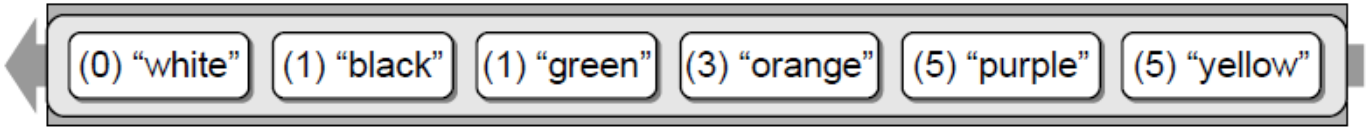
Priority Queue

Anoop S Babu
Faculty Associate
Dept. of Computer Science & Engineering
bsanoop@am.amrita.edu

Priority Queue

- A priority queue is a **special type of queue** in which each **element is associated with a priority** and is served according to its priority.
- When an element is added to a priority queue, the user designates its priority by providing an associated **key**.



- Priority Queue is an extension of queue with following properties.
  - Every item has a priority associated with it.
  - An element with high priority is dequeued before an element with low priority.
  - If two elements have the same priority, they are served according to their order in the queue.

Priority Queue: Basic Operations

P.add(k, v): Insert an item with key k and value v into priority queue P.

P.min( ): Return a tuple, (k,v), representing the key and value of an item in priority queue P with minimum key (but do not remove the item); an error occurs if the priority queue is empty.

P.remove_min( ): Remove an item with minimum key from priority queue P, and return a tuple, (k,v), representing the key and value of the removed item; an error occurs if the priority queue is empty.

P.is_empty( ): Return True if priority queue P does not contain any items.

len(P): Return the number of items in priority queue P.

Example: Priority Queue Operations

Operation	Return Value	Priority Queue
P.add(5,A)		{(5,A)}
P.add(9,C)		{(5,A), (9,C)}
P.add(3,B)		{(3,B), (5,A), (9,C)}
P.add(7,D)		{(3,B), (5,A), (7,D), (9,C)}
P.min( )	(3,B)	{(3,B), (5,A), (7,D), (9,C)}
P.remove_min( )	(3,B)	{(5,A), (7,D), (9,C)}
P.remove_min( )	(5,A)	{(7,D), (9,C)}
len(P)	2	{(7,D), (9,C)}
P.remove_min( )	(7,D)	{(9,C)}
P.remove_min( )	(9,C)	{ }
P.is_empty( )	True	{ }
P.remove_min( )	"error"	{ }

Implementation of Priority Queue

- Priority queue can be implemented using,
  - an array. *(or python list)*
  - a linked list.
  - a heap data structure.
  - a binary search tree.
- Among these data structures, heap data structure provides an efficient implementation of priority queues.

List-Based Implementations

- Implementation with a Unsorted List

(Append new items to the end of the list.)

  - When a **new item is enqueued, simply append** a new instance of the storage class (containing the item and its priority) **to the end of the list**.
  - When an item is dequeued, **search for the item with the highest priority and remove it** from the list.
  - If more than one item has the same priority, the first one encountered during the search will be the first to be dequeued.

List-Based Implementations

- Implementation with a Sorted List

(Keep the items sorted within the list based on their priority)

  - When a new item is enqueued, **find its proper position** within the list **based on its priority and insert** an instance of the storage class at that point.
  - If we order the items from highest priority at the front to lowest at the end, then the dequeue operation simply requires the removal of the first item in the list.

Comparing the Two List-Based Implementations

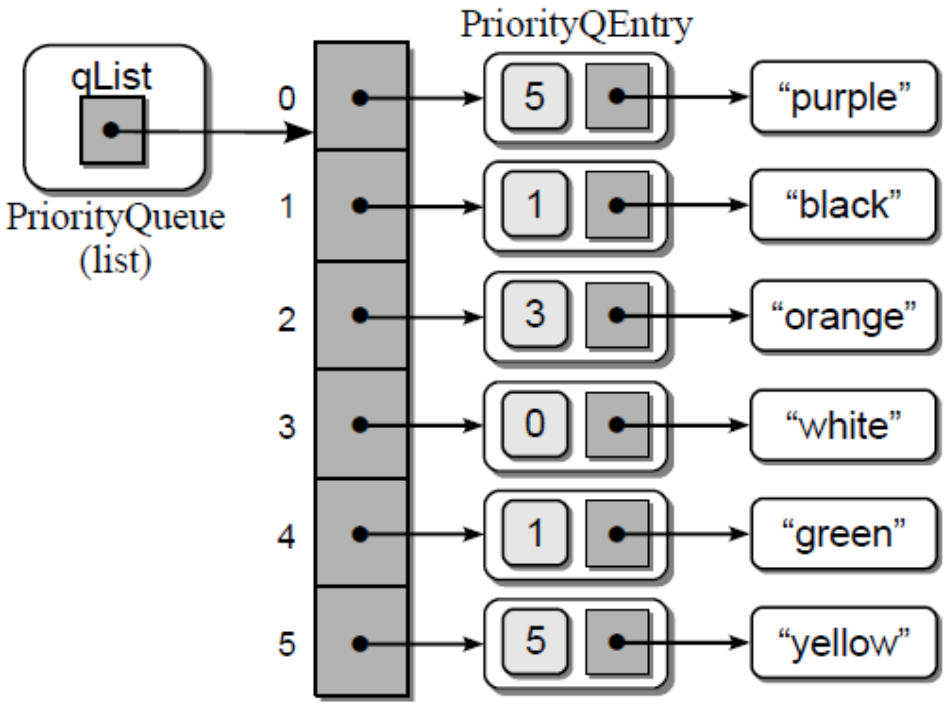
Operation	Unsorted List	Sorted List
len	$O(1)$	$O(1)$
is_empty	$O(1)$	$O(1)$
add	$O(1)$	$O(n)$
min	$O(n)$	$O(1)$
remove_min	$O(n)$	$O(1)$

Worst-case running times of the methods of a priority queue of size n.

List Implementation: Example

```
class PriorityQueue :
   # Create an empty unbounded priority queue.
  def __init__( self ):
    self._qList = list()

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

   # Returns the number of items in the queue.
  def __len__( self ):
    return len( self._qList )

   # Adds the given item to the queue.
  def enqueue( self, item, priority ):
     # Create a new instance of the storage class and append it to the list.
    entry = _PriorityQEntry( item, priority )
    self._qList.append( entry )

   # Removes and returns the first item in the queue.
  def dequeue( self ) :
    assert not self.isEmpty(), "Cannot dequeue from an empty queue."

     # Find the entry with the highest priority.
    highest = self._qList[i].priority
    for i in range( self.len() ) :
       # See if the ith entry contains a higher priority (smaller integer).
      if self._qList[i].priority < highest :
        highest = self._qList[i].priority

     # Remove the entry with the highest priority and return the item.
    entry = self._qList.pop( highest )
    return entry.item

# Private storage class for associating queue items with their priority.
class _PriorityQEntry( object ):
  def __init__( self, item, prioity ):
    self.item = item
    self.priority = priority
```

List Implementation: Example



An instance of the priority queue implemented using a list.