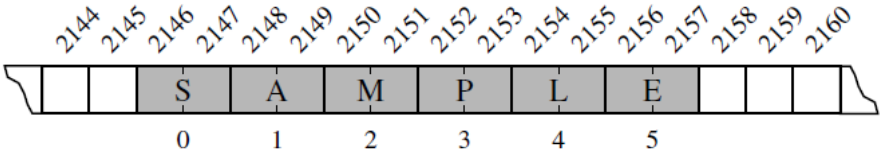
درس ساختمان داده‌ها
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی بهار ۹۹
تدریس توسط: حسین جوهری

خلاصه درس پنجم: آرایه و لیست در پایتون

در این جلسه بصورت خلاصه‌وار به بررسی ساختارهای آرایه‌ای می‌پردازیم. چند کلاس متداول پایتون مثل list و tuple و str از این دسته از ساختارها هستند که هر کدام قابلیتها و ویژگیهای خاص خود را دارند. در اینجا سعی میکنیم مکانیزم داخلی و چگونگی پیاده‌سازی این کلاسها را مورد بحث و بررسی قرار دهیم.

۱ آرایه داده‌ای

ساده‌ترین شکل ساماندهی و ذخیره اطلاعات در حافظه به این صورت است که اطلاعات بصورت دنباله‌ای متوالی در مکانهای پشت سر هم در حافظه کامپیوتر ذخیره شوند. در زبان برنامه‌نویسی به دنباله‌ای از داده‌ها که در واحدهای هم اندازه و پشت سر هم ذخیره شده‌اند یک آرایه گفته می‌شود. همانطور که در شکل زیر مشاهده می‌شود، حروف کلمه SAMPLE در قطعات پشت سر هم حافظه (در یک آرایه) ذخیره شده اند. اعداد بالای هر واحد حافظه، آدرس آن محل از حافظه را نشان می‌دهند. فرض کنید هر واحد حافظه در اینجا ۸ بیت (۱ بایت) باشد. هر حرف برای نگهداری به ۱۶ بیت نیاز دارد، پس هر حرف دو واحد حافظه را اشغال کرده است. آدرس شروع آرایه 2145 است. حرف L واقع در اندیس 4 آرایه دو واحد 2154 و 2155 را اشغال کرده است.



مزایای آرایه داده‌ای مزیت اصلی استفاده از آرایه این است که اگر بخواهیم داده واقع در یک اندیس دلخواه را بخوانیم بسرعت می‌توانیم به آن دسترسی پیدا کنیم. اگر start آدرس شروع آرایه باشد و بخواهیم به عنصر واقع در اندیس k ام دسترسی داشته باشیم (با فرض اینکه هر عنصر آرایه به اندازه cellsize واحد حافظه را اشغال کرده باشد) کافیست سراغ آدرس start + cellsize * k برویم و داده مورد نظر را از آن آدرس بخوانیم. این نشان می‌دهد مزیت دیگر آرایه داده‌ای این است که نیازی به نگهداری اطلاعات بیشتر برای دسترسی به داده‌ها وجود ندارد. تنها داشتن آدرس شروع آرایه کافی است.

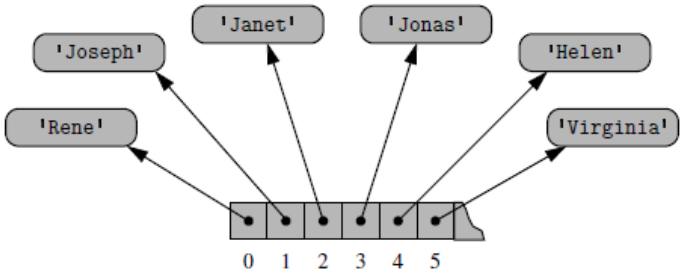
معایب آرایه‌داده‌ای با وجود اینکه آرایه داده‌ای بسیار سریع و از لحاظ فضای نگهداری بسیار مقرون بصرفه است، اما استفاده از آن برای نهگداری داده‌های ناهمگون و غیر متوازن با اتلاف حافظه مواجه خواهد شد. فرض کنید بخواهیم لیستی از اسامی را ذخیره کنیم.

```
['Rene', 'Jonas', 'Helen', 'Virginia', ... ]
```

از آنجا که اسامی افراد غالبا طولهای متفاوت دارند، ناچاریم یک طول ثابت برای هر اسم در نظر بگیریم. فرض کنید مقدار آستانه حداکثر ۲۰ کاراکتر را برای هر اسم در نظر بگیریم. از آنجا که اکثر افراد اسامی با طول کوتاه (مثلا ۴ یا ۵ دارند) درصد قابل توجهی از حافظه بهدر خواهد رفت. علاوه بر این اگر بخواهیم دنباله‌ای از داده‌های ناهمگون (از نوع مختلف) را ذخیره کنیم باز هم ناچاریم برای آدرسدهی سریع یک میزان آستانه برای هر عنصر دنباله تعیین کنیم که باعث اتلاف حافظه خواهد شد. برای رفع این مشکل، آرایه‌های ارجاعی مورد استفاده قرار می‌گیرند که در قسمت بعدی به آن می‌پردازیم.

۲ آرایه ارجاعی

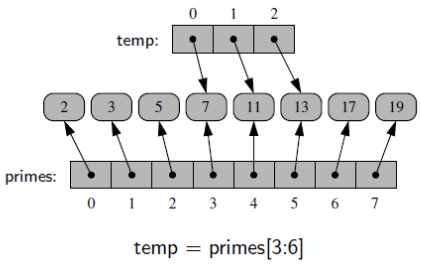
در آرایه ارجاعی به جای اینکه داده واقعی در محلهای پشت سر هم ذخیره شوند، آدرس داده‌ها در هر خانه از آرایه ذخیره می‌شود. شکل زیر یک آرایه ارجاعی را نشان می‌دهد.

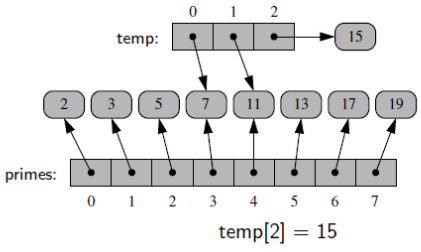


هر خانه از آرایه حاوی اشاره‌گری است که به جایی از حافظه اشاره می‌کند (محلی از حافظه که داده مورد نظر در آنجا ذخیره شده است.). این اشاره‌گر در واقع آدرس آن محل از حافظه است. با این روش می‌توان داده‌های با انواع مختلف و اندازه‌های مختلف را در یک آرایه بدون اتلاف حافظه ساماندهی کرد. تنها بدی کار این است که علاوه بر خود داده‌ها، آدرس داده‌ها نیز ذخیره می‌شود. برای خواندن عنصر k ام آرایه، کافی است که آدرس محل ذخیره شدن عنصر k ام را از خانه k ام آرایه بخوانیم و سپس سراغ محل اصلی ذخیره شدن آن برویم. پس برای خواندن یک قلم داده به دو دسترسی نیاز است (در آرایه داده‌ای تنها یک دسترسی کافی بود.)

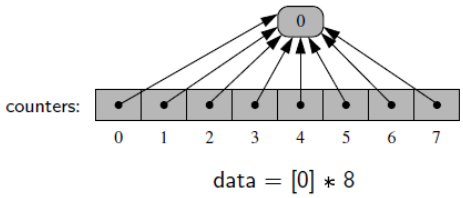
۱.۲ ساختار ارجاعی لیست در پایتون

ساختار list در پایتون از روش آرایه ارجاعی استفاده می‌کند. در شکل زیر لیست primes حاوی هشت عدد اول است. آرایه ارجاعی به محلهایی که اعداد در آنجا ذخیره شده‌اند اشاره می‌کند. حتی وقتی قطعه‌ای از لیست در لیستی دیگر کپی می‌شود (لیست temp)، لیست جدید به همان محلهای حافظه اشاره می‌کند. اما وقتی عناصر لیست جدید تغییر می‌کنند، اشاره‌گر عناصر مربوطه به محلهای جدیدی از حافظه اشاره می‌کنند. در پایین سمت راست، تاثیر دستور انتصاب temp[2]=5 را می‌بینید.

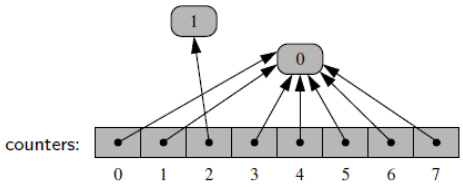




حتی زمانیکه با دستور $\text{data} = [0]*8$ یک لیست حاوی هشت عدد صفر ایجاد می‌کنیم، در واقع تنها یک محل حافظه حاوی صفر ایجاد می‌شود. عناصر لیست اشاره‌گری به محل مربوطه خواهند بود (شکل سمت چپ). وقتی دستور $\text{data}[2]=1$ را اجرا می‌کنیم، یک شی جدید حاوی 1 ایجاد می‌شود و عنصر اندیس دوم لیست به آن اشاره می‌کند (شکل سمت راست.)

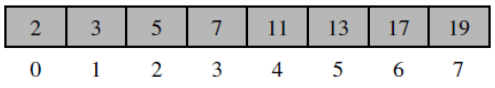


data = [0] * 8



۲.۲ کلاس array در پایتون

اگر بخواهیم یک آرایه‌ای داده‌ای در پایتون ایجاد کنیم (بطوریکه مثل آرایه list ارجاعی نباشد) می‌توانیم از کلاس array استفاده کنیم. در اینجا محتوای هر عنصر آرایه خود داده است و نه اشاره‌گر به آن. البته نوع عناصر آرایه را قبلا باید مشخص کنیم. برای مثال آرگومان i به این معنی است که نوع عناصر لیست عدد صحیح (علامت دار) است. جدول زیر انواع مختلف از پیش تعیین شده را نشان می‌دهد.

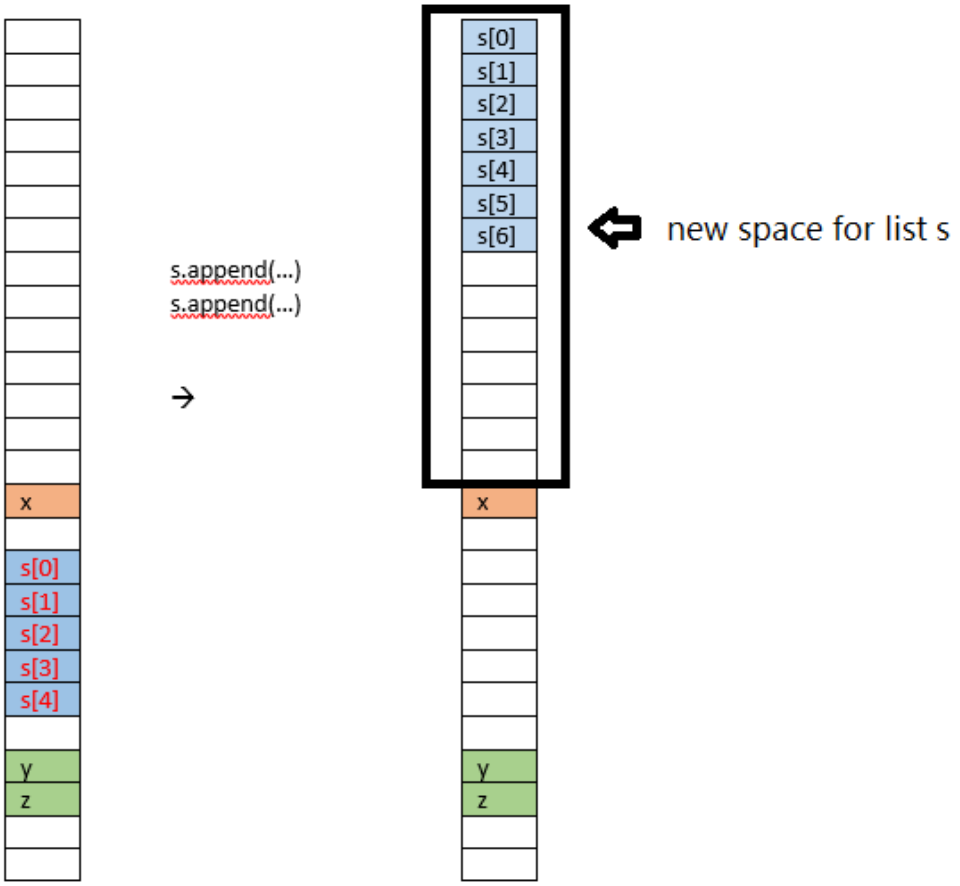


primes = array('i', [2, 3, 5, 7, 11, 13, 17, 19])

Code	C Data Type	Typical Number of Bytes
'b'	signed char	1
'B'	unsigned char	1
'u'	Unicode char	2 or 4
'h'	signed short int	2
'H'	unsigned short int	2
'i'	signed int	2 or 4
'I'	unsigned int	2 or 4
'l'	signed long int	4
'L'	unsigned long int	4
'f'	float	4
'd'	float	8

۳ آرایه پویا

اگر بخواهیم آرایه‌ای داشته باشیم که طولش متغیر باشد و بتوانیم عناصر جدید به آن اضافه کنیم، یک مشکل کوچک پیش می‌آید. برای مثال فرض کنید بخواهیم عناصر جدید را به انتهای آرایه اضافه کنیم (مانند متد append در لیست پایتون). چون عناصر آرایه (چه داده‌ای باشد و چه ارجاعی) در مکانهای پشت سر هم حافظه ذخیره شده‌اند، باید در محل آرایه در حافظه کامپیوتر، فضای خالی برای گسترش آرایه موجود باشد. اگر چنین فضایی موجود نباشد، عمل افزایش طول آرایه با مشکل مواجه می‌شود. یک راه حل برای این مسئله این است که هر بار دیدیم فضای خالی وجود ندارد، کل آرایه را به محلی دیگر از حافظه انتقال دهیم تا مشکل نبود فضا برطرف شود. به شکل زیر توجه کنید. در سمت چپ فضا برای گسترش لیست s وجود ندارد. در سمت راست لیست s به جایی دیگر منتقل شده و سپس دستورات append اجرا شده است.



راه حل انتقال آرایه به مکانی دیگر و تخصیص حافظه جدید (از نظر زمانی) عملی هزینه بر است. اگر بخواهیم هر بار که حافظه کم آوردیم، آرایه را به جایی دیگر منتقل کنیم دستور append بسیار هزینه بر خواهد بود. اگر بخواهیم هر دفعه این کار را انجام ندهیم یک راه این است موقعی که عمل گسترش را انجام می‌دهیم، به اندازه دو برابر اندازه فعلی آرایه حافظه بگیریم. یعنی اگر تعداد عناصر آرایه n باشد، حافظه‌ای به اندازه $2n$ بگیریم و آرایه را در آنجا کپی کنیم. با این تمدید، n عمل append بعدی با خیال راحت و بسرعت انجام می‌شود. در شکل بالا لیست s به مکانی جدید منتقل شده به اندازه دو برابر اندازه فعلی، حافظه به آن تخصیص داده شده.

۱.۳ تحلیل سرشکنی Amortized Analysis

در روش پیشنهادی بالا که به آرایه پویا (dynamic array) موسوم است، اجرای یک دستور append در بدترین حالت به اندازه طول آرایه زمان خواهد برد اما درصد بالایی از اجراهای append زمان اجرایشان $O(1)$ خواهد بود. پس اگرچه در بدترین حالت زمان اجرا بالاست اما بطور متوسط زمان اجرا قابل قبول است. در این حالت گفته می‌شود بصورت سرشکنی amortized زمان اجرای دستور append پایین است.

برای اینکه دقیقتر صحبت کنیم، فرض کنید کپی کردن یک آرایه با k عنصر و اختصاص حافظه‌ای به اندازه $2k$ عنصر به آن به k واحد زمانی نیاز داشته باشد. در این روش هر زمان که تعداد عناصر آرایه به توانی از 2 رسید (برای مثال 2^i) حافظه اختصاصی به آرایه دو برابر می‌شود (2^{i+1}) و به این ترتیب جا برای عناصر بعدی باز می‌شود. زمان اجرا برای ۱۶ اجرای متوالی دستور append در نمودار شکل زیر نشان داده شده است.

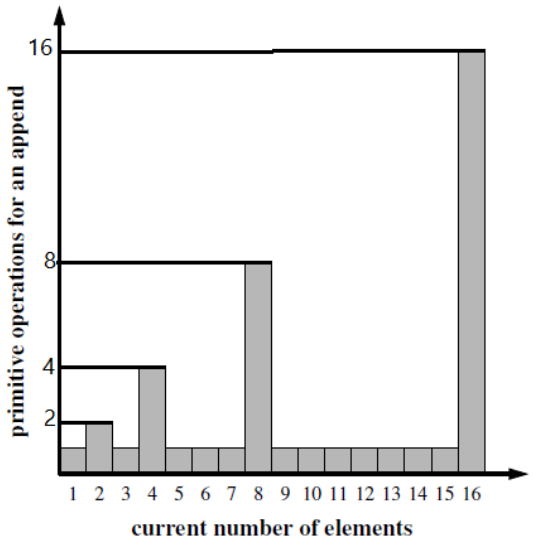


Figure ۱: شکل از صفحه ۱۹۷ کتاب مرجع

لم: اجرای n دستور append روی یک آرایه خالی، با تکنیک آرایه پویا، در $O(n)$ واحد زمانی قابل انجام است.

اثبات: برای اثبات این ادعا نشان می‌دهیم اجرای n دستور append به حداکثر $3n$ واحد زمانی نیاز دارد. دو نوع append را تمییز می‌دهیم:

- regular append (معمولی) زمانی که بعد از اجرا طول آرایه توانی از 2 نیست.
- special append (فوق العاده) زمانی که بعد از اجرا طول آرایه توانی از 2 می‌شود و متعاقبا آرایه گسترش می‌یابد و حافظه جدید به آن تخصیص داده می‌شود.

در اینجا فرض بر این است که اجرای هر دستور append (بدون در نظر گرفتن زمان مورد نیاز برای دو برابر کردن طول آرایه) به یک واحد زمانی نیاز دارد. اگر زمان اختصاص یافته را مثل پولی که می‌پردازیم نگاه کنیم، می‌خواهیم نشان دهیم برای اجرای n دستور متوالی append به تعداد حداکثر $3n$ دلار نیاز داریم. فرض کنید برای هر append 3 دلار کنار بگذاریم (می‌خواهیم نشان دهیم این مقدار برای کل اجرا کافی خواهد بود.) موقع اجرای یک append از 3 دلار 1 دلار را خرج می‌کنیم و 2 دلار را میتوانیم پس انداز کنیم. زمانی که طول آرایه به 2^i برسد، از توان دو قبلی (2^{i-1}) تا حال به تعداد حداقل $2 \times 2^{i-1}$ دلار پس انداز کرده‌ایم. با این مقدار می‌توانیم هزینه کپی کردن آرایه (2^i دلار) را پرداخت کنیم. شکل زیر این ایده را برای موقعی که تعداد عناصر آرایه به 2^3 می‌رسد را نشان می‌دهد.

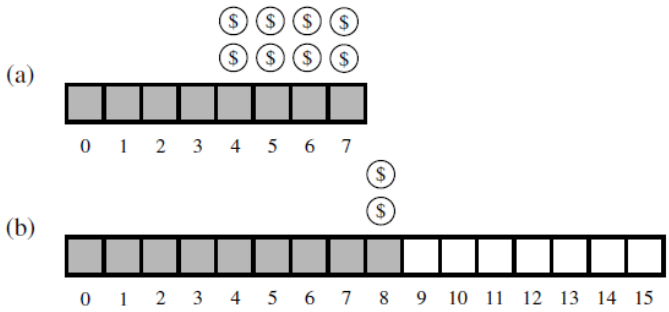


Figure ۲: شکل از صفحه ۱۹۸ کتاب مرجع

با توجه به لم بالا می‌توانیم بگوییم بطور سرشکنی amortized زمان اجرای یک دستور append $O(n)/n$ یا همان $O(1)$ خواهد بود. برای اینکه با زمان بدترین حالت اشتباه نشود، این را بصورت $O(1)^*$ می‌نویسیم.

۲.۳ زمان اجرای متدهای مختلف لیست پایتون

با فرض اینکه لیست پایتون از تکنیک آرایه پویا استفاده می‌کند، با توجه به توضیحاتی که در قسمت قبل داده شد، زمان اجرای متدهای مختلف لیست پایتون بصورت زیر خواهد بود.

Operation	Running Time
data[j] = val	$O(1)$
data.append(value)	$O(1)^*$
data.insert(k, value)	$O(n-k+1)^*$
data.pop( )	$O(1)^*$
data.pop(k) **del** data[k]	$O(n-k)^*$
data.remove(value)	$O(n)^*$
data1.extend(data2) data1 += data2	$O(n_2)^*$
data.reverse( )	$O(n)$
data.sort( )	$O(n \log n)$

*amortized

Figure ۳: شکل از صفحه ۲۰۴ کتاب مرجع