الگوریتم درهم سازی

محسن هوشمند
دانشکده تکنولوژی اطلاعات و علم رایانه
دانشگاه تحصیلات تکمیلی علوم پایه زنجان

درهم‌سازی

درهم‌ساز استفاده در تصادفی‌سازی و نمایش فشرده داده‌ها

ایفاء نقشی اصلی در داده‌ساختار احتمالاتی

تابع درهم‌ساز

- ورودی: قطعات داده ورودی با هر اندازه‌ای
- نگاشت به شناسه‌ای عددی با اندازهٔ کوچکتر (و غالبا ثابت) به نام مقدار درهم‌ساز یا درهم‌ساز

فشرده شدن ورودی با توابع درهم‌ساز ورودی

اجتناب‌ناپذیری یکسانی حاصل از اعمال تابع درهم‌ساز بر دو قطعه دادهٔ متفاوت

- «تصادم»

درهم‌سازی

غالباً نیازمند بررسی عمیق‌تر مستقل از میزان توجه به آن در دوره‌های برنامه‌نویسی، ساختمان داده‌ها و الگوریتم‌ها

حضور گستردهٔ داده‌ساختارهای مبتنی بر درهم‌سازی مانند جدول‌های درهم و نیز توابع درهم، در لایه‌های مختلف سامانه‌های نرم‌افزاری و شبکه‌ای

مثال- نگارش رایانامه

درهم‌سازی در ای‌نامه

نخستین گام
- احراز هویت و ورود به حساب کاربری،
- پردازش گذرواژهٔ کاربر با تابعی درهم‌ساز
- مقایسهٔ مقدار درهم حاصل با مقدار ذخیره‌شده در پایگاه
- رکن اصلی امنیت در بسیاری از سامانه‌های احراز

مرحلهٔ تدوین متن ایمیل
- استفاده از واژه‌نامه‌ای داخلی
- بهره‌گیری سامانهٔ بررسی املایی از درهم‌سازی برای تعیین وجود یا عدم وجود یک واژه در واژه‌نامهٔ داخلی
- نشان‌دهندهٔ نقش مهم درهم‌سازی در تسریع عملیات جست‌وجو

ارسال ایمیل،
- درهم‌سازی زوج نشانی‌های IP مبدأ و مقصد درهم می‌شوند جهت مشخص شدن سرور میانی بعدی جهت بستهٔ داده
- بخشی از سازوکار توزیع بار در شبکه
- منجر به کارایی بیشتر در مسیریابی بسته‌ها

در سمت مقصد
- استفاده از درهم‌سازی محتوای پیام در برخی سامانه‌های پالایش هرزنامه از درهم‌سازی
- شناسائی نشانه‌های مرتبط با پیام‌های ناخواسته و فیلتر پیام‌های مشکوک

درهم‌سازی- تابع و جدول

بررسی دو موضوعِ درهم‌سازی و جدول‌های درهم در ادامه

- شباهت دو موضوع ولی عدم یکسانی و اشتباه به جای یکدیگر
- ولی یکسان نبودن دو مفهوم یکسان نیستند، در این فصل بیش

بیشتر در رابطه با کاربرد درهم‌سازی در جدول‌های درهم و مختصری در زمینهٔ رمزنگاری

پایه‌ای برای استفاده از درهم‌سازی در سایر داده‌ساختارها

جدول‌های درهم، همچون خود درهم‌سازی، ساختارهایی فراگیر

- استفاده بدون اطلاع برنامه‌نویسان در استفادهٔ روزمره از نگاشت‌های کلید-مقدار از جدول درهم بهره می‌برند.

مقایسهٔ جدول‌های درهم با دیگر داده‌ساختارها

جهت درک دلیل استفادهٔ گسترده از جدول‌های درهم

- نیاز به مقایسه آنها با سایر ساختارهای داده در پیاده‌سازی داده‌ساختارِ لغت‌نامه،
- بررسی میزان پشتیبانی هر یک از سه عمل اصلی جست‌وجو، درج و حذف

احتمال بالای استفاده از درهم‌سازی

- در هر حوزه‌ای که امنیت یا سرعت بازیابی داده اهمیت داشته باشد

ساختارهای دادهٔ متعددی امکان ایفای نقش «لغت‌نامه»

- هر یک دارای الگو کارایی متفاوتی

مناسب برای پیرنگ‌های کاربردی گوناگون

آرایهٔ سادهٔ نامرتب

ساختاری ساده

عملیات درج- کارایی زمانی ثابت $O(1)$

- افزودن عضو جدید به انتهای آرایه

عملیات جست‌وجو

- در بدترین حالت نیازمند اسکن کامل آرایه و بنابراین دارای هزینهٔ زمانی خطی O(n)

قابل قبول بودن چنین ساختاری برای پیاده‌سازی لغت‌نامه

- درج بسیار فراوان و جست‌وجو بسیار کم‌تکرار باشد.

آرایهٔ مرتب

آرایه‌های مرتب

فراهم‌سازی امکان جست‌وجوی سریع‌تر با بهره‌گیری از جستجوی درختی

کاهش زمان جست‌وجو به O(log n)

برای بسیاری از اندازه‌های آرایه عملاً بسیار نزدیک به زمان ثابت

- مثال- لگاریتم دودوییِ عدد یک میلیارد یا $۱۰^۹$ کمتر از ۳۰

حفظ ترتیب مرتب در هنگام درج یا حذف

- تحمیل هزینهٔ زمانی خطی
- نیاز به جابجایی زیادی از مقادیر در بدترین حالت
- نامطلوب بودن هزینهٔ خطی در بسیاری از برنامه‌ها

فهرست پیوندی

فهرست‌های پیوندی

درج- برخلاف آرایه‌های مرتب، در زمان ثابت $O(1)$، با افزودن عنصر در ابتدای فهرست

حذف- در صورت داشتن اشاره‌گر مربوطه

- زمان ثابت با تغییر چند پیوند در زمان ثابت انجام می‌شود.

در فهرست‌های پیوندی تک‌سویه

- حذف نیاز به دانستن از عضو پیش از عضو هدف
- یافتن آن مستلزم پیمایش پیوندها
- نیاز مجدد به زمان خطی

در مجموع، در ساختارهای خطی مانند آرایه و فهرست پیوندی

- وجود همیشگی حداقل یک عملیات با هزینهٔ O(n) است،
- نیاز به ساختارهای غیرخطی برای اجتناب از آن

درخت‌های جست‌وجوی دودویی متوازن

درخت‌های جست‌وجوی دودویی متوازن

اجرای عملیات لغت‌نامه متناسب با عمق

نگه‌داری عمق در حد O(log n) با مکانیسم‌های متوازن‌سازی مانند AVL و قرمز–سیاه

- عملیات درج، جست‌وجو و حذف در بدترین حالت زمان لگاریتمی
- نزدیکی زمان لگاریتمی از نظر کارایی بسیار نزدیک‌تر به زمان ثابت است تا زمان خطی
- مناسب برای بسیاری از پیرنگ‌های عملی

حفظ نظم اعضا با درخت‌های دودویی متوازن نظم

انتخابی معمول برای اجرای سریع پرس‌وجوهای بازه‌ای، همچنین یافتن مقادیر قبلی و بعدی

درخت‌های دودویی متوازن بهترین عملکرد ممکن در میان ساختارهای مبتنی بر پایهٔ مقایسهٔ عناصر

مقایسه دو درخت

درختAVL متعادل‌تر

- فراهم‌سازی جستجوئی سریع‌تر

درخت سرخ-سیاه

- هنگام اولویت بیشتر عملیات درج و حذف
- به دلیل چرخش‌های کمتر

نیازهای داده‌ساختارهای جدید

عدم محدودیت طراحی ساختارهای دادهٔ کاراتر محدود به عملیات مقایسه

پشتیبانی رایانه‌ها از عملیات متنوعی مانند انتقال بیتی، محاسبات عددی و سایر تبدیل‌ها

استفاده مناسب و کارای توابع درهم‌ساز از همین عملیات جهت دور زدن محدودیت‌های لگاریتمی

جدول درهم‌ساز

مزیت بنیادین جدول‌های درهم‌ساز
- کاهش هزینهٔ همهٔ عملیات لغت‌نامه‌ای به میانگین زمان ثابت $O(1)$

عدم ضمانت برای بدترین حالت

امکان هزینهٔ خطی O(n) در بدترین حالت

طراحی مناسب جدول درهم‌ساز منجر به نادرسازی همیشگی چنین زمانی

تفاوت مهم با ساختارهای خطی
- خطی بودن جست‌وجو در آرایهٔ نامرتب به شکلی پایدار و قابل پیش‌بینی
- نادر بودن رخداد بدترین حالت در جدول درهم‌ساز

دلیل پخش‌شدگی یکنواخت در سطل‌ها
- استثنا: نگاشت چند ورودی غیرمرتبط به یک مقدار یکسان درهم

جدول درهم‌ساز

مراتب عدم استفاده

- مناسب نبود جدول‌های درهم در کاربردهایی با داشتن اهمیت حفظ ترتیب عناصر
- از بین بردن ذاتی ترتیب داده‌ها

ناکارآمدی در پاسخ‌دادن به پرس‌وجوهای وابسته به نظم

- مثال- پرس‌وجوهای بازه‌ای در پایگاه داده، مانند یافتن تمام سنین بین ۳۵ تا ۵۶ سال یا نقاط هندسی با مختصات x بین ۳ تا ۴۵،

بهترین عملکرد جدول‌های درهم بهترین عملکرد

- یافتن تطابقی دقیق امکان
- استفاده از درهم‌سازی برای پرس‌وجوهای مشابهت نیز
  - در تشخیص سرقت ادبی و انتحال استفاده کرد

مقایسه داده‌ساختارها

	جستجو	درج	حذف	یافتن قبلی و بعدی
آرایهٔ نامرتب	$O(n)$	$O(1)$	$O(n)$	$O(n)$
آرایهٔ مرتب	$O(\log n)$	$O(n)$	$O(n)$	$O(1)$
فهرست پیوندی	$O(n)$	$O(1)$	$O(1)$	$O(n)$
درخت جستجوی دودوئی متعادل	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$
جدول درهم	$O(1)$	$O(1)$	$O(1)$	$O(n)$

مثال-حذف افزونگی داده‌ها در سامانه‌های پشتیبان‌گیری و ذخیره‌سازی

بسیاری از شرکت‌ها مواجه با چالش نگهداری حجم بسیار بزرگی از داده‌های کاربران

مشتریان این سامانه‌ها غالباً سازمان‌های بزرگ

مدیریت مجموعه‌های عظیمی از داده‌ها

پشتیبانی یا **snapshot** منظم و با روال در چنین محیط‌هایی از داده‌ها

زمان صورت‌برداری با فاصله‌های زمانی کوتاه

- عدم تغییر بخش عمده‌ای از داده‌ها میان دو **snapshot** متوالی

اهمیت تشخیص و شناسائی سریع داده‌های تغییر یافته و ذخیره آنها

منجر به صرفه‌جویی قابل توجه در زمان و فضای ذخیره‌سازی

نیاز به تشخیص کارآمد محتوای تکراری

مثال-حذف افزونگی داده‌ها در سامانه‌های پشتیبان‌گیری و ذخیره‌سازی

فرایند حذف داده‌های تکراری
- Deduplication

در اغلب پیاده‌سازی‌های جدید استفاده از توابع درهم‌ساز

مثال- سامانه‌ای به نام ChunkStash که برای ارائهٔ گذردهی بالا با استفاده از حافظهٔ فلش
- تقسیم فایل‌ها به قطعه‌های کوچکی با اندازهٔ ثابت هشت کیلوبایت، تقسیم می‌شوند.
- تبدیل محتوای هر قطعه با استفاده از تابع درهم SHA-1 به اثر انگشتی بیست بایتی
- در صورت وجود ثبت قبلی از اثر انگشت به معنای ذخیره آن
  - نگه‌داری یک اشاره‌گر به دادهٔ موجود
- در صورت جدید بودن اثر انگشت، فرض بر جدید بودن قطعهٔ داده
- ذخیرهٔ قطعه و اثر انگشت همره با اشاره‌گر به محل ذهیرهٔ قطع در جدول درهم ساز در مخزن داده

دیگر مزایای تقسیم فایل‌ها به قطعه‌های
- امکان شناسایی داده‌های تقریباً تکراری
- اگر تنها بخش کوچکی از فایلی بزرگ تغییر کرده باشد، تنها دخیرهٔ همان قطعهٔ تغییر یافته به عنوان دادهٔ جدید
- عدم ذخیرهٔ مجدد سایر قطعه‌های بدون تغییر باقی

مثال–تشخیص انتحال با MOSS و اثرانگشتی‌سازی Rabin–Karp

سامانهٔ Measure of Software Similarity (MOSS)

- خدمتی جهت تشخیص انتحال و سرقت ادبی
- عمدتاً در ارزیابی تکالیف برنامه‌نویسی

از ایده‌های الگوریتمی اصلی در سامانهٔ MOSS

- بهره‌گیری از گونه‌ای اصلاح‌شده از الگوریتم تطبیق رشتهٔ Rabin–Karp
- تکیه الگوریتم مذکور بر تکنیک اثرانگشتی‌سازی k-gram
  - زیررشته‌های پیوسته با طول ثابت k

مثال-تشخیص انتحال با MOSS و اثرانگشتی‌سازی رابین-کارپ

مسئلهٔ تطبیق رشته

- رشتهٔ t نمایانگر متن بزرگ
- رشتهٔ p نمایانگر الگوی کوچک‌تری
- به دنبال وجود p در t

بخش اعظم از پژوهش‌ها

- بر مقایسهٔ مستقیم زیررشته‌های t و p

الگوریتم رابین-کارپ

- بر پایهٔ مقایسهٔ درهمِ زیررشته‌ها
- عملکردی بسیار سریع
- بخشی از این سرعت ناشی از استفادهٔ از درهم‌سازی

مثال-تشخیص انتحال با MOSS و اثرانگشتی‌سازی رابین-کارپ

رابین-کارپ

- مقایسهٔ نویسه‌به‌نویسه تنها در صورت برابر اعلام شدن درهم دو زیررشته
- در بدترین حالت، رخ دادن تصادم‌های زیاد
  - حالتی که دو زیررشتهٔ متفاوت درهم یکسانی ایجاد کنند
  - الگوریتم ناچار به صرف زمان $O(|t||p|)$ همانند روش جستجوی کامل
- مع‌الوصف در بیشتر کاربردهای عملی
  - اندک بودن تعداد انطباق‌های واقعی
  - استفاده از تابع درهم مناسب
- عملیاتی شدن الگوریتم با سرعتی خطی و بسیار کارآمد $O(|t||p|)$

مثال-تشخیص انتحال با MOSS و اثرانگشتی‌سازی رابین-کارپ

مثال- الگوریتم اثرانگشتی رابین-کارپ، هدف یافتن

- الگوی p = BBBBC
- در رشتهٔ بزرگ‌تر t = BBBBBBBBABBBDBBBBC
- مقدار درهم الگوی مدنظر BBBBC، برابر ۱۶۲
- ابتدای رشتهٔ t زیررشتهٔ BBBBB قرار دارد که مقدار درهم آن ۱۶۱
  - بنابراین، عدم‌انطباق رخ می‌دهد.
- با انتقال پنجره یک نویسه به راست
- بارها و بارها به مقادیر درهم متفاوتی برمی‌خوریم
  - همگی متفاوت با درهم الگو (۱۶۲)
- در ادامهٔ جست‌وجو، ظاهر شدن زیررشتهٔ ABBBD با مقدار درهم ۱۶۲
- برابری درهم زیررشتهٔ یافت شده با درهم الگو
  - بررسی نویسه‌به‌نویسه
  - نتیجهٔ بررسی یکی نبودن دو زیررشته
  - انطباق کاذب
- در انتهای رشتهٔ t مشاهدهٔ دوبارهٔ مقدار درهم ۱۶۲
  - بررسی نویسه‌به‌نویسهٔ زیررشتهٔ BBBBC
  - برابری زیررشته با الگو
  - برگرداندن انطباق درست

مثال-تشخیص انتحال با MOSS و اثرانگشتی‌سازی رابین-کارپ

هزینهٔ محاسبهٔ درهم وابسته به اندازهٔ زیررشته

- بنابراین، عدم تاثیر درهم‌سازی در افزایش سرعت
- اما، استفادهٔ رابین-کارپ از درهم غلطان (rolling hash)
- با داشتن درهم k-gram مربوط به بازهٔ $t[j, \dots, j+k-1]$
- محاسبهٔ درهم k-gram انتقال‌یافتهٔ تک نویسه به راست،
  - یعنی $t[j+1, \dots, j+k]$
- تنها زمان ثابت نیاز
- عملی بودن در صورت امکان حذف اثر نویسهٔ نخست از زیررشتهٔ قبلی «حذف» و افزودن اثر نویسهٔ جدید را «
  - روشی ساده از چنین تابع درهمی جمع مقادیر اسکی نویسه‌ها
  - در عمل استفاده از توابع پیچیده‌تری

مثال-تشخیص انتحال با MOSS و اثرانگشتی‌سازی رابین-کارپ

کاربرد مستقیم رابین-کارپ جهت بررسی انتحال

- شامل شکستن دو تکلیف برنامه‌نویسی به قطعه‌های کوچک و اثرانگشتی‌کردن آن‌ها

اما هدف MOSS مقایسهٔ تعداد زیادی از تکالیف دریافتی و یافتن همهٔ موارد بالقوهٔ مشابهت

مستلزم مقایسهٔ همهٔ فایل‌ها با یکدیگر و در نتیجه زمان اجرایی درجه دو

- در عمل غیرقابل‌پذیرش

حل مشکل در MOSS

- انتخاب تعداد محدودی اثر انگشت به‌عنوان نمایندهٔ هر فایل
- ایجاد نمایهٔ معکوس
- نگاشت هر اثر انگشت به موقعیت‌های وقوع آن در اسناد

استخراج مجموعهٔ فایل‌هایی دارای انطباق

جلوگیری از انجام مقایسهٔ نابجای همه‌باهم جلوگیری می‌کند.

وجود روش‌های متفاوتی برای انتخاب اثرانگشت‌های نماینده

- مثال- هر پنجرهٔ متوالی از نویسه‌ها در فایل، برای مثال پنجره‌ای به طول ۵۰ نویسه، انتخاب حداقلِ مقدار درهم میان k-gram-های موجود در آن پنجره
- انتخاب اثرانگشت در هر پنجره منجر به حفظ انطباق‌های طولانی و پیوسته میان دو سند و افزایش کارایی تشخیص

توابع درهم‌ساز جامع

جی. لارنس کارتر و مارک وگمن در سال ۱۳۵۸

پیشنهاد توابع درهم‌ساز جامعی

دارای ویژگی‌های ریاضی خاص

منجر به میانگین کمتری از تصادم‌ها علی‌رغم انتخاب تصادفی از بین کل داده‌های ورودی هستی

خانواده توابع درهم‌ساز جامع H

نگاشت هر عضوی از دامنه ورودی به مجموعه گسستهٔ $\{0,1, \dots, m-1\}$

تضمین احتمال پایین تصادم با انتخاب تصادفی تابعی درهم‌ساز از خانواده

$$pr\big(h(x) = h(y)\big) \leq \frac{۱}{m} \forall x, y: x \neq y$$

توابع درهم‌ساز جامع

انتخاب تصادفی تابعی درهم‌ساز از خانواده با ویژگی مذکور

- دقیقاً معادل انتخاب مقداری با احتمال یکنواخت تصادفی

استفادهٔ بسیاری از برنامه‌ها از خانواده درهم‌ساز

$$h_k(x) = \big((k \cdot x)\%p\big)\%m$$

p عدد اول با شرط $p \geq m$

معمولا برابر با یکی از اعداد اول مرسن

- مثلا، برای $m = 10^9$ از $p = M_{31} = 2^{31} - 1 \approx 2 \cdot 10^9$

k عدد صحیحی تصادفی به پیمانهٔ p و $k \neq 0$

خانواده تقریبا جامع

تضمین میانگین ریاضی احتمال تصادم کمتر از $\frac{2}{m}$

توابع درهم‌ساز جامع

خانواده‌ای پیچیده‌تر از توابع درهم‌ساز جامع برای درهم‌ساز اعداد صحیح

$$h_{k,q}(x) = \big((k \cdot x + q)\%\, p\big)\%\, m$$

- مقدار p عدد اول با شرط $p \geq m$
- k و q اعداد صحیح تصادفی به پیمانهٔ p و $k \neq 0$

خانواده‌های توابع درهم‌ساز فوق محدود به اعداد صحیح

عدم کارائی برای اکثر کاربردهای عملی نیازمند

- به درهم‌ساز بردارهای با اندازه متغیر و
- به توابع درهم‌ساز سریع و مطمئن و دارای ویژگی‌های خاص

سخن کوتاه، وجود رده‌های فراوانی از توابع درهم‌ساز در عمل

- انتخاب هر رده وابسته به طراحی آنها و نیاز مدنظر

معرفی چند مورد مهم از توابع درهم‌ساز مورد استفاده در داده‌ساختارهای احتمالی

توابع درهم‌ساز رمزنگاری

نگاشت‌های ثابت از رشته‌های بیتی ورودی با طول متغیر به رشته‌های بیتی خروجی با طول ثابت

- تفاوت ثابت نخست با ثابت دوم

اجتناب‌ناپذیری تصادم درهم‌ساز

- اما لزوم مقاوم بودن درهم‌ساز امن
- دشواری یافتن تصادم‌ها
- امکان یافتنی بختانهٔ تصادمیا محاسبهٔ از قبل
  - در نتیجه چنین رده‌ای از توابع نیاز به اثبات‌های ریاضی نیاز دارد.

اهمیت توابع درهم‌ساز رمزنگاری در رمزنگاری

- مانند امضاهای دیجیتال، طرح‌های احراز هویت و یکپارچگی پیام

توابع درهم‌ساز رمزنگاری

انتظارات از درهم‌سازهای رمزنگاری انتظار:

الف- ضریب تلاش، تابع درهم‌ساز رمزنگاری برای هر چه سخت‌تر کردن وارونگی جستجوی کامل رایانشا پرهزینه است.

ب- حالت گیرافتادگی، تابع درهم‌ساز رمزنگاری نباید برای یک الگوی ورودی محتمل دارای حالتی مخصوص بدان را دارا باشد.

ج- پراکنش یا انتشار، هر بیت خروجی تابع درهم‌ساز رمزنگاری باید تابعی از تمامی بیت‌های ورودی و با نسبت پیچیدگی یکسان از هر بیت ورودی باشد.

توابع درهم‌ساز رمزنگاری

تقسیم‌بندی توابع رمزنگاری بر اساس استفاده از کلید مخفی

- توابع درهم‌ساز کلیددار و توابع درهم‌ساز بدون

داده‌ساختارهای احتمالاتی استفاده از توابع درهم‌ساز

- شامل توابع درهم‌ساز یک‌طرفه، توابع درهم‌ساز مقاوم در برابر تصادم، و توابع درهم‌ساز جامع یک‌طرفه
- دارای تفاوت در برخی ویژگی‌ها با یکدیگر
- توابع درهم‌ساز یک‌طرفه شرایطی را برآورده می‌کنند.
  - الف- اعمال برای بلوک‌های داده با هر طولی
    - البته، در عمل، محدود به مقدار ثابتی بسیار بزرگ
  - ب-تولید خروجی با طول ثابت
  - ج- رعایت«مقاومت پیش‌تصویر» (خاصیت یک‌طرفگی)
    - یا یافتن ورودی که به خروجی مشخص شده درهم‌ساز شود باید از نظر محاسباتی غیرقابل اجرا باشد

توابع درهم‌ساز رمزنگاری

بعید بودن تولید مقدار درهم‌ساز برابر برای دو ورودی متفاوت در توابع درهم‌ساز مقاوم به تصادم

اگر توابع مذکور مقاوم به تصادم نباشند.

- در عوض باید توابع درهم‌ساز جهانی یک‌طرفه باید «مقاوم به تصادمِ هدف» یا «مقاوم به تصادمِ پیش‌تصویر دوم»
- به معنای از لحاظ رایانشی عملی نبودن یافتن ورودی دوم متمایز که دارای مقدار درهمی برابر با ورودی مشخص
- مقاوم بودن به تصادم برخورد به معنای مقاوم بودن به پیش‌تصویر دوم
  - اما پیچیدگی عمومی یافتن یک تابع مقاوم به پیش‌تصویر دوم بسیار بیشتر از یافتن یک جفت دارای تصادم

کندتر بودن توابع درهم‌ساز رمزنگاری نسبت به توابع غیررمزنگاری به دلیل طراحی

- به ویژه، شرط ضریب تلاش
- تابع SHA-1 در حدود ۵۴۰ مگابایت بر ثانیه است، در مقابل توابع غیررمزنگاری در حدود ۲۵۰۰ مگابایت بر ثانیه و بیشتر

الگوریتم‌های چکیده پیام

معرفی الگوریتم چکیده پیام MD5

- ران رایوست (ر در رش‌ا و ر در س‌ل‌رس) در سال ۱۳۷۰

جانشین استاندارد قدیمی MD4

تابعی درهم‌ساز رمزنگاری

- معرفی شده در IETF RFC 1321

روش

- دریافت پیامی با طول دلخواه و تولید خروجی درهم‌ساز ۱۲۸ بیتی منحصربه‌فرد
- استواری الگوریتم $MD5$ بر طرح مرکل-دامگور
- در مرحلهٔ اول، تقسیم ورودی با اندازه دلخواه را با استفاده از تابع پدگذاری سازگار با MD به تعدادی بلوک با اندازه ثابت
  - بلوک‌های ۵۱۲ بیتی یا شانزده کلمه ۳۲ بیتی
- پردازش یک به یک بلوک‌های آماده شده با استفاده از تابع فشرده‌سازی خاص
- استفادهٔ هر بلوک بعدی از نتیجه خروجی قبلی
- چکیدهٔ نهایی MD5 مقدار ۱۲۸ بیتی
  - تولید پس از پردازش آخرین بلوک

الگوریتم‌های چکیده پیام

پدگذاری پیام

- افزودن، بیت خاتمه، سپس بیت‌های صفر تا طول آن در پیمانه ۵۱۲ مساوی ۴۴۸
- در پایان، افزودن طول ۶۴ بیتی پیام اصلی

تقسیم پیام پدگذاری شده در قالب بلوک‌های ۵۱۲ بیتی

- هر بلوک شامل شانزده کلمه ۳۲ بیتی

مقداردهی اولیه

- چهار ثبات ۳۲ بیتی با مقادیر ثابت اولیه مقداردهی تشکیل دهندهٔ حالت داخلی الگوریتم
- ورود ترتیبی هر بلوک پیام به تابع فشرده‌سازی
  - دارای چهار دور پردازش است و مجموعاً ۶۴
  - هر دور، استفاده از توابع غیرخطی، عملگرهای منطقی، جمع پیمانه‌ای و چرخش‌های بیتی
    - نقش آن افزایش آشفتگی و انتشار در حالت داخلی
  - خروجی هر مرحله به عنوان ورودی مرحله بعد

الگوریتم‌های چکیده پیام

در دور اول استفاده از تابع F و تولید بیت به بیت

- هر بیت بر اساس انتخاب شرطی عمل می‌کند

$$F(B,C,D) = (B \text{ و } C) \text{ یا } (\text{نقیض } B \text{ و } D)$$

- رفتار «انتخابی» شبیه است و ایجاد وابستگی شدیدی به بیت‌های ورودی
- موجب رفتاری غیرخطی

در دور دوم استفاده از تابع G

- ساختاری مشابه F با چیدمان متفاوتی از بیت‌ها

$$G(B,C,D) = (B \text{ و } D) \text{ یا } (C \text{ و نقیض } D)$$

- هدف آن تقویت پخش‌شدگی در مرحله میانی است.

در دور سوم استفاده از تابع H

$$H(B,C,D) = B\ XOR\ C\ XOR\ D$$

- تابعی خطی‌تر است (چرا؟) اما ایجاد تغییرپذیری و ترکیب‌پذیری بیشتری در بیت‌ها ایجاد می‌کند.

در دور چهارم استفاده از تابع I

$$I(B,C,D) = C\ XOR\ (B \text{ یا نقیض } D)$$

- رفتار غیرخطی پیچیده‌تری نسبت به H و ایجاد تغییرات بزرگ در حالت داخلی

الگوریتم‌های چکیده پیام

در هر یک از ۶۴ گام

- ترکیب خروجی این توابع به همراه یکی از کلمات بلوک پیام، یک ثابت از پیش تعریف‌شده و چرخش چپ
- منجر به به‌روزرسانی حالت داخلی

در پایان پردازش آخرین بلوک

- ترکیب چهار ثبات نهایی با یکدیگر
- تولید مقدار درهم ۱۲۸ بیتی
  - چکیده پیام

استفاده گسترده از الگوریتم $MD5$ در بررسی یکپارچگی داده‌ها

- به جای مقایسه مستقیم داده‌های بزرگ، مقایسهٔ مقادیر درهم محاسبه شده از فایل‌ها با یکدیگر
- مشخص شدن تغییر یا عدم داده‌ها

الگوریتم‌های چکیده پیام

با وجود کاربرد گسترده MD5 در گذشته

- پژوهش‌ها تاییدی بر مصون نبودن الگوریتم در برابر حملات تصادم
- گزارش امنیتی VU#8360682 تأیید کنندهٔ آسیب‌پذیری الگوریتم در برابر حملات تصادم
- در چنین حملاتی امکان تولید دو پیام متفاوت با مقدار درهم یکسان

ضعف اجازه به مهاجمان جهت تولید اسناد، گواهی‌ها یا توکن‌های رمزنگاری جعلی

- اما معتبر از نظر سامانه‌های مبتنی بر MD5

توصیه به عدم استفاده از MD5 در کاربردهای امنیتی و رمزنگاری دیگر

اما امکان استفاده در برخی کاربردهای غیرامنیتی مانند ساختارهای داده احتمالاتی یا بررسی ساده یکپارچگی

الگوریتم‌های درهم‌ساز امن

تعریف الگوریتم‌های درهم‌ساز امن در آژانس امنیت ملی ایالات متحده (NSA)
- انتشار با مؤسسهٔ ملی استانداردها و فناوری (NIST)

انتشار اولین الگوریتم از خانواده، به نام SHA-0، در سال ۱۳۷۲

جانشینی به سرعت با SHA-1
- تولید مقدار درهم‌ساز بلندتر ۱۶۰ بیتی (۲۰ بایت) در SHA-1
  - افزایش امنیت آن با رفع نقاط ضعف SHA-0
- استفاده از SHA-1 در طول سالیان در برنامه‌های مختلف
- امضای بیشتر وب‌سایت‌ها با استفاده از الگوریتم‌های مبتنی بر آن امضا
- کشف ضعف در سال ۱۳۸۴
- توقف استفاده از آن در نتیجه در سال ۱۳۸۹ NIST
  - توقف استفاده از آن از سال ۱۳۹۰ در اینترنت
- مانند MD5، نقاط ضعف یافت شده بر استفاده از آن به عنوان تابع درهم‌ساز برای ساختارهای داده احتمالاتی تأثیری نداشت.
- معرفی SHA-2 در سال ۱۳۸۰ منتشر
  - شامل شش تابع درهم‌ساز با اندازه چکیده‌های متمایز SHA-224، SHA-256، SHA-384، SHA-512، و جز اینها
  - قدرت بیشتر SHA-2 نسبت به SHA-1
  - نامحتمل بودن موفقیت حملات علیه SHA-2 با قدرت رایانشی فعلی

الگوریتم‌های درهم‌ساز امن

```
function SHA۰(message):
                                                    # پَد کردن پیام طبق قواعد خانواده SHA
    padded_message = sha_pad(message)
                                                    # مقداردهی اولیه‌ی پنج مقدار داخلی مثل SHA − ۱
    H۰= ۰x۶۷۴۵۲۳۰۱
    H۱= ۰xEFCDAB۸۹
    H۲= ۰x۹۸BADCFE
    H۳= ۰x۱۰۳۲۵۴۷۶
    H۴= ۰xC۳D۲E۱F۰


                                                    # پردازش پیام در بلوک‌های ۵۱۲ بیتی و شکستن بلوک به شانزده کلمه‌ی ۳۲ بیتی
    for each ۵۱۲-bit block M:
        for i from ۰ to ۱۵:
            W[i] = M[i]
                                                    # گسترش برنامه‌ی پیام نسخه SHA-۰
        for t from ۱۶ to ۷۹:
```

```
            W[t] = (W[t-۳] XOR W[t-۸] XOR W[t-۱۴] XOR W[t-۱۶])
                                                    # توجه: SHA-۰ در این مرحله هیچ چرخشِ بیتی انجام نمی‌دهد


                                                    # مقداردهی متغیرهای کاری
        A = H۰, B = H۱, C = H۲, D = H۳, E = H۴
```

الگوریتم‌های درهم

```
# هشتاد دور محاسبه
for t from ۰ to ۷۹:
    if ۰ ≤ t ≤ ۱۹:
        f = (B AND C) OR ((NOT B) AND D)
        K = ۰x۵A۸۲۷۹۹۹
    else if ۲۰ ≤ t ≤ ۳۹:
        f = B XOR C XOR D
        K = ۰x۶ED۹EBA۱
    else if ۴۰ ≤ t ≤ ۵۹:
        f = (B AND C) OR (B AND D) OR (C AND D)
        K = ۰x۸F۱BBCDC
    else:
        f = B XOR C XOR D
        K = ۰xCA۶۲C۱D۶

# گام اصلی
    TEMP = (leftrotate(A, ۵) + f + E + W[t] + K) mod ۲^۳۲
    E = D
    D = C
    C = leftrotate(B, ۳۰)
    B = A
    A = TEMP

#به‌روزرسانی مقادیر هش
    H۰ = (H۰ + A) mod ۲^۳۲
    H۱ = (H۱ + B) mod ۲^۳۲
    H۲ = (H۲ + C) mod ۲^۳۲
    H۳ = (H۳ + D) mod ۲^۳۲
    H۴ = (H۴ + E) mod ۲^۳۲

#خروجی نهایی ۱۶۰ بیت است
return H۰ || H۱ || H۲ || H۳ || H۴
```

الگوریتم‌های درهم‌ساز امن

تفاوت اصلی $SHA-0$ با $SHA-1$ در گام زیر

W[t] = W[t-۳] XOR W[t-۸] XOR W[t-۱۴] XOR W[t-۱۶]

عدم استفاده از مقدار «چرخش به چپ ۱ بیتی» در SHA-0، ولی کارگذاری در SHA-1

- موجب پخش بهتر و امنیت بیشتر

W[t] = rotate(W[t-۳] XOR W[t-۸] XOR W[t-۱۴] XOR W[t-۱۶])

الگوریتم‌های درهم‌ساز امن-مثال

پیام جهت رمزگذاری حرف A با کد اسکی A=0x41 و استفاده از SHA-0

۱. ساخت بلوک ۵۱۲ بیتی: به طوری کلی فرایند کلی به صورت زیر است.
پیام A با پدگذاری در ابتدا ۰x۴۱ قرار می‌گیرد. سپس، ۰x۸۰ قرار گرفته و پس از آن چندین صفر قرار می‌گیرند. در انتها طول پیام (۸ بیت یا ۰x۰۰۰۰۰۰۰۸) اضافه می‌شود. پس، بلوک ۵۱۲ بیتی را به شکل ۱۶ کلمه ۳۲ بیتی نشان می‌دهیم:

W[۰] = ۰x۴۱۰۰۰۰۸۰
W[۱] = ۰x۰۰۰۰۰۰۰۰
W[۲] = ۰x۰۰۰۰۰۰۰۰
W[۳] = ۰x۰۰۰۰۰۰۰۰
...
W[۱۴] = ۰x۰۰۰۰۰۰۰۰
W[۱۵] = ۰x۰۰۰۰۰۰۰۸

الگوریتم‌های درهم‌ساز امن-مثال

۲. گسترش برنامه‌ی پیام (Message Schedule): SHA-۰ از فرمول زیر استفاده می‌کند:

W[t] = W[t−۳] XOR W[t−۸] XOR W[t−۱۴] XOR W[t−۱۶]

«سه مقدار اول» را به صورت عددی حساب می‌کنیم تا روش کاملاً روشن شود.

محاسبه W[۱۶]

W[۱۶] = W[۱۳] XOR W[۸] XOR W[۲] XOR W[۰]

چون همه صفر هستند جز W[۰]:

W[۱۶]=۰x ۰۰۰۰۰۰۰۰XOR ۰x ۰۰۰۰۰۰۰۰XOR ۰x ۰۰۰۰۰۰۰۰XOR ۰x ۴۱۰۰۰۰۸۰= ۰x۴۱۰۰۰۰۸۰

W[۱۷] = W[۱۴] XOR W[۹] XOR W[۳] XOR W[۱]

تمام ورودی‌ها صفر هستند:

W[۱۷] = ۰x۰۰۰۰۰۰۰۰

W[۱۸] = W[۱۵] XOR W[۱۰] XOR W[۴] XOR W[۲]

W[۱۵] = ۰x ۰۰۰۰۰۰۰۸

و بقیه صفر:

W[۱۸] = ۰x۰۰۰۰۰۰۰۸

همان‌طور که از مثال برمی‌آید گسترش پیام در SHA-۰ ساختاری نسبتاً ساده دارد.

الگوریتم‌های درهم‌ساز امن-مثال

۳. مقداردهی اولیه متغیرهای کاری

A = ۰x ۶۷۴۵۲۳۰۱

B = ۰xEFCDAB ۸۹

C = ۰x۹۸BADCFE

D = ۰x۱۰۳۲۵۴۷۶

E = ۰xC۳D۲E۱F۰

که همان مقدار اولیه استاندارد SHA-۰ و SHA-۱ است.

الگوریـ

۴. اجرای چند دور اول: برای قابل پیگیری ماندن مثال صرفا سه دور اول را از هشتاد دور را نمایش می‌دهیم

دور صفر: در بازه $t = ۰$ تا ۱۹ از تابع f ۱ استفاده می‌شود:

f = (B AND C) OR ((NOT B) AND D)

اگر مقداردهی انجام شود:

f ≈ ۰x ۱۰۳۲۵۴۷۶

K = ۰x۵A۸۲۷۹۹۹

W[۰] = ۰x۴۱۰۰۰۰۸۰

محاسبه TEMP:

TEMP = leftrotate(A,۵) + f + E + W[۰] + K

leftrotate(A,۵) ≈ ۰xE۸A۴۶۰۲C

محاسبه جمع:

TEMP ≈ 0xE۸A۴۶۰۲C + ۰x۱۰۳۲۵۴۷۶+ ۰xC۳D۲E۱F۰+۰x۴۱۰۰۰۰۸۰+۰x۵A۸۲۷۹۹۹

= ۰xA۰F۸A ۷۱۱(mod ۲^۳۲)

به‌روزرسانی متغیرها:

A = TEMP = ۰xA۰F۸A۷۱۱

B = قبلی A = ۰x۶۷۴۵۲۳۰۱

C = leftrotate(B,۳۰) = ۰x۵۹D۴۸D۸C

D = قبلی C = ۰x۹۸BADCFE

E = قبلی D = ۰x۱۰۳۲۵۴۷۶

الگوریتم

دور ۱: f همان تابع (چون $t = ۱$ هنوز در ۰–۱۹ است)

W[۱] = ۰x ۰۰۰۰۰۰۰۰

K = ۰x۵A۸۲۷۹۹۹

leftrotate(A,۵)

≈۰x۱F۱۴E۲۳۳

TEMP = ۰x۱F۱۴E ۲۳۳ + f(جدید) + E + W[۱] + K ≈۰x۹D۲۲۸۹AB

به‌روزرسانی:

A = ۰x۹D۲۲۸۹AB

B = ۰xA۰F۸A ۷۱۱

C = leftrotate(B,۳۰) = ۰x۲۸۳E۲۹C ۴

D = ۰x۵۹D۴۸D۸C

E = ۰x۹۸BADCFE

دور دوم:

W[۲] = ۰

K = همان مقدار

leftrotate(A,۵) ≈۰xA۴۵۱۳۷۵۷

TEMP ≈ ۰x۸D۳FA۷۶۰

به‌روزرسانی:

A = ۰x۸D۳FA۷۶۰

B = ۰x۹D۲۲۸۹AB

C = ۰x۲۸۳EA۹C۲

D = ۰x۲۸۳E۲۹C٤

E = ۰x۵۹D۴۸D۸C

تا اینجا سه دور اول از ۸۰ دور انجام شده است. روند مزبور تا دور ۷۹ ادامه می‌یابد. در پایان، مقادیر A,B,C,D,E به مقدارهای اولیه افزوده می‌شوند تا خروجی نهایی بلوک تولید شود.

الگوریتم‌های درهم‌ساز امن

الگوریتم SHA-2

ورودی: پیامM

خروجی: هش(M)SHA-2

۱ .پدگذاری پیام با یک بیت ۱ و چند بیت ۰ به اندازه مناسب تا طول آن برابر با k

۲. تقسیم پیام پد شده به بلاک‌هایی با طول ثابت

۳. تنظیم مقدار اولیه درهم‌ها

۴ .برای هر بلاک:
- تعیین مقادیر اولیه word
- ۶۴ دور (round) اجرا می‌شود:
  - محاسبه تابع چرخش و انتخاب(choose, majority)
  - به‌روزرسانی مقادیر word
- به‌روزرسانی مقدار درهم‌ها با نتایج دورها

۵ .خروجی، ترکیب نهایی مقدار درهم

توابع درهم‌ساز غیررمزنگاری

توابع غیر رمزنگاری عدم طراحی برای دفع حملات با هدف یافتن تصادم

- عدم نیاز امنیت و مقاومت بالا در برابر تصادم

صرفاً باید سریع باشند و تضمین احتمال پایین تصادم

فراهم کردن سریع درهم‌سازی مقدار زیادی داده با احتمال خطای معقول

فاولر-نول-وو FOWLER/NOLL/VO

معرفی در سال ۱۳۷۰ بر اساس داوری مقاله، توسط گلن فاولر و فونگ وو

- ارسال به کمیته IEEE POSIX P1003.2
- سپس بهبود آن توسط لندون کرت نول

استفاده در جدول‌های درهم، اثرانگشتی‌سازی داده‌ها، اندیس‌گذاری فایل‌ها و کاربردهای سبک درهم‌سازی

اساس طراحی آن

- عملیات بسیار کوچک و سریع
- تولید توزیع مناسبی پیاده‌سازی آسان آن در هر زبان برنامه‌نویسی
- پردازش بیت به بیت ورودی جریانی

فاولر-نول-وو FOWLER/NOLL/VO

نگهداری حالت داخلی مقداردهی شده با پایه افست ویژه

تکرار بر روی بلوک‌های ورودی ۸ بیتی

ضرب حالت در یک ثابت عددی بزرگ، به نام عدد اول ف‌ن‌و

اعمال OR انحصاری منطقی (XOR) به بلوک ورودی

گزارش مقدار حاصل از حالت به عنوان مقدار درهم پس از پردازش آخرین ورودی

عدد اول ف‌ن‌و و ثابت‌های پایه افست پارامترهای طراحی

- وابسته به طول بیتی مقادیر درهم‌ساز تولید شده

انتخاب اعداد اول بخشی از مزیت الگوریتم ف‌ن‌و

- تولید درهم بهتر در برخی از اعداد اول از بقیه

الگوریتم فَن‌و-۱الف

نسخه بهبودیافته FNV1a تغییری جزئی در ترتیب عملیات

در ابتدا اعمال عملگر XOR روی حالت و بایت ورودی

سپس ضرب حاصل در عدد اول FNV

با وجود استفاده FNV1a از همان پارامترهای اولیه و عدد اول مانند FNV1

- این جابه‌جایی ساده بدون آنکه تأثیر منفی بر عملکرد پردازنده داشته باشد
- موجب پراکندگی (توزیع) اندکی بهتر بیت‌های خروجی

خانواده FNV شامل الگوریتم‌هایی برای مقادیر درهم ۳۲، ۶۴، ۱۲۸، ۲۵۶، ۵۱۲ و ۱۰۲۴ بیتی

الگوریتم ف‌ن‌و-۱

الگوریتم FNV با وجود سادگی بسیار زیاد در پیاده‌سازی

عملکرد مناسب برای رشته‌های تقریباً یکسان

اما خروجی آن نسبتاً «تنک» (sparse) است

- بسیاری از بیت‌ها صفر هستند
- این ضعف، نامناسب کردن آن برای کاربردهای رمزنگاری

استفاده در سرورهای DNS، توییتر، درهم‌سازهای نمایه پایگاه داده، موتورهای جستجوی وب و بسیاری سامانه‌های دیگر

الگوریتم فَن‌و-۱

همان‌طور که اشاره شد، فَن‌و بر سه مؤلفهٔ اصلی استوار است:

1. پایهٔ افست **Offset basis**: عددی ثابت و بزرگ که حالت اولیهٔ درهم را تعیین می‌کند. به عنوان مثال (برای نسخه‌های استاندارد): نسخهٔ ۳۲ بیتی 0x811C9DC5 و نسخه ۶۴ بیتی 0xCBF29CE484222325 است.
2. عدد اول **FNV prime**: عدد اول بزرگ که برای پخش کردن (**mixing**) بیت‌ها استفاده می‌شود. در نسخهٔ ۳۲ بیتی 0x01000193 و در نسخهٔ ۶۴ بیتی 0x100000001B3 است.
3. دو عمل ساده برای هر بایت ورودی اعم از یای انحصاری و ضرب در عدد اول است و در پایان همیشه عملیات به پیمانهٔ ۲ به توان **n** (مثلاً 2^{32} یا 2^{64}) نگه داشته می‌شود.

سادگی دلیل سرعت بالای FNV

الگوریتم فَن‌و-۱

برای هر بایت ورودی b:

```
hash = offset_basis
for each byte b:
    hash = hash * FNV_prime
    hash = hash XOR b
return hash
```

در پایان: مقدار درهم hash نتیجهٔ نهایی

الگوریتم فَن‌و-۱الف

نسخه مزبور رایج‌تر و مناسب‌تر است زیرا این نسخه ترتیب عملیات را تغییر می‌دهد و کیفیت پراکندگی را بیشتر می‌کند:

```
hash = offset_basis
for each byte b:
    hash = hash XOR b
    hash = hash * FNV_prime
return hash
```

الگوریتم فـن‌و-۱الف -- مثال

درهم رشتهٔ A با استفاده از فـن‌و-۱الف نسخهٔ ۳۲ بیتی

پایه آفست برابر

Hash= 0x811C9DC5

گام ۱: عمل XOR با بایت 0x41

hash = ۰x۸۱۱C۹DC۵XOR ۰x۴۱= ۰x۸۱۱C۹D۸۴

گام ۲: ضرب در FNV prime

hash = (۰x۸۱۱C۹D ۸۴* ۱۶۷۷۷۶۱۹)% 2^{32} = ۰xE۴۰C۲۹۲C

درهم نهایی A:

۰xE۴۰C۲۹۲C

الگوریتم فن‌و-۱

ضرب در عدد اول بزرگ

- منجر به پخش شدن موثر بیت‌ها به شکل مؤثری
- اجتناب از ایجاد الگوهای تکراری

یای انحصاری

- ترکیب مستقیم بایت ورودی با حالت فعلی
- ترکیب این دو منجر به تولید درهمی سبک و بسیار سریع ولی با توزیع مناسب

برتری فن‌و-۱الف نسبت به نسخهٔ اولیه

- در «رفتار بهمنی» (Avalanche effect) بهتر و تصادم‌های کمتر

الگوریتم فَن‌و-۱

دلیل رفتار بهمنی بهتر در فَن‌و-۱الف نسبت به فَن‌و-۱

- در فَن‌و-۱الف
  - انجام عمل XOR با بایت ورودی پیش از ضرب
  - تأثیر بایت جدید کاملاً در حالت داخلی ترکیب می‌گردد و سپس پخش شدن در ضرب
- در فَن‌و-۱
  - ضرب پیش از XOR رخ می‌دهد
  - اضافه شدن بایت ورودی به حالت بدون پخش کافی

مناسب نبودن فَن‌و برای درهم رمزنگاری و گذرواژه یا امنیت

مناسب در جدول‌های درهم، مقایسهٔ سریع فایل‌ها (checksums)، کلیدهای درهم در پروتکل‌های شبکه، حذف افزونگی، اثرانگشت‌های ۶۴ بیتی، و تولید بذر اولیهٔ درهم‌های غلطان یا سریع

الگوریتم درهم مورمور

روش‌های درهم مورمور MurmurHash

- خانواده‌ای از توابع درهم‌ساز غیررمزنگاری
- آستین اپلبی در سال ۱۳۸۶ خورشیدی
- نسخه نهایی، $MurmurHash3$، در سال ۱۳۹۰

هدف اصلی درهم مورمور

- سرعت بسیار بالا همراه با توزیع عالی (اثر بهمنی خوب)
- طراحی برای کاربردهایی مانند جدول‌های درهم، فیلترهای بلوم، و کش‌های کلید-مقدار
- نامناسب در اهداف رمزنگاری

نسخه‌هائی شامل $MurmurHash3$ برای ۳۲ بیت ($x86$)، ۶۴ بیت ($x64$) و ۱۲۸ بیت

الگوریتم درهم مورمور

در نسخهٔ $MurmurHash3$ برای ۳۲ بیت $(x86)$،
- پردازش ورودی به صورت بلوک‌های ۴ بایتی (۳۲ بیتی)
- در صورت مضرب ۴ نبودن طول ورودی: پردازش بایت‌های باقی‌مانده جداگانه

دو ثابت طراحی که
- انتخاب شده به صورت تجربی
- جهت اختلاط (mixing) بیت‌ها ۱c =۵۱d۲e۹xcc۰

c۲= ۰x۱b۸۷۳۵۹۳
- انتخاب به نحوی که منجر به پخش‌شدگی (اثر بهمن) بهینه

تابع کمکی یا چرخش به چپ به اندازه r
- انتقال بیت‌ها به چپ
- بیت‌های بیرون زده از سمت چپ، به سمت راست وارد می‌شوند.
- مثال- در ۳۲ بیت زیر داریم: $rotl32(0x12345678,4) = 0x23456781$
  - پیاده‌سازی با سه شیفت و یک OR (نشان دهید)

الگوریتم درهم مورمور- مراحل

مرحلهٔ ۱

- مقداردهی اولیه صورت
- مقدار h (حالت داخلی) برابر با دانه اولیه که کاربر مشخص می‌کند

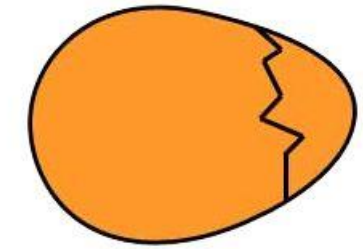
مرحلهٔ ۲

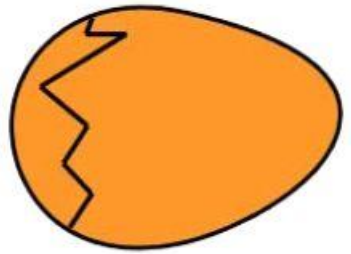
- پردازش بلوک‌های ۴ بایتی:

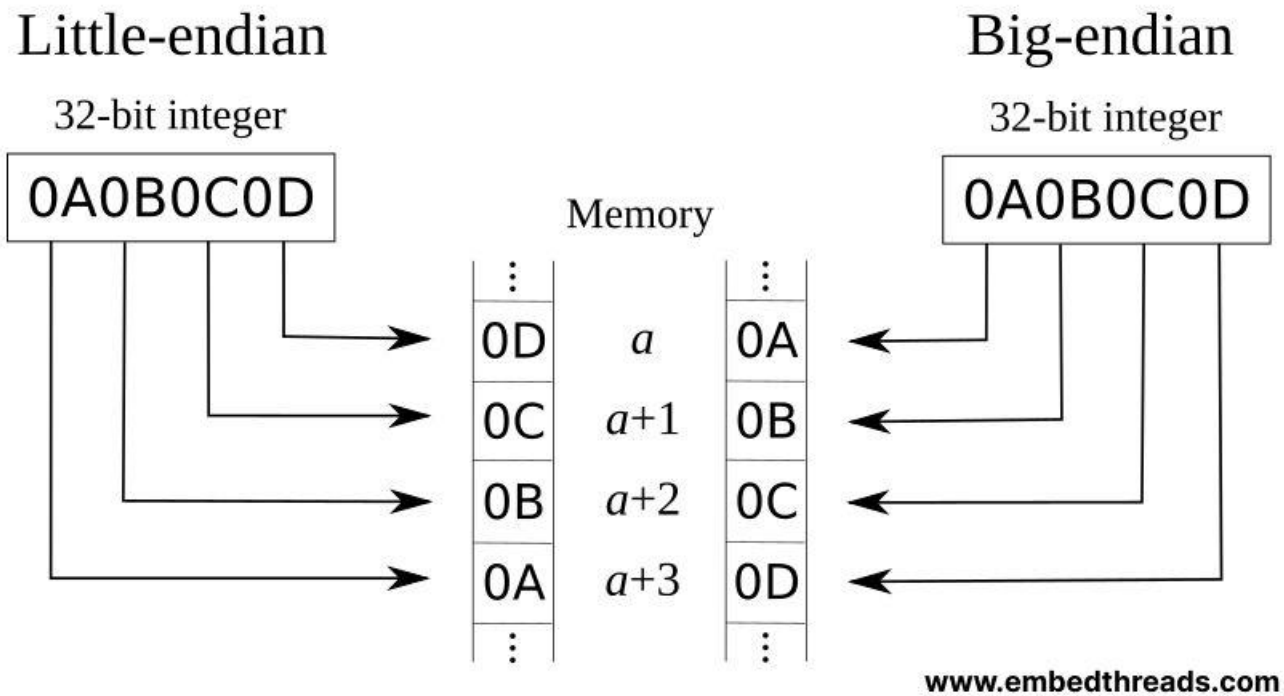
1. مقدار k برابر مقدار بلوک جاری است که به صورت little-endian از بایت‌ها خوانده می‌شود.
2. مقدار k در ثابت نخست ضرب می‌شود: k = k * c1.
3. مقدار حاصل ۱۵ بیت به چپ چرخش داده می‌شود: k = rotl32(k, 15).
4. حاصل در ثابت دوم ضرب می‌شود: k = k * c2.
5. حاصل با حالت ترکیب می‌شود: h = h xor k.
6. حالت چرخش می‌یابد: h = rot32(h, 13)
7. اختلاط دیگر اجرا می‌شود: h = h * 5 + 0xe6546b64.

LITTLE ENDIAN - The Lilliputians break their eggs at the smaller end

BIG ENDIAN - The Blefuscudians break their eggs at the big end.







www.embedthreads.com

الگوریتم درهم مورمور

مرحلهٔ ۳

- پردازش بایت‌های باقی‌مانده (ذیل)
- باقی ماندن یک تا سه بایت در صورت مضرب ۴ نبودن طول ورودی
- قرار گرفتن بایت‌های مذکور در متغیر ۳۲ بیتی k
  - بایت اول در کم‌ارزش‌ترین بیت
- مقداردهی صفر k یا k=0
- در صورت وجود بایت سوم k = k XOR(B[2]<<16)
- در صورت وجود بایت دوم k = k XOR(B[1]<<8)
- در صورت وجود بایت اول k = k XOR(B[0])
- سپس، k حاصل مانند بلوک کامل (مراحل دو-۱ تا دو-۴) پردازش شده و ترکیب با h

الگوریتم درهم مورمور

مرحلهٔ ۴

- افزودن طول پیام جهت اجتناب از تصادم بین پیام‌های با طول متفاوت و محتوای یکسان
- h = h XOR length

مرحلهٔ ۵

- نهایی‌سازی و استفاده از تابعی خاص مشهور به تابع $fmix32$
- هدف این مرحله از بین بردن هرگونه الگوی خطی باقی‌مانده و اطمینان از تاثیر هر بیت از ورودی روی همه بیت‌های خروجی
- h = h XOR (h >> ۱۶)
- h = h * ۰x۸۵ebca۶b
- h = h XOR (h >> ۱۳)
- h = h * ۰xc۲b۲ae۳۵
- h = h XOR (h >> ۱۶)

در مرحله ۶: برگرداندن مقدار h به عنوان درهم‌ساز نهایی

الگوریتم درهم مورمور

مثال- اجرای $MurmurHash3$ (نسخه ۳۲ بیتی)

- ورودی کوچک abc در قالب بایت‌ها و بر اساس کد اسکی‌در قالب شانزده شانزدهی
- دانه یا مقدار اولیه برابر صفر
- مرحلهٔ ۱ تبدیل ورودی به بلوک‌های۴ بایتی
  - چون طول پیام ۳ بایت، عدم وجود بلوک کامل
  - ورود به مرحلهٔ سوم یا مرحلهٔ پردازش ذیل

tail =[61,62,63]

- k = ۰
- k ^= ۶۳<< ۱۶
- k ^= ۶۲<< 8
- k ^= ۶۱

جدول‌های درهم‌ساز

محاسبه

- k =
- ۶۵۵۳۶ * ۶۳= ۴۱۲۸۷۶۸
- ۲۵۶ * ۶۲= ۱۵۸۷۲
- ۶۱

پس

- k = ۴۱۲۸۷۶۸+ ۱۵۸۷۲+ ۶۱

عدد دقیق به هگز

- k = ۰x۰۰۳e۳d۳d

حال وارد مراحل الگوریتم می‌شویم

- k = k * c۱
- c۱= ۰xcc۹e۲d۵۱

محاسبه

- k = ۰x۰۰۳e۳d۳d × ۰xcc۹e۲d ۵۱

نتیجه مقدار بزرگ ۶۴ بیتی ، اما در نهایت نگهداری فقط ۳۲ بیت (ضرب پیمانه ۳۲ بیت)

جدول‌های درهم‌ساز

اعمال چرخش چپ ۱۵ بیتی بر k

- k = rotl32(k, 15)

در ادامه،

- k = k * c2
- c2 = 0x1b873593

در نهایت،

- h = seed XOR k

چون $seed = 0$، پس

- h = k

جدول‌های درهم‌ساز

مرحله ۳، افزودن طول پیام و طول=۳، پس،

- h = h XOR 3

مرحله ۴ مرحله نهایی‌سازی و اعمال مراحل $fmix32$ روی h

- h ^= h >>16
- h *= 0x85ebca6b
- h ^= h >> 13
- h *= 0xc2b2ae35
- h ^= h >> 16

خروجی نهایی $MurmurHash3("abc")$

$0xb3dd93fa$

و به صورت دهدهی ۳۰۱۷۶۴۵۰۵۰

جدول‌های درهم‌ساز

جدول درهم‌ساز

- داده‌ساختاری لغت‌نامه‌ای تشکیل شده از «آرایه‌ای انجمنی» نامرتب به طول m
- هر ورودی نگاشت به مدخل یا سطل
  - دارای کلیدی در محدوده $\{0,1, \dots ,m-1\}$

استفاده از تابع درهم‌ساز برای درج مقداری در جدول درهم،

- محاسبه کلید برای انتخاب سطل مناسب برای ذخیره مقدار
- معمولا دامنه‌ای که عناصر ورودی از آن گرفته می‌شوند بسیار بزرگتر از ظرفیت m جدول درهم‌ساز
  - بنابراین اجتناب‌ناپذیری تصادم در کلیدها
  - افزایش تعداد تصادم‌ها هنگام افزایش تعداد عناصر در جدول درهم‌ساز
  - ضریب بار α مفهوم بنیادی جدول‌های درهم‌ساز
    - نسبت تعداد کلیدهای استفاده شده n به طول کل جدول m

$$\alpha = \frac{n}{m}$$

جدول‌های درهم‌ساز

ضریب بار معیاری از میزان پر بودن جدول درهم

- چون $n \leq m$، کران بالای آن یک
- وقتی α به مقدار حداکثر خود میل می‌کند، افزایش قابل توجه احتمال تصادم می‌کند
  - ایجاد نیاز به افزایش ظرفیت
- نیاز به تدبیر مشکل تصادم‌ها در پیاده‌سازی‌های جدول درهم و پیشنهاد راهکاری برای مدیریت آنها

دو تکنیک اصلی در این زمینه

- آدرس‌دهی بسته و آدرس‌دهی باز است.

آدرس‌دهی بسته

- ذخیرهٔ مقادیر تصادمی در ذیل همان کلید در داده‌ساختاری ثانویه

آدرس‌دهی باز

- ذخیرهٔ مقادیر تصادمی در موقعیت‌هایی غیر از موقعیت‌های ترجیحی خود با ارائه راهی برای آدرس‌دهی آنها

جدول‌های درهم‌ساز

تکنیک آدرس‌دهی بسته

- دم دست‌ترین راه حل تصادم‌ها و دارای پیاده‌سازی‌های مختلفی

زنجیره‌سازی جداگانه

- ذخیرهٔمقادیر تصادمی در فهرستی پیوندی
- استفاده درهم‌ساز کامل که از توابع درهم‌ساز خاص و جدول‌های درهم ثانویه با طول‌های مختلف

به جای ایجاد داده‌ساختاری ثانویه به هر شکلی

- امکان حل تصادم‌ها با ذخیره عناصر تصادمی در جای دیگر جدول اصلی و ارائه الگوریتمی برای آدرس‌دهی آنها
- مشخص نبودن آدرس مقدار معروف به آدرس‌دهی باز
  - «کاوش خطی» و «فاخته» دو پیاده‌سازی آدرس‌سازی باز

کاوش خطی

از پیاده‌سازی‌های ساده جدول درهم‌ساز با آدرس‌دهی باز

در سال ۱۳۳۳ جین آم دال، الین ام. مک‌گرا، و آرتور ساموئل

دونالد کنوث تحلیل آن در سال۱۳۴۲

قرار دادن مقدار تصادمی در مدخل خالی بعدی

نام آن: جابجائی موقعیت نهایی عضو به صورت خطی از مدخل ترجیحی

- کاوش مدخلی پس از دیگری

جدول درهم‌ساز کاوش خطی: مانند آرایهٔ دواری ذخیره کنندهٔ مقادیر اندیس شده در سطل‌ها

برای درج مقدار جدید x

- محاسبهٔ کلید آن با $k = h(x)$ با استفاده از تابع درهم‌ساز واحد h
- در صورت پر بودن سطل مطابق با کلید و حاوی مقدار متفاوتی باشد، به معنای تصادم
- ادامّ جستجو در جهت عقربه‌های ساعت در سطل‌های بعدی ادامه تا یافتن فضای خالی
  - اندیس x در آن

نظارت بر ضریب بار جدول درهم تضمین کنند» یافتن فضای خالی

کاوش خطی

جستجوی مقدار x در جدول

- محاسبهٔ کلید آن، k با استفاده از تابع درهم‌ساز h
- بررسی سطل‌ها در جهت عقربه‌های ساعت
  - آغاز از سطل ترجیحی با کلید k=h(x)
  - تا زمان یافتن مقدار موردنظر x یا پدیدار شدن اولین سطل خالی
  - نتیجه بر عدم وجود عنصر در جدول

کاوش خطی

مثال- جدول درهم کاوش خطی به طول $m = ۱۲$ و تابع درهم‌ساز مبتنی بر MurmurHash3 با اندازه ۳۲ بیتی که دامنه را به محدوده $\{0,1, \dots, m-1\}$ نگاشت می‌دهد

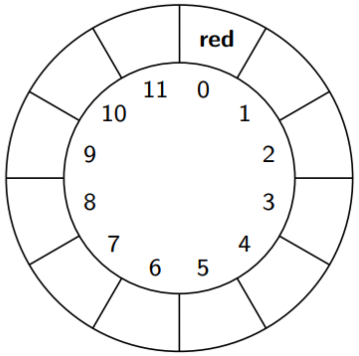
$$h(x) = MurmurHash3(x) \ \% \ m$$

حال، نام‌های مختلف رنگ‌ها را در جدول درهم‌ساز ذخیره می‌کنیم، از سرخ شروع می‌کنیم. مقدار تابع درهم‌ساز برای عنصر است

$$h = h(red) = 2352586584 \ \% \ 12 = 0$$

از آنجایی که جدول درهم‌ساز کاوش خطی در ابتدا خالی است، سطل با کلید k=0 هیچ عنصری ندارد، بنابراین ما فقط عضو را در آنجا اندیس می‌کنیم:

کاوش خطی



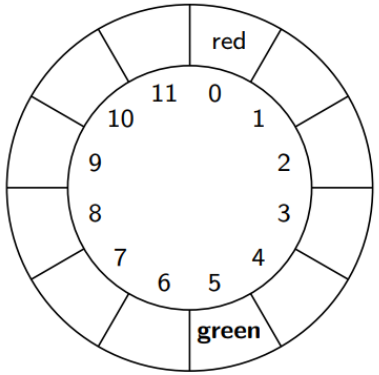
مثال- جدول درهم کاوش خطی به طول $m = ۱۲$ و تابع درهم‌ساز مبتنی بر MurmurHash3 با اندازه ۳۲ بیتی که دامنه را به محدوده $\{0,1, \dots, m-1\}$ نگاشت می‌دهد

$$h(x) = MurmurHash3(x) \ \% \ m$$

حال، نام‌های مختلف رنگ‌ها را در جدول درهم‌ساز ذخیره می‌کنیم، از سرخ شروع می‌کنیم. مقدار تابع درهم‌ساز برای عنصر است

$$h = h(red) = 2352586584 \ \% \ 12 = 0$$

از آنجایی که جدول درهم‌ساز کاوش خطی در ابتدا خالی است، سطل با کلید k=0 هیچ عنصری ندارد، بنابراین ما فقط عضو را در آنجا اندیس می‌کنیم:

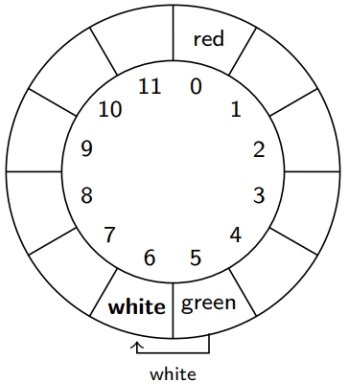


کاوش خطی

سپس، عنصر سبز را می‌گیریم که مقدار درهم‌ساز آن است

$$h = h(green) = 150831125 \,\%\, 12 = 5$$

کلید k=5 است، زیرا این سطل خالی است، می‌توان دوباره عنصر را بدون هیچ عواقبی ذخیره می‌کنیم.

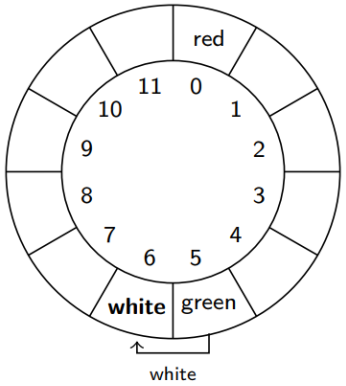


کاوش خطی

حال، عنصر سفید را در نظر می‌گیریم. مقدار درهم‌ساز آن است

$$h = h(green) = 16728905 \,\%\, 12 = 5$$

سطل ترجیحی برای این عنصر، سطلی با کلید k=5 است. با این حال، سطل قبلاً با عنصری متفاوت اشغال شده است، پس تصادم واقع شده است. در این حالت، الگوریتم کاوش خطی را اعمال می‌کنیم و سعی می‌کنیم سطل خالی بعدی را در جهت عقربه‌های ساعت از موقعیت سطل ترجیحی پیدا کنیم. خوشبختانه، سطل بعدی، با کلید k=6 آزاد است و عنصر سفید را در آنجا ذخیره می‌کنیم.



کاوش خطی

جستجوی عنصر سفید در جدول درهم کاوش خطی

- ابتدا بررسی سطل ترجیحی آن را با کلید k=5
  - پر بودن با مقداری دیگر
  - بررسی سطل‌ها در جهت عقربه‌های ساعت
  - با شروع از کلید $k + 1 = 6$
  - سطل بعدی با کلید $k = 6$ حاوی مقدار موردنظر
  - نتیجه وجود مقدار مدنظر در جدول درهم

کاوش خطی

الگوریتم برای هر عملیات نیازبه زمان O(1)

- تا زمانی که جدول درهم‌ساز کاوش خطی پر نباشد (ضریب بار به شدت کمتر از یک است). طولانی‌ترین دنباله کاوش در کاوش خطی به طول مورد انتظار O(log(n))است.
- الگوریتم کاوش خطی به انتخاب تابع درهم‌ساز hبسیار حساس است زیرا باید توزیع یکنواخت ایده‌آل را ارائه دهد.
- در عمل این امکان‌پذیر نیست و عملکرد الگوریتم با انحراف توزیع واقعی به سرعت کاهش می‌یابد.
- برای رفع این مشکل، از تکنیک‌های مختلفی برای تصادفی‌سازی اضافی به طور گسترده استفاده می‌شود.

درهم فاخته

از دیگر پیاده‌سازی‌های آدرس‌دهی باز

راسموس پاگ و فلمینگ فریشه رودلر در سال ۱۳۸۰ معرفی و در سال ۱۳۸۳ منتشر

به جای بهره از یک تابع درهم‌ساز از دو تابع درهم‌ساز

جدول درهم فاخته آرایه‌ای از سطل‌ها

به جای یک سطل ترجیحی مانند کاوش خطی و بسیاری از الگوریتم‌های دیگر

- هر مقدار ورودی دارای دو سطل نامزد است که

تعیین شدن با دو تابع درهم‌ساز متفاوت

درهم فاخته

برای اندیس کردن مقدار جدید x در جدول فاخته

محاسبهٔ کلیدهای دو سطل نامزد با توابع درهم‌ساز h_1 و h_2

در صورت خالی بودن حداقل یکی از آن سطل‌ها

- درج مقدار در آن

در غیر این صورت، به طور تصادفی یکی از آن سطل‌ها انتخاب و ذخیرهٔ مقدار x در آنجا

انتقال مقدار قبلی از آن سطل به سطل نامزد جایگزین

ادامه روش تا زمان یافتن سطل خالی پیدا شود یا رسیدن به حداکثر تعداد جابجایی‌ها برسیم

در صورت عدم وجود سطل خالی، پر فرض کردن جدول درهم

با وجود انکان اجرای دنباله‌ای از جابجایی‌های با درهم‌ساز فاخته، اما حفظ زمان ثابت O(1) برای تکمیل

درهم فاخته

سادگی روش جستجو و امکان اجرا در زمان ثابت

تعیین سطل‌های نامزد برای مقدار ورودی با محاسبه درهم‌سازهای h_1 و h_2

بررسی وجود چنین مقداری در یکی از آن سطل‌ها وجود دارد

روش حذف نیز به طور مشابه

درهم فاخته

مثال- جدول درهم‌ساز فاخته به طول $m = ۱۲$ با دو تابع درهم‌ساز ۳۲ بیتی MurmurHash3 و FNV1a که مقادیری در محدوده $\{0,1, \dots, m-1\}$ تولید می‌کنند

$$h_1(x) = MurmurHash3(x) \% \, m$$
$$h_2(x) = FNV1a(x) \% \, m$$

اندیس کردن نام رنگ‌ها را در جدول درهم

آغاز با سرخ و محاسبهٔ کلیدهای سطل نامزد را با اعمال آن توابع درهم‌ساز

$$h_1(red) = 2352586584 \% \, 12 = 0$$
$$h_2(red) = 1089765596 \% \, 12 = 8$$

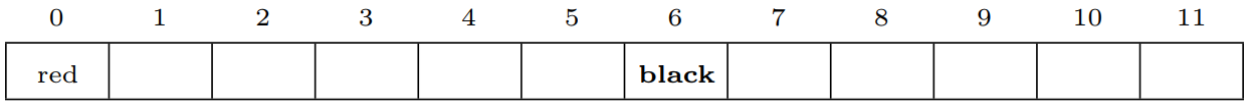
خالی بودن جدول درهم

0	1	2	3	4	5	6	7	8	9	10	11
red											

درهم فاخته

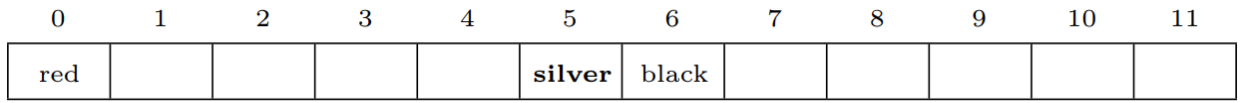
مقدار بعدی رنگ سیاه

- سطل‌های نامزد آن $h\_1(black) = 6$ و $h\_2(black) = 0$
- $k = 0$ اشغال شده مقداری دیگر
- صرفا امکان ذخیره در سطل جایگزین $k = 6$
  - آزاد است



درهم فاخته

وضعیت مشابهی برای عنصر نقره‌ای با $h_1(silver) = 5$ و $h_2(silver) = 0$ وجود دارد. این عنصر را در سطل با کلید $k = 5$ ذخیره می‌کنیم زیرا ۰ اشغال شده است.

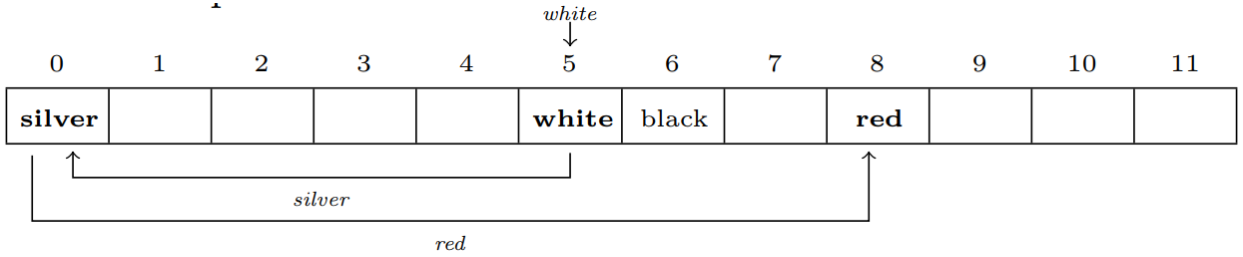


درهم فاخته

- حال عنصر سفید را در نظر می‌گیریم. مقادیر درهم‌ساز این عنصر هستند:

$$h_1(white) = 16728905 \,\%\, 12 = 5$$
$$h_2(white) = 3724674918 \,\%\, 12 = 6$$

- هر دو سطل پر
- نیاز به جابجایی طبق طرح درهم‌ساز فاخته
- ابتدا، انتخاب تصادفی یکی از سطل‌های نامزد
- فرض سطل با کلید $k = 5$ و قرار دادن عنصر سفید در آن
- عنصر نقره‌ای از سطل ۵ باید به سطل جایگزین خود، که ۰ است، منتقل شود.
  - پر بودن سطل با کلید ۰
  - ذخیرهٔ مقدار نقره‌ای را
  - انتقال عنصر قرمز از آن سطل به دیگر سطل نامزد
  - سطل جایگزین با کلید ۸ برای عنصر قرمز آزاد
  - پس از ذخیره آن در آن سطل، و پایان روش درج



درهم فاخته

جستجوی عنصر نقره‌ای

- بررسی سطل‌های نامزد آن: ۵ و ۰
- وجود در کلید ۰
  - پس وجود عنصر نقره‌ای در جدول درهم فاخته

درهم فاخته

تضمین درهم‌ساز فاخته بر اشغال فضای بالا

اما نیاز به بزرگتری بودن طول جدول درهم‌ساز از فضای مورد نیاز برای نگهداری همه عناصر

با اصلاح طرح درهم‌ساز فاخته، داده‌ساختار احتمالاتی به نام فیلتر فاخته

منابع درس

[Medjedovic22] D, Medjedovic, E. Tahirovic, “Algorithms and Data Structures for Massive Datasets,” Simon and Schuster, 2022.

[Gakhov20] A. Gakhov, Probabilistic Data Structures And Algorithms For Big Data Applications, BoD, 2020.

[Rocca21] M. La Rocca, Advanced Algorithms and Data Structures, Simon and Schuster, 2021.

[Neapolitan]

[CLRS]

[Anand2011] J. Leskovec, A. Rajaraman, J.D. Ullman, Mining of Massive Datasets. Cambridge university press, 2020.