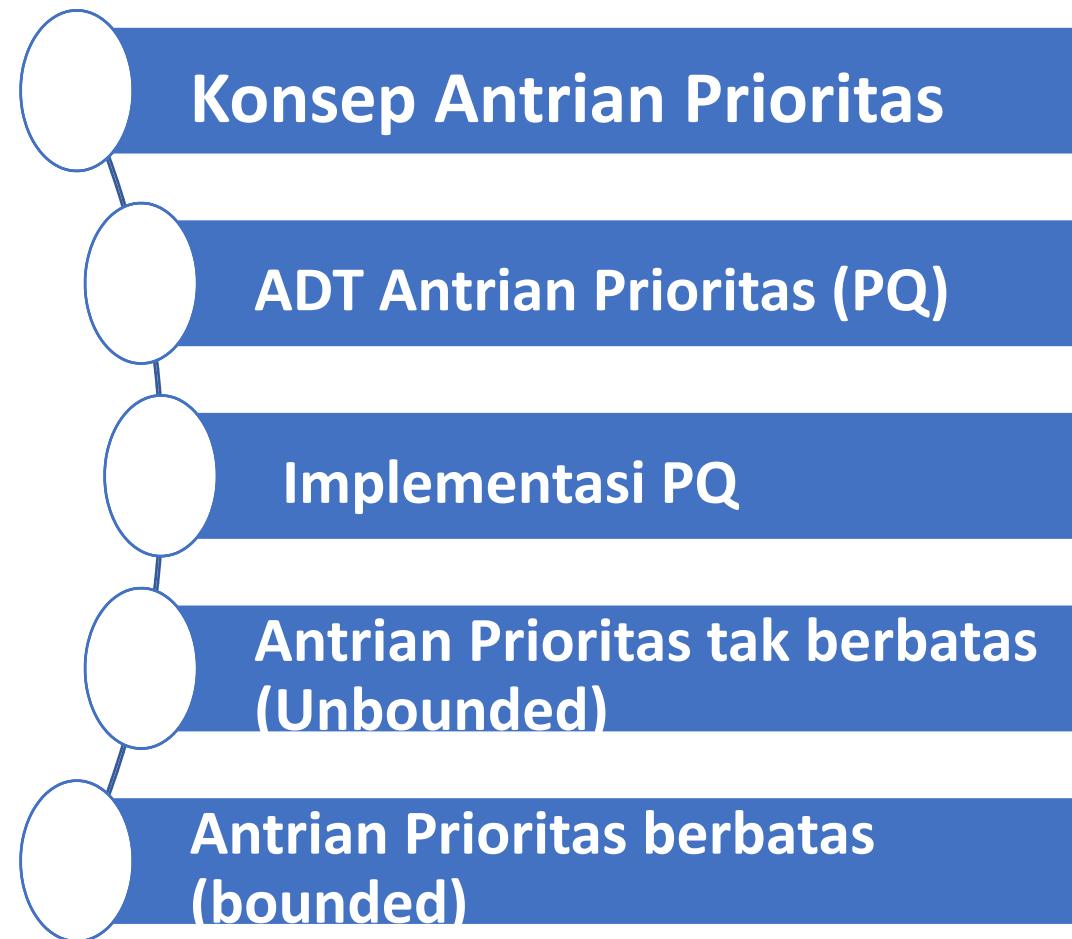


# **STRUKTUR DATA (PYTHON)**

## **“Antrian Prioritas”**

[@SUARGA] | [Pertemuan 12]

# OutLine





# Konsep Antrian Prioritas

- Antrian (queue) biasanya memiliki kebijakan FIFO, yang pertama datang akan dilayani terlebih dahulu, namun
- Antrian Prioritas (Priority Queue) adalah antrian yang dilengkapi dengan skala prioritas, jadi selain menurut urutan, prioritas juga mendapat perhatian utama.
- contoh: antrian naik pesawat terbang, biasa orang-orang penting mendapat prioritas utama mengalahkan orang yang sudah antri
- Selanjutnya data akan diurutkan bukan menurut kedatangan-nya (bukan FIFO) tetapi menurut urutan prioritas atau kunci k, dengan urutan ascending.

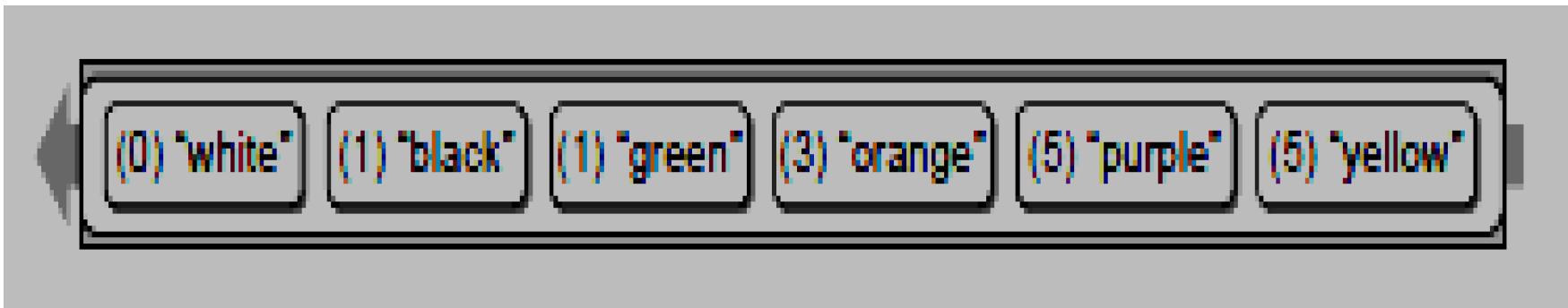
# Priority Queue ADT

Beberapa fungsi ADT dari antrian prioritas :

- **add(k, v)** : menambahkan item v dengan prioritas k
- **min()** : menampilkan data (k,v) dimana nilai k terkecil
- **remove\_min()** : memberi pelayanan pada data dengan k minimum
- **is\_empty()** : memeriksa apakah antrian kosong
- **len(P)** : menghitung banyaknya item dalam antrian prioritas P

Catatan: antrian diberi angka 0 sd n, dimana 0 adalah prioritas tertinggi.

- **Contoh operasi dari fungsi-fungsi ADT :**
- P=PQ () #inisialisasi Priority Queue (PQ)
- P.add(5, 'purple') #data 'purple' dengan prioritas 5
- P.add(1, 'black') #data 'black' dengan prioritas 1
- P.add(3, 'orange') # dan seterusnya ...
- P.add(0, 'white')
- P.add(1, 'green')
- P.add(5, 'yellow')
- P.min() #mencari k min, hasilnya (0, 'white')
- len(P) #banyaknya item, hasilnya 6



Data akan disimpan dengan urutan prioritas, misalnya ‘white’ walaupun dimasukkan belakangan tetapi di-simpan didepan karena prioritas-nya 0, adalah prioritas tertinggi. Apabila prioritas sama maka data disimpan sesuai urutannya dimasukkan, misalnya (1, ‘black’) lebih dulu dari (1,’green’), karena (1, black) lebih dahulu dimasukkan.

- P.remove\_min()
  - #mengeluarkan (0, 'white') dari antrian
  - len(P) # panjang = 5
- 
- #mengambil data secara ber-urut
  - while not P.is\_empty() :
    - item = P.remove\_min()
    - print( item )
  - 
  - Hasilnya:
  - black
  - green
  - orange
  - purple
  - yellow

# Model PriorityQueue (PQ)

- Item / anggota antrian prioritas (PQ) memiliki dua elemen penting:
  - **key** : kunci atau prioritas
  - **value** : nilai data
- Maka basis class (PQBase class) dari antrian prioritas harus memuat kedua elemen tersebut

# class PQBase

```
class PQBase:  
    # kelas abstrak untuk priority queue  
  
    class _Item:  
        __slots__ = '_key' , '_value'  
  
        def __init__(self, k, v):  
            self._key = k  
            self._value = v  
  
        def __lt__(self, other):  
            #membandingkan prioritas k  
            return self._key < other._key  
  
    def is_empty(self):    # apakah kosong ?  
        return len(self) == 0
```

# Implementasi PQ tak-berurutan

- Pada implementasi tak-berurutan (UnsortedPQ) semua node baru akan dimasukkan seperti pada proses enqueue struktur antrian biasa (FIFO), dimana node yang baru ditambahkan dari belakang antrian, dengan demikian **proses penambahan node menjadi cepat**, karena tidak perlu menelusuri antrian untuk mencari tempat yang sesuai.
- Namun **proses untuk mencari suatu kunci tertentu atau proses untuk mengambil satu node untuk dilayani (remove\_min) akan memerlukan waktu lebih lama**, karena node tidak ber-urut menurut kunci prioritas minimum maka antrian harus di-telusuri untuk mendapatkan posisi prioritas minimum.

# UnsortedPQ Implementation

```
#UnsortedPQ.py == @Suarga
from PQBase import PQBase
from PositionalList import PositionalList

class UnsortedPQ(PQBase):

    def _find_min(self):
        # mencari node dengan kode prioritas minimum
        if self.is_empty():
            raise Exception('Antrian kosong')
        small = self._data.first()
        walk = self._data.after(small)
        while walk is not None:
            if walk.element() < small.element():
                small = walk
            walk = self._data.after(walk)
        return small

    def __init__(self):
        # inisialisasi dengan positional-list
        self._data = PositionalList()
```

```
def is_empty(self):
    return (len(self._data) == 0)

def __len__(self):
    # memberikan panjang list
    return len(self._data)

30 def add(self, key, value):
    # menambah data
    self._data.add_last(self._Item(key,value))

def min(self):
    # mencari data minimum
    p = self._find_min()
    item = p.element()
    return (item._key, item._value)

40 def remove_min(self):
    # membuang data minimum
    p = self._find_min()
    item = self._data.delete(p)
    return (item._key, item._value)
```

```
def traverse(self):
    walk = self._data.first()
    while walk is not None:
        item = walk.element()
        print(item._key, item._value)
        walk = self._data.after(walk)

50

def testUSPQ():
    USPQ = UnsortedPQ()
    USPQ.add(3, 'red')
    USPQ.add(5, 'purple')
    USPQ.add(0, 'white')
    USPQ.add(4, 'green')
    USPQ.add(2, 'blue')
    USPQ.add(1, 'yellow')
    n = USPQ.__len__()
    print('Jumlah elemen = ', n)
    print('Isi PQ : ')
    USPQ.traverse()
    (mink, minv) = USPQ.min()
    print('elemen minimum key = ', mink, ' isi = ', minv)
    (mink, minv) = USPQ.remove_min()
    print('elemen yg diambil key = ', mink, ' isi = ', minv)
    print('Isi PQ setelah remove : ')
    USPQ.traverse()

70

if __name__ == '__main__':
    testUSPQ()
```

# Hasil Test

Python Interpreter

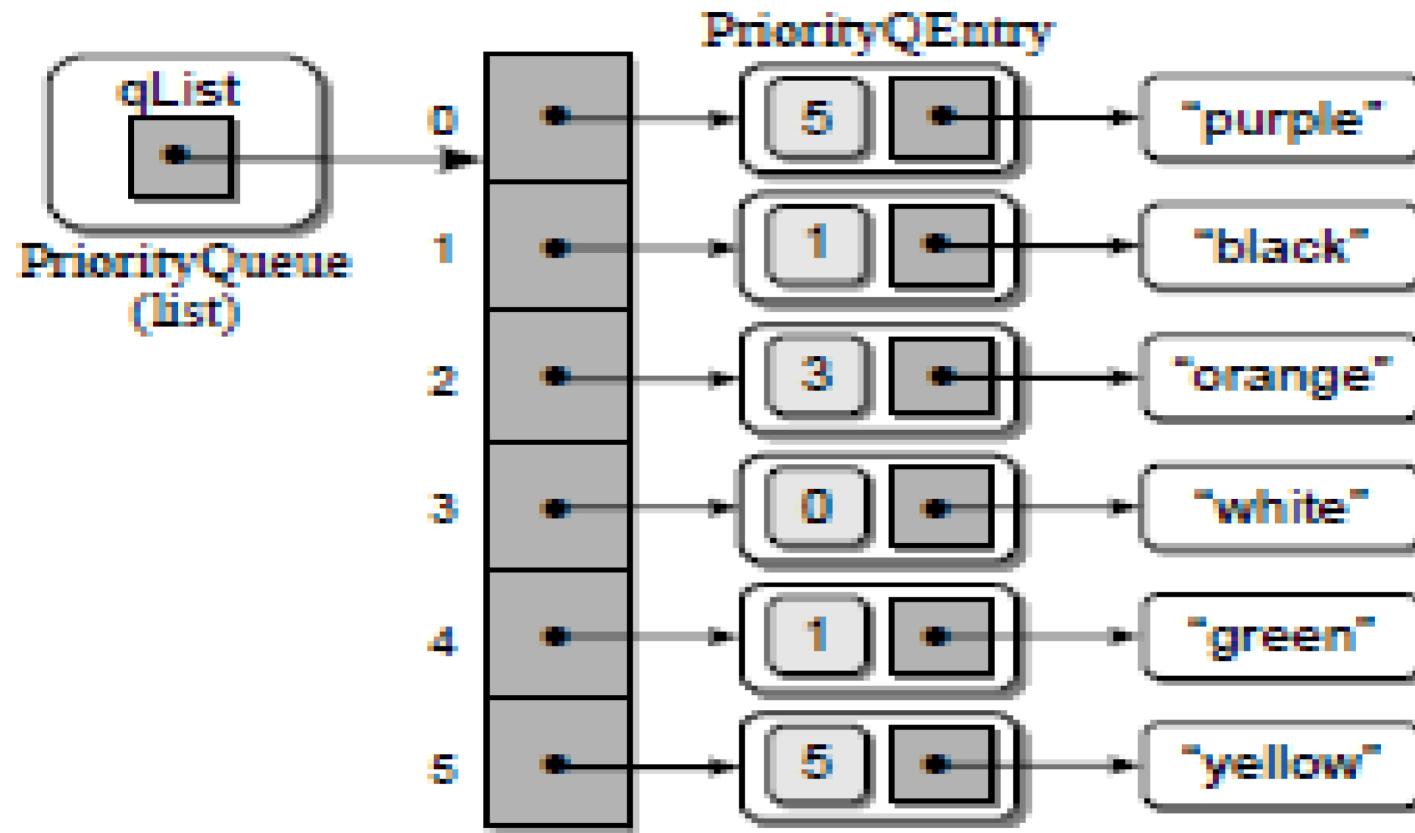
```
>>>
*** Remote Interpreter Reinitialized ***
Jumlah elemen =  6
Isi PQ :
3 red
5 purple
0 white
4 green
2 blue
1 yellow
elemen minimum key =  0  isi =  white
elemen yg diambil key =  0  isi =  white
Isi PQ setelah remove :
3 red
5 purple
4 green
2 blue
1 yellow
>>> |
```

# UnboundedPQ

- Antrian Prioritas tak berbatas (unbounded priority queue), dapat diciptakan dengan memanfaatkan Python struktur data internal Python yaitu List().
- Menggunakan struktur List() membuat implementasi ini tidak dibatasi berapa banyak anggota / item yang boleh disimpan
- Namun PQ ini bersifat unsortedPQ

# Contoh: PQ Python List Implementation

Andaikan suatu Priority queue akan dibentuk dengan enam item didalamnya seperti pada gambar berikut ini.



**Maka perintah untuk membentuk antrian prioritas ini bisa dilakukan sebagai berikut:**

- from PriorityQueue import \*
- #testPriorityQueue
- Q=PriorityQueue()
- Q.add('purple',5)
- Q.add('black',1)
- Q.add('orange',3)
- Q.add('white',0)
- Q.add('green',5)
- Q.add('yellow',1)
- print('size = ',Q.\_\_len\_\_())
- print('Items in order: ')
- while not Q.isEmpty():
- item = Q.remove\_item()
- print(item)

## **Hasil percobaan diatas menghasilkan berikut ini.**

- size = 6
  - Items in order:
  - white
  - black
  - yellow
  - orange
  - purple
  - green
- >>>

```
1 # Implementasi ADT antrian prioritas tdk-berbatas (unbounded)
# memakai Python list, dimana item baru ditambahkan di belakang.
#
2 class PriorityQueue :
3     # mencipta unbounded priority queue kosong.
4         def __init__( self ):
5             self._qList = list()
6
7     # memeriksa apakah PQ kosong?
8         def isEmpty( self ):
9             return len( self ) == 0
10
11     # memberikan jumlah item dalam PQ
12         def __len__( self ):
13             return len( self._qList )
14
15     # menambah entri baru ke dalam PQ
16         def add( self, item, priority ):
17             # Create a new instance of the storage class
18             # and append it to the list.
19                 entry = _PriorityQEntry( item, priority )
20                 self._qList.append( entry )
21                 #self._qList.insert(priority, entry)
```

```
# mencari item dengan prioritas tertinggi (key minimum)
def min( self ) :
    assert not self.isEmpty(),"Cannot dequeue from an empty queue."
    # cari entri dengan prioritas tertinggi.
    ix=0
    highest = self._qList[0].priority
    size = len(self._qList)
    for i in range( size ) :
        # periksa apakah yang ke-i tertinggi
        # priority (smaller integer).
        if self._qList[i].priority < highest :
            highest = self._qList[i].priority
            ix=i
    entry = self._qList[ix]
    # tampilkan yang tertinggi prioritas-nya.
    return (entry.priority, entry.item)

# Removes and returns the first item in the queue.
def remove_item( self ) :
    assert not self.isEmpty(),"Cannot dequeue from an empty queue."
    # cari entri dengan prioritas tertinggi.
```

```
# cari entri dengan prioritas tertinggi.  
ix=0  
highest = self._qList[0].priority  
size = len(self._qList)  
for i in range( size ) :  
    # periksa apakah yang ke-i tertinggi  
    # priority (smaller integer).  
    if self._qList[i].priority < highest :  
        highest = self._qList[i].priority  
        ix=i  
  
    # ambil/keluarkan yang tertinggi prioritas-nya.  
entry = self._qList.pop( ix )  
return (entry.priority, entry.item)  
  
def traverse(self):  
    #menampilkan isi PQ  
    n = len(self._qList)  
    for i in range(n) :  
        print (self._qList[i].priority, self._qList[i].item)
```

```
· # Private storage class for associating queue
· # items with their priority.
70 · class _PriorityQEntry( object ):
·     def __init__( self, item, priority ):
·         self.item = item
·         self.priority = priority
74
·     def testPQ():
·         Q = PriorityQueue()
·         Q.add('purple',5)
·         Q.add('black',1)
·         Q.add('orange',3)
·         Q.add('white',0)
·         Q.add('green',1)
·         Q.add('yellow',5)
·         n = Q.__len__()
·         print('Jumlah elemen = ',n)
·         print('Isi PQ : ')
·         Q.traverse()
·         (mink,minv) = Q.min()
·         print('elemen minimum key = ',mink,' isi = ',minv)
```

```
    (mink,minv) = Q.min()
    print('elemen minimum key = ',mink,' isi = ',minv)
89   (mink,minv) = Q.remove_item()
90   print('elemen yg diambil key = ',mink,' isi = ',minv)
    print('Isi PQ setelah remove : ')
    Q.traverse()
    print('elemen sisa urut menurut key : ')
    print('Items in order: ')
    while not Q.isEmpty():
        (key,item) = Q.remove_item()
        print(key,item)

100 if __name__ == '__main__':
     testPQ()
```

# Hasil Test

Python Interpreter

```
*** Remote Interpreter Reinitialized ***
Jumlah elemen =  6
Isi PQ :
5 purple
1 black
3 orange
0 white
1 green
5 yellow
elemen minimum key =  0  isi =  white
elemen yg diambil key =  0  isi =  white
Isi PQ setelah remove :
5 purple
1 black
3 orange
1 green
5 yellow
elemen sisa urut menurut key :
Items in order:
1 black
1 green
3 orange
5 purple
5 yellow
>>> |
```

# PQ berbatas (Bounded PQ)

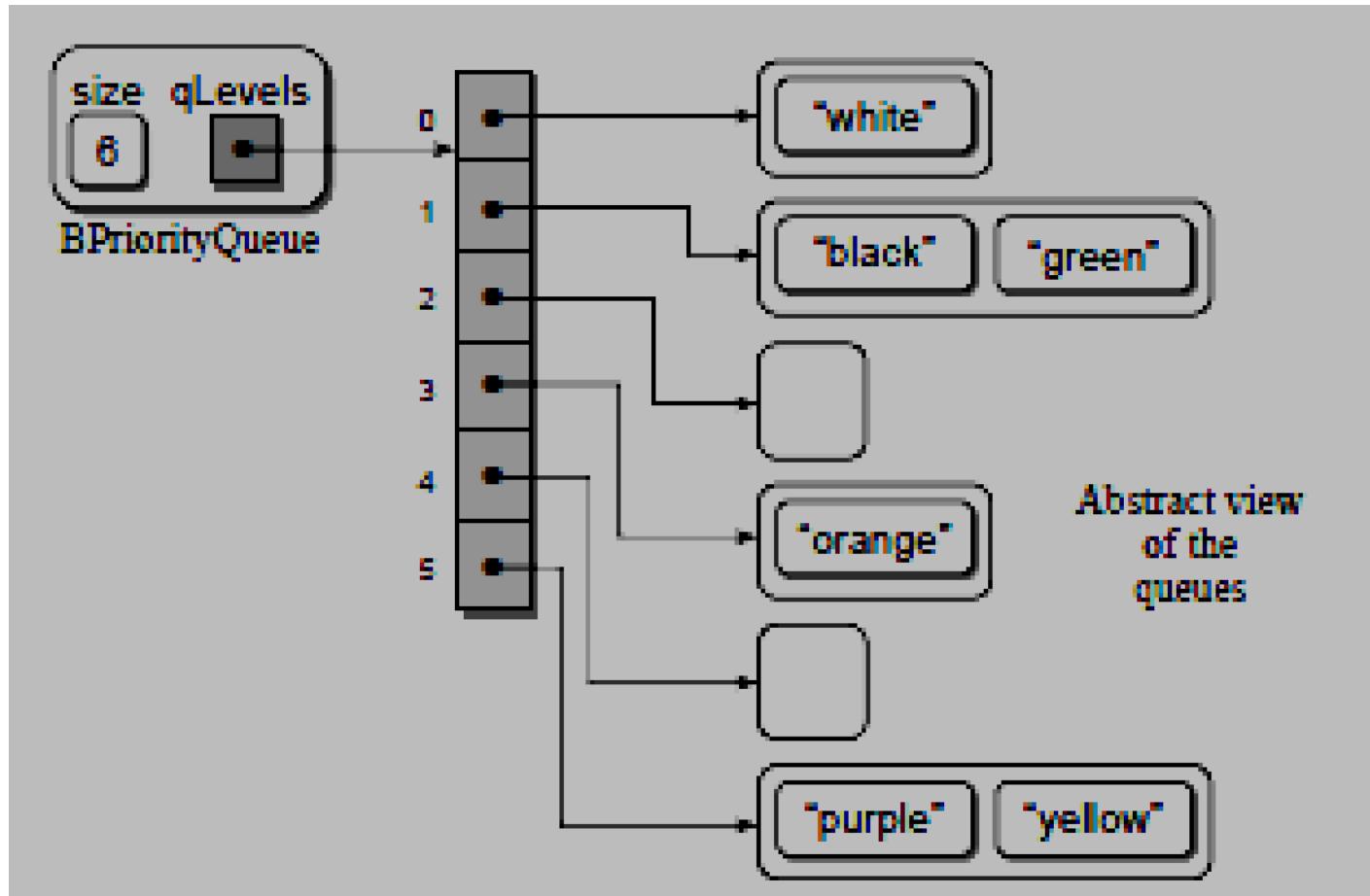
- **Bounded-Priority Queue** adalah antrian prioritas yang terbatas, dimana prioritas dibatasi maksimal sama dengan suatu integer  $p$ , atau  $[ 0 \dots p ]$ .
- Antrian ini yang diberi nama **BpriorityQueue** di-implementasikan memakai **LinkedList**, dengan ADT yang sama dengan antrian prioritas biasa.
- Pada implementasi-nya, constructor menciptakan dua data field, yaitu larik (array) yang diberi nama **\_qLevels**, untuk menyimpan prioritas dan berfungsi menjadi head ke antrian linked-list, yaitu **LinkedQueue()**. **LinkedQueue** akan di-isi dengan item/data sesuai urutan pemasukan-nya ke dalam antrian.

# Contoh

- BPQ berikut andaikan dibatasi hanya 6 (0..5) prioritas

```
BP=PriorityQueue(6)
BP.add(3, 'orange')
BP.add(1, 'black')
BP.add(0, 'white')
BP.add(5, 'purple')
BP.add(1, 'green')
BP.add(5, 'yellow')
```

# Hasil-nya



# Implementasi BPQ

```
# Implementation of the bounded Priority Queue ADT using an array of
# queues in which the queues are implemented using a linked list.
# BPriorityQueue.py == @Suarga
from Array import *
#from llistqueue import *
from LinkedQueue import *

class BPriorityQueue :
    # Creates an empty bounded priority queue.
    def __init__( self, numLevels ):
        self._qSize = 0
        self._qLevels = Array( numLevels )
        for i in range( numLevels ) :
            self._qLevels[i] = LinkedQueue()

    # Returns True if the queue is empty.
    def isEmpty( self ) :
        return len( self ) == 0
```

```
20     # Returns the number of items in the queue.
21     def __len__( self ):
22         return self._qSize
23
24     # Adds the given item to the queue.
25     def add( self, priority, item ):
26         assert priority >= 0 and priority < len(self._qLevels), \
27             "Invalid priority level."
28         self._qLevels[priority].enqueue( item )
29         self._qSize += 1
30
31     # Removes and returns the next item in the queue.
32     def remove_min( self ) :
33         # Make sure the queue is not empty.
34         assert not self.isEmpty(), "Cannot dequeue from an empty queue."
35         # Find the first non-empty queue.
36         i = 0
37         p = len(self._qLevels)
38         while i < p and self._qLevels[i].is_empty() :
39             i += 1
40
41         # We know the queue is not empty, so dequeue from the ith queue.
42         self._qSize -= 1
43         return self._qLevels[i].dequeue()
```

# test BPriorityQueue

```
- BP=BPriorityQueue(6)
- BP.add(3,'orange')
- BP.add(1,'black')
- BP.add(0,'white')
50 BP.add(5,'purple')
- BP.add(1,'green')
- BP.add(5,'yellow')

- print('size = ', BP.__len__())
- n = len(BP._qLevels)
- print('\n Isi Priority Queue: ')
- for i in range(n):
-     BP._qLevels[i].traverse()

60 print('\n Isi dihapus dengan urutan : ')
- while not BP.is_Empty():
-     item = BP.remove_min()
-     print(item)

- print('size setelah isi dibuang = ', BP.__len__())
```

# Hasil test:

```
>>>
*** Remote Interpreter Reinitialized ***
size = 6

    Isi Priority Queue:
white
black
green
orange
purple
yellow

    Isi dihapus dengan urutan :
white
black
green
orange
purple
yellow
size setelah isi dibuang = 0
>>>
```