جزوه جلسه هفدهم داده ساختارها و الگوریتم

۲ آذر ۱۴۰۰

فهرست مطالب

۱ درهم سازی کامل یا Perfect Hashing

این متد درهم سازی اولین بار در سال ۱۹۸۴ در ژورنال ACM معرفی شد.
در این روش درهم سازی، یک جدول $\mathrm{m = n}$ برای ذخیره سازی اشیا نیاز است. همچنین در نسخه اصلی این روش، همه اشیا از ابتدا در دسترس هستند و فقط میتوانیم اشیا را جست و جو کنیم (insert، delete و update نداریم) و این دیگر عملیات در نسخه های دیگر وجود دارند. حال به بررسی عملیات مورد انتظار در این روش می پردازیم:

۱.۱ delete و search

در روش آدرس دهی باز، در هر مرحله، یک بار تابع Hash محاسبه می شد؛ اما در این روش برای هر جست و جو (و در ادامه آن حذف) نیازمند ۲ بار محاسبه Hash می باشد.

۲.۱ insert

در مرحله اول با تابع درهم سازی اولیه h، کلید را به یک خانه از جدول می نگاریم (Mapping). در حالت نامطلوب، از قبل تعدادی شئ دیگر نیز در این خانه وجود دارد. فرض کنیم تعداد کلیدهای موجود در یک خانه برابر با k باشد. حال یک جدول دیکر به اندازه k^2 در نظر میگیریم و با تابع درهم سازی ثانویه h'، k شی مدنطر را در این جدول قرار می دهیم.
به احتمال حداقل $1/2$ برخورد صورت نمی گیرد. اگر در این مرحله نیز برخورد رخ داد، تابع ثانویه یا h' را عوض می کنیم. همچنین در نظر داریم که h' برای خانه های مختلف جدول متفاوت است.
ثابت می شود حافظه مورد نیاز، به صورت میانگین برابر $O(n)$ می باشد.(اثبات با در نظر گرفتن یک بودن ضریب بارگذاری (α) و انجام محاسبات روی توان دوم آن انجام می گیرد.)

۱.۲.۱ پارادوکس روز تولد

برای اثبات احتمال $1/2$ ذکر شده، به ذکر پارادوکس روز تولد می پردازیم.
پارادوکس روز تولد می گوید احتمال اینکه روز تولد دو نفر در یک جمع ۲۰ نفره یکسان باشد، حدودا $1/2$ است. ($20 \approx \sqrt{365}$)
به همین دلیل می توان ادعا کرد احتمال برخورد حداکثر برابر $1/2$ می باشد.

$$E[\#conflicts] = \binom{k}{2} \times \frac{1}{k^2} = \frac{k^2-k}{2k^2} = \approx 1/2 \ (< 1/2)$$

در عبارت بالا، $\binom{k}{2}$ برابر تعداد حالات انتخاب ۲ کلید و $\frac{1}{k^2}$ احتمال برخورد ۲ عضو انتخاب شده می باشد.
حال می توان نتیجه گرفت که برای میانگین تعداد برخورد ها در حالت های مختلف کمتر از $1/2$ در فضای صفر و یکی (برخورد میکند یا نمیکند) داریم:

$$Pr[conflicts] < 1/2$$

پس احتماب برخوردها نیز کمتر از $1/2$ است و در نتیجه برای نصف خروجی های h' برخورد نداریم.

۲ درهم سازی کوکو یا Cuckoo Hashing

روشی دیگر برای رفع مشکل برخورد، استفاده از درهم سازی کوکو است. ایده حل مشکل، استفاده از ۲ جدول (T_1 و T_2) و ۲ تابع مجزا و مستقل از هم h_1 و h_2 برای هر کدام از جدول هاست. اندازه جدول نیز در این حالت برابر $m = 2n$ می باشد و هر شی در یکی از جدول ها درج میشود.

۱.۲ search و delete

برای جست و جوی یک کلید، یک یا دو بار نیاز به فراخوانی تابع درهم سازی داریم؛ بدین صورت که ابتدا $h_1(key)$ را محاسبه میکنیم؛ اگر $T_1[h_1(key)]$ حاوی شی مدنظر بود آنرا برمی گردانیم. در غیر این صورت به سراغ $T_2[h_2(key)]$ می رویم. اگر شی مورد نظر پیدا شد آن را برمی گردایم و در غیر این صورت ادعا میکنیم شی مدنظر در جدول وجود ندارد.
برای حذف نیز مشابه مراحل بالا، اگر شی در هرکدام از مراحل پیدا شد آن را حذف میکنیم.

۲.۲ insert

برای درج هر شئ، ۲ کاندید داریم؛ $T_1[h_1(key)]$ و $T_2[h_2(key)]$. اگر خانه مربوط به T_1 خالی بود، درج در آن خانه انجام میگیرد و در غیر اینصورت به سراغ T_2 می رویم. اگر این خانه خالی بود درج صورت می گیرد و در صورتی که هردو خانه پر باشند، عنصر موجود در خانه $T_2[h_2(key)]$ را بیرون انداخته و شئ را در انجا درج می کنیم. حال عنصری را که بیرون انداخته بودیم مطابق الگوریتم ذکر شده در یکی از دو جدول درج می کنیم.
حالتی نامطلوب است که این روند بیرون انداختن عناصر و درج مجدد طولانی شود. اگر تعداد مراحل از $2logn$ بیشتر شد، جدول ها را از ابتدا با ۲ تابع درهم سازی جدید

میسازیم.
به دلیل اینکه رخ دادن چنین حالتی بسیار کم است، پس میتوان گفت هزینه درج به صورت سرشکن برابر $O(1)$ می باشد.

۳ توابع درهم سازی در رمزنگاری

توابع درهم سازی که در رمز نگاری استفاده می شوند پیچیده تر از توابع درهم سازی که بررسی کردیم (یا حداقل حاصل چندین مرتبه فراخوانی توابع معمول درهم سازی) هستند. همچنین خروجی ها نیز به دلیل افزایش امنیت دارای تعداد بیت زیادی هستند. (حداقل ۱۶۰ بیت)
حال به بررسی ویژگی های توابع مناسب رمزنگاری می پردازیم.

۱.۳ یک طرفه بودن (OW)

این ویژگی به زبان ساده یعنی برای هر y نتوان کلیدی مانند x پیدا کرد که $\mathrm{h(x)=y}$ طولانی بودن اندازه خروجی توابع درهم سازی به تقویت این ویژگی کمک می کنند؛ بدین صورت که با افزایش طول خروجی تولید حالت های مختلف برای هر بررسی $\mathrm{h(x)=y}$ وقت گیرتر می شود.
در سیستم عامل لینوکس، hash رمز تمامی اکانت کاربران در $\mathrm{/etc/shadow}$ نگه داری می شود. اگر تابع درهم سازی مورد استفاده این ویژگی را دارا نباشد، میتوان یک ورودی را به عنوان رمز یک اکانت وارد کرد که hash آن با hash رمز اصلی یکسان باشد و وارد اکانت شد. در این حالت لزومی بر برابر بودن رمز اصلی و رمز وارد شده وجود ندارد.
اگر کاربر نتواند وارد حساب خود شود، یعنی رمز وارد شده آن به صورت قطعی نادرست است ولی درصورت وارد شدن به حساب، نمیتوان ادعا کرد رمز وارد شده همان رمز اصلی کاربر است.

۲.۳ مقاوم در برابر برخورد (CR)

در این ویژگی انتظار داریم برای تابع درهم سازی نتوان x_1 و x_2 متفاوتی پیدا کرد که خروجی hash آنها یکسان باشد ($h(x_1) = h(x_2)$)
سناریویی را در نظر بگیرید که یک متن را امضا میکنیم. امضا کردن متن زمان بر است به همین دلیل Hash آنرا امضا میکنیم. (منظور از امضا کردن یک متد خاص برای رمز کردن یک فایل است که فرد امضا کننده فقط میتواند فایل را رمز نگاری کند و همه می توانند آن را رمزگشایی کنند). حال اگر این ویژگی برقرار نباشد، بعد از امضا کردن متن x_1 می توان ادعا کرد متن x_2 امضا شده است؛ زیرا که خروجی تابع Hash آن ها یکسان است.

۳.۳ مقاومت در برابر برخورد هدف دار (TCR)

این ویژگی نیز به معنی این است که برای کلید x_1 داده شده، نتوان x_2 پیدا کرد که $h(x_1) = h(x_2)$ و خروجی Hash آنها یکسان باشد ($x_1 \neq h_2$)
سناریوی امضای متن را مجددا در نظر بگیرید. در این حالت برای متنی که امضا کردیم نباید متن دیگری تولید کرد که Hash آنها یکسان باشد. (توجه شود در قسمت قبل هر دو متن x_1 و x_2 از ابتدا در دسترس بودند.)
برای مثال دیگر، سرویس DropBox را در نظر بگیرید. به دلیل حجیم بودن فایل ها به نسب Hash آن ها، برای تضخیص تغییرات فایل ها و سینک کردن آن، از Hash فایل در سرور استفاده می شود. اگر این ویژگی برقرار نباشد، می توان فایل را تغییر داد به گونه ای که این تغییر مخفی بماند.
بدین منظور این سرویس به هنگام دریافت یک فایل، Hash آن را نیز در اختیار کاربر قرار میدهد تا از تغییرات فایل در مسیر مطلع شود. احتمال برابر بودن Hashها و یکسان نبودن فایل ها در صورت تغییر، بسیار کم است.

۴.۳ امنیت توابع درهم سازی

توابه بسیاری برای رمزنگاری پیشنهاد شده اند. اما هنگامی که مشخص شود تابع معرفی شده حداقل یکی از ویژگی های فوق را ندارد ناامن طلقی میشود. برای مثال تابع MD-5 که نسل پنجم توابع MD می باشد و برای درهم سازی رمز های کاربران در لینوکس نیز استفاده میشد ثابت شده که ویژگی CR را ندارد.
تابع SHA-1 نیز بطور مشابه به دلیل نقض ویژگی CR کنار گذاشته شد.
تابع SHA-2 که در سال ۲۰۱۵ معرفی شده، جدیدترین تایع پیشنهاد شده و مورد استفاده در این زمان است.