان Deleted برخورد می‌کند جستجو را ادامه می‌دهد، زیرا مکانی که با Deleted علامتگذاری شده در هنگام درج کلید مورد جستجو اشغال بوده است.
- زمان جستجو در روش آدرس‌دهی باز به علت استفاده از این علامتگذاری پرهزینه است و ممکن است نیاز باشد همهٔ جدول را بررسی کنیم. به همین دلیل در کاربردی که می‌خواهیم تعداد زیادی از کلیدها را حذف کنیم از روش زنجیرسازی استفاده می‌کنیم که زمان جستجوی بهتری دارد.

آدرس‌دهی باز

- دو روش برای طراحی توابع درهم‌سازی توسط آدرس‌دهی باز وجود دارد که در اینجا در مورد آنها صحبت خواهیم کرد : درهم‌سازی دوگانه [1] و وارسی خطی [2] .

[1] double hashing
[2] linear probing

آدرس‌دهی باز

- درهم‌سازی دوگانه [1] : درهم‌سازی دوگانه یکی از بهترین روش‌های موجود برای آدرس‌دهی باز است.
- درهم‌سازی دوگانه از تابع درهم‌سازی به صورت زیر استفاده می‌کند.

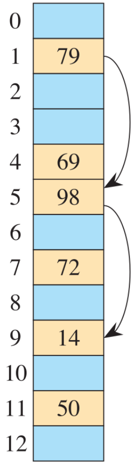
$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m$$

- به طوری که h_1 و h_2 دو تابع درهم‌سازی هستند.
- اولین مکان انتخابی $T[h_1(k)]$ است و مکان‌های انتخابی بعدی گام‌هایی به اندازه ضریبی از $h_2(k)$ بعد از مکان انتخابی اول هستند.

[1] double hashing

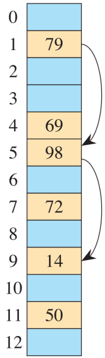
آدرس‌دهی باز

- در شکل زیر یک مثال از درهم‌سازی دوگانه نشان داده شده است.



آدرس‌دهی باز

- اندازه جدول 13 است و $h_1(k) = k \mod 13$ و $h_2(k) = 1 + (k \mod 11)$.
- کلید 14 بعد از وارسی مکان‌های 1 و 5 که اشغال شده‌اند وارد مکان 9 می‌شود.



آدرس‌دهی باز

- برای این که همه جدول مورد جستجو قرار بگیرد مقدار $h_2(k)$ باید نسبت به اندازه جدول یعنی m اول باشد. یک روش مناسب برای اطمینان حاصل کردن از برقراری این شرط است که m را توانی از ۲ انتخاب کنیم و h_2 را به گونه‌ای طراحی کنیم که همیشه عدد فرد تولید کند. یک روش دیگر این است که m را اول انتخاب کنیم و h_2 را به گونه‌ای طراحی کنیم که عددی مثبت کوچکتر از m تولید کند.
- برای مثال می‌توانیم m را یک عدد اول انتخاب کنیم و توابع را به صورت زیر طراحی کنیم.

$$h_1(k) = k \mod m$$
$$h_2(k) = 1 + (k \mod (m-1))$$

آدرس‌دهی باز

- به عنوان مثال، اگر $k = 123456$ و $m = 701$ باشد، آنگاه $h_1(k) = 80$ و $h_2(k) = 257$ خواهد بود، بنابراین اولین وارسی مکان 80 را انتخاب می‌کند و در وارسی‌های بعدی ضرایبی از 257 مکان بعدی (با محاسبه باقیمانده بر m) انتخاب می‌شوند تا وقتی که یک مکان خالی پیدا شود.
- اگرچه مقادیر m که اول یا توانی از ۲ نیستند می‌توانند در درهم‌سازی دوگانه استفاده شوند، اما در عمل انتخاب $h_2(k)$ وقتی از توابع دیگر استفاده می‌کنیم، سخت‌تر می‌شود، زیرا باید اطمینان حاصل شود که نسبت به m اول است.
- وقتی m عدد اول یا توانی از ۲ باشد، درهم‌سازی دوگانه تعداد $\Theta(m^2)$ دنباله وارسی تولید می‌کند زیرا هر جفت $(h_1(k), h_2(k))$ یک دنباله وارسی متفاوت است.

آدرس‌دهی باز

- وارسی خطی [1] : وارسی خطی یک حالت خاص از درهم‌سازی دوگانه است و یکی از روش‌های بسیار ساده در آدرس‌دهی باز برای رفع تصادم است.
- همانند درهم‌سازی چندگانه ابتدا مکان $T[h_1(k)]$ بررسی می‌شود. اگر این مکان اشغال بود، مکان بعدی یعنی $T[h_1(k)+1]$ بررسی می‌شود. بررسی تا $T[m-1]$ ادامه پیدا می‌کند و مجددا به $T[0]$ باز می‌گردد تا به $T[h_1(k)-1]$ برسد.
- بنابراین گام وارسی که در درهم‌سازی دوگانه برابر با $h_2(k)$ بود در وارسی خطی برابر با ۱ است پس $h_2(k) = 1$ به ازای همه مقادیر k .

[1] linear probing

آدرس‌دهی باز

- پس تابع درهم‌سازی در وارسی خطی به ازای $i = 0, 1, \cdots, m-1$ برابر است با :

$$h(k, i) = (h_1(k) + i) \mod m$$

- مقدار $h_1(k)$ می‌تواند هریک از مقادیر $\{0, 1, \cdots, m-1\}$ باشد.

آدرس‌دهی باز

- آدرس‌دهی باز را بر اساس ضریب بار جدول درهم‌سازی یعنی $\alpha = n/m$ تحلیل می‌کنیم.
- در آدرس‌دهی باز، حداکثر یک عنصر در یک مکان ذخیره می‌شود و بنابراین $n \leqslant m$ است و در نتیجه $\alpha \leqslant 1$ است.
- همانند قبل فرض می‌کنیم درهم‌سازی جایگشت‌ها را یکنواخت و مستقل تولید می‌کند. بنابراین دنباله وارسی $\langle h(k,0), h(k,1), \cdots, h(k,m-1) \rangle$ که برای درج یا جستجو برای کلید k استفاده می‌شود با احتمال برابر می‌تواند هریک از جایگشت‌های $\langle 0, 1, \cdots, m-1 \rangle$ باشد. البته هر کلید یک دنباله وارسی یکتا خواهد داشت.

آدرس‌دهی باز

- قضیه : به ازای یک جدول درهم‌سازی با آدرس‌دهی باز و ضریب بار $\alpha = n/m < 1$ ، تعداد وارسی‌ها در یک جستجوی ناموفق حداکثر $1/(1-\alpha)$ است.
- اثبات: اولین وارسی همیشه رخ خواهد داد، اگر مکان اول انتخاب شده با احتمال α اشغال باشد، انتخاب دوم بررسی می‌شود. احتمال این‌که هر دو انتخاب اول و دوم اشغال باشند برابر است با α^2 و به همین ترتیب الی آخر.
- کران به دست آمده برابر است با $1 + \alpha + \alpha^2 + \alpha^3 + \cdots$.
- با محاسبه این دنباله هندسی به دست می‌آوریم: $1 + \alpha + \alpha^2 + \alpha^3 + \cdots < 1/(1-\alpha)$.

آدرس‌دهی باز

- اگر α ثابت باشد، طبق این قضیه، جستجوی ناموفق در زمان $O(1)$ اجرا می‌شود.
- برای مثال، اگر جدول درهم‌سازی نیمه‌پر باشد، تعداد متوسط وارسی‌ها در یک جستجوی ناموفق حداکثر $1/(1-0.5) = 2$ است. اگر ۹۰ درصد اشغال باشد، تعداد متوسط وارسی‌ها حداکثر $1/(1-0.9) = 10$ است.