FibonacciHeap Class Reference

Public Member Functions

FibonacciHeap ()
Description
Constructor to create an empty Fibonacci Heap

FibonacciHeap (**Node** *)
Description
Constructor to create a Fibonacci Heap with one node More...

void **FibonacciHeap_Insert** (**Node** *)
Description
Insert Operation to insert a new node n into a Fibonacci Heap H. It has O(1) actual cost and O(1) amortized cost More...

FibonacciHeap * **FibonacciHeap_Union** (**FibonacciHeap** *)

Node * **FibonacciHeap_extractMin** ()
Description
Operation used to extract the node with the highest priority (node with minimum key), the operation removes it from the Heap and returns it. It has O(D(n) + t(H)) actual cost, and O(D(n)) amortized cost which is O(lg n) More...

void **FibonacciHeap_decreaseKey** (**Node** *, int)
Description
Operation used to decrease a key k of a node x. It has O(c) actual cost, and O(1) amortized cost More...

void **FibonacciHeap_delete** (**Node** *)
Description
This operation is used to delete a node n from Heap. Its amortized cost is O(lg n) More...

int **getnumberOfNodes** ()
Description
Getter to get the number of nodes n[H] More...

Node * **getMin** ()
Description
Getter to get the node with the minimum key More...

Private Member Functions

void **FibonacciHeap_Link** (**Node** *, **Node** *)
Description
This operation is used to link nodes together, it does so by removing a node y from the root list of heap H, and making it a child of a node x More...

void **consolidate** ()
Description
This operation is used by extractMin(). It is used to link roots that has equal degrees until each root in the rool List of H, has a unique degree. More...

void **cut** (**Node** *, **Node** *)
Description
This operation is used by decreaseKey(). it is used to cut the link between node x and its parent y making x a new root in the root list More...

void **cascadingCut** (**Node** *)
Description
This operation is used by decreaseKey(). It cuts y from its parent and makes it a new root, however the parent of y is the new y. It calls itself recursivly, to cut all children of node n and make them new roots. It recurses until y is a root or it finds an unmarked node More...

Private Attributes

int **numberOfNodes**
number of nodes in Heap H, n[H]

Node * **root**
pointer to the root in Heap H

Node * **min**
pointer the **Node** with the minimum key value

Constructor & Destructor Documentation

◆ FibonacciHeap()

FibonacciHeap::FibonacciHeap (**Node** *)

Description

Constructor to create a Fibonacci Heap with one node

Parameters

node single node to insert to the newly created Fibonacci Heap

Member Function Documentation

◆ cascadingCut()

void FibonacciHeap::cascadingCut (**Node** *) private

Description

This operation is used by decreaseKey(). It cuts y from its parent and makes it a new root, however the parent of y is the new y. It calls itself recursivly, to cut all children of node n and make them new roots. It recurses until y is a root or it finds an unmarked node

Parameters

y **Node** y, which we want to cut from its parent parent[y]

Pseudocode

```
CASCADING-CUT(H, y)
 1 z ← p[y]
 2 if z ≠ NIL
 3    then if mark[y] = FALSE
 4            then mark[y] ← TRUE
 5            else CUT(H, y, z)
 6                 CASCADING-CUT(H, z)
```

◆ consolidate()

void FibonacciHeap::consolidate ( ) private

Description

This operation is used by extractMin(). It is used to link roots that has equal degrees until each root in the rool List of H, has a unique degree.

Pseudocode

```
CONSOLIDATE(H)
  1 for i ← 0 to D(n[H])     //D(n[H]) Maximum degree of any node in Heap H
  2      do A[i] ← NIL       //A[] buffer we want to use to link the trees together
  3 for each node w in the root list of H
  4      do x ← w
  5         d ← degree[x]
  6         while A[d] ≠ NIL
  7             do y ← A[d]       ▷ Another node with the same degree as x.
  8                if key[x] > key[y]
  9                   then exchange x ↔ y
 10                FIB-HEAP-LINK(H, y, x)
 11                A[d] ← NIL
 12                d←d + 1
 13         A[d] ←x
 14 min[H] ← NIL
 15 for i← 0 to D(n[H])
 16      do if A[i] ≠ NIL
 17            then add A[i] to the root list of H
 18                 if min[H] = NIL or key[A[i]] < key[min[H]]
 19                    then min[H] ←A[i]
```

◆ cut()

void FibonacciHeap::cut (**Node** * ,
Node *
) private

Description

This operation is used by decreaseKey(). it is used to cut the link between node x and its parent y making x a new root in the root list

Parameters

x **Node** x, which we want to cut from its parent y

y **Node** y, parent of x

Pseudocode

```
CUT(H, x, y)
 1 remove x from the child list of y, decrementing degree[y]
 2 add x to the root list of H
 3 parent[x] ← NIL 4 mark[x] ← FALSE
```

◆ FibonacciHeap_decreaseKey()

void FibonacciHeap::FibonacciHeap_decreaseKey (**Node** * ,
int
)

Description

Operation used to decrease a key k of a node x. It has O(c) actual cost, and O(1) amortized cost

Parameters

node The node, whose key we want to decrease

k New key k that we want to assign to node

Pseudocode

```
FIB-HEAP-DECREASE-KEY(H, x, k)
 1  if k > key[x]
 2     then error "new key is greater than current key"
 3  key[x] ← k
 4  y ← parent[x]
 5  if y≠ NIL and key[x] < key[y]
 6     then CUT(H, x, y)            //CUT() is included in the documentation
 7          CASCADING-CUT(H, y)     //CASCADING-CUT() is included in the documentation
 8  if key[x] < key[min[H]]
 9      then min[H] ←x
```

◆ FibonacciHeap_delete()

void FibonacciHeap::FibonacciHeap_delete (**Node** *)

Description

This operation is used to delete a node n from Heap. Its amortized cost is O(lg n)

Parameters

n **Node** n, which we want to delete from H

Pseudocode

```
FIB-HEAP-DELETE(H, x)
 1 FIB-HEAP-DECREASE-KEY(H, x, -∞)
 2 FIB-HEAP-EXTRACT-MIN(H)
```

◆ FibonacciHeap_extractMin()

Node * FibonacciHeap::FibonacciHeap_extractMin ( )

Description

Operation used to extract the node with the highest priority (node with minimum key), the operation removes it from the Heap and returns it. It has O(D(n) + t(H)) actual cost, and O(D(n)) amortized cost which is O(lg n)

Returns

Node x with the minimum key (node with highest priority)

Pseudocode

```
FIB-HEAP-EXTRACT-MIN(H)
 1  z←min[H]
 2  if z≠ NIL
 3     then for each child x of z
 4               do add x to the root list of H
 5                  parent[x] ← NIL
 6          remove z from the root list of H
 7          if z = right[z]
 8             then min[H] ← NIL
 9             else min[H] ← right[z]
10                  CONSOLIDATE(H)
11          n[H] ← n[H] - 1
12  return z
```

◆ FibonacciHeap_Insert()

void FibonacciHeap::FibonacciHeap_Insert (**Node** *)

Description

Insert Operation to insert a new node n into a Fibonacci Heap H. It has O(1) actual cost and O(1) amortized cost

Parameters

n **Node** to insert into the Fibonacci Heap H

Pseudocode

```
FIB-HEAP-INSERT(H, x)
 1  degree[x] ← 0    //degree of node x
 2  parent[x] ← NIL  //Parent of node x
 3  child[x] ← NIL   //Child of node x
 4  left[x] ←x       //Node left to node x
 5  right[x] ←x      //Node right to node x
 6  mark[x] ← FALSE  //Mark of x
 7  concatenate the root list containing x with root list H
 8  if min[H] = NIL or key[x] < key[min[H]]
 9  then min[H] ←x 10  n[H] ←n[H] + 1    //n[H]= number of nodes in H
```

◆ FibonacciHeap_Link()

void FibonacciHeap::FibonacciHeap_Link (**Node** * ,
Node *
) private

Description

This operation is used to link nodes together, it does so by removing a node y from the root list of heap H, and making it a child of a node x

Parameters

y The node y, which will be removed from the root list and assigned as a child to node x

x **Node** x, which will be the parent of y

Pseudocode

```
FIB-HEAP-LINK(H, y, x)
 1  remove y from the root list of H
 2  make y a child of x, incrementing degree[x]
 3  mark[y] ← FALSE
```

◆ FibonacciHeap_Union()

FibonacciHeap * FibonacciHeap::FibonacciHeap_Union (**FibonacciHeap** *)

Description

Operation to Merge two Fibonacci Heap H1 and H2 into one Fibonacci Heap H. It has O(1) actual cost and O(1) amortized cost

Parameters

H2 A Fibonacci Heap H2 to merge with Fibonacci Heap H1

Returns

Heap H which is the result of merging H1 and H2 together

Pseudocode

```
FIB-HEAP-UNION(H1, H2)
 1  H← MAKE-FIB-HEAP()
 2  min[H] ←min[H1]
 3  concatenate the root list of H2 with the root list of H
 4  if (min[H1] = NIL) or (min[H2] ≠ NIL and min[H2] < min[H1])
 5    then min[H] ←min[H2]
 6  n[H] ←n[H1] + n[H2]     //n[H]= number of nodes in the heap
 7  free the objects H1 and H2
 8  return H
```

◆ getMin()

Node * FibonacciHeap::getMin ( )

Description

Getter to get the node with the minimum key

Returns

The node that has the minimum key

◆ getnumberOfNodes()

int FibonacciHeap::getnumberOfNodes ( )

Description

Getter to get the number of nodes n[H]

Returns

The number of nodes n in Fibonacci Heap H

The documentation for this class was generated from the following file:

- **FibonacciHeap.hpp**