Priority Queues

Priority Queue ADT

- A priority queue stores a collection of items
- Each item is a pair (key, value)
- Main methods of the Priority Queue ADT
  - add (k, x) inserts an item with key k and value x
  - remove_min() removes and returns the item with smallest key
- Additional methods
  - min() returns, but does not remove, an item with smallest key
  - len(P), is_empty()
- Applications:
  - Standby flyers
  - Auctions
  - Job scheduler in Operating systems

Priority Queue Example

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"	{ }

Total Order Relations

- Keys in a priority queue can be arbitrary objects on which an order is defined
- Two distinct entries in a priority queue can have the same key
- Mathematical concept of total order relation $\leq$
  - Reflexive property: $x \leq x$
  - Antisymmetric property: $x \leq y \wedge y \leq x \Rightarrow x = y$
  - Transitive property: $x \leq y \wedge y \leq z \Rightarrow x \leq z$

e.g. Alphabetical order: Define x <= y if 'x' is before 'y' in alphabetical order

Composition Design Pattern

- An item in a priority queue is simply a key-value pair
- Priority queues store items to allow for efficient insertion and removal based on keys

```python
class PriorityQueueBase:
  """Abstract base class for a priority queue."""

  class _Item:
    """Lightweight composite to store priority queue items."""
    __slots__ = '_key', '_value'

    def __init__(self, k, v):
      self._key = k
      self._value = v

    def __lt__(self, other):
      return self._key < other._key          # compare items based on their keys

  def is_empty(self):                        # concrete method assuming abstract len
    """Return True if the priority queue is empty."""
    return len(self) == 0
```

Sequence-based Priority Queue

- Implementation with an unsorted doubly linked list

- Performance:
  - add takes $O(1)$ time since we can insert the item at the beginning or end of the sequence
  - Remove_min and min take $O(n)$ time since we have to traverse the entire sequence to find the smallest key

- Implementation with a sorted doubly linked list

- Performance:
  - add takes $O(n)$ time since we have to find the place where to insert the item
  - remove_min and min take $O(1)$ time, since the smallest key is at the beginning

Runtime of implementing PQ using Sorted and Unsorted List

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	$O(n)$	$O(1)$

Priority Queue Sorting

- We can use a priority queue to sort a set of comparable elements
  1. Insert the elements one by one with a series of add operations
  2. Remove the elements in sorted order with a series of remove_min operations
- The running time of this sorting method depends on the priority queue implementation

Algorithm *PQ-Sort*(*S*, *C*)
    Input sequence *S*, comparator *C* for the elements of *S*
    Output sequence *S* sorted in increasing order according to *C*
    $P \leftarrow$ priority queue with comparator *C*
    while $\neg$*S.is_empty* ()
        $e \leftarrow$ *S.remove_first* ()
        P.add (*e*, $\varnothing$)
    while $\neg$*P.is_empty*()
        $e \leftarrow$ *P.removeMin*().*key*()
        S.add_last(*e*)