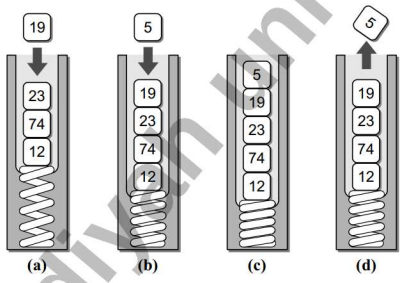
Modul 8

Stacks and Queues

Di modul ini kita akan belajar struktur *stack* (tumpukan) dan *queue* (antrian).

8.1 Pengertian *Stack*



Gambar 8.1: Tampilan abstrak sebuah stack: (a) mendorong (*push*) angka 19; (b) mendorong angka 5; (c) hasil setelah angka 19 dan 5 didorong; dan (d) mengeluarkan (*pop*) angka paling atas.

Stack, diterjemahkan menjadi *tumpukan*, digunakan untuk menyimpan data sedemikian hingga item terakhir yang disisipkan adalah item pertama. Stack digunakan untuk mengimplementasikan protokol *last-in-first-out* (LIFO). Stack adalah sebuah struktur data linear di mana item baru ditambahkan dan diambil dari ujung yang sama, yakni puncak tumpukan (***top** of the stack*). Di bagian paling bawah adalah *base* atau dasar-nya.

Konsep stack sangat umum dalam literatur ilmu komputer dan banyak digunakan di berbagai penyelesaian masalah. Stack juga ditemui dalam kehidupan sehari-hari[1]. Perhatikan gambaran abstrak stack di Gambar 8.1[2].

Dalam definisi yang lebih formal, sebuah stack adalah sebuah struktur data yang menyimpan koleksi linear yang mempunyai akses terbatas pada urutan terakhir-masuk-keluar-pertama.

[1]Seperti: tumpukan piring, tumpukan buku di kardus, atau parkir mobil dengan satu gerbang saja di mana mobil yang terakhir masuk adalah yang pertama keluar.

[2]Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.

Menambah dan mengambil item dibatasi pada satu ujung saja, yakni *puncak* tumpukan (*top* of the stack). Stack kosong (*empty stack*) adalah stack yang tidak mempunyai item.

8.2 *Features* dan *properties* sebuah stack

Bayangkan kamu mempunyai sebuah kardus kosong *Indomie*® dan akan menyimpan buku-buku ukuran A4 dengan posisi tidur di situ. Apa saja yang bisa kita tanyakan, dan kita lakukan, pada tumpukan itu? Setidaknya:

1. Apakah kosong?
2. Apakah penuh?
3. Berapa banyakkah buku di tumpukan itu?
4. Taruh satu buku di tumpukan itu
5. Ambil satu buku dari tumpukan itu
6. Intip, buku apakah yang paling atas

Dari *requirements* ini kita bisa mulai merancang beberapa perintah yang harus ada di class Stack yang akan dibuat.

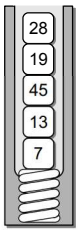
- `Stack()`: membuat stack baru yang kosong
- `isEmpty()`: mengembalikan nilai boolean yang menunjukkan apakah stack itu kosong.
- `length()`: mengembalikan banyaknya item di dalam stack
- `pop()`: mengembalikan nilai dari item yang paling atas dan menghapusnya, jika stack-nya tidak kosong.
- `peek()`: mengembalikan nilai dari item yang paling atas, tanpa menghapusnya.
- `push( item )`: menambah `item` ke puncak stack.

Bisa kamu lihat bahwa *untuk saat ini* kita tidak mengimplementasikan perintah yang berasosiasi dengan "Apakah penuh". Jika diperlukan hal ini bisa dibuat, namun secara default kita bisa menumpuk sebanyak-banyaknya ke tumpukan yang dibentuk dengan list Python[3].

Berikut ini adalah contoh pemanggilannya.

```
PROMPT = "Masukkan bilangan positif (<0 untuk mengakhiri):"
myStack = Stack()               # Membuat stack baru (program di halaman berikut)
value = int(input( PROMPT ))
while value >= 0 :              # Memasukkan satu per satu
    myStack.push( value )
    value = int(input( PROMPT ))
while not myStack.isEmpty() :   # Mengeluarkan satu per satu
    value = myStack.pop()
    print( value )
```

[3] Jika kamu memakai bahasa C, kamu bisa menumpuk sebanyak-banyaknya di suatu stack dengan memakai struktur data linked-list. Sesungguhnya list di bahasa Python diimplementasikan dengan doubly-linked-list di bahasa C (yakni bahasa yang digunakan untuk membuat bahasa Python)



Gambar 8.2: Hasil stack setelah mengeksekusi contoh.

Ketika user memasukkan (*push*), secara berurutan, nilai 7, 13, 45, 19, 28, -1, maka kita mempunyai stack yang sepeti ditunjukkan di Gambar 8.2. Perhatikan bahwa kita mempunyai nilai yang paling dasar adalah nilai yang pertama kita masukkan, dan yang paling atas adalah yang paling terakhir kita masukkan. Kalau kita mengeluarkan (*pop*), maka urutannya adalah dari yang paling atas. Lihat Gambar 8.2[4].

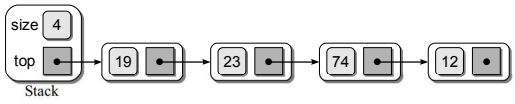
8.3 Implementasi Stack

Stack dapat diimplementasikan dengan beberapa cara. Dua yang paling umum adalah memakai list dan linked list.

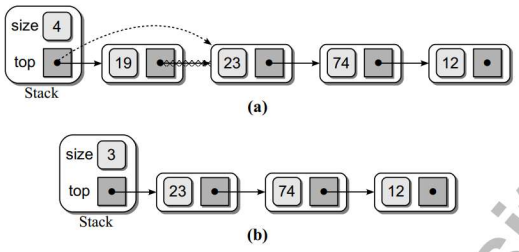
Menggunakan list

Berikut ini implementasi stack menggunakan list Python.

```
class  Stack(object):
    def __init__(self):         # Membuat stack kosong.
        self.items = []         # List untuk menyimpan stack.

    def isEmpty(self):          # Mengembalikan True kalau kosong,
        return len(self)==0     #   selain itu False

    def __len__(self):          # Mengembalikan banyaknya item di stack.
        return len(self.items) #

    def peek(self):             # Mengembalikan nilai posisi atas tanpa menghapus.
        assert not self.isEmpty(), "Stack kosong. Tidak bisa diintip"
        return self.items[-1]

    def pop(self):              # Mengembalikan nilai posisi atas lalu menghapus.
        assert not self.isEmpty(), "Stack kosong. Tidak bisa di-pop"
        return self.items.pop()

    def push(self, data):       # Mendorong item baru ke stack.
        self.items.append(data)
```



Gambar 8.3: Contoh object stack yang diimplementasikan dengan linked-list.



Gambar 8.4: Menge-pop sebuah item dari stack. (a) modifikasi link yang diperlukan, (b) hasil setelah mengeluarkan item yang paling atas.

Menggunakan linked-list

Berikut ini adalah implementasi stack menggunakan linked-list.

```
1  class StackLL(object):
2
3      def __init__(self):
4          self.top = None
5          self.size = 0
6
7      def isEmpty(self):
8          return self.top is None
9
10     def __len__(self):
11         return self.size
12
13     def peek(self):
14         assert not self.isEmpty(), "Tidak bisa diintip. Stack kosong."
15         return self.top.item
16
17     def pop(self):
18         assert not self.isEmpty(), "Tidak bisa pop dari stack kosong."
19         node = self.top
20         self.top = self.top.next
21         self.size -= 1
22         return node.item
23
24     def push(self, data):
25         self.top =  _StackNode(data, self.top)
26         self.size += 1
27
28 class _StackNode(object):           # Ini adalah kelas privat
29     def __init__(self, data, link): #  untuk menyimpan data. Dia
30         self.item = data            #  dipanggil oleh class StackLL
```

[4]Gambar-gambar di bab ini diambil dari Rance D. Necaise, *Data Structures and Algorithms Using Python*, John Wiley and Sons, 2011.

```
31          self.next = link            #  di atas.
```

Perhatikan bahwa materi linked-list-nya dibuat dengan class yang berbeda, yakni `_StackNode`. Gambar 8.3 dan 8.4 memperlihatkan contoh operasinya.

8.4 Contoh program

Stack digunakan secara sangat luas dalam *computing*. Interupsi program dan sifat multi-tasking komputer modern bergantung salah satunya pada konsep stack ini. Kita belum akan menyentuh materi itu di matakuliah ini, namun kamu bisa 'merasakan' mekanisme algoritmanya melalui contoh-contoh program.

Mengubah bilangan desimal ke biner

Program di bawah ini mengubah representasi suatu bilangan, dari basis sepuluh ke basis dua, yang penjelasannya telah kamu dapatkan di kelas.

```
1  def cetakBiner(d):
2      f = Stack()          # Atau: f =  StackLL()
3      if d==0: f.push(0);
4      while d !=0:
5          sisa = d%2
6          d = d//2
7          f.push(sisa)
8      st = ""
9      for i in range(len(f)):
10         st = st + str(f.pop())
11     return st
```

Berikut ini adalah contoh pemanggilan dan keluarannya.

```
>>> cetakBiner(11)
'1011'
>>> cetakBiner(53)
'110101'
```

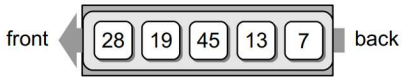
8.5 Pengertian *Queue*

Queue[5] atau antrian adalah sebuah struktur data yang memodelkan fenomena antrian. Seperti sudah kamu duga, antrian memiliki sifat "yang duluan datang adalah yang duluan keluar", *First in First out*, disingkat FIFO. Lihat Gambar 8.5. Apa saja yang bisa kita *query* ke sebuah antrian?

- Buatlah sebuah antrian. Ini akan diimplementasikan dengan class `Queue()`.
- Apakah antriannya kosong? Ini akan kita diimplementasikan dengan metode `isEmpty()`.
- Berapa panjangkah antriannya? Ini akan diimplementasikan dengan metode `__len__()`.

[5] Dibaca [kyu]

- Masukkan item *xyz* ini ke antrian. Ini akan diimplementasikan dengan metode `enqueue(data)`.
- Ambil item *pqr* ini dari antrian. Ini akan diimplementasikan dengan metode `dequeue()`.



Gambar 8.5: Tampilan abstrak sebuah sebuah antrian. Bagian depan (front) adalah yang terlebih dulu datang, bagian belakang (back) adalah yang datang belakangan. Pada antrian yang biasa, yang terlebih dulu datang akan keluar terlebih dahulu.

Dari sini bisa kita tulis kode untuk membuat antrian seperti diilustrasikan pada Gambar 8.5 di atas seperti di bawah ini.

```
Q = Queue()
Q.enqueue( 28 )
Q.enqueue( 19 )
Q.enqueue( 45 )
Q.enqueue( 13 )
Q.enqueue( 7 )
```

Setelah membuat object baru, kita lalu mengantrikan lima angka dengan urutan seperti urutan tampilnya angka-angka itu di antrian. Kita lalu bisa mengambil angka atau menambahkan angka ke antrian itu. Gambar 8.6 mengilustrasikan beberapa operasi tambahan pada contoh antrian di atas.

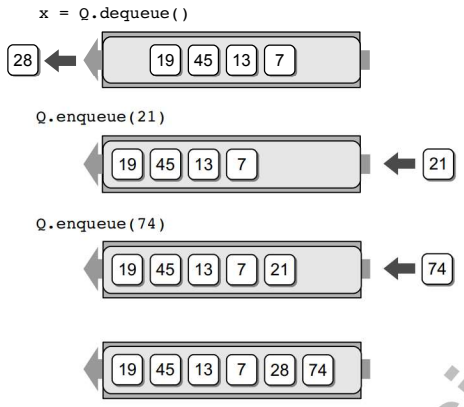
8.6 Implementasi

Implementasi antrian paling sederhana adalah memakai list di Python, seperti kode di bawah ini.

```
1  class Queue(object):
2      def __init__(self):
3          self.qlist = []
4
5      def isEmpty(self):
6          return len(self) == 0
7
8      def __len__(self):
9          return len(self.qlist)
10
11     def enqueue( self, data ):
12         self.qlist.append(data)
13
14     def dequeue(self):
15         assert not self.isEmpty(), "Antrian sedang kosong."
16         return self.qlist.pop(0)
```

Cobalah beberapa perintah untuk memeriksa validitas program di atas.

```
## Ini melanjutkan kode di atas
Q.dequeue()  # Perhatikan urutan hasilnya.
Q.dequeue()  #  Apakah sesuai dengan urutan masuknya?
Q.dequeue()
```



Gambar 8.6: Sebuah antrian saat kepadanya dilakukan beberapa operasi tambahan. Mengambil 28 dari antrian, mengantrikan 21, lalu mengantrikan 74.

```
Q.dequeue()
Q.dequeue()
Q.dequeue()  # Apakah ada pesan error?
Q.enqueue(98)
Q.enqueue(54)
Q.dequeue()
```

8.7 *Priority Queues*

Priority queue adalah antrian dengan prioritas. Berikut ini adalah perbedaan antara antrian biasa dengan antrian dengan prioritas:

- Ketika suatu item akan dimasukkan ke antrian (`enqueue`), item ini disertai dengan suatu penanda prioritas. Ini biasanya berupa bilangan bulat dari 0 sampai suatu bilangan tertentu yang dipilih sesuai aplikasi (misalnya kita memilih 4, yang berarti ada 5 prioritas dengan 0 sebagai prioritas tertinggi).
- Ketika suatu item akan diambil dari antrian (`dequeue`), maka yang diambil adalah yang prioritasnya tertinggi, dengan tidak memperdulikan urutan kedatangan. Jadi misal kita mempunyai lima item di suatu antrian dengan semuanya mempunyai prioritas 4, dan lalu datang item baru (`enqueue`) dengan prioritas 2, maka ketika "dipanggil" (dengan `dequeue`), yang keluar terlebih dahulu adalah item dengan prioritas 2 tadi.
- Jika ada dua item atau lebih dengan prioritas yang sama, yang didahulukan keluar adalah yang terlebih dahulu masuk. Prinsip FIFO tetap berlaku pada item-item dengan prioritas sama.

Berikut ini contoh pemanggilannya.

```
S = PriorityQueue()
S.enqueue("Jeruk", 4)
S.enqueue("Tomat", 2)
```

```
S.enqueue("Mangga", 0)
S.enqueue("Duku", 5)
S.enqueue("Pepaya", 2)
S.dequeue()          # Akan mengeluarkan "Mangga" karena prioritasnya tertinggi.
S.dequeue()          # Akan mengeluarkan "Tomat".
S.dequeue()          # Kalau ini "Pepaya".
```

Untuk mengimplementasikan *priority queue* ini, kita membuat suatu kelas baru yang dipakai untuk menyatukan item dengan prioritas yang dilekatkan padanya. Berikut ini sebagian dari implementasi *priority queue* ini.

```
class PriorityQueue(object):

    def __init__(self):
        self.qlist = []

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

    def isEmpty(self):
        return len(self) == 0

    def enqueue(self, data, priority):
        entry = _PriorityQEntry( item, priority ) # Memanggil object _PriorityQEntry
        self.qlist.append(entry)

    def dequeue(self):
        pass # .... lihat Soal-soal untuk Mahasiswa.
```

Berikut ini adalah class untuk menyimpan item beserta prioritasnya.

```
class _PriorityQEntry(object):
    def __init__(self, data, priority):
        self.item = data
        self.priority = priority
```

8.8 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal di bawah, kerjakan dulu latihan dan percobaan di atas.

Stacks

1. Buatlah program untuk mengubah representasi suatu bilangan dari basis sepuluh ke basis dua. Berikut ini contoh pemanggilannya.

```
>>> cetakHexa(12)
'C'
>>> cetakHexa(31)
'1F'
>>> cetakHexa(229)
'E5'
>>> cetakHexa(255)
'FF'
>>> cetakHexa(31519)
```

```
'7B1F'
```

Perhatikan bahwa sisa pembagian tidak hanya 0 dan 1, namun bisa 0 sampai 9 dan bahkan 10, 11, 12, 13, 14, 15. Kamu harus memetakan angka-angka yang lebih dari 9 ke lambang A, B, C, D, E, dan F.

2. Eksekusi program berikut dengan pensil dan kertas, dan tunjukkan isi stack-nya pada setiap langkah.

```
nilai = Stack()
for i in range(16 ):
    if i % 3 == 0:
        nilai.push( i )
```

3. Eksekusi program berikut dengan pensil dan kertas, dan tunjukkan isi stack-nya pada setiap langkah.

```
nilai = Stack()
for i in range( 16 ) :
    if i % 3 == 0 :
        nilai.push( i )
    elif i % 4 == 0 :
        nilai.pop()
```

Queues

4. Tulis dua metode berikut ke class `Queue` dan class `PriorityQueue` di atas

   - Metode untuk mengetahui item yang paling depan tanpa menghapusnya

     ```
     def getFrontMost(self):
         ## Tulis perintahnya di sini
     ```

   - Metode untuk mengetahui item yang paling belakang tanpa menghapusnya

     ```
     def getRearMost(self):
         ## Tulis perintahnya di sini
     ```

5. Pada class `PriorityQueue` di atas, metode `dequeue()` belum diimplementasikan. Tulislah metode dequeue() ini dengan memperhatikan syarat-syarat seperti yang telah dicantumkan di halaman 81.