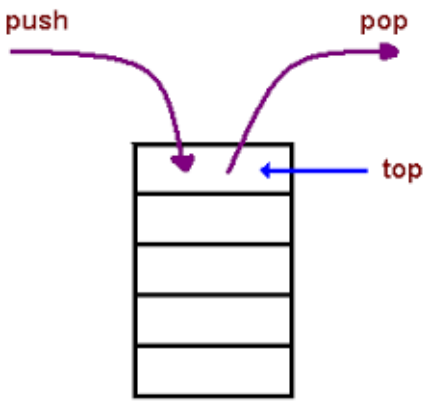
ساختار داده‌های ساده : پشته و صف

۱ پشته stack

پشته مجموعه‌ای مرتب از عناصر است که اعمال دسترسی، درج و حذف از آن تنها در یک سوی آن (اصطلاحا بالای پشته) انجام پذیر است. در واقع به یک عنصر داخل پشته نمی‌توان دسترسی پیدا کرد مگر اینکه عناصر بالای آن از پشته حذف شوند. علارغم این محدودیت، این ساختار داده بطور طبیعی در پیاده‌سازی بعضی از الگوریتم‌ها و حل مسائل پرکاربرد ظاهر می‌شود. در این درس چند نمونه از کاربرد پشته در طراحی الگوریتم‌ها را ذکر می‌کنیم.

در پایین سه عمل اصلی ویژه ساختار داده انتزاعی پشته ذکر شده است.

- push(x) عنصر x را در بالای پشته درج می‌کند.
- pop() عنصر بالای پشته را حذف و آن را برمی‌گرداند
- top() عنصر بالای پشته را برمی‌گرداند



۱.۱ پیاده‌سازی پشته

پشته را می‌توان با کمک گرفتن از آرایه و یک متغیر کمکی که تعداد عناصر موجود در پشته را نگه می‌دارد به راحتی پیاده‌سازی کرد (حتی می‌توان از اندیس اول آرایه به عنوان متغیر کمکی که اندازه پشته را نگه می‌دارد استفاده کرد.) در پایتون می‌توان از ساختار لیست برای پیاده‌سازی پشته استفاده کرد. در اینجا عمل push را می‌توان با متد append پیاده‌سازی کرد. در زیر پیاده‌سازی کلاس ArrayStack با استفاده از لیست پایتون نشان داده شده است. موقعی که پشته خالی باشد و اعمال pop و top فراخوانی شود یک وضعیت استثنائی (exception) گزارش می‌شود که مانند گزارش یک پیغام خطاست.

```
class Empty(Exception):
   pass

class ArrayStack:

 def __init__ (self):
  self.data = [ ] # nonpublic list instance

 def __len__(self):
  return len(self.data)

 def is_empty(self):
  return len(self.data) == 0

 def push(self, e):
  self.data.append(e) # new item stored at end of list

 def top(self):
  if self.is empty( ):
   raise Empty('Stack is empty')
  return self.data[-1] # the last item in the list

 def pop(self):
  if self.is empty( ):
   raise Empty( 'Stack is empty' )
  return self.data.pop( ) # remove last item from list
```

با توجه به کد بالا و بحثهایی که در دروس قبل در مورد زمان اجرای append کردیم، جدول زیر زمان اجرای اعمال اصلی پشته را نشان می‌دهد. (دقت کنید در تکنیک آرایه پویا، زمانی که عناصر از انتهای آرایه حذف می‌شوند، در صورتی که تعداد عناصر از حافظه اختصاصی خیلی کمتر شود، عمل فشرده سازی shrink انجام می‌شود و بخشی از حافظه آرایه آزاد می‌شود. این مانند حالت append است با این تفاوت که اینجا حافظه اختصاص یافته آزاد می‌شود. به همین دلیل زمان اجرای عمل pop بصورت سرشکنی $O(1)$ محاسبه شده است.)

Operation	Running Time
S.push(e)	$O(1)^*$
S.pop( )	$O(1)^*$
S.top( )	$O(1)$
S.is_empty( )	$O(1)$
len(S)	$O(1)$

*amortized

پشته با اندازه ثابت می‌توانیم از یک آرایه با طول ثابت برای پیاده‌سازی پشته استفاده کنیم. در اینجا فرض بر این است که تعداد عناصر پشته از یک مقدار MAX (که از قبل تعیین می‌شود) فراتر نمی‌رود. در این پیاده‌سازی زمان

حذف و اضافه به پشته $O(1)$ خواهد بود اما همانطور که گفته شد تعداد عناصر پشته نمی‌تواند از مقدار آستانه‌ای بیشتر باشد.

۲.۱ کاربرد پشته در طراحی الگوریتمها

مسئله تطبیق پرانتزها در متون و عبارات ریاضی، بطور معمول از پرانتزها (کروشه‌ها و آکولادها) استفاده می‌شود. چک کردن اینکه یک عبارت ریاضی (یا کد برنامه نویسی) درست پرانتزبندی شده است یکی از وظایف ویرایشگرهای کامپیوتری است. برای تطبیق پرانتزها و چک کردن درستی آنها میتوان از ساختار داده پشته استفاده کرد. کد زیر این کار را کرده است.

```
1  def is_matched(expr):
2    """Return True if all delimiters are properly match; False otherwise."""
3    lefty = '({['                                  # opening delimiters
4    righty = ')}]'                                 # respective closing delims
5    S = ArrayStack( )
6    for c in expr:
7      if c in lefty:
8        S.push(c)                                  # push left delimiter on stack
9      elif c in righty:
10       if S.is_empty( ):
11         return False                             # nothing to match with
12       if righty.index(c) != lefty.index(S.pop( )):
13         return False                             # mismatched
14   return S.is_empty( )                           # were all symbols matched?
```

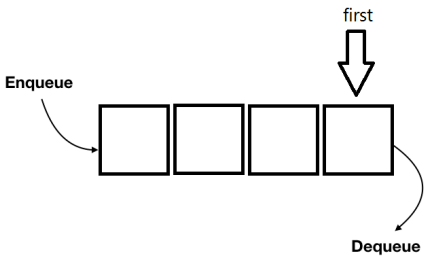
Code Fragment 6.4: Function for matching delimiters in an arithmetic expression.

از کاربردهای دیگر پشته، می‌توان به مسئله تطبیق تگهای HTML اشاره کرد (به صفحات ۲۳۷ و ۲۳۸ کتاب مراجعه شود.)

۲ صف Queue

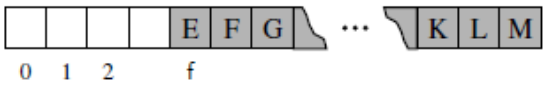
صف همانند پشته مجموعه‌ای مرتب از اشیا است با این تفاوت که حذف و اضافه در آن بر اساس اصل FIFO (آنکه زود آمده زودتر می‌رود) استوار است. در پایین سه عمل اصلی ویژه ساختار داده انتزاعی صف ذکر شده است.

- enqueue(x) عنصر x را به انتهای صف اضافه می‌شود.
- dequeue() عنصر اول صف را برمی‌گرداند و آن را صف خارج می‌کند.
- first() عنصر اول صف را برمی‌گرداند.



۱.۲ پیاده‌سازی صف

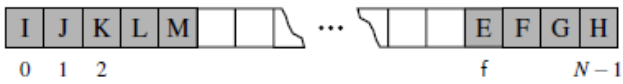
برای پیاده‌سازی صف در پایتون (همانند پشته) می‌توان از ساختار لیست استفاده کرد. می‌توان با استفاده از دستور append عمل enqueue را پیاده‌سازی کرد و از دستور pop(۰) برای پیاده‌سازی dequeue استفاده کرد (دستور pop(۰) اولین عنصر لیست را حذف می‌کند.) اما این پیاده‌سازی چندان کارا نیست. حذف عنصر اول لیست بسیار پرهزینه است و هر بار انجام آن مستلزم شیفت عناصر داخل لیست است. یک راه حل این است که عناصر را واقعا از لیست حذف نکنیم و از یک متغیر اضافه (که شروع صف را در لیست نشان می‌دهد) استفاده کنیم. به شکل زیر توجه کنید. اندیس f در لیست به محل اولین عنصر لیست اشاره می‌کند.



اما این راه حل مشکل دیگری ایجاد میکند. طول لیست با ورود عناصر به صف مدام افزایش می‌یابد اما خروج از صف باعث آزادسازی حافظه و کاهش طول لیست نمی‌شود. برای مثال موقعیت می‌تواند جوری شود که طول لیست به ۱۰۰۰۰ رسیده اما تنها یک عنصر در صف وجود دارد. برای رفع این مشکل می‌توان از تمهید لیست چرخشی استفاده کرد. در این راه حل، در ابتدای کار یک مقدار آستانه MAX برای بیشترین تعداد عناصر حاظر در صف درنظر گرفته می‌شود و سپس لیستی خالی با طول MAX ایجاد می‌شود.

```
MAX = 100000          # for example
data = [None]*MAX
```

متغیر f کماکان به اندیس اول صف اشاره می‌کند.



وقتی می‌خواهیم عنصر اول صف را خارج کنیم کافی است مقدار f را یک واحد به سمت راست (بصورت گردشی) با استفاده از دستور زیر شیفت دهیم:

```
f = (f + 1) % MAX
```

اگر متغیر size تعداد عناصر در صف را در خود جای داده باشد برای پیدا کردن اندیس آخر صف می‌تواند از دستور زیر استفاه کرد.

```
end = (f + size) % MAX
```

در پیاده‌سازی لیست چرخشی برای صف، زمان اجرای dequeue و enqueue برابر با $O(1)$ خواهد بود اما این محدودیت را دارد که تعداد عناصر صف نمی‌تواند از یک مقداری بیشتر باشد. برای رفع این محدودیت، هر بار که لیست پر شد و تعداد عناصر به مقدار آستانه MAX رسید می‌توان اندازه لیست را دو برابر کرد (که این البته خود عملی زمانبر است.)

کلاس ArrayQueue در صفحات ۲۴۳ و ۲۴۴ کتاب مرجع ساختار داده صف را با استفاده از لیست پایتون (با حذف و اضافه چرخشی) پیاده‌سازی کرده است. در این پیاده‌سازی هر وقت لیست پر شد، اندازه آن با استفاده از متد resize دو برابر می‌شود.

```
1  class ArrayQueue:
2    """FIFO queue implementation using a Python list as underlying storage."""
3    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues
4
5    def __init__(self):
6      """Create an empty queue."""
7      self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
8      self._size = 0
9      self._front = 0
```

```
28   def dequeue(self):
29     """Remove and return the first element of the queue (i.e., FIFO).
30
31     Raise Empty exception if the queue is empty.
32     """
33     if self.is_empty():
34       raise Empty('Queue is empty')
35     answer = self._data[self._front]
36     self._data[self._front] = None                  # help garbage collection
37     self._front = (self._front + 1) % len(self._data)
38     self._size -= 1
39     return answer
```

```
40   def enqueue(self, e):
41     """Add an element to the back of queue."""
42     if self._size == len(self._data):
43       self._resize(2 * len(self.data))          # double the array size
44     avail = (self._front + self._size) % len(self._data)
45     self._data[avail] = e
46     self._size += 1
47
48   def _resize(self, cap):                       # we assume cap >= len(self)
49     """Resize to a new list of capacity >= len(self)."""
50     old = self._data                            # keep track of existing list
51     self._data = [None] * cap                   # allocate list with new capacity
52     walk = self._front
53     for k in range(self._size):                 # only consider existing elements
54       self._data[k] = old[walk]                 # intentionally shift indices
55       walk = (1 + walk) % len(old)              # use old size as modulus
56     self._front = 0                             # front has been realigned
```