داده ساختار جدول درهم ساز (Hash Table):

یادآوری: هدف از معرفی و بررسی داده ساختارها، ذخیره و بازیابی مجموعه‌های پویا به صورت کارا می‌باشد.

در این راستا، داده ساختارهای زیادی معرفی شده اند و بررسی کردیم که عملیات مختلفی متناسب با کاربرد مورد نیاز تعریف شده اند.

در حالت کلی داده ساختارهایی که تا این نقطه از درس فرا گرفتیم در دو رده کلی زیر قابل تقسیم بندی هستند:

- داده ساختارهای مقدماتی، نظیر: لیست، لیست پیوندی، صف، صف حلقوی و پشته.
- داده ساختارهای پیشرفته تر، نظیر: درخت Trie، درخت دودویی جستجو (BST)، درخت دودویی عمارت (BET) و درخت هرم بیشینه (Max-Heap tree).

سؤال: پیچیدگی زمانی عملیات پایه روی داده ساختارهای فوق، نظیر: عملیات درج، حذف و جستجو، را قبلاً بررسی کردیم، در اینجا سؤال آن است که آیا می‌توان این عملیات پایه را در $O(1)$ انجام داد، یا سعی کرد تا جایی که ممکن است، حداقل پیچیدگی زمانی را برای این عملیات داشت؟

در ادامه به بررسی داده ساختار جدول درهم ساز برای پاسخگویی به این سؤال می‌پردازیم.

تعریف داده ساختار جدول درهم ساز (Hash Table):

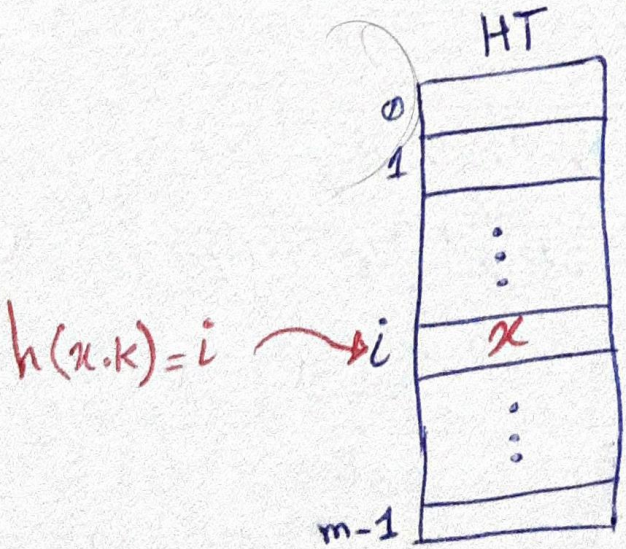
فرض کنید:

* m خانه از حافظه در اختیار داریم که از صفر تا $m-1$ اندیس گذاری شده است،
* n کلید که مقادیر آن از صفر تا $n-1$ است نیز در اختیار داریم،
* همچنین یک تابع درهم ساز h با توصیف زیر نیز داده شده است:

$$h: \{0,\dots,n-1\} \longrightarrow \{0,\dots,m-1\}$$

یک جدول درهم ساز HT با m خانه حافظه، داده ساختاری است که هر کلید $k \in \{0,\dots,n-1\}$ را با استفاده از تابع درهم ساز h به طور مناسبی به یکی از m خانه حافظه نگاشت کرده و عنصر حاوی کلید k را در آن خانه از حافظه درج می‌کند.

شکل زیر یک جدول درهم ساز را ارائه کرده است. با فرض اینکه x یک عنصر است و $h(x.k)=i$ داریم:



در ادامه سعی داریم به چالش های اصلی داده ساختار جدول درهم ساز، یعنی ① تعیین مقدار خانه‌های حافظه، ② تعیین یک تابع درهم ساز خوب بپردازیم.

بررسی یک حالت ایده آل از نظر کارایی:

دسترسی مستقیم (Direct Access) یا آدرس دهی مستقیم (Direct Address)

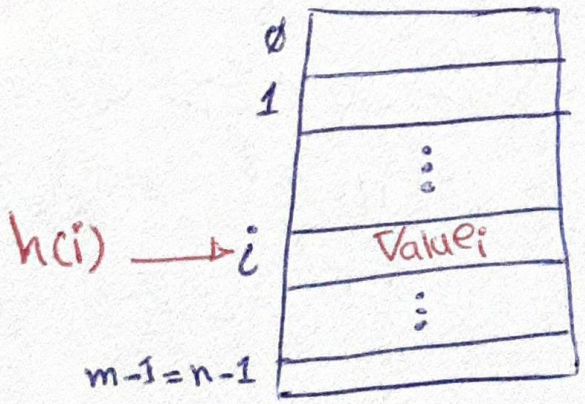
در این روش، با فرض عدم محدودیت روی حافظه مثال تشریحات زیر را برای جدول درهم ساز در نظر گرفت:

$m = \{0, \dots, m-1\} = [m]$

$n = \{0, \dots, n-1\} = [n]$

$[m] = [n]$, $h(i) = i$

$HT(h(i)) = \begin{cases} Value_i & \text{اگر کلید عناصر جا بافتد} \\ Null & \text{در غیر اینصورت} \end{cases}$



در این روش، عملیات درج، حذف و جستجو به صورت زیر قابل اعمال است:

- درج مقدار Value برای عنصر با کلید k به صورت $HT[h(k)] = HT[k] = Value$ انجام می‌شود.
- حذف عنصر با کلید k به صورت $HT[h(k)] = HT[k] = Null$ انجام می‌شود.

- جست وجوی یک عنصر با کلید k به صورت زیر قابل انجام است:

- اگر $HT[h(k)] = HT[k] = Null$ باشد، آنگاه عنصر با کلید k در مجموعه ی داده های خود نداریم.

- اگر $HT[h(k)] = HT[k] \neq Null$ باشد، عنصر با کلید k موجود است و می توانیم آن عنصر را بازیابی کنیم.

نکته: روش آدرس دهی مستقیم، یک راه حل تئوری (Theoritical) و نه عملی (Practical) است و فقط فضای مصرفی زیادی را می طلبد. با این حال پیچیدگی زمانی عملیات پایه: درج، حذف و جست وجو در آن از مرتبه $O(1)$ است.

حرکت به سوی جداول درهم ساز عملی (با قید محدودیت حافظه):

در ادامه به دنبال راه حلی هستیم که با اعمال محدودیت حافظه بتوانیم داده ساختار جدول درهم ساز تا حد امکان کارایی را برای عملیات پایه ارایه دهیم.

به عبارت دیگر می خواهیم به سراغ تنظیماتی برویم که $m \ll n$ (m خیلی کوچکتر از n) باشد.

$[m] = \{0, 1, \cdots, m-1\}$

$k \in [n] = \{0, 1, \cdots, n-1\}$

$h: [n] \longrightarrow [m]$

$m \ll n$