Implementation of a priority queue with an unsorted sequence

Variables

sequence: sequence, the elements of which are items (pairs of keys and elements)
invariant: the elements of *sequence* are the items of the priority queue

Initialization

sequence ← empty sequence

Algorithms

size():
  output: size of priority queue
return size of *sequence*

isEmpty():
  output: priority queue is empty?
return *sequence* is empty?

insertItem(*key*, *element*):
  postcondition: item (*key*, *element*) has been inserted in the priority queue
  input: item to be inserted
insert (*key*, *element*) into *sequence* (at the end)

minPosition():
  precondition: *sequence* is nonempty
  output: position of *sequence* with minimal key
position ← first position of *sequence*
minimum ← first position of *sequence*
while *position* ≠ last position of *sequence* **do**
loop-invariant: *minimum* is the position with minimal key from the first position of *sequence* upto (and excluding) *position*
    position ← position after *position* in *sequence*
    if key of *position* < key of *minimum* **then**
        minimum ← *position*
return *minimum*

minElement():
  precondition: priority queue is nonempty
  output: element with smallest key in priority queue
minimal ← minPosition()
return element stored in *minimal*

minKey():
  precondition: priority queue is nonempty
  output: smallest key in priority queue
minimal ← minPosition()
return key stored in *minimal*

removeMinElement():
  precondition: priority queue is nonempty
  postcondition: item of returned element has been removed from the priority queue
  output: element with smallest key in priority queue
minimal ← minPosition()
element ← element stored in *minimal*
remove *minimal* from *sequence*
return *element*

Implementation of a priority queue with a sorted sequence

Variables

sequence: sequence, the elements of which are items (pairs of keys and elements)
invariant: the elements of *sequence* are the items of the priority queue and *sequence* is sorted by key from biggest to smallest

Initialization

sequence ← empty sequence

Algorithms

size():
  output: size of priority queue
return size of *sequence*

isEmpty():
  output: priority queue is empty?
return *sequence* is empty?

insertItem(*key*, *element*):
  postcondition: item (*key*, *element*) has been inserted in the priority queue
  input: item to be inserted
if *sequence* is empty **then**
    insert (*key*, *element*) into *sequence*
else if key of last position of *sequence* $\geq$ *key* **then**
    insert item (*key*, *element*) as last element of *sequence*
else
    position ← first position of *sequence*
    while key of *position* $>$ *key* **do**
    loop-invariant: the first position of *sequence* upto (and excluding) *position* contain bigger keys than *key*
        position ← position after *position* in *sequence*
        insert item (*key*, *element*) before *position* in *sequence*

minElement():
  precondition: priority queue is nonempty
  output: element with smallest key in priority queue
return element stored in last position of *sequence*

minKey():
  precondition: priority queue is nonempty
  output: smallest key in priority queue
return key stored in last position of *sequence*

removeMinElement():
  precondition: priority queue is nonempty
  postcondition: item with returned element has been removed from the priority queue
  output: element with smallest key in priority queue
return element stored in last position of *sequence*
remove last position from *sequence*