

# Implementation of Dijkstra's algorithm using Binary, Binomial and Fibonacci Heaps

Anant Maheshwari 13CO111 Shrukul Habib 13CO143

Generated by Doxygen 1.8.12



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	BinomialHeap Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Member Function Documentation . . . . .	6
3.1.2.1	Binomial_link(node *, node *) . . . . .	6
3.1.2.2	Create_node(int, int) . . . . .	6
3.1.2.3	Decrease_key(node *, int, int) . . . . .	6
3.1.2.4	Delete(node *, int) . . . . .	7
3.1.2.5	Display(node *) . . . . .	7
3.1.2.6	Extract_Min(node *) . . . . .	7
3.1.2.7	Initializeheap() . . . . .	7
3.1.2.8	Insert(node *, node *) . . . . .	8
3.1.2.9	Merge(node *, node *) . . . . .	8
3.1.2.10	Revert_list(node *) . . . . .	8
3.1.2.11	Search(node *, int) . . . . .	8
3.1.2.12	Union(node *, node *) . . . . .	9
3.2	FibonacciHeap Class Reference . . . . .	9
3.2.1	Detailed Description . . . . .	10

3.2.2	Member Function Documentation	10
3.2.2.1	Cascade_cut(nodef *, nodef *)	10
3.2.2.2	Consolidate(nodef *)	10
3.2.2.3	Create_node(int, int)	11
3.2.2.4	Cut(nodef *, nodef *, nodef *)	11
3.2.2.5	Decrease_key(nodef *, int, int)	11
3.2.2.6	Delete_key(nodef *, int)	12
3.2.2.7	Display(nodef *)	12
3.2.2.8	Extract_Min(nodef *)	12
3.2.2.9	Fibonnaci_link(nodef *, nodef *, nodef *)	12
3.2.2.10	Find(nodef *, int)	13
3.2.2.11	InitializeHeap()	13
3.2.2.12	Insert(nodef *, nodef *)	13
3.2.2.13	pop()	13
3.2.2.14	top()	14
3.2.2.15	Union(nodef *, nodef *)	14
3.3	node Struct Reference	14
3.3.1	Detailed Description	14
3.3.2	Member Data Documentation	15
3.3.2.1	child	15
3.3.2.2	degree	15
3.3.2.3	index	15
3.3.2.4	parent	15
3.3.2.5	sibling	15
3.4	nodef Struct Reference	15
3.4.1	Detailed Description	16
3.4.2	Member Data Documentation	16
3.4.2.1	child	16
3.4.2.2	degree	16
3.4.2.3	index	16
3.4.2.4	left	16
3.4.2.5	mark	16
3.4.2.6	parent	16
3.4.2.7	right	16

<b>4 File Documentation</b>	<b>17</b>
4.1 dijkstra.cpp File Reference . . . . .	17
4.1.1 Detailed Description . . . . .	18
4.1.2 Function Documentation . . . . .	19
4.1.2.1 binary_heap() . . . . .	19
4.1.2.2 binomial_heap() . . . . .	19
4.1.2.3 deleteBinaryHeap() . . . . .	19
4.1.2.4 fibonacci_heap() . . . . .	19
4.1.2.5 getLeft(int index) . . . . .	19
4.1.2.6 getRight(int index) . . . . .	20
4.1.2.7 heapifyDownBinaryHeap(int index) . . . . .	20
4.1.2.8 heapifyUpBinaryHeap(int index) . . . . .	20
4.1.2.9 insertBinaryHeap(pair< int, int > element) . . . . .	20
4.1.2.10 main() . . . . .	21
4.1.2.11 parent(int index) . . . . .	21
4.1.2.12 value(int index) . . . . .	21
<b>Index</b>	<b>23</b>



# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">BinomialHeap</a>	Binomial Heap Data Structure class . . . . .	5
<a href="#">FibonacciHeap</a>	Fibonacci Heap Data Structure class . . . . .	9
<a href="#">node</a>	*-----BINOMIAL HEAP-----*/ . . . .	14
<a href="#">nodef</a>	*-----FIBONACCI HEAP----- -----*/ . . . . .	15





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

[dijkstra.cpp](#)

Implementation of Dijkstra's algorithm using Binary Heap, Binomial Heap and Fibonacci Heap . 17



## Chapter 3

# Class Documentation

### 3.1 BinomialHeap Class Reference

Binomial Heap Data Structure class.

#### Public Member Functions

- `node * Initializeheap ()`  
*Initialize the Binomial Heap.*
- `int Binomial_link (node *, node *)`  
*Adds edges between two nodes of a tree and makes appropriate changes in parent,child, sibling and degree parameters.*
- `node * Create_node (int, int)`  
*Creates a new node and assigns value and the index.*
- `node * Union (node *, node *)`  
*Union of the the two Binomial Heaps.*
- `node * Insert (node *, node *)`  
*Insertion of a new Binomial Tree in the Binomial Heap.*
- `node * Merge (node *, node *)`  
*Merging of the nodes of the given Binomial Heap.*
- `node * Extract_Min (node *)`  
*Extract the minimum from the Binomial Heap.*
- `void pop (node *)`
- `int Revert_list (node *)`  
*Reverts the Binomial Heap list.*
- `int Display (node *)`  
*Display the nodes of the Binomial Heap.*
- `node * Search (node *, int)`  
*Search for a node in the given Binomial Heap.*
- `int Decrease_key (node *, int, int)`  
*Decrease the key value of a given node.*
- `int Delete (node *, int)`  
*Delete a given node from the Binomial Heap.*

### 3.1.1 Detailed Description

Binomial Heap Data Structure class.

### 3.1.2 Member Function Documentation

#### 3.1.2.1 `int BinomialHeap::Binomial_link ( node * y, node * z )`

Adds edges between two nodes of a tree and makes appropriate changes in parent, child, sibling and degree parameters.

##### Parameters

<i>y</i>	pointer of the child node
<i>z</i>	pointer of it's parent node

##### Returns

Returns int

#### 3.1.2.2 `node * BinomialHeap::Create_node ( int k, int ind )`

Creates a new node and assigns value and the index.

##### Parameters

<i>k</i>	Value of the node
<i>ind</i>	index of the new node

##### Returns

Returns the pointer of the newly created node

#### 3.1.2.3 `int BinomialHeap::Decrease_key ( node * H, int i, int k )`

Decrease the key value of a given node.

##### Parameters

<i>H</i>	The pointer to the existing Binomial Heap
<i>i</i>	The value of the node to be decreased
<i>k</i>	The user entered value of the new key

**Returns**

Returns int value

**3.1.2.4 int BinomialHeap::Delete ( node \* *H*, int *k* )**

Delete a given node from the Binomial Heap.

**Parameters**

<i>H</i>	The pointer of Root node of the Heap
<i>k</i>	The value of the node to be deleted

**Returns**

Returns 0 if the node is not found

**3.1.2.5 int BinomialHeap::Display ( node \* *H* )**

Display the nodes of the Binomial Heap.

**Parameters**

<i>H</i>	The pointer to the Binomial Heap
----------	----------------------------------

**Returns**

Returns 0 if the Heap is empty, 1 otherwise

**3.1.2.6 node \* BinomialHeap::Extract\_Min ( node \* *H1* )**

Extract the minimum from the Binomial Heap.

**Parameters**

<i>H1</i>	The pointer to the Binomial Heap
-----------	----------------------------------

**Returns**

Returns the pointer of the new Binomial Heap Root

**3.1.2.7 node \* BinomialHeap::Initializeheap ( )**

Initialize the Binomial Heap.

**Returns**

Returns the pointer of the root node

**3.1.2.8 node \* BinomialHeap::Insert ( node \* *H*, node \* *x* )**

Insertion of a new Binomial Tree in the Binomial Heap.

**Parameters**

<i>H</i>	The pointer to the existing Heap
<i>x</i>	The pointer to the new Binomial Tree

**Returns**

Returns the pointer of the final Binomial Heap

**3.1.2.9 node \* BinomialHeap::Merge ( node \* *H1*, node \* *H2* )**

Merging of the nodes of the given Binomial Heap.

**Parameters**

<i>H1</i>	The pointer to the existing Heap
<i>H2</i>	The pointer to the new Binomial Heap

**Returns**

Returns the pointer of the final Binomial Heap

**3.1.2.10 int BinomialHeap::Revert\_list ( node \* *y* )**

Reverts the Binomial Heap list.

**Parameters**

<i>y</i>	The pointer of the given node
----------	-------------------------------

**Returns**

Returns int value

**3.1.2.11 node \* BinomialHeap::Search ( node \* *H*, int *k* )**

Search for a node in the given Binomial Heap.

## Parameters

<i>H</i>	The pointer of the Heap, passed the value of the Root node
<i>k</i>	The value of the node to be Searched For

## Returns

Returns the pointer of the required node if found, else NULL otherwise

3.1.2.12 `node * BinomialHeap::Union ( node * H1, node * H2 )`

Union of the the two Binomial Heaps.

## Parameters

<i>H1</i>	The pointer to the existing Heap
<i>H2</i>	The pointer to the new Binomial Heap

## Returns

Returns the pointer of the final Binomial Heap

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

- [dijkstra.cpp](#)

## 3.2 FibonacciHeap Class Reference

Fibonacci Heap Data Structure class.

### Public Member Functions

- `nodef * InitializeHeap ()`  
*Initialization of the FIBONACCI Heap.*
- `int Fibonnaci_link (nodef *, nodef *, nodef *)`  
*Linking the nodes in a FIBONACCI Heap.*
- `nodef * Create_node (int, int)`  
*Creation of a new Node of a FIBONACCI Heap.*
- `nodef * Insert (nodef *, nodef *)`  
*Inserting new node to the existing FIBONACCI Heap.*
- `nodef * Union (nodef *, nodef *)`  
*Unifying FIBONACCI Heaps.*
- `nodef * Extract_Min (nodef *)`  
*Extract the Min Node from the existing FIBONACCI Heap.*
- `void pop ()`

- Pops the node with the min value in the heap.*
- int [top](#) ()  
*Returns the index of the node with the least value in the Heap.*
- int [Consolidate](#) (nodef \*)  
*Consolidate the node in FIBONACCI Heap.*
- int [Display](#) (nodef \*)  
*Displays the elements of the FIBONACCI Heap.*
- nodef \* [Find](#) (nodef \*, int)  
*Finds a node with value k in the FIBONACCI Heap.*
- int [Decrease\\_key](#) (nodef \*, int, int)  
*Decrease the key value of a given node.*
- int [Delete\\_key](#) (nodef \*, int)  
*Deletes a node in FIBONACCI Heap.*
- int [Cut](#) (nodef \*, nodef \*, nodef \*)  
*Cuts the FIBONACCI Heap.*
- int [Cascase\\_cut](#) (nodef \*, nodef \*)  
*Cascade the Fibonnaci Heap cuts.*

### 3.2.1 Detailed Description

Fibonacci Heap Data Structure class.

### 3.2.2 Member Function Documentation

#### 3.2.2.1 int FibonacciHeap::Cascase\_cut ( nodef \* H1, nodef \* y )

Cascade the Fibonnaci Heap cuts.

##### Parameters

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
<i>y</i>	The pointer to the another FIBONACCI Heap

##### Returns

Returns int value

#### 3.2.2.2 int FibonacciHeap::Consolidate ( nodef \* H1 )

Consolidate the node in FIBONACCI Heap.

##### Parameters

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
-----------	--



**Returns**

Returns int value

**3.2.2.3    `nodef * FibonacciHeap::Create_node ( int value, int ind )`**

Creation of a new Node of a FIBONACCI Heap.

**Parameters**

<i>value</i>	The value of the new node to be inserted
<i>ind</i>	The index of the new node to be inserted

**Returns**

Returns the pointer to the newly created node

**3.2.2.4    `int FibonacciHeap::Cut ( nodef * H1, nodef * x, nodef * y )`**

Cuts the FIBONACCI Heap.

**Parameters**

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
<i>x</i>	The pointer to the another FIBONACCI Heap
<i>y</i>	The pointer to the third FIBONACCI Heap

**Returns**

Returns int value

**3.2.2.5    `int FibonacciHeap::Decrease_key ( nodef * H1, int x, int k )`**

Decrease the key value of a given node.

**Parameters**

<i>H1</i>	The pointer to the existing FIBONACCI Heap
<i>x</i>	The value of the node to be decreased
<i>k</i>	The user entered value of the new key

**Returns**

Returns int value

### 3.2.2.6 int FibonacciHeap::Delete\_key ( nodef \* *H1*, int *k* )

Deletes a node in FIBONACCI Heap.

#### Parameters

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
<i>x</i>	The node with value k gets deleted
<i>y</i>	The pointer to the third FIBONACCI Heap

#### Returns

Returns 0 if key not found

### 3.2.2.7 int FibonacciHeap::Display ( nodef \* *H* )

Displays the elements of the FIBONACCI Heap.

#### Parameters

<i>H</i>	The pointer of the Root node of the FIBONACCI Heap
----------	--

#### Returns

Returns 0 if Heap is empty

### 3.2.2.8 nodef \* FibonacciHeap::Extract\_Min ( nodef \* *H1* )

Extract the Min Node from the existing FIBONACCI Heap.

#### Parameters

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
-----------	--

#### Returns

Returns the minimum node

### 3.2.2.9 int FibonacciHeap::Fibonnaci\_link ( nodef \* *H1*, nodef \* *y*, nodef \* *z* )

Linking the nodes in a FIBONACCI Heap.

#### Parameters

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
<i>y</i>	The node to be merged with z
<i>z</i>	Add new Children to this node

**Returns**

Returns the new Root of the FIBONACCI Heap

**3.2.2.10 `nodef * FibonacciHeap::Find ( nodef * H, int k )`**

Finds a node with value *k* in the FIBONACCI Heap.

**Parameters**

<i>H</i>	The pointer of the Root node of the FIBONACCI Heap
<i>k</i>	The value of the node to be found

**Returns**

Returns the pointer to the node if found, NULL otherwise

**3.2.2.11 `nodef * FibonacciHeap::InitializeHeap ( )`**

Initialization of the FIBONACCI Heap.

**Returns**

Returns the pointer to the root of the new FIBONACCI Heap

**3.2.2.12 `nodef * FibonacciHeap::Insert ( nodef * H, nodef * x )`**

Inserting new node to the existing FIBONACCI Heap.

**Parameters**

<i>H</i>	The pointer of the Root node of the FIBONACCI Heap
<i>x</i>	The pointer to the newly created node to be inserted

**Returns**

Returns the new Root of the FIBONACCI Heap

**3.2.2.13 `void FibonacciHeap::pop ( )`**

Pops the node with the min value in the heap.

**Returns**

Doesn't return anything, void

**3.2.2.14** `int FibonacciHeap::top ( )`

Returns the index of the node with the least value in the Heap.

**Returns**

Returns the index of the node with least value

**3.2.2.15** `nodef * FibonacciHeap::Union ( nodef * H1, nodef * H2 )`

Unifying FIBONACCI Heaps.

**Parameters**

<i>H1</i>	The pointer of the Root node of the FIBONACCI Heap
<i>H2</i>	The pointer to the another FIBONACCI Heap

**Returns**

Returns the new Root of the FIBONACCI Heap

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

- [dijkstra.cpp](#)

**3.3 node Struct Reference**

\*-----BINOMIAL HEAP-----\*/

**Public Attributes**

- int **n**
- int [index](#)
- int [degree](#)
- [node](#) \* [parent](#)
- [node](#) \* [child](#)
- [node](#) \* [sibling](#)

**3.3.1 Detailed Description**

\*-----BINOMIAL HEAP-----\*/

Strucure of a node, used to store various info about node.

### 3.3.2 Member Data Documentation

#### 3.3.2.1 `node* node::child`

Pointer to the parent of the given node

#### 3.3.2.2 `int node::degree`

Index of the node

#### 3.3.2.3 `int node::index`

Value of the node

#### 3.3.2.4 `node* node::parent`

Stores the degree of the given node

#### 3.3.2.5 `node* node::sibling`

Pointer to the child of the given node

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

- [dijkstra.cpp](#)

## 3.4 nodef Struct Reference

\*-----FIBONACCI HEAP-----\*/

### Public Attributes

- `int n`
- `int index`
- `int degree`
- `nodef * parent`
- `nodef * child`
- `nodef * left`
- `nodef * right`
- `char mark`
- `char C`

### 3.4.1 Detailed Description

\*-----FIBONACCI HEAP-----\*/

Structure of a node, used to store various info about node in a FIBONACCI Heap.

### 3.4.2 Member Data Documentation

#### 3.4.2.1 **nodef\*** nodef::child

Pointer to parent of the node

#### 3.4.2.2 **int** nodef::degree

Index of the node

#### 3.4.2.3 **int** nodef::index

Stores value of the node

#### 3.4.2.4 **nodef\*** nodef::left

Pointer to child of the node

#### 3.4.2.5 **char** nodef::mark

Pointer to right child of the node

#### 3.4.2.6 **nodef\*** nodef::parent

Degree of the node

#### 3.4.2.7 **nodef\*** nodef::right

Pointer to left child of the node

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

- [dijkstra.cpp](#)

## Chapter 4

# File Documentation

### 4.1 dijkstra.cpp File Reference

Implementation of Dijkstra's algorithm using Binary Heap, Binomial Heap and Fibonacci Heap.

```
#include <bits/stdc++.h>
```

#### Classes

- struct `node`  

```
*-----BINOMIAL HEAP-----*/
```
- class `BinomialHeap`  
*Binomial Heap Data Structure class.*
- struct `nodef`  

```
*-----FIBONACCI HEAP-----*/
```
- class `FibonacciHeap`  
*Fibonacci Heap Data Structure class.*

#### Macros

- `#define MAX 100001`
- `#define INF (1<<20)`
- `#define pii pair< int, int >`
- `#define pb(x) push_back(x)`

## Functions

- int `parent` (int index)  
*Finds the parent of a node in a tree.*
- int `value` (int index)  
*Returns the value of the node for a given index.*
- void `heapifyUpBinaryHeap` (int index)  
*Heapifies the given binary heap after insertion or deletion. Min Heap.*
- void `insertBinaryHeap` (pair< int, int > element)  
*Insertion of a new node in the given Binary Heap.*
- int `getLeft` (int index)  
*Gets the left child index of a given node.*
- int `getRight` (int index)  
*Gets the right child index of a given node.*
- void `heapifyDownBinaryHeap` (int index)  
*Heapify the binary heap after Deletion Operation (Sort)*
- pair< int, int > `deleteBinaryHeap` ()  
*Obtains the lowest value from the heap and deleted the node and then heapifies the binary heap.*
- void `binary_heap` ()  
*Create a Binary Heap.*
- void `binomial_heap` ()  
*Create a Binomial Heap.*
- void `fibonacci_heap` ()  
*Create a FIBONACCI Heap.*
- int `main` ()  
*The Main Function of the Program.*

## Variables

- int `index_node`
- vector< pair< int, int > > `binaryHeap`  
*\*-----BINARY HEAP-----\*/*
- vector< pii > `G` [MAX]
- int `D` [MAX]
- bool `F` [MAX]
- int `i`
- int `u`
- int `v`
- int `w`
- int `sz`
- int `nodes`
- int `edges`
- int `starting`

### 4.1.1 Detailed Description

Implementation of Dijkstra's algorithm using Binary Heap, Binomial Heap and Fibonacci Heap.

#### Date

15 April 2016

#### Author

Shrukul Habib 13CO143  
Anant Maheshwari 13CO111



## 4.1.2 Function Documentation

### 4.1.2.1 void binary\_heap ( )

Create a Binary Heap.

#### Returns

Returns void

### 4.1.2.2 void binomial\_heap ( )

Create a Binomial Heap.

#### Returns

Returns void

### 4.1.2.3 pair<int,int> deleteBinaryHeap ( )

Obtains the lowest value from the heap and deleted the node and then heapifies the binary heap.

#### Returns

Returns the deleted (minimum-valued) node

### 4.1.2.4 void fibonacci\_heap ( )

Create a FIBONACCI Heap.

#### Returns

Returns void

### 4.1.2.5 int getLeft ( int *index* )

Gets the left child index of a given node.

#### Parameters

<i>index</i>	The index of the node whose left child is to be found
--------------	---

**Returns**

Returns the index of the left child of the given node

**4.1.2.6 int getRight ( int *index* )**

Gets the right child index of a given node.

**Parameters**

<i>index</i>	The index of the node whose right child is to be found
--------------	--

**Returns**

Returns the index of the right child of the given node

**4.1.2.7 void heapifyDownBinaryHeap ( int *index* )**

Heapify the binary heap after Deletion Operation (Sort)

**Parameters**

<i>index</i>	The index of the node from which the the heapify operation begins, called the function with parameter 0 initially
--------------	---

**Returns**

Doesn't return anything, void

**4.1.2.8 void heapifyUpBinaryHeap ( int *index* )**

Heapifies the given binary heap after insertion or deletion. Min Heap.

**Parameters**

<i>index</i>	The index of the node where any operation was made
--------------	--

**Returns**

Doesn't return anything, void

**4.1.2.9 void insertBinaryHeap ( pair< int, int > *element* )**

Insertion of a new node in the given Binary Heap.

**Parameters**

<i>element</i>	the value and the index of the new node to be inserted combined in pair data structure
----------------	--

**Returns**

Doesn't return anything, void

**4.1.2.10 int main ( )**

The Main Function of the Program.

**Returns**

Return int value

**4.1.2.11 int parent ( int *index* )**

Finds the parent of a node in a tree.

**Parameters**

<i>index</i>	The index of the node whose parent is to be found
--------------	---

**Returns**

returns the index of the parent if it exist, if the node is the root, then returns -1

**4.1.2.12 int value ( int *index* )**

Returns the value of the node for a given index.

**Parameters**

<i>index</i>	The index of the node whose value is to be found
--------------	--

**Returns**

returns the value of the node for the given index



# Index

- binary\_heap
  - dijkstra.cpp, 19
- binomial\_heap
  - dijkstra.cpp, 19
- Binomial\_link
  - BinomialHeap, 6
- BinomialHeap, 5
  - Binomial\_link, 6
  - Create\_node, 6
  - Decrease\_key, 6
  - Delete, 7
  - Display, 7
  - Extract\_Min, 7
  - Initializeheap, 7
  - Insert, 8
  - Merge, 8
  - Revert\_list, 8
  - Search, 8
  - Union, 9
- Cascase\_cut
  - FibonacciHeap, 10
- child
  - node, 15
  - nodef, 16
- Consolidate
  - FibonacciHeap, 10
- Create\_node
  - BinomialHeap, 6
  - FibonacciHeap, 11
- Cut
  - FibonacciHeap, 11
- Decrease\_key
  - BinomialHeap, 6
  - FibonacciHeap, 11
- degree
  - node, 15
  - nodef, 16
- Delete
  - BinomialHeap, 7
- Delete\_key
  - FibonacciHeap, 11
- deleteBinaryHeap
  - dijkstra.cpp, 19
- dijkstra.cpp, 17
  - binary\_heap, 19
  - binomial\_heap, 19
  - deleteBinaryHeap, 19
  - fibonacci\_heap, 19
  - getLeft, 19
  - getRight, 20
  - heapifyDownBinaryHeap, 20
  - heapifyUpBinaryHeap, 20
  - insertBinaryHeap, 20
  - main, 21
  - parent, 21
  - value, 21
- Display
  - BinomialHeap, 7
  - FibonacciHeap, 12
- Extract\_Min
  - BinomialHeap, 7
  - FibonacciHeap, 12
- fibonacci\_heap
  - dijkstra.cpp, 19
- FibonacciHeap, 9
  - Cascase\_cut, 10
  - Consolidate, 10
  - Create\_node, 11
  - Cut, 11
  - Decrease\_key, 11
  - Delete\_key, 11
  - Display, 12
  - Extract\_Min, 12
  - Fibonnaci\_link, 12
  - Find, 13
  - InitializeHeap, 13
  - Insert, 13
  - pop, 13
  - top, 13
  - Union, 14
- Fibonnaci\_link
  - FibonacciHeap, 12
- Find
  - FibonacciHeap, 13
- getLeft
  - dijkstra.cpp, 19
- getRight
  - dijkstra.cpp, 20
- heapifyDownBinaryHeap
  - dijkstra.cpp, 20
- heapifyUpBinaryHeap
  - dijkstra.cpp, 20
- index
  - node, 15

