# Chapter 9

# Priority Queues

## 9.1 Priority Queues ( p_queue )

### 1. Definition

An instance $Q$ of the parameterized data type *p_queue*<$P, I$> is a collection of items (type *pq_item*). Every item contains a priority from a linearly ordered type $P$ and an information from an arbitrary type $I$. $P$ is called the priority type of $Q$ and $I$ is called the information type of $Q$. If $P$ is a user-defined type, you have to define the linear order by providing the compare function (see Section 2.3). The number of items in $Q$ is called the size of $Q$. If $Q$ has size zero it is called the empty priority queue. We use $\langle p, i\rangle$ to denote a *pq_item* with priority $p$ and information $i$.
Remark: Iteration over the elements of $Q$ using iteration macros such as *forall* is not supported.

*#include* < *LEDA/core/p_queue.h* >

### 2. Types

| | |
|---|---|
| *p_queue*<$P, I$>:: *item* | the item type. |
| *p_queue*<$P, I$>:: *prio_type* | the priority type. |
| *p_queue*<$P, I$>:: *inf_type* | the information type. |

### 3. Creation

*p_queue*<$P, I$> $Q$; creates an instance $Q$ of type *p_queue*<$P, I$> based on the linear order defined by the global compare function *compare*(*const P*&, *const P*&) and initializes it with the empty priority queue.

*p_queue*<$P, I$> $Q$(*int* (∗*cmp*)(*const P*& , *const P*& ));

creates an instance $Q$ of type *p_queue*<$P, I$> based on the linear order defined by the compare function *cmp* and initializes it with the empty priority queue. *Precondition*: *cmp* must define a linear order on $P$.

### 4. Operations

| | | |
|---|---|---|
| *const P*& | $Q$.prio(*pq_item it*) | returns the priority of item *it*.<br>*Precondition*: *it* is an item in $Q$. |
| *const I*& | $Q$.inf(*pq_item it*) | returns the information of item *it*.<br>*Precondition*: *it* is an item in $Q$. |
| *I*& | $Q$[*pq_item it*] | returns a reference to the information of item *it*.<br>*Precondition*: *it* is an item in $Q$. |
| *pq_item* | $Q$.insert(*const P*& *x*, *const I*& *i*) | adds a new item <$x, i$> to $Q$ and returns it. |
| *pq_item* | $Q$.find_min( ) | returns an item with minimal priority (nil if $Q$ is empty). |
| *P* | $Q$.del_min( ) | removes the item $it = Q$.find_min() from $Q$ and returns the priority of it.<br>*Precondition*: $Q$ is not empty. |
| *void* | $Q$.del_item(*pq_item it*) | removes the item *it* from $Q$.<br>*Precondition*: *it* is an item in $Q$. |
| *void* | $Q$.change_inf(*pq_item it*, *const I*& *i*) | makes $i$ the new information of item *it*.<br>*Precondition*: *it* is an item in $Q$. |
| *void* | $Q$.decrease_p(*pq_item it*, *const P*& *x*) | makes $x$ the new priority of item *it*.<br>*Precondition*: *it* is an item in $Q$ and $x$ is not larger then *prio*(*it*). |
| *int* | $Q$.size( ) | returns the size of $Q$. |
| *bool* | $Q$.empty( ) | returns true, if $Q$ is empty, false otherwise. |
| *void* | $Q$.clear( ) | makes $Q$ the empty priority queue. |

### 5. Implementation

Priority queues are implemented by binary heaps [91]. Operations insert, del_item, del_min take time $O(\log n)$, find_min, decrease_p, prio, inf, empty take time $O(1)$ and clear takes time $O(n)$, where $n$ is the size of $Q$. The space requirement is $O(n)$.