HeapNode:

public HeapNode nodeDecreaseKey

Helper method for heap.DecreaseKey() - works with list.listDecreaseKey().

This method decreases the node’s key with the gived delta value, and cuts the node into the heap forest, if necessary. It also takes care to mark the cut node’s parent node, if necessary.

Time complexity: O(n).

public boolean hasPrev()

Returns true iff this node’s ‘prev’ reference is not null.

Time complexity: O(1)

/\*\*  
 \* Check if this node has a next node  
 \* @return true if this node has a next node, false otherwise  
 \* Time Complexity: O(1)  
 \*/  
public boolean hasNext()

Returns true iff this node’s ‘next’ reference is not null.

Time complexity: O(1)

public boolean hasParent()

Returns true iff this node’s ‘parent’ reference is not null.

Time complexity: O(1)  
  
public int getKey()

Returns the key of this node

Time complexity: O(1).

public HeapNode getParent()

Returns the parent of this node, nullable return value is possible.

Time complexity: O(1).

public int rank():

Returns the length of the children linked list of this node.

Time complexity: O(1)

public int getSize()

Returns the size of the children linked list of this node, plus one for the node itself.

Time complexity: O(1).

public boolean getMark()

Returns the mark of this node.

Time complexity: O(1)

public HeapNode findRecursive(int key):

recursively searches for the node with the given key in the tree rooted at this node,

including this node.

Prioritized direct search in children list over deep search to the depth of the tree.

Time complexity: O(n), since the tree is not a search tree.

public void setPrev(HeapNode node):

Sets the ‘prev’ reference to the given node.

Time complexity: O(1).

public void setNext(HeapNode node):

Sets the ‘next’ reference to the given node.

Time complexity: O(1).

public boolean setMark(boolean mark):

Set the mark of this node.

Time complexity: O(1)

public void insertPrev(HeapNode node):

Insert a node as the prev node of this node,

Update both node’s relevant references to each other.

Time complexity: O(1).

public void insertChild(HeapNode node)

Insert a node to the header of the children list of this node.

Time complexity: O(1).

public void plantPrev(LinkedList list):

Plant a linked list previous to this node.

Updates length, size, and relevant node pointers.

Time Complexity: O(1)

public void eject()

Remove this node from the doubly linked list:

set pointers to next, prev and 'siblings' to 'null'

Time Complexity: O(1)

/\*\*  
 \* cut this node from its siblings list.  
 \* Time Complexity: O(1)  
 \*/  
public void cut()

Cut this node from

Time complexity: O(1)

/\*\*  
 \* Node factory which takes the key as an argument and also  
 \* sets a reference to the heap of the node.  
 \*/  
static class NodeFactory

LinkedList  
class LinkedList implements Iterable<FibonacciHeap.HeapNode>:

LinkedList class that stores HeapNodes, also has a pointer to the minNode

and a length and size attributes that update according to the stored node's  
 size. a size is the total number of nodes stored in the current list, and all their offsprings.

public void deleteMin():

Helper method for heap.deleteMin()

Deletes the given node from the list, and plants the children in his place by order

Time complexity: O(1).

public FibonacciHeap.HeapNode listDecreaseKey(int key, int d, FibonacciHeap heap)

Helper method for heap.DecreaseKey() – based on Node.nodeDecreaseKey()

O(n)

public String toString()

[key\_root, key\_2, ..., key\_tail]

Time Complexity: O(n)  
  
public boolean isEmpty():

Returns true iff root is null

Time complexity: O(1).

public boolean hasParent()

Returns true iff parent is not null.

O(1).

public FibonacciHeap.HeapNode getMin():

Return the minimal node from a designated reference ‘minNode’

Time complexity: O(1).

public FibonacciHeap.HeapNode findRecursive(int key):

recursively search for the node with the provided key within the children list of each node,

only if the requested key is not directly inside the children list.

Returns the node with the corresponding key.

Time complexity: O(n).

private void setSize(int size)

Set the size of this list

O(1).

public void increaseSize(int delta):

Call setSize() based on the delta parameter, which indicates the increase of the list’s size.

Recursively call tha parent’s siblings’ increaseSize() with the given delta parameter.

Time complexity: O(logn).

public void decreaseSize(int delta)

Call increaseSize with -delta

Time Complexiy: O(logn)  
  
public void insertFirst(FibonacciHeap.HeapNode node)

Insert the parameter node as the header of this list,  
set the parameter node's siblings list to this list,  
update root attribute to the parameter node, and also the tail node if necessary.  
updates length and size of this list  
updates minNode pointer of this list, if necessary.   
Time Complexity: O(logn)

public void annex(LinkedList list2)

Concatenate the param 'list2' to the tail of this list.  
 updates list pointers as relevant, including tail, root, and minNode  
 updates length and size of this list

Time Complexity: O(logn)

public void plantBefore(LinkedList list2, FibonacciHeap.HeapNode nodeAfter)

Param ‘list2’ is the list to be planted to this list

Param ‘nodeAfter’ is the node to be after the planted list, null calls annex(list2).

Time complexity: O(logn).

public void cutNode(FibonacciHeap.HeapNode node)

Removes the node with its children from the linked list.  
updates length and size of list, also sends an update command upwards in the tree.  
updates the minNode reference of this linked list, if necessary.  
Time Complexity: O(logn)

public void ripNode(FibonacciHeap.HeapNode node)

Removes the node with its children from the linked list.  
updates length and size of list, also sends an update command upwards in the tree.  
Time Complexity: O(logn)

public void updateMin()

Go through all the nodes in the list to find the minimal keyed node,

And stores a reference to it in the list.

Time complexity: O(m), where m is the length of this list. (n is the size of the heap).

@Override  
public Iterator<FibonacciHeap.HeapNode> iterator()

The node iterator of this linked list.

/\*\*  
 \* A factory to create a linked list with a reference to its parent.  
 \*/  
class LinkedListFactory

/\*\*  
 \* A node iterator that goes through the linked list.  
 \*/  
class LinkedListIterator implements Iterator<FibonacciHeap.HeapNode>