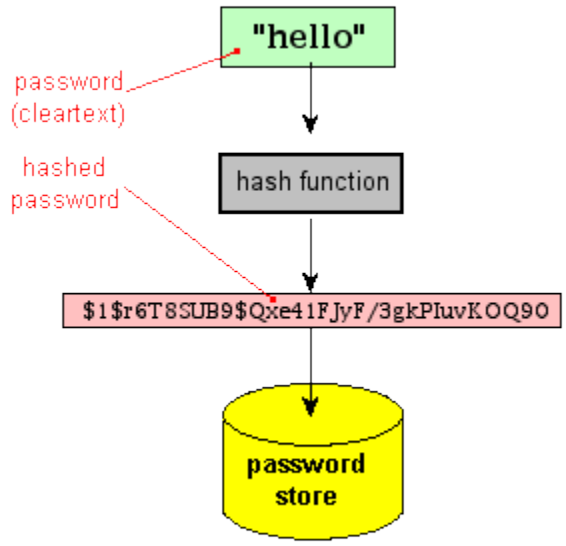
هش چیست؟

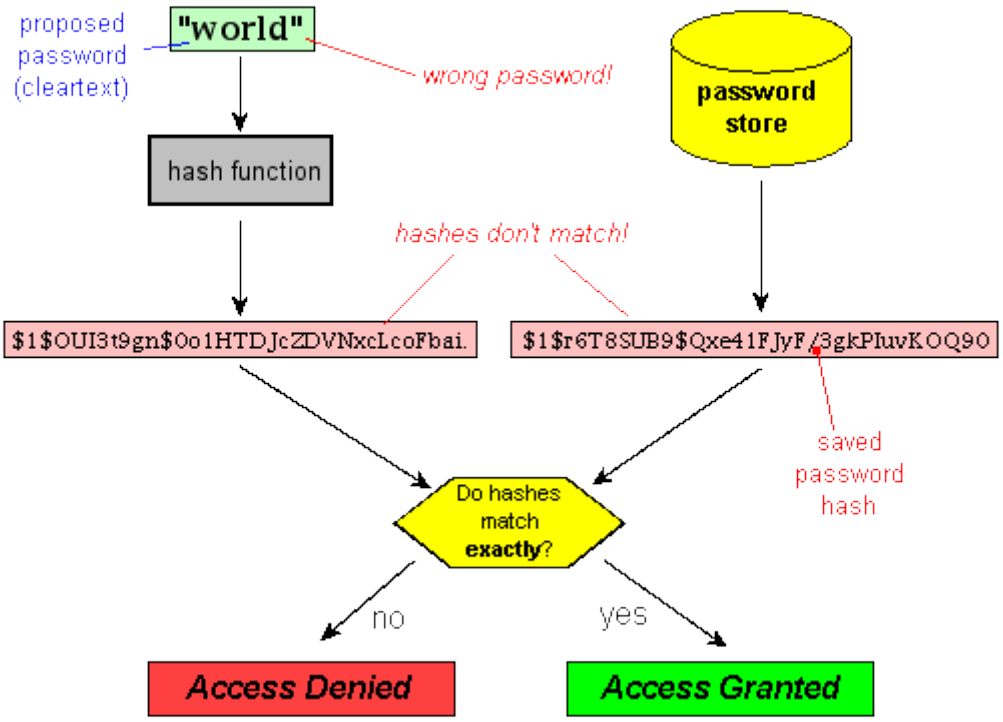
هش (Hash, Hash Code, Digest, Message Digest هم نامیده می شود) را می توان به صورت اثر انگشت دیجیتالی یک داده در نظر گرفت. با این روش شما می توانید رشته ای اندازه-ثابت (fixed length) از یک داده به دست آورید که با روش های ریاضی به صورت "یک طرفه" رمزنگاری شده است. کشف رشته اصلی از رشته هش آن (عملیات معکوس) به صورت کارا تقریبا غیر ممکن است. نکته دیگر اینکه هر داده یک رشته هش شده کاملا منحصر به فرد ایجاد می کند(احتمال یکی شدن رشته های هش دو رشته متفاوت در الگوریتم MD5 یک در e+383.40282366920938463463374607431768 می باشد.. این خواص ، هش کردن را به روشی کارا و ایده آل برای ذخیره سازی کلمات عبور در برنامه های شما تبدیل می کند. چرا؟ برای این که حتی اگر یک نفوذگر(Hacker) بتواند به سیستم و بانک اطلاعاتی شما نفوذ کند و بخشی از اطلاعات شما را به دست آورد (شامل کلمات عبور هش شده) نمی تواند کلمات عبور اولیه را از روی آن ها بازیابی کند.



یکی از دو خصوصیت الگوریتم های HASH اینه که **معکوس پذیر نیستند!** دومی اینه **که هرگز دو ورودی متفاوت به خروجی یکسان منجر نمی شوند**. هر یک از این دو خصوصیت اگر نقض بشه می گیم الگوریتم شکسته!!!

شناسایی اعضا با استفاده از Hash :

تا کنون نشان داده ایم که بازیابی کلمه عبور اصلی از روی رشته هش تقریبا غیر ممکن است ، خب چگونه برنامه های ما تشخیص دهند که کلمه عبور وارد شده توسط کاربر صحیح است ؟ به سادگی ! با تولید رشته هش کلمه عبور وارد شده توسط کاربر و مقایسه آن با رشته هش ذخیره شده در رکورد بانک اطلاعاتی مربوط به کاربر می توانید متوجه شوید که آیا دو رشته با هم برابرند یا نه. بگذارید با ذکر یک مثال این بحث را ادامه دهیم.



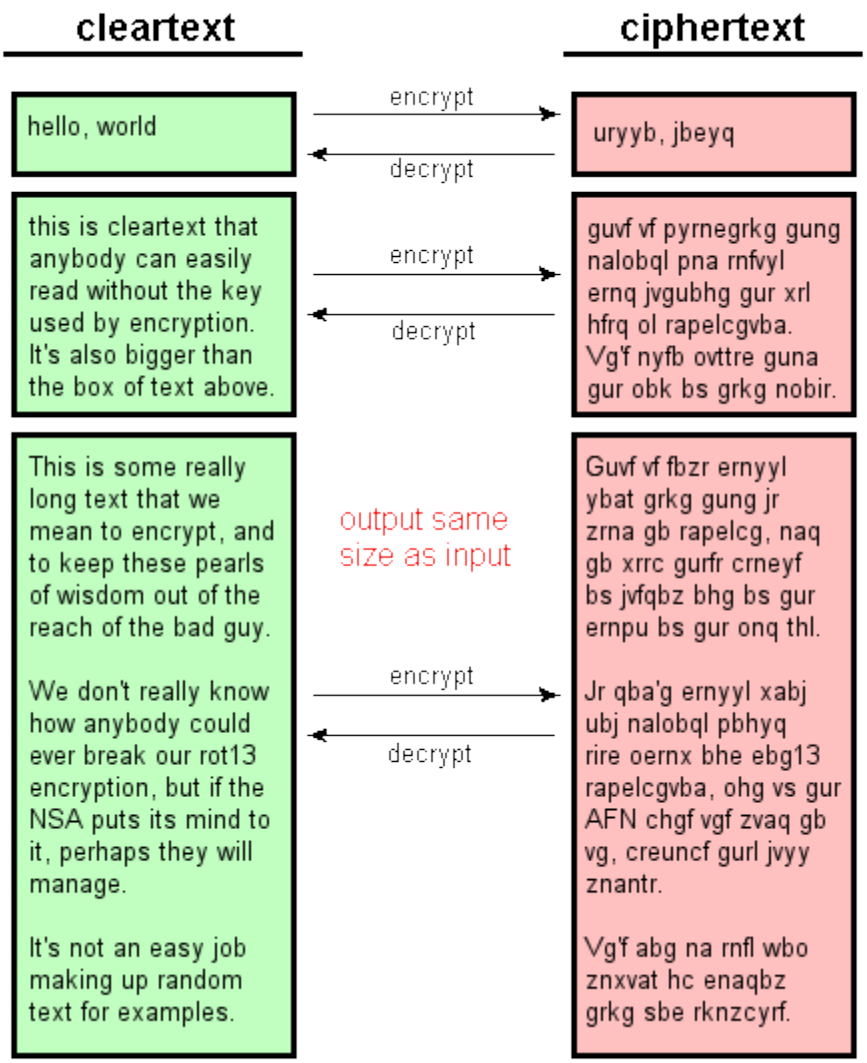
Hashes are "digests", not "encryption"

Hash یک عمل خلاصه سازی (digest) را روی جریان ورودی انجام می دهد نه یک عمل رمز نگاری (encryption) .

Encryption داده را از یک متن صریح (Clear text) به یک متن برمز در آورده تبدیل می کند (Cipher text). encryption ک عمل دو طرفه می باشد . که هرچه حجم Clear text بیشتر باشد حجم Cipher text نیز بیشتر می شود.

که این ارتباط در شکل صفحه بعد به خوبی بیان شده است:

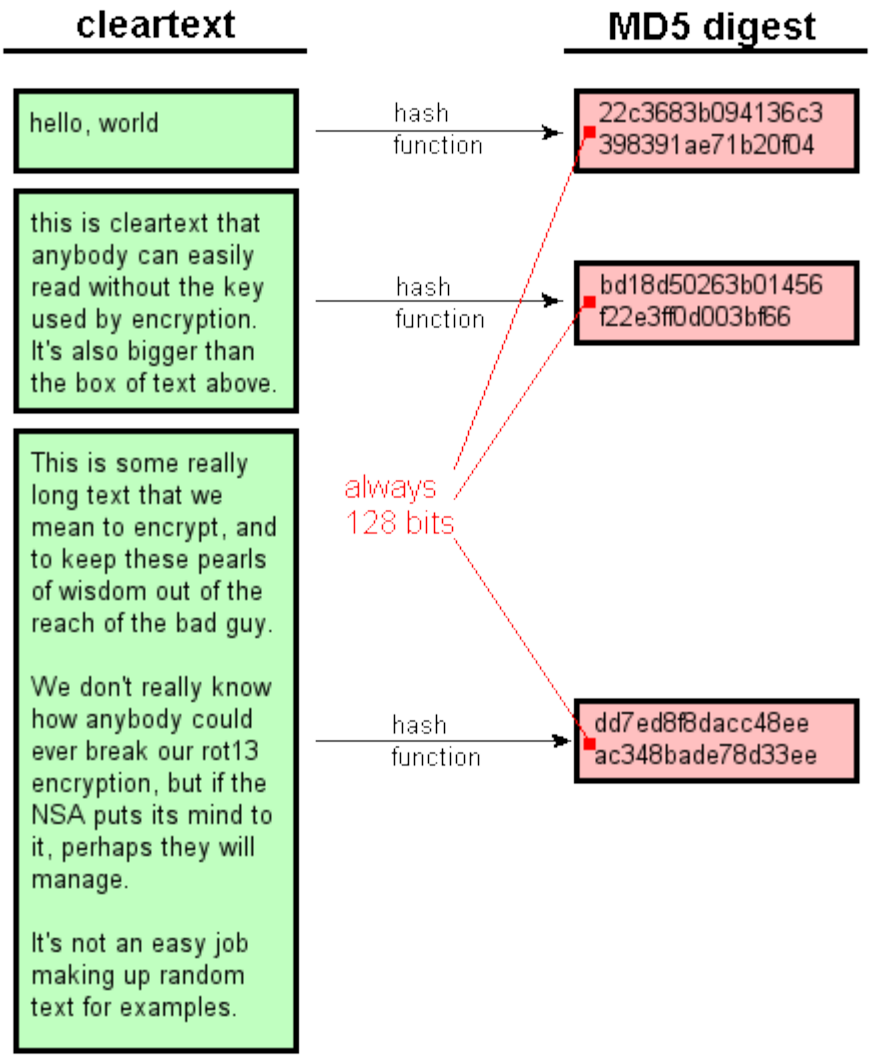
Encryption - a two-way operation



Hashe ها جریان داده ورودی را به یک خلاصه کوچک تبدیل می کنند. که این یک عمل یک طرفه(غیر قابل بازگشت) می باشد. و جریان داده ورودی آنها با هر حجمی که باشد خروجی یک مقدار ثابت میشود.

شکل بعدی این ارتباط را در خلاصه سازی توسط الگوریتم MD5 به خوبی نشان می دهد.

Hashing - a one-way operation



موارد استفاده از Hash ها :

Hash ها کلا موارد استفاده کمی دارند که ما در ادامه بحث آن ها را بیان می کنیم:

1. تشخیص درستی یک فایل Verifying file integrity

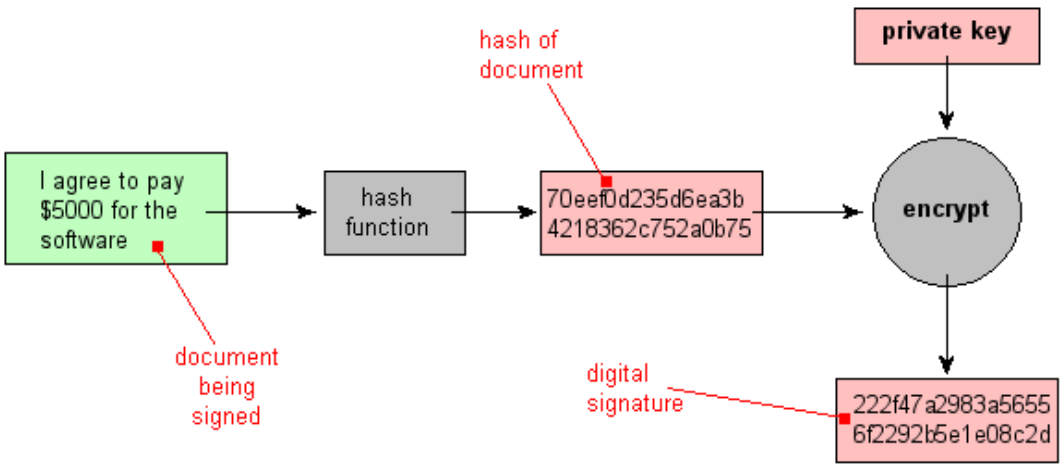
برای مثال زمانی که یک فایل با حجم بالا را دانلود می نماییم می توانیم با به دست آوردن مقدار MD5 آن فایل توسط دستور md5sum و مقایسه آن با مقدار Md5 داده شده توسط سایت مورد نظر از درستی فایلمان اطمینان حاصل کنیم.

2. hash کردن کلمه عبور Hashing passwords

در صفحات قبلی به طور کامل توضیح داده شده است .

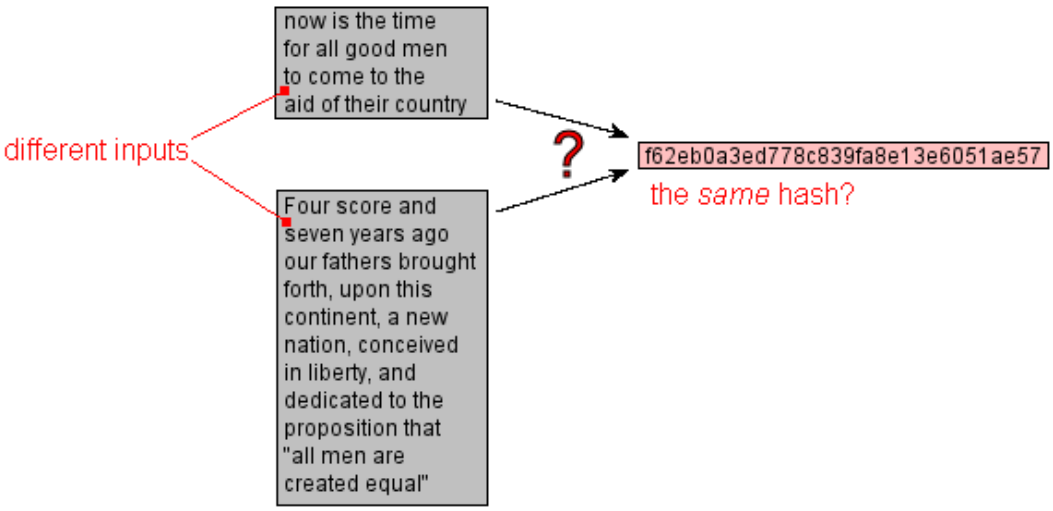
3. نشانه گذاری اسناد به روش digitally (امضاهای digitally)

که نحوه انجام ان به وضوح در شکل زیر نشان داده شده است:



Collision (تصادم) در Hash :

زمانی که مقدار Hash دو ورودی متفاوت یکسان باشند می گوییم Collision رخ داده است.



اما تا کنون هیچ موردی از Collision دیده نشده است . این امر از این حقیقت ناشی می شود که تعداد مقادیر یک الگوریتم hash بسیار زیاد می باشند.برای مثال یک Hash 128 بیتی میتواند 3.4×10^{38}

مقدار ممکن داشته باشد.که معادل 340,282,366,920,938,463,463,374,607,431,768,211,456 میباشد.

انواع هش :

- (128 bits, obsolete) MD4
- (128 bits) MD5
- (160 bits) RIPEMD-160
- (160 bits) SHA-1
- SHA-256, SHA-384, and SHA-512 (longer versions of SHA-1, with slightly different designs)

انواع مختلفی از الگوریتم های قوی هش کردن برای استفاده در برنامه های کاربردی موجود هستند، محبوب ترین آنها که مورد استفاده برنامه نویسان هستند MD5 و SHA-1(Secure hash algorithm)می باشند. سیستم های قدیمی تر از(DES(Data Encryption Standard استفاده می کردند. این روش 56 بیتی دیگر یک روش قوی هش کردن محسوب نمی گردد. ، الگوریتم های قوی تری مانند SHA-256 و SHA-512 برای موارد خاص مانند امضاهای دیجیتالی توصیه می گردد ولی برای هش کردن کلمات عبوردر برنامه های امروزی SHA-1 هنوز سطح امنیت بسیار خوبی را فراهم می کند.

جدول هش :

همانطور که در جستجوی دودویی دیده شد با استفاده از یک ساختمان داده به خصوص به اسم درخت جستجوی دودویی کارایی جستجو را بهبود بخشید یم.

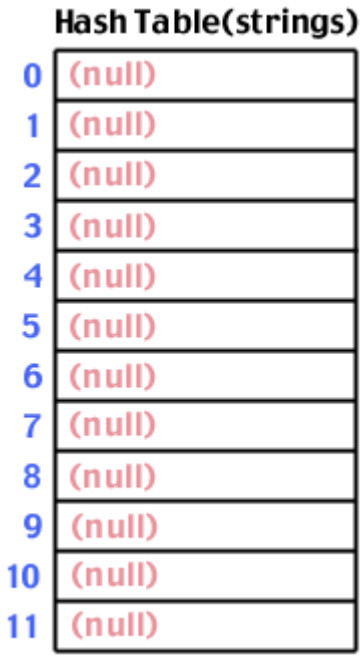
رسیدیم.O(logn) به جستجوی دودویی باO(n) در واقع از جستجوی خطی با

حال می خواهیم یک ساختمان داده جدید به نام **جدول هش** را معرفی کنیم که کارایی عمل جستجو را تا افزایش می دهد.O(1)

یک جدول هش از دو قسمت تشکیل شده : یک آرایه (جدول واقعی جایی که دادهایی که جستجو می شود) که به آن تابع هش می گویند.mapping function در آن ذخیره می شود (و یک تابع نگاشت)

اندیس های آرایه که (integer space) تابع هش نگاشتی است از فضای ورودی به فضای اعداد صحیح را مشخص می کند . به عبارت دیگر تابع هش روشی را فراهم می کند برای اختصاص دادن اعداد به داده های ورودی به گونه ای که سپس داده ورودی می تواند در اندیس آرایه مطابق باعدد تخصیص داده ذخیره شود.در ادامه مثالی در این رابطه بیان می کنیم:

مثال را آغاز می کنیم. یعنی از رشته ها به عنوان داده هایی ابتدا با یک آرایه جدول هش از رشته ها که ذخیره و جستجو می شوند استفاده می کنیم . اندازه جدول هش را در این مثال 12 می گیریم.



The empty hash table of strings

در مرحله بعد ما به یک تابع هش نیاز دارم. راههای زیادی برای ساختن یک تابع هش وجود دارد. در حال حاضر یک تابع هش ساده را در نظر می گیریم.

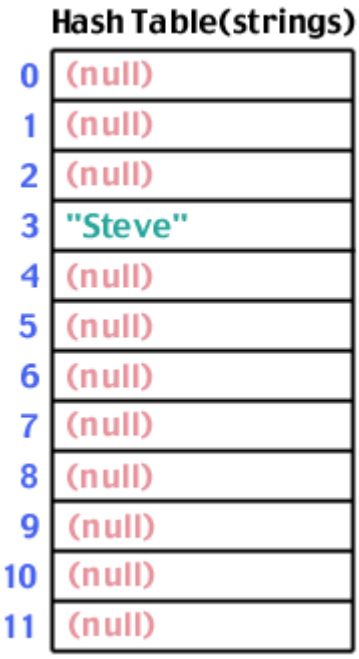
که یک رشته را به عنوان ورودی میگیرد. مقدار هش برگشتی از تابع برابر مجموع کاراکتر های اسکی است که رشته ورودی را می سازند به پیمانه اندازه

جدول:

```
int hash(char *str, int table_size)
{
    int sum;

    /* Make sure a valid string passed in */
    if (str==NULL) return -1;

    /* Sum up all the characters in the string */
    for( ; *str; str++) sum += *str;

    /* Return the sum mod the table size */
    return sum % table_size;
}
```

اجرا می کنیم که مقدار 3 حاصل می شود.("Steve",12) تابع هش را باپارامتر های

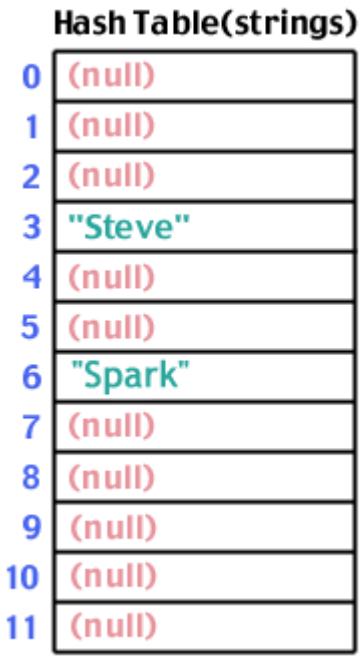
. را در جدول ذخیره می کنیم Steve حال رشته



The hash table after inserting "Steve"

اجرا میکنیم("Spark",12) تابع هش را با پارامتر های "Spark". بیایید رشته دیگری را امتحان کنیم:

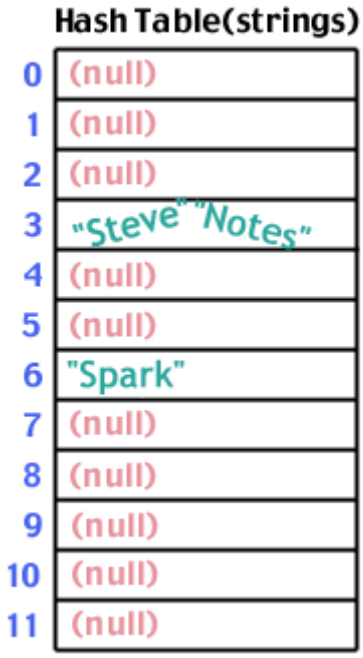
که عدد 6 حاصل می شود. سپس آن را نیز در جدول ذخیره میکنیم.



The hash table after inserting "Spark"

اجرا میکنیم . که این بار نیز مقدار 3 حاصل می شود. ("Notes",12) حال تابع هش را با پارامتر های

را نیز در جدول درج می کنیم.Notes رشته



A hash table collision

مشاهده می شود که دو ورودی متفاوت مقدارهش یکسانی دارند و هر دو عنصر باید در یک مکان در آرایه درج شوند که این مساله غیر ممکن است . و همانطور که قبلا نیز ذکر شد به این مساله تصادم گفته (linear probing) می شود. در رابطه با تصادم الگوریتم های زیادی وجود دارد از قبیل اکتشاف خطی و زنجیرسازی جداگانه.

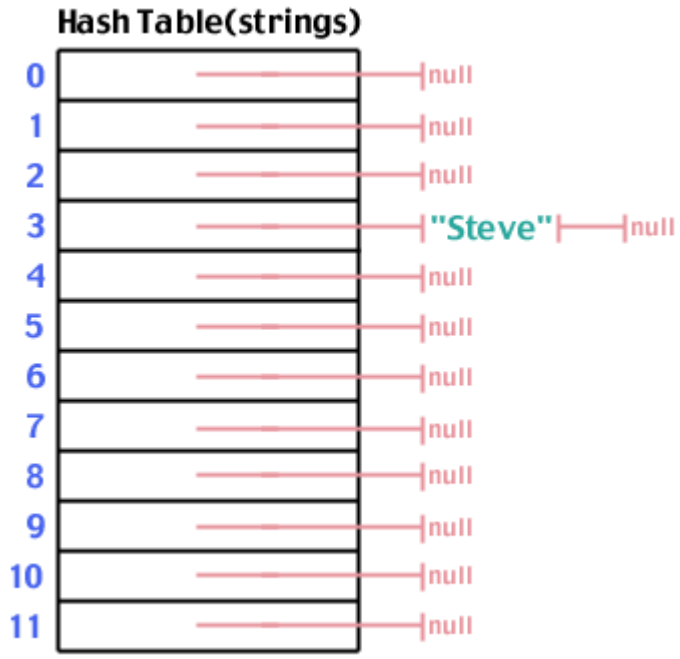
بررسی می کنیم. ر ا (Separate Chaining) که ما در اینجا زنجیره سازی جداگانه

زنجیره سازی جداگانه به مقدار کمی تغییر در ساختمان دادهمان نیاز دارد. به جای ذخیره سازی مستقیم داده ها در آرایه آنها را در لیست های پیوندی ذخیره می کنیم. هر عنصر آرایه به یکی از این لیست های پیوندی اشاره می کند. زمانی که مقدار هش یک ورودی به دست آمد آن عنصر به لیست پیوندی که در اندیس مورد نظر در آرایه وجود دارد اضافه می شود . از ان جایی که لیست های پیوندی محدودیت در طولشان ندارند مساله تصادم ها حل شده به حساب می آید. اگر دو داده متفاوت مقدار هش یکسانی داشته باشند هر دوی آنها در لیست پیوندی ذخیره می شوند.

بیایید مثال بالا را این بار با ساختمان داده تغییر یافته مان بررسی نماییم :

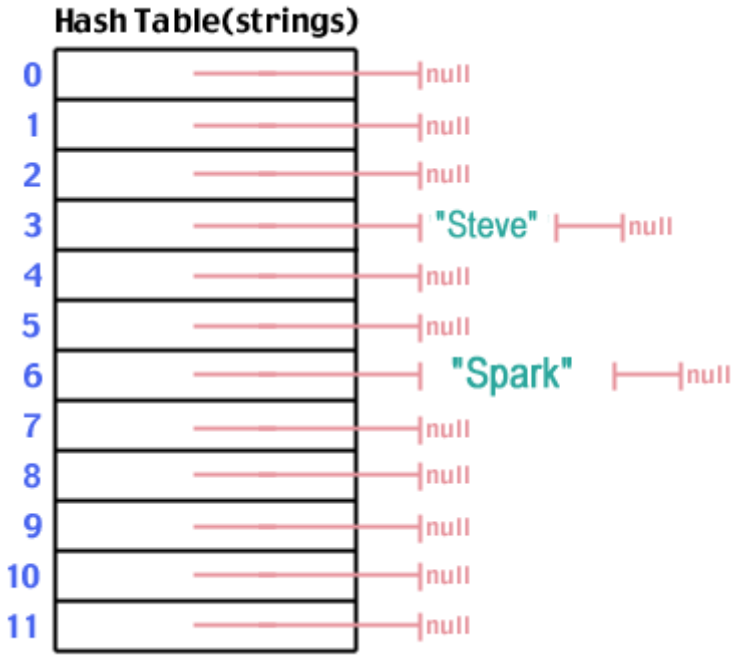
Modified table for separate chaining

دوباره Steve را با مقدار هش 3 به جدولمان اضافه میکنیم:



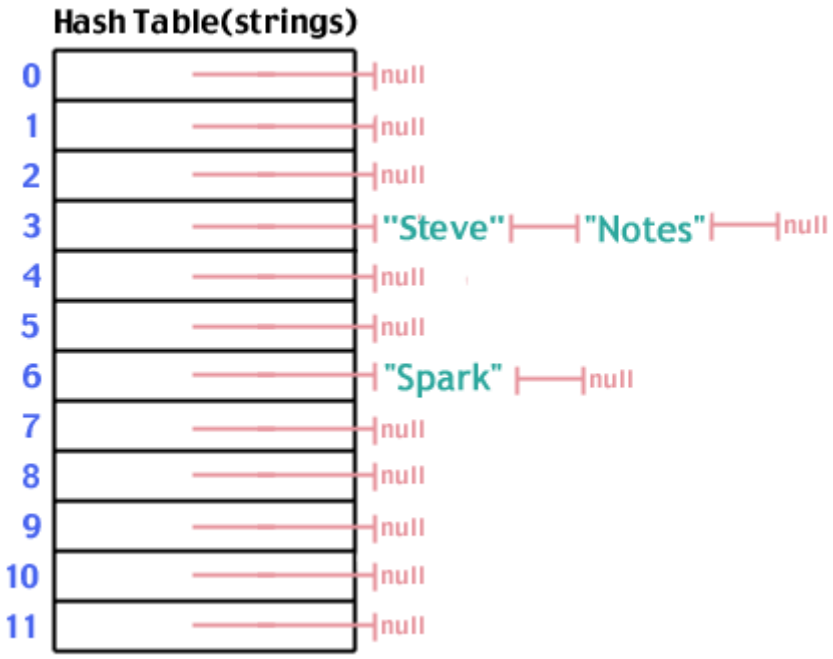
After adding "Steve" to the table

هم چنین Spark با مقدار هش 6 را نیز به جدولمان اضافه می کنیم:



After adding "Spark" to the table

حال Notes با مقدار هش 3 (همانند مقدار هش Steve) را اضافه می کنیم:



Collision solved - "Notes" added to table

مراحل جستجو مشابه با عمل درج در جئول می باشد. ما مقدار هش داده ای را که می خواهیم جستجو شود را به دست می آوریم. سپس به ان مکان از آرایه می رویم. لیستی که از آن مکان سرچشمه می گیرد (آغاز میشود) را بررسی می کنیم. و می بینیم که چیزی که به دنبال آن هستیم در لیست وجود دارد یا نه .

=>تعداد مراحل $O(1)$ می باشد.

زنجیره سازی جداگانه (Separate chaining) به ما این امکان را می دهد که مساله تصادم را در یک روش ساده و قدرتمند حل نماییم . با این حال هنوز مشکلاتی وجود دارد . حالتی را فرض کنید که تمام داده های ورودی دارای یک مقدار هش باشند. در این مورد برای جستجوی یک داده باید یک جستجوی خطی با $O(n)$ در لیست پیوندی انجام دهیم . پس در بدترین حالت این روش $O(n)$ را خواهیم داشت که احتمال آن بسیار کم است . در حال که در بهترین حالت و حالت متوسط $O(1)$ را خواهیم داشت!

جداول هش کامل:

اگر تمام کلیدها در یک زمان پیش بینی شده معلوم شوند، جدول هش کامل perfect hash functionمیتواند مورد استفاده قرار گیرد تا یک جدول هش کامل ساخته شودکه در آن هیچ تصادمی رخ ندهد. اگرجدول کامل کمینه minimal perfect hashingاستفاده شود، تمام مکانهای جدول هش به خوبی استفاده خواهند شد.

زمان جستجو در هش کامل در بدترین حالت بر خلاف متدهای زنجیرواری و آدرس دهی باز مقدار ثابتی است. اما با وجود اینکه زمان متوسط جستجودر اینجا کو تاه است، ممکن است در مواردی(به تناسب اعداد ورودی) برای مجموعه‌ای از اعداد بسیار بزرگ شود.

تجزیه و تحلیل تصادم:

وقتی زیرمجموعه‌ای تصادفی از یک مجموعهٔ بزرگ اعداد را درهم سازی میکنیم، جلوگیری از رخ دادن تصادم عملاً غیر ممکن خواهد بود.برای مثال اگر بخواهیم 2500 کلید را در یک میلیون خانه هش کنیم،95٪ احتمال وجود دارد که حداقل دو کلید در یک مکان هش شوند.

بنابر این، بسیاری از پیاده سازی‌های توابع هش، استراتژی خاصی برای تحلیل مساله تصادم در نظر می‌گیرند تا این قبیل پیشامدها را کنترل کنند. تعدادی ازاستراتژی‌های رایج در زیر توصیف خواهند شد.همهٔ این متدها احتیاج دارند که تمام کلیدها یا ارجاعات آنها به همراه مقدار هش آنها با هم در جدول ذخیره شوند.

عامل بار :

کارایی بیشتر متدهای تحلیل تصادم مستقیما به تعداد اعداد ورودی بستگی ندارد، اما به طور قوی به عامل بار جدول مربوط است.عامل بار یعنی نسبت تعداد اعداد ورودی(تعداد کلیدها) به سایز آرایه. با یک تابع هش خوب، متوسط هزینه جستجو تقریبا ثابت است بهمان اندازه که عامل بار از 0 تا 0.7 یا بیشتر افزایش می‌یابد.علاوه بر این، احتمال رخداد تصادم و هزینهٔ کنترل آن افزایش می‌یابد.

به عبارت دیگر، هر چقدر عامل بار به صفر نزدیک شود، سایز جدول هش با کمی بهبود در هزینهٔ جستجو افزایش می‌یابد و حافظه به هدر می‌رود.

زنجیر سازی جداگانه:

در استراتژی ای به نام زنجیرکردن جداگانه، زنجیر کردن مستقیم، یا زنجیر کردن ساده، هر خانه از آرایه یک اشاره گر است به یک لیست پیوندی که شامل کلید به همراه مقدار هش آن با هم در یک خانه است. جستجو در این روش نیازمند پیمایش تمام لیست برای یافتن کلید داده شده است.برای اضافه کردن یک رکورد باید آن را به ته لیست اضافه کنیم و برای حذف کردن آن باید لیست را جستجو کرده و عنصر را حذف کنیم.(این تکنیک درهم سازی باز یا آدرس دهی بسته نیز نامیده می‌شود که نباید با درهم سازی بسته یا همان آدرس دهی باز اشتباه گرفته شود)

جدول درهم سازی زنجیره شده با لینک لیست بسیار معروف است، زیرا به یک ساختمان داده‌ٔ پایه‌ای با الگوریتمی ساده نیازمند است و می‌تواند از تابع هش ساده‌ای استفاده کند که ممکن است برای متدهای دیگر مفید واقع نشود.

اگر توزیع کلیدها به حد کافی یکنواخت باشد، هزینه‌ٔ جستجو تنها به میانگین تعداد کلیدها در هر لیست بستگی دارد-یا به عبارتی به عامل بار.حتی اگر تعداد کلیدها خیلی بیشتر از خانه‌های آرایه باشد جدول درهم سازی زنجیره‌ای باز هم کارامد است.کارایی این جدول‌ها به طور خطی با عامل بار متناسب است.

برای مثال، یک جدول زنجیره‌ای با 1000 خانه و 10.000 کلید ذخیره شده(عامل بار=10) 5 تا 10 بار کندتر از جدولی با 10.000 خانه (عامل بار=1) خواهد بود؛ اما هنوز 1000 بار سریع تر از یک لیست متوالی ساده است و ممکن است که حتی از یک درخت متوازن جستجو هم سریع تر باشد.

برای روش زنجیره‌ای سازی مجزا، بدترین حالت وقتی اتفاق می افتد که همه‌ٔ کلیدها وارد یک خانه شوند که در این حالت جدول بسیار نا کارامد بوده و هزینه‌ٔ جستجو در آن بالاست. اگر جدول یک لیست خطی باشدبرای رویه‌ٔ جستجو ممکن است مجبور باشیم که تمام کلیدها را پیمایش کنیم؛ بنابراین در بدترین حالت هزینه‌ٔ جستجو با تعداد کلیدهای وارد شده در جدول متناسب خواهد بود.

آرایه‌ٔ زنجیره‌ای مثل یک لیست مرتب که بر اساس کلیدها سورت شده پیاده سازی می‌شود.که هزینه‌ٔ جستجوی موفق در این روش نسبت به یک لیست نا مرتب تقریبا نصف است.با این حال، اگر انتظار برود که احتمال آمدن یک سری کلید نسبت به بقیه بیشتر باشد، ممکن ایت که لیست نا مرتب که در آن عنصر یافت شده به اول لیست منتقل می‌شود کارامد تر باشد.

داده ساختارهای پیچیده تر، از جمله درخت متوازن جستجو قابل اهمیت تر خواهند بود اگر:عامل بار بزرگ باشد(در حدود 10 یا بیشتر)، احتمال این باشد که توزیع درهم سازی نا متوازن شود،...با این حال، استفاده از یک جدول با سایز بزرگ و/یا یک تابع هش خوب ممکن است در آن موارد کارامد تر باشد.علاوه بر این، جدول هش زنجیره‌ای معایب لینک لیست را به همراه دارد.

زنجیرسازی جداگانه با سر لیست‌ها:

برخی از متدهای زنجیرکردن بر اساس ذخیره کردن ابتدای هر لیست یک خانه پیاده سازی شده اند.هدف از این کار بالا بردن سرعت دسترسی به جدول است.در نتیجه در حافظه صرفه جویی می‌شود چرا که این قبیل جدول‌ها طوری طراحی شده اند که به تعداد کلیدها خانه حافظه داشته باشند در حالیکه بسیاری از خانه‌ها دارای دو یا تعداد بیشتری کلید خواهند بود.

زنجیرسازی جداگانه با دیگر ساختارها :

به جای یک لیست، هر داده ساختار دیگری که اعمال موجود را پشتیبانی کند می‌تواند مورد استفاده قرار گیرد.

برای مثال، با استفاده از یک درخت خود متوازن، دربدترین حالت زمان یک جدول هش از O(n) به O(log n) کاهش می‌یابد. همه‌ٔ روشهای متنوعی که درهم سازی آرایه نامیده می‌شوند، از یک آرایه‌ٔ پویا برای ذخیره‌ٔ مقادیری که قرار است وارد آرایه شود استفاده می‌کنند.هر کلیدی که قرار است در یک خانه اضافه شود به انتهای یک آرایه‌ٔ پویا اضافه می‌شود که که به آن خانه اختصاص داده شده است.این تغییر باعث می‌شود که از حافظه‌ٔ نهان پردازنده استفاده‌ٔ کاراتری شود چرا که کلیدها در خانه‌های متوالی از حافظه ذخیره

می‌شوند.همچنین اشاره گر به خانه بعدی در لینک لیست هم وقتی کلیدها کوچکند باعث حفظ فضای حافظه می‌شود، مثل اشاره گرها یا اعداد صحیح تک کلمه ای(8 بیتی)

از دیگر جزئیات این روش پدیده ایست که درهم سازی کامل پویا نامیده می‌شود.یعنی یک آرایه با k کلید ورودی از یک جدول هش کامل با k2 شکاف تشکیل شود.وقتی این روش از حافظهٔ بیشتری استفاده می‌کند (n2 شکاف یا خانه برای n ورودی در بدترین حالت) تضمین می‌شود که در بدترین حالت زمان جستجو یک زمان از اوردر ثابت خواهد بود و همچنین زمان سرشکن برای اضافه کردن هم پایین خواهد بود.

آدرس دهی باز:

در یک استراتژی دیگر که به روش آدرس دهی باز مشهور است، تمام کلیدها در خود آرایه ذخیره می‌شوند.زمانیکه قرار است یک کلید وارد آرایه شود، جستجواز خانه‌ای که قرار است به آنجا هش شود آغاز می‌شود تا اینکه یک خانهٔ خالی پیدا شود و کلید درآن درج شود.برای یافتن یک کلید هم آرایه با همین منوال پیمایش می‌شود تا اینکه یا کلید مورد نظر یافت شود یا به خانهٔ خالی برخورد کنیم که در این صورت چنین کلیدی در جدول وجود نداشته است.

نام گذاری آدرس دهی بازبه این موضوع اشاره دارد که جایی که کلید در آنجا درج می‌شود الزاما همان جایی نیست که مقدار هش آن کلید به ما نشان می‌دهد.(این روش درهم سازی بسته نیز نامیده می‌شود که نباید با آدرس دهی بسته یا درهم سازی باز که معمولاً همان زنجیر سازی جداگانه است، اشتباه گرفته شود)

معروف‌ترین ترتیب‌های جستجو عبارتند از:

جستجوی خطی که فاصلهٔ بین جستجوها ثابت است(معمولا 1)

جستجوی درجه 2

هش مضاعف

از معایب روش آدرس دهی باز این است که تعداد کلیدهای درج شده نمی‌تواند از خانه‌های آرایه فراتر رود.در واقع، حتی با یک تابع هش خوب هم وقتی عامل بار به سمت 0.7 یا بیشتر رشد می‌کند، کارایی جدا کاهش می‌یابد.