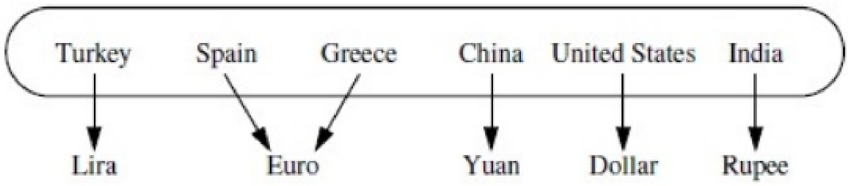
ساختمان داده
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی. پاییز ۱۴۰۲

۱ ساختار داده انتزاعی MAP

Map (نگاشت) یک ساختار داده است که هر قلم داده در آن یک زوج (کلید، مقدار) است بطوریکه کلیدها محنصربفرد بوده و تکرار در آنها رخ نمی‌دهد. متناظر با هر کلید key یک مقدار value وجود دارد. برای مثال کلید می‌تواند شماره دانشجویی و مقدار نام و نام خانوادگی دانشجو باشد. برای مثالی دیگر، کلید می‌تواند پلاک خودرو باشد و مقدار مدل خودرو باشد. شکل زیر نمایشی از یک نمونه ساختار داده Map می‌باشد. کلیدها در آن اسم کشورها و مقدار هر کلید واحد پول آن کشور می‌باشد.



اعمال اصلی ساختار داده عبارتند از:

- جستجوی کلید و برگرداندن مقدار مربوطه در صورت وجود.

$$\text{return} \;\; M[k]$$

- درج کلید به همراه مقدار مربوطه

$$M[k] = v$$

- حذف کلید

$$\text{del} \;\; M[k]$$

در پایتون ساختار داده دیکشنری dict ساختار داده‌ای از نوع Map است که امکاناتی چون جستجوی یک کلید، پیداکردن مقدار متناظر با یک کلید، درج یک زوج کلید و مقدار آن، حذف یک کلید و غیره را فراهم می‌آورد. برای تسریع اجرای اعمال مربوطه پایتون از تکنیک درهم‌سازی Hashing استفاده می‌کند. در این درس و درس آینده به معرفی و بررسی این تکنیک می‌پردازیم.

۲ پیاده‌سازی ساختار داده MAP

با فرض اینکه کلیدها اعداد صحیح 0 تا $N-1$ هستند می‌توانیم از یک آرایه به طول N برای پیاده‌سازی MAP استفاده کنیم.

$$MAP = \{(11, L), (7, Q), (6, C), (3, Z), (1, D)\}$$

0	1	2	3	4	5	6	7	8	9	10	11
	D		Z			C	Q				L

با این روش اعمال درج و حذف و جستجو در زمان $O(1)$ قابل انجام است.

معایب این روش:

- فضای مصرفی بالا. اگر تعداد کلیدها از N خیلی کمتر باشد، مکانهای زیادی از آرایه خالی خواهد بود.
- فقط برای کلیدهای عددی قابل استفاده است.

برای رفع مشکلات بالا می‌توانیم از درهمسازی استفاده کنیم. اولا اینکه می‌توانیم از تابعی استفاده کنیم که کلیدهای غیرعددی را تبدیل به عدد کند.

$$g(\text{non-numeric keys}) \to \{0, 1, 2, \cdots, N'-1\}$$

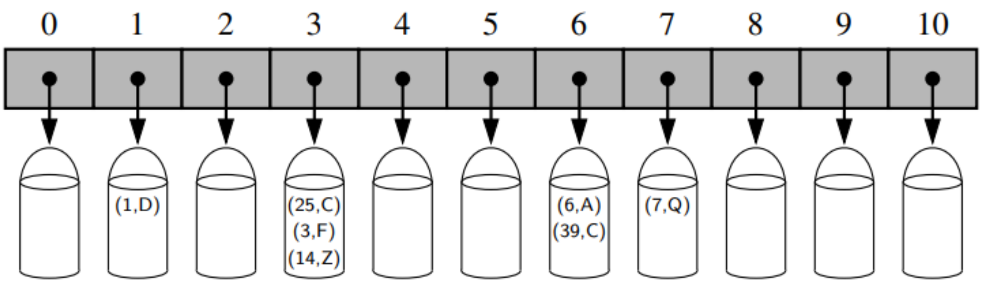
دوما برای اینکه مشکل اتلاف حافظه را حل کنیم از تابعی دیگر مانند

$$h : \{0, 1, \cdots, N'-1\} \to \{0, 1, 2, \cdots, N-1\}$$

استفاده می‌کنیم که اعداد بزرگ را به اعدد کوچک (یک بازه کوچك از اعداد صحیح و مثبت) نگاشت کند. یک ایده ساده برای h می‌تواند باقیمانده تقسیم بر عدد N باشد.

$$h(x) = x \mod N$$

برای مثال در زیر یک نگاشت داریم که از تابع $h(x) = x \mod 11$ استفاده می‌کند. طبیعتا در این حالت تصادم collision می‌تواند رخ دهد. به عبارت دیگر بعضی از کلیدها به یک خانه واحد نگاشت شوند.

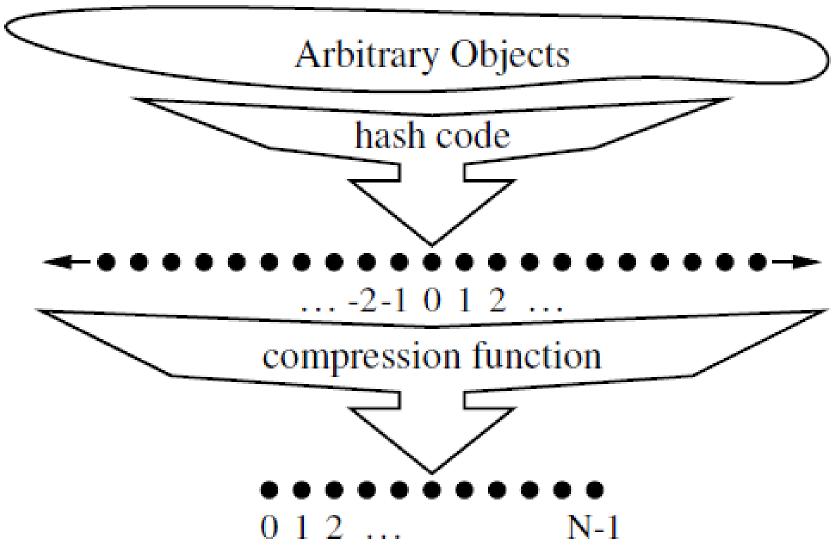


MAP = { (1,D), (25,C), (3,F), (14,Z), (6,A), (39,C), (7,Q) **}**

مطلوب است که تابع درهمسازی داشته باشیم که تعداد تصادمها تحت آن کم باشد و علاوه بر این خود تابع بصورت سریع محاسبه شود و فضای زیادی برای نگه داری در حافظه نخواهد.

۳ ایده مختلف برای تابع درهمسازی

همانطور که گفتیم یک تابع درهمسازی می‌تواند دو لایه داشته باشد. در لایه اول کلید غیر عددی تبدیل عدد می‌شود. به تابعی که این کار را انجام می‌دهد کد درهمسازی hash code گفته می‌شود. در لایه دوم اعدد بزرگ با استفاده از یک تابع فشرده سازی compression function به اعداد کوچک نگاشت می‌شوند.



۱.۳ ایده های مختلف برای hash code

- استفاده از نمایش باینری و تابع XOR

فرض کنید کلید x از n قطعه ۳۲ بیتی تشکیل شده است.

$$x = (x_0 \underbrace{x_1}_{32\text{ bits}} \cdots x_{n-1})$$

تابع XOR معمولا با نماد $\oplus$ نمایش داده می‌شود. اینجا دو رشته ۳۲ بیتی را گرفته و بیتهای متناظر آنها را XOR می‌کند. حاصل یک رشته ۳۲ بیتی است. این کار بصورت زنجیر وار روی همه قطعات کد انجام می‌شود و حاصل که یک رشته ۳۲ بیتی است برگردانده می‌شود. بدین ترتیب کلید ورودی تبدیل به یک عدد در بازه 0 تا $2^{32}-1$ خواهد شد.

$$\text{hash code}(x) = x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1}$$

- کد درهمسازی چند جمله‌ای polynomial hash code

در این روش عدد دلخواه a انتخاب می‌شود و عبارت چند جمله‌ای زیر محاسبه می‌شود. برای نگه داری حاصل از 32 بیت استفاده می‌شود. بدیهی است که بدلیل بزرگی عدد حاصل سرریز overflow ممکن است اتفاق بیافتد که اهمیتی ندارد چون تصادم در هر صورت اجتناب ناپذیر است.

$$x = (x_0 \underbrace{x_1}_{32\text{ bits}} \cdots x_{n-1})$$

$$\text{hash code}(x) = x_0 + x_1 a^1 + x_2 a^2 + \cdots + x_{n-1} a^{n-1}$$

- کد درهمسازی با استفاده از شیفت چرخشی

مانند روش قبلی است با این تفاوت که ضرب در عدد a را بصورت شیفت چرخشی بیتها به تعداد a بار به سمت راست یا چپ تلقی می‌کنیم.

نکته: تابع hash در پایتون یک کد درهمسازی را برای شیء داده شده برمی‌گرداند. شیء داده شده باید از نوع immutable باشد.

۲.۳ تابع فشرده سازی

- روش باقیمانده تقسیم
$h(x) = x \mod N$
خیلی ساده اما غالبا به تنهایی ناکارآمد. برای مثال اگر کلیدها ضرایب N باشند همه به یک خانه نگاشت می‌شوند.

- روش ضرب و تقسیم Multiply and Divide (MAD)

در این روش عدد اول p انتخاب می‌شود که بزرگتر یا مساوی با N باشد. سپس اعداد $a \in \{1, \cdots, p-1\}$ و $b \in \{0, 1, \cdots, p-1\}$ بصورت تصادفی انتخاب می‌شوند.

$$h(x) = ((ax + b) \mod p) \mod N$$

دقت کنید قسمت $(ax+b) \mod p$ وقتی a و b تصادفی هستند باعث می‌شود که کلید ورودی در بازه صحیح $[0, p-1]$ بصورت یکنواخت توزیع شود. بدین ترتیب با احتمال خوبی اعدادی ورودی به خانه‌های مختلف نگاشت می‌شوند.

لم: با فرض اینکه $x \neq y$ داریم

$$Pr(h(x) = h(y)) \leq \frac{1}{N}$$

اثبات: با فرض اینکه زوج $x \neq y$ مشخص شده است، چند انتخاب مختلف برای (a, b) باعث می‌شوند که رخداد نامطلوب $h(x) = h(y)$ اتفاق بیافتند؟ یک کران بالا برای این تعداد بدست می‌آوریم. فرض کنید $h(x) = h(y)$ اتفاق افتاده. داریم

$$((ax + b) \mod p) \mod N = ((ay + b) \mod p) \mod N$$

پس عدد k وجود دارد که

$$((ax + b) \mod p) = ((ay + b) \mod p) + kn$$

از این می‌توانیم نتیجه بگیریم

$$ax + b \equiv ay + b + kn \pmod p$$

پس

$$ax \equiv ay + kn \pmod p$$

چون $x \neq y$ پس $(x - y)$ در میدان F_p وارون دارد. لذا

$$a \equiv (x - y)^{-1} kn \pmod p$$

چند k مختلف می‌توانیم داشته باشیم؟ حداکثر $\frac{p-1}{N}$. لذا از میان $p - 1$ انتخاب ممکن برای a حداکثر به تعداد $\frac{p-1}{N}$ از آنها می‌توانند بد باشد. لذا داریم

$$Pr(h(x) = h(y)) \leq \frac{\frac{p-1}{N}}{p-1} = \frac{1}{N}$$

□

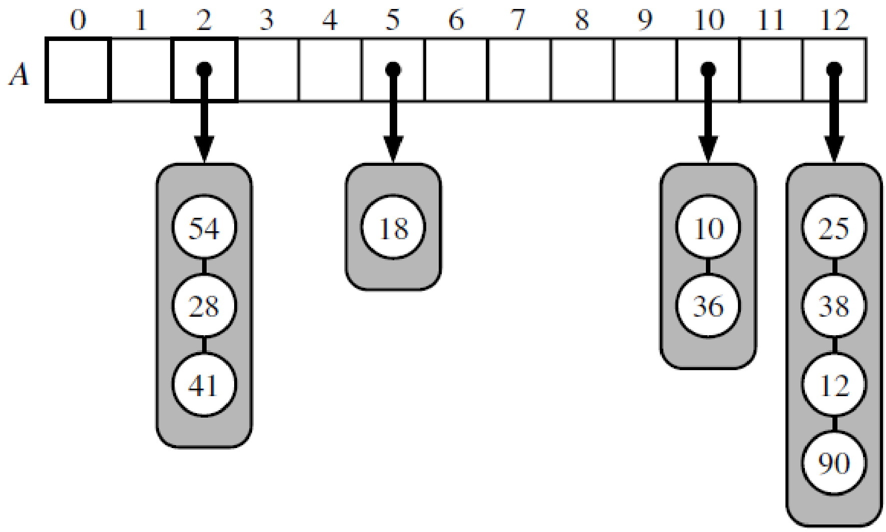
از مشاهدات بالا نتیجه می‌شود با فرض اینکه m کلید عددی داریم، بطور متوسط تعداد کلیدهایی که با کلید x به یک خانه نگاشت می‌شوند حداکثر $\frac{m-1}{N}$ است.

۴ روشهای مدیریت تصادم

با فرض اینکه تصادم رخ می‌دهد و تعدادی کلید به یک خانه نگاشت می‌شود، مدیریت تصادم و روش نگهداری کلیدها در حافظه می‌تواند با ایده‌های مختلف انجام شود.

- روش زنجیره‌های مجزا Separate Chaining

در این روش برای هر خانه پر، یک ساختار داده مجزا نگهداری می‌شود. این ساختار داده می‌تواند یک آرایه مجزا، یک لیست پیوندی، یک درخت BST یا هر ساختار داده دیگری باشد. در این حالت زمان جستجو و درج و حذف بستگی به حداکثر تعداد کلیدها در هر خانه دارد. اگر کلیدها بطور یکنواخت توزیع شوند، هر خانه بطور متوسط $O(m/N)$ کلید خواهد داشت. به پارامتر $\lambda = \frac{m}{N}$ اصطلاحا فاکتور بار load factor گفته می‌شود. اگر فاکتور بار کم باشد، میزان اتلاف حافظه می‌تواند زیاد باشد.

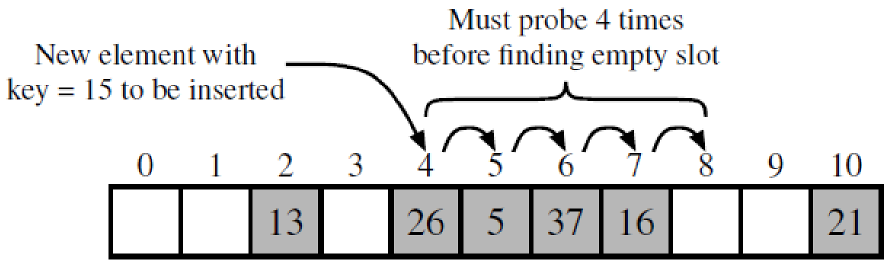


- روش آدرس دهی باز Open Addressing

در آدرس دهی باز بطور کلی از تصادم اجتناب می‌شود. به شرطی که $\lambda \leq 1$ می‌توان از خانه‌های خالی آرایه برای ذخیره‌سازی کلیدهای بدشانس استفاده کرد. لذا در صورت نیاز وقتی خانه $h(x)$ قبلا توسط کلید دیگر اشغال شده باشد، کلید x در اندیس دیگری غیر از $h(x)$ ذخیره می‌شود. ایده های مختلفی برای انتخاب خانه خالی موجود برای ذخیره سازی x وجود دارد.

۱. Linear Probing در این روش ابتدا سعی می‌شود x در خانه $A[h(x)]$ ذخیره شود. اگر $A[h(x)]$ پر باشد، به راست حرکت می‌کنیم و اولین خانه خالی که پیدا کردیم x را در آنجا درج می‌کنیم. اگر به انتهای آرایه رسیدیم به شروع آرایه برمی‌گردیم و جستجو برای خانه خالی را از آنجا از سر می‌گیریم.

بدین ترتیب، موقع جستجو برای x، وقتی دیدیم خانه $A[h(x)]$ پر است و توسط کلید دیگری اشغال شده است، به سمت راست حرکت می‌کنیم و دنبال x می‌گردیم. اگر سر راه به یک خانه خالی برخورد کنیم آنجا می‌توانیم نتیجه بگیریم که کلید x وجود ندارد.



h(x) = x mod 11

موقع حذف کلید x، نمی‌توانیم بسادگی خانه مربوطه را خالی کنیم چون خالی کردن یک خانه باعث می‌شود که جستجو برای کلیدهای واقع در خانه‌های سمت راست مختل شود. یک ایده ساده این است که خانه‌ای که در آن حذف اتفاق می‌افتد را علامت بزنیم. موقع جستجو از روی خانه‌ای خالی علامت زده رد می‌شویم. طبیعی است بعد از مدتی که خانه‌های خالی علامت زده تعدادشان زیاد شد، نیاز به یک پاکسازی داریم و معمولا درهمسازی دوباره انجام می‌شود.

نتایج تجربی نشان می‌دهد که روش Linear Probing مشکل ایجاد قطعات پر و پراکنده را دارد. در نتیجه توزیع کلیدها در خانه‌های خالی بعد از مدتی از حالت یکنواخت خارج خواهد شد.

۲. Quadratic Probing برای رفع مشکل قطعات پراکنده در روش قبلی می‌توانیم جستجو برای خانه خالی را بصورت خطی و پشت سر هم انجام ندهیم و از پرشهایی با طولهای متفاوت استفاده کنیم. جستجو برای اولین خانه خالی در روش Linear Probing را می‌توان با حلقه زیر مدل کرد

$$A[h(x)+i] \quad \text{for } i = 1, 2, 3, \cdots$$

در تعمیمی از این روش، خانه مورد جستجوی بعدی می‌تواند از الگوی زیر تبعیت کند.

$$A[h(x)+f(i)] \quad \text{for } i = 1, 2, 3, \cdots$$

در روش Quadratic Probing از تابع $f(i) = i^2$ استفاده می‌شود.