اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

1– Hashing

علی محمد فروتن نژاد 810192429

نسیم شیروانی مهدوی 810192554

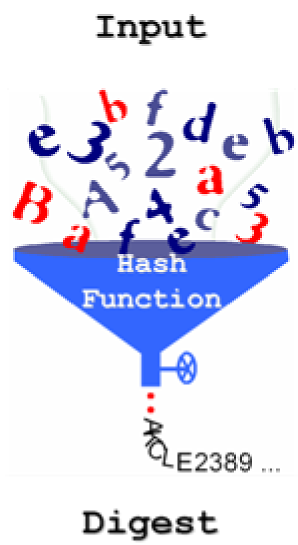
2– ساختمان داده‌ی هش

بسیاری از کاربردها در جهان کنونی به مجموعه‌های پویایی که توانایی وارد کردن، جستجو و حذف‌کردن اطلاعات را داشته باشند احتیاج دارند، که به اصطلاح به این مجموعه‌ها دیکشنری اطلاق می‌شود.

برای پیاده‌سازی دیکشنری می‌توانیم از لیست پیوندی، آرایه و جدول ادغامی (hash table) استفاده کنیم. که جدول ادغامی دارای مفهوم ساده‌تری از آرایه معمولی است.

یک جدول ادغامی(hash table) ساختمان داده مناسبی برای پیاده‌سازی دیکشنری است. برای مثال اگر برای پیاده‌سازی دیکشنری از یک لیست پیوندی یا آرایه استفاده کنیم برای جستجوی یک عنصر در بدترین حالت زمانی معادل o(n) خواهیم داشت. اما زمان لازم برای جستجوی یک عنصر در یک جدول ادغامی o(1) است.

برای پیاده‌سازی hash table توسط آدرس‌دهی مستقیم از یک آرایه استفاده می‌کنیم که به ما توانایی بررسی موقعیتی دلخواه را در زمان O(1) می‌دهد .البته آدرس‌دهی مستقیم هنگامی امکان‌پذیراست که ما توانایی تخصیص‌دادن آرایه‌ای را داشته‌باشیم که برای هر کلید ممکن، یک مکان داشته‌باشد. درادامه با تکنیک آدرس‌دهی مستقیم بیشتر آشنا خواهیم شد. عدد طبیعی حاصل از تابع درهم‌سازی معمولا به عنوان اندیس یک آرایه مورد استفاده قرارمی‌گیرد. مقادیری حاصل از این تابع را معمولاً **مقدار هش** یا فقط **هش** می‌خوانند.

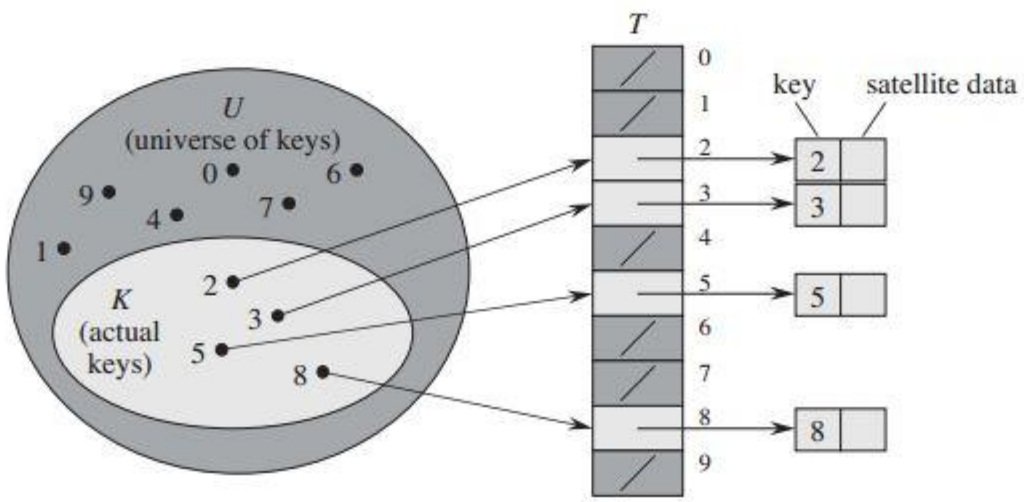


۱,۱– جدول‌هایی با آدرس‌دهی مستقیم :

استفاده از روش آدرس‌دهی مستقیم در شرایطی مناسب است که مجموعه‌ی عناصر ما و کلیدهای متناظرشان کوچک باشد، با فرض این که هیچ دو عنصری کلید مشابهی نداشته باشند.

به همین منظور برای نمایش این مجموعه از آرایه یا جدول آدرس‌دهی مستقیم، که با T[0 .. m-1] مشخص می‌شود، استفاده می‌کنیم که هر مکان آرایه، متناظر با کلیدی در مجموعه جهانی U است و هر شکاف (slot) k به عنصری از مجموعه با کلید k اشاره می‌کند، اگر مجموعه عنصری با کلید k نداشته باشد آنگاه T(k) = NULL است.

شکل 1 این روش را شرح می‌دهد:



شکل 1- روش آدرس‌دهی مستقیم

پیاده سازی اعمال این دیکشنری به صورت زیر می باشد:

Direct-Address-Search (T, k)

1. Return T[k]

Direct-Address-insert (T, x)

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

1. T [x.key] = x

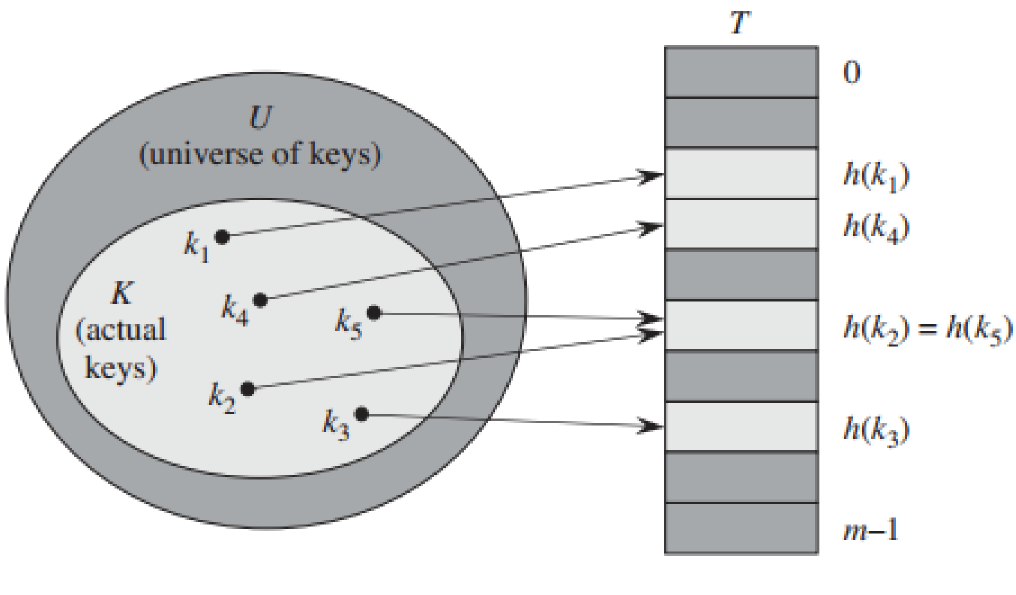
Direct-Address-Delete (T, x)

1. T [x.key] = NIL

همه این اعمال سریع هستند و تنها به زمان o(1) احتیاج دارند. با این روش دیگر ضرورتی نیست که کلید شی را داشته‌باشیم چون اندیس شی در جدول همان کلید شی است در نتیجه در مصرف حافظه صرفه‌جویی کرده‌ایم.

جدول ادغامی

یکی از مشکلات آدرس‌دهی مستقیم این است که اگر مجموعه جهانی U (مجموعه جهانی از کلیدها) بزرگ باشد، ممکن است ذخیره ی جدول T با اندازه |U| عملی نباشد ، از طرف دیگر ممکن است مجموعه K (مجموعه کلید های واقعی) برای کلید های ذخیره شده نسبت به U خیلی کوچک باشد و اغلب فضای تخصیص یافته برای T به هدر رود.



شکل 2-   جدول ادغامی

در آدرس‌دهی مستقیم عنصری با کلید K مسقیما در شکاف K ذخیره می‌شد. می توانیم از جدول ادغامی برای ذخیره‌سازی عنصر با کلید K در جدول استفاده‌کنیم. در این صورت عنصر با کلید K در شکاف h(k) ادغام می‌شود، که به h(k) تابع ادغامی نیز گفته می‌شود.

در اینجا h، مجموعه جهانی U از کلیدها را به شکاف های جدول ادغامی T[0...m-1] نگاشت می‌کند:

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

انتخاب تابع مناسب برای ادغام کردن

در این قسمت نکاتی را برای ادغام‌کردن مناسب ارایه می‌دهیم سپس دو طرح را برای توابع ادغامی توزیع با استفاده از ایده تقسیم و ضرب را مطرح می کنیم.

به‌طور کلی توزیع یکنواخت کلیدها در جدول در عمل بطور کامل امکان‌پذیر نیست اما معمولا توابعی، با استفاده از روش‌های مکاشفه‌ای (heuristic)، براساس دامنه کلیدها انتخاب می‌شوند. یک تابع ادغامی خوب تقریبا فرضیه ادغام یکنواخت را ارضا می‌کند هر کلید به طور مساوی می‌تواند به هر کدام از m قسمت ادغام شود، مستقل از اینکه بقیه کلیدها کجا ادغام شده‌اند. متاسفانه، نوعا چک‌کردن این شرایط ممکن نیست.

گاهی ما توزیع را می‌دانیم. مثلاً اگر کلیدها به طور تصادفی از اعداد حقیقی k باشند که به طور مستقل و یکنواخت در بازه توزیع شده‌اند آن گاه تابع ادغامی h(k) = [km] شرایط ادغام ساده یکنواخت را برمی‌آورد.

اغلب، توابع پراکندگی با فرض اینکه کلیدها اعداد طبیعی هستند طراحی می شوند و اگر کلیدها اعداد طبیعی نیستند، باید به گونه‌ای آنها را به اعداد طبیعی تبدیل‌کرد.

دو نوع روش برای توابع ادغامی معرفی می‌شوند:

1- روش تقسیم :

در این روش برای ایجاد تابع ادغام سازی از باقیمانده تقسیم صحیح k بر m برای نگاشت k به یکی از m شکاف استفاده می کنیم یعنی تابع ادغامی به صورت زیر است:

$$h(k) = k \bmod m$$

به عنوان مثال :

m = 15 , k = 46 □ h(k) = 1

m = 10 , k = 32 □ h(k) = 2

این روش روش سریعی است و فقط با یک عمل تقسیم انجام می‌شود.

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

در هنگام انتخاب روش تقسیم از بعضی از مقادیر m اجتناب می‌کنیم مثلا m نباید توانی از 2 باشد زیرا اگر که $m = 2^p$ باشد انگاه h(k)، p بیت مرتبه پایین k خواهد بود. بهتر است تابع ادغامی وابسته به همه بیت‌های کلید باشد. در واقع عدد اولی که توانی از 2 نباشد، می‌تواند انتخاب خوبی برای m باشد.

2- روش ضرب:

در این روش ابتدا کلید k را در ثابت A ضرب می کنیم که A<1>0 سپس قسمت کسری kA را استخراج می‌کنیم سپس آن را در m ضرب کرده و براکت کف نتیجه را می‌گیریم پس به طور خلاصه مشاهده می‌کنیم که:

$$h(k) = \lfloor m\,(kA \bmod 1)\rfloor = \lfloor m\,(kA - \lfloor kA \rfloor)\rfloor,\quad 0 < A < 1$$

به عنوان مثال اگر m = 1000 و k =132 و A = (√5-1)/2=0.618 آنگاه:

$$h(k) = \lfloor 1000(132*0.618 \bmod 1)\rfloor = 81.576$$

برخورد در ادغام کردن

مشکلی که برای ادغام کردن (hashing) وجود دارد، این است که 2 کلید ممکن است در یک قسمت ادغام شوند (چون در تابع ادغامی مقدار یکسانی دارند)، ما این اتفاق را برخورد می‌نامیم. به عنوان مثال در شکل 2، k2 و k5 به یک خانه جدول نگاشت شده‌اند.

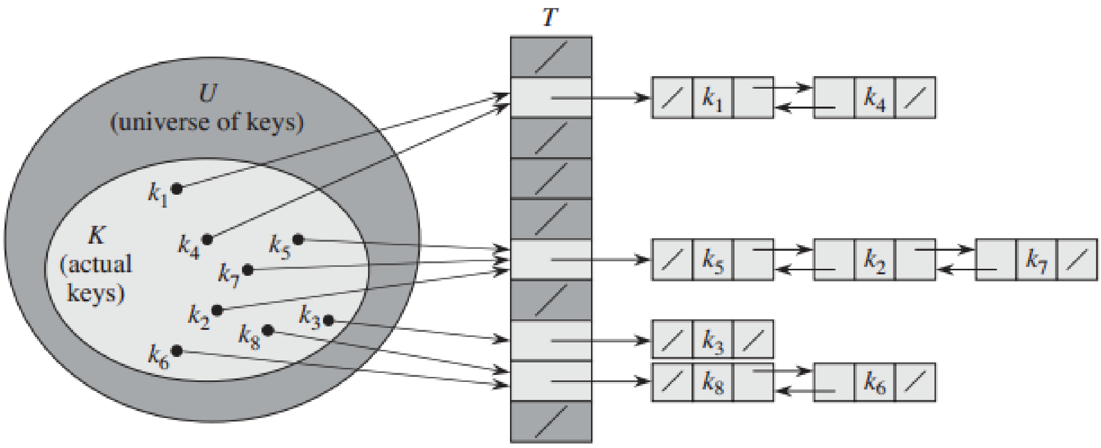
برای حل مشکل برخورد دو راه حل وجود دارد

1- حل برخورد با استفاده از زنجیره ای کردن
2- تکنیک آدرس دهی باز

در ادامه با این دو روش بیشتر آشنا خواهیم شد.

1.1.1– حل برخورد با استفاده از زنجیره ای کردن

در حل برخورد با استفاده از زنجیره‌ای‌کردن، ما همهٔ عناصری را که به یک شکاف (slot) مشابه ادغام (hash) می‌شوند را در یک لیست پیوندی قرار می‌دهیم که شکاف (slot) j شامل یک اشاره‌گر به سر لیست عناصر ذخیره شده‌ای است که در j ادغام شده‌اند. اگر در j عنصری ذخیره نشده‌باشد آنگاه شکاف j مقدار NULL را دارد یا به عبارت دیگر خالی است (مطابق شکل3)



شکل 3-   حل برخورد با استفاده از روش زنجیره ای کردن

پیاده‌سازی اعمال دیکشنری جدول ادغامی ای که با استفاده از روش زنجیره‌ای کردن برخورد آن رفع شده‌است:

CHAINED-HASH-INSERT(T,x)

1. insert x at the head of list T[h(x.key)]

CHAINED-HASH-SEARCH(T,k)

1. search for an element with key k in list T[h(k)]

CHAINED-HASH-DELETE(T,x)

1. delete x from the list T[h(x.key)]

بدترین حالت اجرا برای درج کردن $o(1)$ است. عمل درج تقریباً سریع است زیرا فرض می شود که عنصر x ای که باید در جدول درج (insert) شود هم اکنون در جدول حاضر نیست؛ در صورت لزوم می توان این فرض را چک کرد.(با هزینه ای اضافی به وسیله جستجو کردن قبل از درج عنصر)

برای جستجو بدترین زمان اجرا متناسب با طول لیست است و برای حذف کردن یک عنصر از جدول ادغامی ابتدا عنصر مورد نظر را در جدول می یابیم سپس آن را حذف می‌کنیم که بدترین زمان اجرا در اینجا هم متناسب با طول لیست می باشد.

البته اگر لیست‌های پیوندی دو طرفه باشند حذف عنصر x می تواند در زمان $o(1)$ اجرا شود. (مثلا عنصر آخر را حذف کنیم).

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

تحلیل جست و جوی Chaining Hashing

اگر m را تعداد slot های جدول ادغامی در نظر بگیریم و n را تعداد کل عناصر ذخیره شده در hash table باشد ضریبی به نام α را تعریف می‌کنیم که پراکندگی عناصر را در جدول نشان می‌دهد و به صورت زیر است :

Load factor □$=n/m$ -------> متوسط تعداد کلیدها در هرشکاف

بدترین حالت در زنجیره ای کردن این بود که همه عناصر در یک slot از جدول قرار بگیرند که زمان محاسبه □(n)+h(k) را دارد که h(k) زمان مورد نیاز برای دست‌یابی به slot مورد نظر در hash table و □(n) زمان مورد نیاز برای جست و جوی عنصر درhash table می باشد.

حالت متوسط وابسته به این است که چگونه کلیدها بین m شیار (slot) توزیع می شود. اگر فرض کنیم پراکندگی یکنواخت باشد آنگاه کلیدها به طور مساوی بین m شیار توزیع می‌شوند بنابراین زمان جستجو برای یک عنصر با کلید k برابر □(|T[h(k)]|) می‌باشد و طول مورد انتظار برای هر لیست در هر شکاف α می‌باشد.

پس اگر $n = O(m)$ آنگاه □$=n/m = O(m)/m = O(1)$ باشد

بنا بر این:

هزینه جستجو بطور متوسط : O(1)

هزینه اضافه کردن در بدترین حالت : O(1)

هزینه حذف در بدترین حالت : O(1)

پس تمامی عملیات dictionary با استفاده از hash table با chaining در صورتی که n=O(m) باشد، در حالت متوسط با O(1) انجام می شود (در بهترین حالت)

1.1.2– آدرس‌دهی باز (Open Addressing)

با استفاده از تکنیک آدرس‌دهی باز تمام عناصر در جدول ادغامی خودشان ذخیره می‌شوند و و برخلاف روش زنجیره ای کردن با استفاده از این روش هیچ عنصری خارج از جدول ادغامی ذخیره نمی شود. ورودی هر جدول شامل یک

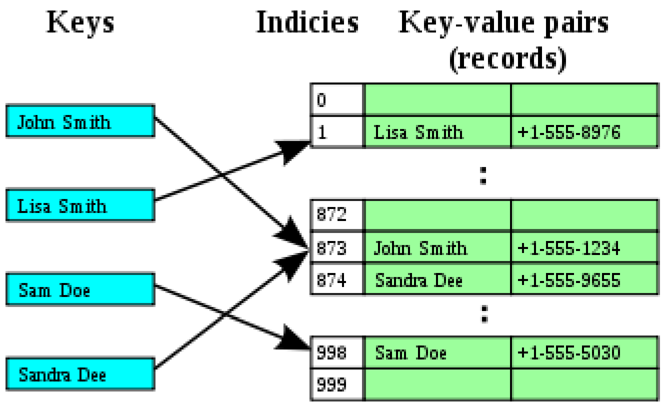
عنصر از مجموعه پویا یا NULL (پوچ) می‌باشد. برای جستجوی یک عنصر، ما تمام slot های جدول را جستجو می‌کنیم تا عنصر مورد نظر پیدا شود یا مشخص شود که عنصر در جدول وجود ندارد. همچنین به منظور انجام جایگذاری با استفاده از آدرس‌دهی باز، بطور پی‌درپی شکاف‌ها (slot) را در جدول ادغامی جستجو می‌کنیم تا زمانیکه یک قسمت خالی برای اینکه کلید را در آن قراردهیم پیدا کنیم. در آدرس‌دهی باز، جدول ادغامی می‌تواند پر شود تا زمانیکه هیچ جایگذاری اضافه‌ای نتواند ایجاد شود.

مزیت آدرس‌دهی باز نسبت به دیگر روش‌ها نظیر زنجیره‌ای کردن در این است که از داشتن اشاره‌گرها جلوگیری می‌کند. در واقع به جای دنبال کردن اشاره‌گرها، تنها کافی است دنباله ای از slot های (شکاف‌ها) جدول را بررسی کنیم. در واقع حافظه‌ای که برای ذخیره‌کردن اشاره‌گرها مصرف می‌کنیم را می‌توان برای ذخیره‌سازی عناصر مورد استفاده قرار داد پس دستیابی به عناصر سریع‌تر می‌شود.

نحوه ذخیره‌سازی به این صورت است که دنباله‌ای از مکان‌های پیمایش‌شده را به وسیله تابع ادغامی مناسب بدست می‌آوریم یعنی دنباله مکان‌های پیمایش شده به کلیدی که باید درج شود بستگی دارد و ما برای پیدا کردن دنباله‌ای برای ذخیره‌سازی عناصر تابع ادغامی را بسط می‌دهیم و عناصر را به ترتیب در slot های متناظر با این دنباله قرار می‌دهیم پس تابع ادغامی به صورت زیر است:

$$h : U * \{0,1,\ldots,m-1\} \dashrightarrow \{0,1,\ldots,m-1\}$$

درواقع با آدرس‌دهی باز لازم است که برای هر کلید k دنباله پیمایشی به فرم $(h(k,0),h(k,1),\ldots,h(k,m-1))$ از جایگشت (m-1,...,0,1,2) داشته باشیم.



در ادامه شبه کد Hash insert را مشاهده می‌کنیم که هر شکاف شامل یک کلید یا NULL (در صورتی که slot خالی باشد) است.

این تابع به عنوان ورودی جدول ادغامی T و کلید K را می‌گیرد، و یا شماره شکافی را بر می‌گرداند که کلید K در آن ذخیره شده است یا به دلیل پر بودن جدول ادغامی خطا اعلان می‌کند.

Hash-Insert(T, k)

1. i □ 0

2. repeat j □ $h(k, i)$

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

```
        if T[j] = NIL  or  T[j]= DELETED
                then T[j] □ k
                return j
        else
                i □ i + 1
until i = m
error "hash table overflow"
```

رویه HASH-SEARCH (T,k) جدول ادغامی Tو کلید k را به عنوان ورودی می‌گیرد و در صورتی که شکاف j حاوی کلید k باشد مقدار j را برمی‌گرداند و اگر k در جدول T نباشد مقدار NULL را برمی‌گرداند.

در ادامه شبه کد HASH-SEARCH را می بینیم:

```
Hash-Search (T, k)
i □ 0
repeat  j □ h(k, i)
        if T[j] = k
            then return j
        i □ i + 1
until T[j] = NIL or i = m
return NIL
```

حذف‌کردن عنصر از جدول ادغامی با استفاده از آدرس‌دهی باز دشوار است. وقتی ما یک کلید را از شکاف i حذف می‌کنیم، به راحتی نمی‌توانیم این شکاف را به عنوان قسمت خالی بوسیلۀ گذاشتن NULL داخل آن علامت‌گذاری کنیم.

انجام این کار باعث می شود تا که بازیابی کلید K غیر ممکن شود چون ما ابتدا در حین درج، آن شکاف را پیمایش میکنیم و ان را اشغال شده در نظر می گیریم، یک راه حل این است که در آن قسمت به جای مقدار NIL، مقدار خاص Deleted را قرار دهیم سپس رویه درج را تغییر دهیم تا با این شکاف به گونه ای رفتار کند که خالی است وعنصرجدید میتواند در آن قرار بگیرد . نیازی به تغییر Hash-search نداریم زیرا به هنگام جست و جو از مقادیر deleted عبور می کند.

سه تکنیک بطور معمول برای محاسبه توالی جستجو که برای آدرس‌دهی باز، احتیاج است استفاده می‌شود: جستجوی خطی، جستجوی مربعی، و ادغام دوتائی.

این تکنیک‌ها همگی تضمین می‌کنند که {h(k,0),h(k,1),….,h(k,m-1)} برای هر کلید k جایگشتی از {m-1,..,0,1,2} است. به این معنا که اگر m ، تعداد slotها باشد، با m بار فراخوانی تابع باید همه اندیسها هر کدام فقط یک بار تولید شوند.

هیچ کدام از این تکنیک‌ها فرض ادغام یکنواخت را برآورده نمی‌کنند، زیرا هیچ یک از آنها قادر به تولید کردن بیش از m^2دنباله ای از پیمایش های متفاوت نیست (به جای m! پیمایش متفاوت که برای ادغام یکنواخت مورد نیاز است). ادغام دوتایی بیشترین تعداد دنباله پیمایش را دارد و، همچنانکه مورد انتظار است، به نظر می‌رسد که بهترین نتایج را بدهد.

جستجوی خطی:

روش پیمایش خطی برای ادغام کردن از تابع ادغام زیر استفاده می کند ،

$$h(k, i) = (h'(k)+i) \bmod m \qquad i=0,1,..m-1$$

و ترتیب slot هایی از جدول را که مورد بررسی قرار می دهد به صورت زیر است :

$$T[h'(k)], T[h'(k)+1], \ldots, T[m-1], T[0], T[1], \ldots, T[h'(k)-1]$$

پیاده‌سازی جستجوی خطی آسان است، اما آن از یک مسئله شناخته شده با عنوان (Primary clustering) متأثر می‌شود.

به این معنی ممکن است تعداد زیادی از کلیدها در یک محدوده از جدول قرار بگیرند که این باعث می شود تا زمان جست و جو و اضافه کردن افزایش بیابد و به جست و جوی خطی نزدیک شود .

به عنوان مثال فرض کنیم که یک جدول پراکندگی با 26 خانه داریم میخواهیم کلمات زیر را در جدول به گونه ای قرار دهیم که شماره ی قرار گرفتن کلمه معادل با اندیس الفبایی اولین حرف باشد .

25	24	23	22	21	10	9	8	7	6	5	4	3	2	1	0

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

z	...	E	ZA	G	GA	A4	A3	D	A1	A2	A

Insert: GA, D, A, G, L, A2, A1, A3, A4, Z, ZA, E

Search: A1

Delete: A1

Search: A3

همانطور که می بینیم بیشتر عناصر در 10 خانه ابتدایی جدول قرار گرفته اند ، و برای درج عنصر A3 در جدول ناچاریم تا از A شروع کنیم و چون 5 خانه بعدی جدول پر است ناچارا در خانه 6 ام قرار می دهیم، به این پدیده primary clustering می گوییم مشاهده می کنیم که clustering باعث می شود که زمان انجام عملیات های درج و جست و جو و حذف کردن افزایش یابد .

جستجوی مربعی:

جستجوی مربعی از تابع ادغامی به شکل زیر استفاده می کند:

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m \quad c_2 \neq 0 \quad i=0,1,..m-1$$

که در آن c1 و c2 ثابت های مثبتی هستند و با انتخاب درست c1 و c2 می توان می توان اطمینان داشت که تابع پراکندگی به ترتیب اندیس تمام شکافها(slot) را تولید خواهد کرد .در این روش اولین مکان پیمایش شده T[h'(k)] است و مکان های بعدی به صورت مجذوری به پیمایش شماره i بستگی دارند.

اشکالی که ممکن است در این روش با آن برخورد کنیم این است که در تابع ادغام اگر h(k1)=h(k2) باشد ،در این صورت برای k1 و k2 ترتیب یکسانی از slot های جدول hash مورد بررسی قرار می گیرند. به عنوان مثال اگر

$h(k,i) = (h□(k) + i^2) \bmod m$ تابع ادغامی ما باشد به ازای

$m=5$ اگر $h□(k)=0$ ترتیبی که برای بررسی تولید می شود: 0,1,4,4,1,...

$m=6$ اگر $h□(k)=0$ ترتیبی که برای بررسی تولید می شود: 0,1,4,3,4,..

بنابراین مشاهده می کنیم که اگر دو کلید مکان پیمایش اولیه یکسانی داشته باشند، دنباله پیمایشی آنها یکسان می شود که این مشکل secondary clustering نام دارد و از معایب روش پیمایش مربعی است.

ادغام دوتائی:

در این روش از یک تابع hash دوم برای انتخاب slot بعدی استفاده می شود، ادغام دو تایی از یک تابع ادغام سازی به صورت زیر استفاده میکند :

$$h(k,i) = (h_1(k)+i\, h_2(k)) \bmod m \qquad i=0,1,..m-1$$

h1 و h2 توابع کمکی هستند که پیمایش اولیه همیشه در T[h1(k)] اتفاق می افتد و مکان پیمایش های بعدی به میزان h2(k) به پیمانه m فاصله دارند. در واقع ترتیب slot هایی که بررسی میشوند به صورت زیر است :

$$h(k),\ h(k)+h_2(k),\ h(k)+2h_2(k),\ h(k)+3h_2(k), \ldots$$

انتخاب h2 اهمیت بالایی دارد و h2 نباید 0 باشد.

ادغام دوتائی یکی از بهترین روشهای موجود برای آدرس‌دهی باز است زیرا جایگشت های تولید شده در آن بسیاری از مشخصات جایگشت هایی که به صورت تصادفی تولید شده اند را دارد.

مثال: $m=13$، $h_1(k)=k \bmod 13$، $h_2(k)= 1+(k \bmod 11)$

$k=14$، $h_1(14)=1$، $h_2(14)=4$

ترتیب مورد بررسی : 1,5,9,0,4,8,12,3,7,11,2,6,10

$k=15$، $h_1(15)=2$، $h_2(15)=5$

ترتیب مورد بررسی :2,7,12,4,9,1,6,11,3,8,0,5,10

نکته : یک انتخاب مناسب برای h2، h2(k)=p- (k mod p) است که p یک عدد اول کوچکتر از m است.

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

تابع درهم‌ساز رمزنگارانه (Cryptographic Hash function)

یک تابع درهم‌سازی رمزنگارانه نوعی تابع هش است که یک رشته را به رشته‌ای دیگر با طول ثابت تبدیل می‌کند. مقدار هش حاصل در واقع نمایشی از کل محتوای رشته ورودی است که به همین سبب آن رو نوعی "اثر انگشت دیجیتالی" به حساب می‌آوردیم.

از معروف‌ترین توابع درهم‌ساز رمزنگارانه می‌توان به md5, SHA-1, SHA-2 اشاره‌کرد

کاربردها

شکل 4- **تشخص صحت و درستی فایل‌ها و یا پیام‌ها**: به طور کلی می‌توان به همراه دیتای ورودی، هش آن را که معمولا با نام "Check Sum" شناخته می‌شود نیز دریافت‌کرد و با مقایسه این هش با هش اصلی دیتای ورودی به صحت و درستی فایل یا پیام مورد نظر پی‌برد.

شکل 5- **تایید پسورد**: ذخیره‌سازی پسوردها به همان شکلی که کاربر انتخاب می‌کند کار خطرناکی است چرا که در صورتی که شخص ثالثی بتواند به دیتابیس پسوردها دست پیداکند، می‌تواند تمام پسوردها را به طور دقیق بخواند و از آن‌ها سواستفاده‌کند. به همین منظور پسوردها اغلب به صورت هش نگهداری می‌شوند و همواره هش پسورد وردی با هش پسورد کاربر مقایسه می‌شود.

شکل 6- **شناسه فایل یا دیتا**: هش‌ها می‌توانند به عنوان تشخیص شناسه‌ی فایل‌ها و اطلاعات مورد استفاده قرار بگیرند. این سیستم توسط برنامه‌های مدیریت پروژه و نیز انتقال فایل همتا به همتا (Peer to Peer) استفاده می‌شود

شکل 7- **ساختن کلیدها**: یکی دیگر از موارد استفاده از توابع هش ساختن کلیدهایی (همچون کلیدهای فعال‌سازی نرم‌افزارها) برای برنامه‌هاست. بدین صورت که از یک سری اطلاعات معنادار هش گرفته شده و هش تولیدشده به عنوان کلید فعال‌سازی استفاده می‌شود

شکل 8- **نشانه‌های امنیتی (Token)**: توکن‌ها در دنیای کامپیوتر برای نگه‌داشتن اعتبار، ورود به یک سیستم هستند. یکی از روش‌های ساختن توکن‌ها استفاده از توابع درهم‌ساز رمزنگارانه است

درجه‌ی سختی

در رمزنگاری درجه‌ی سختی به میزان زمان لازم برای شکستن یک الگوریتم قبل از شناسایی شدن گفته می‌شود. این تعریف در واقع میزان هزینه و زمان لازم برای شکستن یک الگوریتم رمزنگاری را بیان می‌کند

مشکلات

یکی از بزرگترین مشکلات هش فانکشن‌ها محدود بودن طول رشته‌ی خروجی است. این مشکل می‌تواند مشکلات زیادی را پدید بیاورد. به این صورت که جدول‌هایی به وجود می‌آید که در آن‌ها به ازای هش موجود، رشته‌ای که آن هش را تولید می‌کند می‌توان پیداکرد که اصطلاحا به آن‌ها "Rainbow table" گفته می‌شود. البته شایان ذکر است که این جدول‌ها به طور کلی برای همه‌ی موارد کار نمی‌کنند چرا که برای یک الگوریتم ساده هشینگ مانند md5 به تنهایی 2 ^ 128 حالت مختلف وجود دارد

مشکل دیگر که نیز با عنوان "Collision attack" شناخته می‌شود نیز وجود دارد که به طور کلی مشکل بزرگ تمام هش‌ها ست. چرا که امکان تصادم ۲ کلید همواره و در همه‌ی الگوریتم‌های هشینگ وجود دارد

MD5:

یکی از رایجترین الگوریتم‌های موجود برای ساختن هش، الگوریتم md5 است. این الگوریتم یک رشته با طول دلخواه را به عنوان ورودی گرفته و سپس یک رشته‌ی با طول ۱۲۸ بیت به عنوان خروجی برمی‌گرداند.

این الگوریتم به طور کلی الگوریتم قابل اعتمادی محسوب نمی‌شود. یک کامپیوتر با پردازنده ۴ هسته‌ای GHz 2.6 می‌تواند در حدود ۱ ثانیه، به این الگوریتم حمله‌ی تصادم بزند.

SHA-1:

این الگوریتم بر اساس روش کار الگوریتم MD5 اما به صورت محافظه‌کارانه‌تری یک پیغام ۱۶۰ بیتی تولید می‌کند. این الگوریتم نیز در برابر حمله‌ی تصادم ناکارآمد است.

این الگوریتم در بسیاری از پروتوکول‌های ارتباطی نظیر SSL, PGP, SSH و ... استفاده می‌شود. نرم‌افزارهای مدیریت سیستم نیز از این الگوریتم برای شناسایی و تشخیص تخریب داده‌ای استفاده می‌کنند

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

نمونه سوال حل شده:

می خواهیم اعداد 22،31،4،15،28 و 10 را به یک جدول هش به طول 11 با استفاده از روش open addressing اضافه کنیم. جدول نهایی را پس از اضافه کردن این اعداد از چپ به راست برای هر یک از حالت های زیر رسم کنید.

h(k) = k mod m

Linear probing

Quadratic probing, c1 = 1, c2 = 3

Double hashing, h2(k) = 1 + (k mod (m-1))

حل:

قسمت 1:

0	1	2	3	4	5	6	7	8	9	10
22				4	15	28			31	10

قسمت 2:

0	1	2	3	4	5	6	7	8	9	10
22				4		28		15	31	10

قسمت 3:

اولین عددی که در هنگام قرار دادن آن به collision برخورد می کنیم، عدد 15 برابر است که برای آن داریم:

H(15,0) = 4c

H(15,1) = 4 + 1*(1 + 15 mod 10) = 10 collision

H(15,2) = (4 + 2*(1+15 mod 10)) mod 11 = 5

برای بقیه اعداد برخورد رخ نخواهد داد.

0	1	2	3	4	5	6	7	8	9	10
22				4	15	28			31	10

2- آیا توابع H(k,i) در روش open addressing مناسب است؟ در صورت مناسب نبودن علت آن را بیان کنید.(m برابر است با اندازه ی جدول)

H(k,i) = (k mod m + 3i mod m) mod m   m = 2t+1

H(k,i) = (k^2 + 2*k + i(k mod m)) mod m

باید تابع به صورتی باشد که به ازای هر $0 \le I < m$ یک خانه را مشخص کند.

الف) اگر m = 3t باشد این تابع قبول نیست چون همیشه 3i mod 3t = 3x و تمام خانه ها را پوشش نمی دهد، اما در غیر این صورت تابع قابل قبول خواهد بود.

ب) به دلیل اینکه ضریب i در مواردی می تواند صفر باشد، (k=tm) پس این تابع مناسب نیست.

3- فرض کنید که از درهم سازی با آدرس دهی باز برای درج و جستجوی عناصر A تا G در جدول درهم سازی Hاستفاده می کنیم همچنین فرض کنید که تابع در هم سازی به صورت زیر است و از وارسی خطی استفاده میکنیم .

key	A	B	C	D	E	F	G
hash	3	5	3	4	5	6	3

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

کدام یک از موارد زیر نمی تواند حاصل درج این عنصر با ترتیبی معین در جدول h (که از چپ به راست نوشته شده است) باشد؟

1) EFGACBD
2) CEBGFDA
3) BDFACEG
4) CGBADEF

راه حل:
اگر عناصر را به ترتیب ACBDEFGوارد کنیم گزینه ی اول حاصل میشود. و اگر این ترتیب ACEGBDFباشد گزینه 3 و اگر ADEFCGB باشد آرایه ی H مانند گزینه ی 4 میشود اما گزینه ی 2 چون در جای F (درایه ی آخر)A نشسته است ،باید A قبل از F آمده باشد . اما از سوی دیگر چون A از درایه ی سوم تا درایه ی ششم جلو آمده است ،چس هنگام آمدنش باید درایه های سوم تا پنجم (شامل درایه ی چهارم که F در آن قرار دارد) پر بوده باشد که در نتیجه ترتیب آمدن A و F در هر حال متناقض است.

تعداد حالت های مختلف ممکن برای جدول نهایی H پس از درج این 7 عنصر چند تاست؟

راه حل: میدانیم هیچ گاه عنصری با یک دور کامل وارسی در درایه ی 3 نمینشیند پس درایه ی 3 را بی گمان یکی از عناصر A,C و یا G پر خواهد کرد . پس برای این درایه 3 حالت داریم . برای هر کدام از 3 حالت پر شدن درایه ی 3، درایه ی 4 نیز 3 انتخاب دارد از بین A,C,G و D منهای عناصر درایه ی سوم.

با استدلالی مشابه به ازای هر روش پر شدن درایه های 3 و 4 درایه ی 5 تنها چهار گزینه دارد که شامل B یا E یا « A,C,G و D منهای عناصر درایه ی چار و سوم » هستند .

به روشی مشابه میتوان تعداد حالت های درایه ی ششم پس از پر بودن درایه های سوم تا پنجم را برابر 4 به دست آورد (تمامی حروف منهای حروفی که در درایه های سوم و چهارم و پنجم استفاده شده اند) برای درایه ی صفرم و اول و دوم نیز به ترتیب 3 و 2و یک انتخاب باقی می ماند . پس جواب برابر با 4*4*3*3*2 است.

4-اعمال زیر را بر روی جدول نماد های یک کامپایلر انجام می دهیم :

درج : یک شناسه وارد جدول می شود .

جستجو : یک شناسه را در جدول پیدا می کند.

فهرست : فهرست همه ی شناسه های موجود در جدول را به ترتیب الفبایی چاپ میکند .

برای پیاده سازی این جدول از روش در هم سازی باز استفاده میکنیم . کدام یک از گزینه های زیر زمان اجرای یک پیاده سازی کارا از این جدول با n عنصر را نشان میدهد؟

1) درج : O(lg n) جستجو : O(lg n) فهرست : O(n)
2) درج : O(1) جستجو : O(n) فهرست : O(nlgn)
3) درج : O(1) جستجو : O(lg n) فهرست : O(n)
4) درج : O(1) جستجو : O(n) فهرست : O(n)

راه حل : میدانیم که هزینه ی درج ضربدر n به علاوه ی هزینه ی فهرست باید از باشد . چرا که عمل مرتب سازی را انجام میدهد . پس گزینه های 3 و 4 اشتباه می باشند .
برای پیاده سازی گزینه ی اول نیز کافی است هر درایه از جدول در هم سازی یک درخت جستجوی متوازن (نظیر درخت قرمز سیاه یا AVL) باشد . در این صورت می توان عناصر را در O(lgn)در آن حذف و یا جستجو کرد و با یک پیمایش مناسب آن را درO(n) لیست کرد.