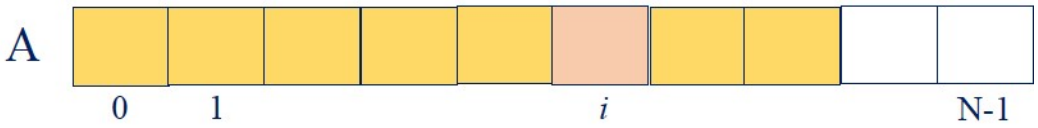
Data Structures and Algorithms

دنباله‌های مبتنی بر آرایه



Dr. *Ali Valinejad*

valinejad.ir
valinejad@umz.ac.ir

فهرست مطالب

انواع دنباله‌ها در پایتون

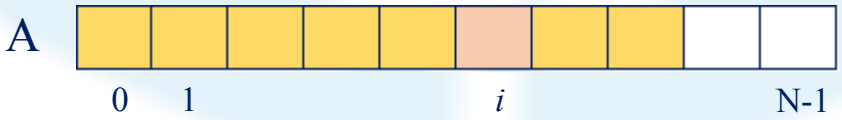
انواع کلاس‌های مبتنی بر دنباله در پایتون

1. list
2. tuple
3. str

اشتراکات:

- حمایت از استراتژی اندیس‌گذاری برای دسترسی به یک عنصر مجزا از دنباله (seq[k])
- استفاده از مفهوم سطح پایین آرایه برای نمایش دنباله

آرایه، یک مجموعه مکان‌هایی از حافظه است که با استفاده از اندیس‌گذاری متوالی آدرس دهی می شود. در پایتون و c/c++ اندیس آرایه‌ها از صفر شروع می شود).



تفاوت‌ها:

- تفاوت در ADT و نمونه سازی

آرایه‌های سطح پایین

- حافظه اولیه کامپیوتر از بیت اطلاعاتی تشکیل شده است ،
- این بیت‌ها معمولاً به واحدهای بزرگتری به نام ***بایت*** تقسیم می‌شوند که به دقت معماری سیستم بستگی دارند.
- یک سیستم رایانه‌ای تعداد زیادی بایت حافظه دارد. برای ردیابی اینکه چه اطلاعاتی در چه بایت ذخیره می‌شود، رایانه از انتزاعی به عنوان آدرس حافظه استفاده می‌کند.
- گروهی از متغیرهای مرتبط می‌توانند یکی پس از دیگری در یک قسمت مجاور و پیوسته از حافظه کامپیوتر ذخیره شوند. چنین نمایشی را به عنوان یک آرایه نامند.

آرایه‌های سطح پایین

آرایه ارجاعی

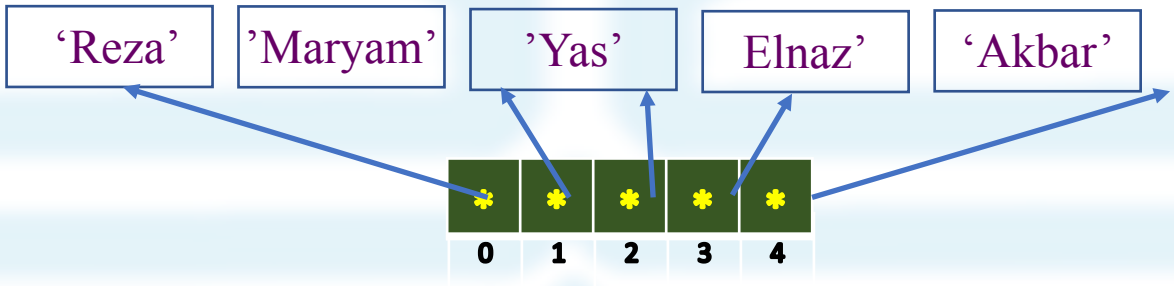
سوال: چگونه می‌توان لیست اسامی زیر را در یک ساختار مبتنی بر آرایه نمایش داد؟

['Reza','Maryam','Yas','Elnaz', 'Akbar']

- **راه اول:** در پایتون می‌توان فضای کافی برای ذخیره سازی هر سلول را طوری در نظر گرفت که طول حداکثر هر رشته را حفظ کند (نه فقط رشته های در حال حاضر ذخیره شده ، بلکه از هر رشته ای که ممکن است بخواهیم آن را بعدا ذخیره کنیم)

'Reza'	'Maryam'	'Yas'	'Elnaz'	'Akbar'

- راه حل دوم: پایتون یک نمونه از list یا tuple را با استفاده از مکانیسم ذخیره‌سازی داخلی از آرایه‌ای از ارجاعات به اشیاء نشان می‌دهد. در پایین ترین سطح ، آنچه ذخیره می‌شود، دنباله‌ای متوالی از آدرس‌های حافظه است که در آن عناصر دنباله قرار می گیرند.



آرایه‌های فشرده

آرایه های فشرده در پایتون

- رشته ها با استفاده از آرایه ای از کاراکترها (نه آرایه ای از ارجاعات) نمایش داده می‌شوند.

- به آرایه ای که به جای ارجاعات بیت‌هایی را ذخیره می کند که داده‌های اولیه را نشان می‌دهند (کاراکترها، در مورد رشته ها)، ***آرایه فشرده*** گویند.

- مزیت آرایه‌های فشرده بر ساختارهای ارجاعی:
  - حافظه مورد استفاده در آرایه های فشرده کمتر از ساختارهای اجرایی است. زیرا نیازی به ذخیره سازی دنباله‌ای از حافظه‌ای ارجاعی علاوه بر داده‌های اولیه نیست.
  - برخلاف ساختارهای ارجاعی، در یک ساختار فشرده، داده‌های اولیه به طور متوالی در حافظه ذخیره می‌شوند.

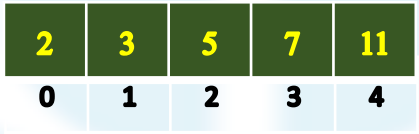
- پشتیبانی اولیه از آرایه‌های فشرده در ماژولی به نام *array* وجود دارد.
  - ماژول *array* کلاسی به نام *array* تعریف می کند که شرایط ذخیره سازی فشرده برای آرایه‌ای از داده‌های اولیه را مهیا می‌سازد

2	3	5	7	11
0	1	2	3	4

primes = array(i , [2, 3, 5, 7, 11])

آرایه‌های فشرده

آرایه‌های فشرده در پایتون



primes = array(i , [2, 3, 5, 7, 11])

Type Code	C Type	Minimum size in Bytes
‘b’	signed char	**1**
‘B’	unsigned char	**1**
‘u’	Unicode character	**2 or 4**
‘h’	signed short	**2**
‘H’	unsigned short	**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’	double	**8**

Type codes supported by the array module

آرایه‌های پویا

- در هنگام ایجاد یک آرایه سطح پایین در یک سیستم رایانه‌ای، باید اندازه دقیق آن آرایه به طور صریح اعلام شود تا سیستم بطور صحیح یک قطعه از حافظه ثابت را برای ذخیره‌سازی آرایه اختصاص دهد.

- از آنجا که سیستم ممکن است مکان‌های حافظه همسایه را برای ذخیره داده‌های دیگر اختصاص دهد، ظرفیت یک آرایه را نمیتوان با گسترش در سلول‌های بعدی افزایش داد.
  - این محدودیت در نمایش یک نمونه از list و tuple در پایتون وجود ندارد.

اگرچه یک لیست پایتون در هنگام ساخت دارای طول خاص است، اما کلاس list به ما امکان می دهد تا بدون در نظر گرفتن محدودیت ظاهری در ظرفیت کلی لیست، عناصر جدیدی را به لیست اضافه کنیم. به این منظور، پایتون به یک ترفند الگوریتمی به عنوان ***آرایه پویا*** تکیه می کند.

- طول یک آرایه پویا (vector در C ++)، وقتی که بخواهیم یک عنصر جدید به آرایه اضافه کنیم در حالی که فضا برای مورد جدید وجود نداشته باشد، بطور خودکار افزایش می‌یابد.

آرایه‌های پویا

- یک نمونه از کلاس list در پایتون، یک آرایه زیرین با ظرفیتی بیشتر از طول فعلی لیست درنظر می‌گیرد. این ظرفیت اضافی باعث می شود تا بتوان به آسانی عنصر جدیدی را به لیست اضافه نمود.

```
import sys
data = [ ]
n=10
for k in range(n):
    a = len(data)
    b =sys.getsizeof(data)
    print( 'Length:%d; Size in bytes:%d'%(a, b),end=' ')
    print(data)
    data.append(None)
```

Output:

```
Length:0;  Size in bytes:64  [ ]
Length:1;  Size in bytes:96  [None]
Length:2;  Size in bytes:96  [None, None]
Length:3;  Size in bytes:96  [None, None, None]
Length:4;  Size in bytes:96  [None, None, None, None]
Length:5;  Size in bytes:128 [None, None, None, None, None]
Length:6;  Size in bytes:128 [None, None, None, None, None, None]
Length:7;  Size in bytes:128 [None, None, None, None, None, None, None]
Length:8;  Size in bytes:128 [None, None, None, None, None, None, None, None]
Length:9;  Size in bytes:192 [None, None, None, None, None, None, None, None, None]
```

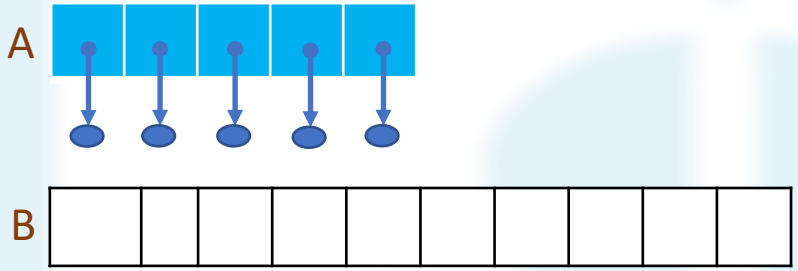
پیاده‌سازی آرایه‌های پویا

- اگرچه کلاس لیست پایتون اجرای بسیار بهینه‌ای از آرایه‌های پویا را ارائه می‌دهد، اما در ادامه پیاده‌سازی چنین کلاسی را مورد بررسی قرار می‌دهیم.
- نکته کلیدی مهیا نمودن امکانی برای برای رشد آرایه A برای ذخیره‌سازی عناصر لیست است. البته، در واقع نمی‌توان این آرایه را رشد داد، زیرا ظرفیت آن ثابت است. اما می‌توان از یک آرایه کمکی استفاده نمود.
- وقتی که آرایه مربوط به یک لیست پر باشد و بخواهیم عنصر جدیدی را به لیست اضافه کنیم، مراحل زیر را انجام می‌دهیم:
  - Allocate a new array **B** with larger capacity.
  - Set **B**[i]=**A**[i], for i=0,...,n, where n denotes current number of items.
  - Set **A**=**B**, that is, we henceforth use **B** as the array supporting the list.
  - Insert the new element in the new array.

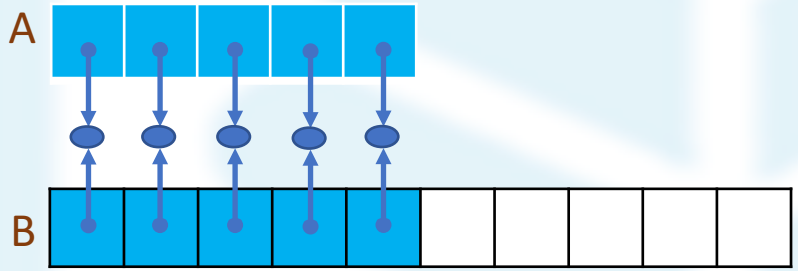
تذکر مهم:

موضوع مهمی که در پیاده‌سازی آریه‌های پویا باید در نظر گرفت، تعیین ظرفیت آرایه جدید است. یک قانون متداول این است که ظرفیت آرایه جدید دو برابر ظرفیت آرایه موجود باشد.

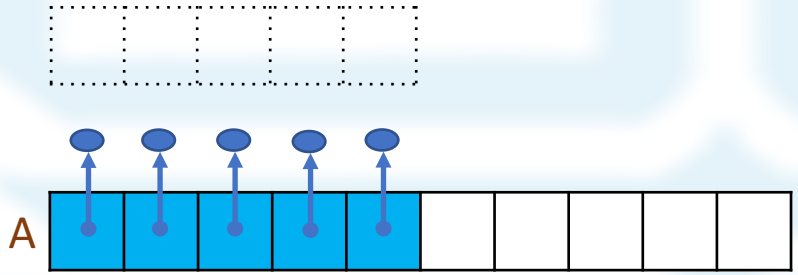
پیاده‌سازی آرایه‌های پویا



Create new array B



Store elements of A in B



Reassign reference A to the new array

پیاده‌سازی آرایه‌های پویا

```
import ctypes
class DynamicArray:
    """A dynamic array class akin to a simplified Python list."""

    def __init__(self):
        """Create an empty array."""
        self._n = 0
        self._capacity = 1
        self._A = self._make_array(self._capacity)

    def _make_array(self, c):
        """Return new array with capacity c."""
        return (c * ctypes.py_object)()

    def __len__(self):
        """Return number of elements stored in the array."""
        return self._n

    def __getitem__(self, k):
        """Return element at index k."""
        if not 0 <= k < self._n:
            raise IndexError('invalid index')
        return self._A[k]
```

پیاده‌سازی آرایه‌های پویا

```
    def _resize(self, c):
        """Resize internal array to capacity c."""
        B = self._make_array(c)
        for k in range(self._n):
            B[k] = self._A[k]
        self._A = B
        self._capacity = c

    def append(self, obj):
        """Add object to end of the array."""
        if self._n == self._capacity:
            self._resize(2 * self._capacity)
        self._A[self._n] = obj
        self._n += 1
```

فرض کنید **S** یک دنباله پیاده‌سازی شده توسط یک آرایه پویا با ظرفیت اولیه یک و با استفاده از *استراتژی دو برابر کردن* اندازه آرایه در زمان پر بودن باشد، در این صورت زمان کلی انجام یک سری از n عملیات append در **S**، با شروع با یک دنباله خالی **S**، از مرتبه O(n) است.

پیاده‌سازی آرایه‌های پویا

```
def insert(self, k, value):
    if self._n == self._capacity:
        self._resize(2 * self._capacity)
    for j in range(self._n, k, -1):
        self._A[j] = self._A[j-1]
    self._A[k] = value
    self._n += 1

def print_array(self):
    for k in range(self._n):
        print(self.__getitem__(k),end=' ')
    print()
```

پیاده‌سازی آرایه‌های پویا

```
if __name__ == '__main__':
    A = DynamicArray()
    print(len(A),A._capacity)
    for k in range(5):
        A.insert(k,k/2)
    A.print_array()
    print(len(A),A._capacity)
    for k in range(9):
        A.append(10*k/2)
    A.print_array()
    print(len(A),A._capacity)
    A._resize(14)
    print(len(A),A._capacity)
    A.print_array()
```

Output:

```
0 1
0.0 0.5 1.0 1.5 2.0
5 8
0.0 0.5 1.0 1.5 2.0 0.0 5.0 10.0 15.0 20.0 25.0 30.0 35.0 40.0
14 16
14 14
0.0 0.5 1.0 1.5 2.0 0.0 5.0 10.0 15.0 20.0 25.0 30.0 35.0 40.0
```

کارایی انواع دنباله‌ها در پایتون

کلاس‌های list و tuple در پایتون

- رفتارهای بدون تغییر(*nonmutating behaviors*) کلاس list، دقیقاً همانند رفتارهای بدون تغییر کلاس tuple است.
  (محاسبه طول- دسترسی به یک اندیس خاص- مقایسه و ...)
- ***tuple*** معمولاً در استفاده از حافظه بسیار کاراتر از list عمل می‌کند، زیرا نیازی به آرایه پویا با ظرفیت مازاد ندارد.

کارآیی مجانبی رفتارهای بدون تغییر کلاس‌های list و tuple

عملیات	زمان اجرا
len(data)	O(1)
data[j]	O(1)
data.count(value)	O(n)
data.index(value)	O(k+1)
value in data	O(k+1)
data1 == data2 (similarly !=, <, <=, >, >=)	O(k+1)
data[j:k]	O(k− j+1)
data1 + data2	O(n1 +n2)
c *data	O(cn)

شناسه‌های **data1**، **data2** و **data** نمونه‌هایی از کلاس list یا tuple و **n1**، **n** و **n2** طول متناظر با آن‌ها را نشان می‌دهند. **k** اندیس چپ‌ترین وقوع یک عنصر را نشان می‌دهد (اگر **k=n**، اندیس وقوع وجود ندارد). برای مقایسه دو دنباله، k نشانگر چپ‌ترین اندیسی است که در آن دو عنصر مخالف هم هستند در غیر این صورت، $\mathbf{k} = min(\mathbf{n1},\mathbf{n2})$.

آرایه‌های دوبعدی

- یک آرایه دوبعدی گاهی اوقات ماتریس نیز نامیده می شود.
- معمولا از دو اندیس مانند i و j به ترتیب برای اشاره به عنصر واقع روی سطر iام و ستون jام ماتریس استفاده می‌شود. در پایتون و C/C++ این اندیس‌ها از صفر شروع می‌شوند.

A

20	**10**	**17**	**95**	**63**
42	**23**	**87**	**51**	**12**

v

20	**10**	**17**	**95**	**63**	**42**	**23**	**87**	**51**	**12**

if $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{v} \in \mathbb{R}^{(m \times n)}$ and $\mathbf{v}[k] = \mathbf{A}[i,j]$ **then** $k = i \times n + j$

[1] Text Book, *Data Structures and Algorithms in Python* [Goodrich, Tamassia & Goldwasser 2013.
[2] http://xpzhang.me/teach/DS18_Fall