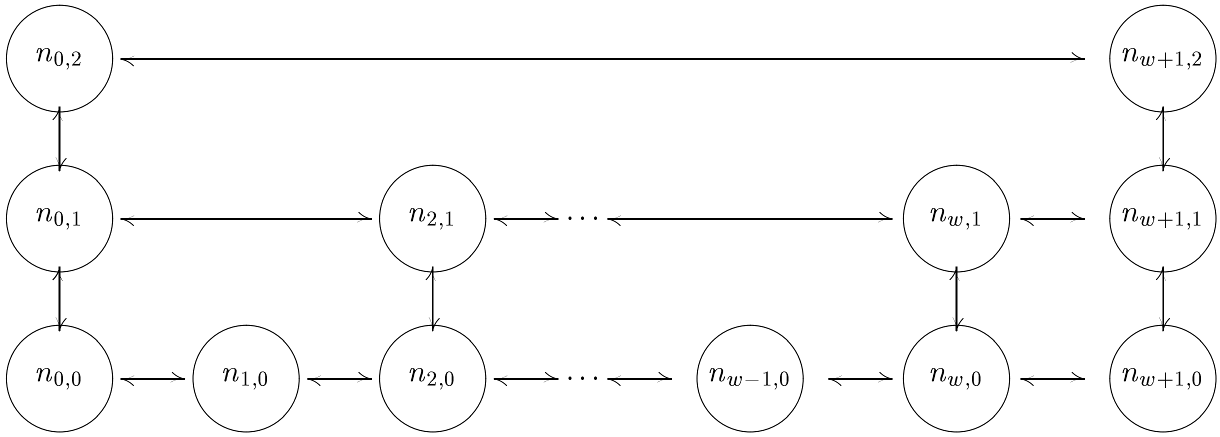
Implementation of a dictionary by means of a skip list

Variables

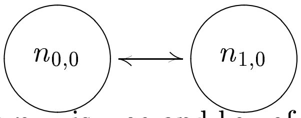
size: integer
skip-list: a two-dimensional collection of nodes; each node contains an item (an element and a key) and pointers to the nodes before, after, above and below the node (if these exist)



start: pointer to node
invariant: the nodes $n_{1,0}$, ..., $n_{w,0}$ of *skip-list* contain the items of the dictionary sorted by key. For every tower, the nodes in the tower contain the same item. The keys of the first and last tower are $-\infty$ and ∞, respectively. Only the top level only contains two nodes: $n_{0,h}$ and $n_{w+1,h}$. *size* is the size of the dictionary.[1] *start* points to $n_{0,h}$.

Initialization

size $\leftarrow 0$

skip-list $\leftarrow$ 

Key of node $n_{0,0}$ is $-\infty$ and key of node $n_{1,0}$ is ∞
start points to $n_{0,0}$

Algorithms

size():
  output: size of dictionary
return *size*

isEmpty():
  output: dictionary is empty?
return (*size* $= 0$)

skipSearch(*key*):
  input: key to be searched for

[1] The auxiliary insertAfterAbove does not preserve this property. We can remedy this by changing either the invariant or the algorithms. For simplicity, we will not do that.

output: node of *skip-list* with the largest key that is less than or equal to *key* on the bottom level
node ← *start*
while there is a node below *node* **do**
loop-invariant: the keys of all the nodes before *node* are smaller than or equal to *key*
    node ← node below *node*
    while key of node after *node* $\leq$ *key* **do**
    loop-invariant: the keys of all the nodes before *node* are smaller than or equal to *key*
        node ← node after *node*
return *node*

findElement(*key*):
  input: key to be searched for
  output: element of item with *key* in the dictionary; NO-SUCH-KEY if no such item exists
node ← skipSearch(*key*)
if key of *node* = *key* **then**
    return element of *node*
else
    return NO-SUCH-KEY

insertAfterAbove(*after*, *above*, *key*, *element*):
  input: item of node to be inserted; position in *skip-list* where node is to be inserted
  output: inserted node
  postcondition: node with item *(key, element)* has been inserted after node *after* and above node *above* in *skip-list*; if *after* = *start* one level is added to the first and last tower
node ← node with item *(key, element)*
insert *node* after node *after* and above node *above*
if *after* = *start* **then**
    insert a node with key $-\infty$ on top of the first tower
    start ← inserted node
    insert a node with key ∞ on top of the last tower
return *node*

skipInsert(*key*, *element*):
  input: item to be inserted
  postcondition: tower with item (*key*, *element*) has been inserted in *skip-list*
after ← skipSearch(*key*)
insert a node with item (*key*, *element*) after node *after*[2]
above ← inserted node
flip *coin*
while *coin* = heads **do**
loop-invariant: *above* is the top node of the tower being inserted
    while there is no node above node *after* **do**
    loop-invariant: there is no node above the nodes in between the node following *after* and the node before *above*
        after ← node before node *after*
    after ← node above node *after*
    above ← insertAfterAbove(*after*, *above*, *key*, *element*)

[2]There is no node above or below the inserted node.

```
      flip coin
size ← size + 1

insertItem(key, element):
   input: item to be inserted
   postcondition: item (key, element) has been inserted into the dictionary
skipInsert(key, element)

skipRemove(key):
   input: key to be searched for
   output: element of item with key in skip-list; NO-SUCH-KEY if no such item exists
   postcondition: tower with key is has been removed from skip-list
node ← skipSearch(key)
if key of node = key then
      element ← element of node
      while there is a node above node do
      loop-invariant: nodes below node have been removed
            node ← node above node
            remove node below node
      remove node
      while one but top level has only two (dummy) nodes do
            remove one but top level from skip-list
      size ← size − 1
      return element
else
      return NO-SUCH-KEY

remove(key):
   input: key to be searched for
   output: element of item with key in dictionary; NO-SUCH-KEY if no such item exists
   postcondition: item has been removed from dictionary
skipRemove(key)
```