

Atishay's Project

Generated by Doxygen 1.9.6

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 BinarySearchTree Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Enumeration Documentation	6
3.1.2.1 order	6
3.1.3 Constructor & Destructor Documentation	6
3.1.3.1 BinarySearchTree()	6
3.1.4 Member Function Documentation	6
3.1.4.1 height()	6
3.1.4.2 insert()	7
3.1.4.3 traverse()	7
3.2 BSTNode Class Reference	7
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 BSTNode()	8
3.3 DoublyLinkedList Class Reference	8
3.3.1 Detailed Description	9
3.3.2 Constructor & Destructor Documentation	9
3.3.2.1 DoublyLinkedList()	9
3.3.3 Member Function Documentation	9
3.3.3.1 insert()	9
3.3.3.2 printer()	9
3.3.3.3 reverse()	10
3.4 DoublyLinkedListNode Class Reference	10
3.4.1 Detailed Description	10
3.4.2 Constructor & Destructor Documentation	11
3.4.2.1 DoublyLinkedListNode() [1/2]	11
3.4.2.2 DoublyLinkedListNode() [2/2]	11
3.5 Heap Class Reference	11
3.5.1 Detailed Description	12
3.5.2 Constructor & Destructor Documentation	12
3.5.2.1 Heap()	12
3.5.3 Member Function Documentation	13
3.5.3.1 deleteMin()	13
3.5.3.2 Heapify()	13
3.5.3.3 insert()	13
3.5.3.4 left()	13

3.5.3.5 min()	14
3.5.3.6 parent()	14
3.5.3.7 right()	14
3.6 SinglyLinkedList Class Reference	15
3.6.1 Detailed Description	15
3.6.2 Constructor & Destructor Documentation	16
3.6.2.1 SinglyLinkedList()	16
3.6.3 Member Function Documentation	16
3.6.3.1 deleteVal()	16
3.6.3.2 find()	16
3.6.3.3 insert()	17
3.6.3.4 printer()	17
3.6.3.5 reverse()	17
3.7 SinglyLinkedListNode Class Reference	17
3.7.1 Detailed Description	18
3.7.2 Constructor & Destructor Documentation	18
3.7.2.1 SinglyLinkedListNode() [1/2]	18
3.7.2.2 SinglyLinkedListNode() [2/2]	18
3.8 Trie Class Reference	19
3.8.1 Detailed Description	19
3.8.2 Member Function Documentation	19
3.8.2.1 checkPrefix()	19
3.8.2.2 countPrefix()	20
3.8.2.3 find()	20
3.8.2.4 insert()	20
3.8.3 Member Data Documentation	21
3.8.3.1 count	21
4 File Documentation	23
4.1 DSA.cpp File Reference	23
4.1.1 Detailed Description	24
4.1.2 Function Documentation	24
4.1.2.1 merge()	24
4.1.2.2 operator<<() [1/3]	25
4.1.2.3 operator<<() [2/3]	25
4.1.2.4 operator<<() [3/3]	25
Index	27

Chapter 1

Class Index

1.1 Class List

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

BinarySearchTree	Binary Search Tree Data Structure	5
BSTNode	Node of a Binary Search Tree	7
DoublyLinkedList	Doubly Linked List Data Structure	8
DoublyLinkedListNode	Node of a Doubly Linked List	10
Heap	This class implements the Heap Data Structure	11
SinglyLinkedList	Singly Linked List Data Structure	15
SinglyLinkedListNode	Node of a Singly Linked List. Stores Data stored in a Node and a pointer to its next Node . . .	17
Trie	The Trie Data Structure	19

Chapter 2

File Index

2.1 File List

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

[DSA.cpp](#)

DSA Library consisting of implementation of several Data Structures including Binary Search Tree, Linked Lists (Single and Double) and [Trie](#) [23](#)

Chapter 3

Class Documentation

3.1 BinarySearchTree Class Reference

Binary Search Tree Data Structure.

Public Types

- enum `order` { `PRE` , `IN` , `POST` }

Order of Traversal in BST.

Public Member Functions

- `BinarySearchTree` ()
Default Constructor.
- void `insert` (ll val)
Insert a New Node in BST with a given Value.
- void `traverse` (BSTNode *T, order tt)
Print the Values stored in BST in a given Traversal Order.
- ll `height` (BSTNode *T)
Function to get the Height of a given Node. Height here is defined as the longest distance of a Node to a leaf.

Public Attributes

- `BSTNode` * `root`
Root Node of BST.

3.1.1 Detailed Description

Binary Search Tree Data Structure.

BST is a Binary Tree Data Structure in which -

1. Left Subtree of each Node stores Nodes with values lesser than the Node
2. Right Subtree of each Node stores Nodes with values greater than the Node
3. Left and Right Subtree each must also be a Binary Search Tree

3.1.2 Member Enumeration Documentation

3.1.2.1 order

```
enum BinarySearchTree::order
```

Order of Traversal in BST.

Enumerator

PRE	Current Node, left, right.
IN	left, Current Node, right
POST	left, right, Current Node

3.1.3 Constructor & Destructor Documentation

3.1.3.1 BinarySearchTree()

```
BinarySearchTree::BinarySearchTree ( ) [inline]
```

Default Constructor.

Initializes root to NULL

3.1.4 Member Function Documentation

3.1.4.1 height()

```
11 BinarySearchTree::height (
    BSTNode * T ) [inline]
```

Function to get the Height of a given Node. Height here is defined as the longest distance of a Node to a leaf.

Parameters

<i>T</i>	- Node whose height is to be found
----------	------------------------------------

Returns

Height of the given Node

3.1.4.2 insert()

```
void BinarySearchTree::insert (
    ll val ) [inline]
```

Insert a New Node in BST with a given Value.

Parameters

<i>val</i>	- Value to be stored in New Node
------------	----------------------------------

3.1.4.3 traverse()

```
void BinarySearchTree::traverse (
    BSTNode * T,
    order tt ) [inline]
```

Print the Values stored in BST in a given Traversal Order.

Parameters

<i>T</i>	- Node starting from which the Tree is to be printed
<i>tt</i>	- Order of Traversal of Tree. It can be - <ol style="list-style-type: none"> 1. Pre Order 2. In Order 3. Post Order

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

- [DSA.cpp](#)

3.2 BSTNode Class Reference

Node of a Binary Search Tree.

Public Member Functions

- [BSTNode](#) (ll val)
Construct a new [BSTNode](#) object with a given value stored in it.

Public Attributes

- **ll info**
Data stored in the Node.
- **ll level**
Height of the Node.
- **BSTNode * left**
Pointer to the Left Child of the Node.
- **BSTNode * right**
Pointer to the Right Child of the Node.

3.2.1 Detailed Description

Node of a Binary Search Tree.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BSTNode()

```
BSTNode::BSTNode (
    ll val ) [inline]
```

Construct a new **BSTNode** object with a given value stored in it.

Parameters

val	- Data to be stored in the BSTNode
------------	---

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

- [DSA.cpp](#)

3.3 DoublyLinkedList Class Reference

Doubly Linked List Data Structure.

Public Member Functions

- **DoublyLinkedList ()**
Construct a new Doubly Linked List object.
- void **insert** (ll data)
Insert new Node with a given data into the Linked List.
- void **printer** (string sep=" ", "
Function to print the Linked List with a given separation between the Data stored in each Node.
- void **reverse** ()
Reverse the Order of Elements in the Linked List.

Public Attributes

- `DoublyLinkedListNode * head`
Head Node i.e. The Node at starting Index.
- `DoublyLinkedListNode * tail`
Tail Node i.e. The Node at last Index.

3.3.1 Detailed Description

Doubly Linked List Data Structure.

A Doubly Linked List (DLL) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DoublyLinkedList()

```
DoublyLinkedList::DoublyLinkedList ( ) [inline]
```

Construct a new Doubly Linked List object.

Sets head and tail to NULL

3.3.3 Member Function Documentation

3.3.3.1 insert()

```
void DoublyLinkedList::insert (
    ll data ) [inline]
```

Insert new Node with a given data into the Linked List.

Parameters

<i>data</i>	- Data to be stored into the New Node
-------------	---------------------------------------

3.3.3.2 printer()

```
void DoublyLinkedList::printer (
```

```
string sep = ", " ) [inline]
```

Function to print the Linked List with a given separation between the Data stored in each Node.

Parameters

<code>in</code>	<code>sep</code>	- Separation to be used between Datam of each node while printing the linked list
-----------------	------------------	---

3.3.3.3 reverse()

```
void DoublyLinkedList::reverse ( ) [inline]
```

Reverse the Order of Elements in the Linked List.

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

- [DSA.cpp](#)

3.4 DoublyLinkedListNode Class Reference

Node of a Doubly Linked List.

Public Member Functions

- [DoublyLinkedListNode](#) ()
Default Contructor.
- [DoublyLinkedListNode](#) (ll val)
Construct a new Doubly Linked List Node object with a given value to be stored.

Public Attributes

- [ll data](#)
Data stored in the Node.
- [DoublyLinkedListNode](#) * **next**
Pointer to Next Node.
- [DoublyLinkedListNode](#) * **prev**
Pointer to Previous Node.

3.4.1 Detailed Description

Node of a Doubly Linked List.

It Stores an additional pointer as compared to a Singly Linked List Node, mainly to point to Previous Node as well

3.4.2 Constructor & Destructor Documentation

3.4.2.1 DoublyLinkedListNode() [1/2]

```
DoublyLinkedListNode::DoublyLinkedListNode ( ) [inline]
```

Default Contructor.

It initializes :

- data to -1
- next to NULL
- prev to NULL

3.4.2.2 DoublyLinkedListNode() [2/2]

```
DoublyLinkedListNode::DoublyLinkedListNode (
    ll val ) [inline]
```

Construct a new Doubly Linked List Node object with a given value to be stored.

Parameters

<i>val</i>	- The value to be stored in the Node
------------	--------------------------------------

It initializes :

- data to val
- next to NULL
- prev to NULL

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

- [DSA.cpp](#)

3.5 Heap Class Reference

This class implements the [Heap](#) Data Structure.

Public Member Functions

- [Heap](#) (int cap)
Construct a new [Heap](#) object.
- int [parent](#) (int i)
Function to get the Parent Node of an element in [Heap](#).
- int [left](#) (int i)
Function to get the Left child of a Node in [Heap](#).
- int [right](#) (int i)
Function to get the Right child of a Node in [Heap](#).
- void [insert](#) (int val)
Inserts a new Node into the [Heap](#) with a given value.
- int [min](#) ()
Function to get the Minimum value stored in the [Heap](#).
- void [Heapify](#) (int root)
Heapify is a process of creating a [Heap](#) from a list of elements or Restoring the [Heap](#) property if it is violated at any Node. This function implements Heapify on the present [Heap](#) from a given node.
- void [deleteMin](#) ()
Deletes the Minimum Element and Heapifies the heap after deleting the minimum element.

Public Attributes

- int **n**
Number of elements in current [Heap](#).
- int **Cap**
Maximum Capacity of [Heap](#).
- int * **arr**
Array in which [Heap](#) elements are stored.

3.5.1 Detailed Description

This class implements the [Heap](#) Data Structure.

A [Heap](#) is a special Tree-based data structure in which the tree is a Binary Tree that stores priorities (or priority-element) pairs at nodes.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 [Heap](#)()

```
Heap::Heap (
    int cap ) [inline]
```

Construct a new [Heap](#) object.

Parameters

<i>cap</i>	- The Maximum Capacity of Heap
------------	--

3.5.3 Member Function Documentation

3.5.3.1 deleteMin()

```
void Heap::deleteMin ( ) [inline]
```

Deletes the Minimum Element and Heapifies the heap after deleting the minimum element.

3.5.3.2 Heapify()

```
void Heap::Heapify (
    int root ) [inline]
```

Heapify is a process of creating a [Heap](#) from a list of elements or Restoring the [Heap](#) property if it is violated at any Node. This function implements Heapify on the present [Heap](#) from a given node.

Parameters

<i>root</i>	Index of Node, starting from which Heap will be made
-------------	--

3.5.3.3 insert()

```
void Heap::insert (
    int val ) [inline]
```

Inserts a new Node into the [Heap](#) with a given value.

Parameters

<i>val</i>	The value to be stored in the New Node
------------	--

3.5.3.4 left()

```
int Heap::left (
```

```
int i ) [inline]
```

Function to get the Left child of a Node in [Heap](#).

Parameters

<i>i</i>	- Index of element whose Left child is to be found
----------	--

Returns

Index of the Left child

3.5.3.5 min()

```
int Heap::min ( ) [inline]
```

Function to get the Minimum value stored in the [Heap](#).

Returns

Minimum value stored in [Heap](#)

3.5.3.6 parent()

```
int Heap::parent (
    int i ) [inline]
```

Function to get the Parent Node of an element in [Heap](#).

Parameters

<i>i</i>	- Index of element whose Parent Node is to be found
----------	---

Returns

Index of the Parent Node

3.5.3.7 right()

```
int Heap::right (
    int i ) [inline]
```

Function to get the Right child of a Node in [Heap](#).

Parameters

<i>i</i>	- Index of element whose Right child is to be found
----------	---

Returns

Index of the Right child

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

- [DSA.cpp](#)

3.6 SinglyLinkedList Class Reference

Singly Linked List Data Structure.

Public Member Functions

- [SinglyLinkedList](#) ()
Construtor of a Singly Linked List.
- void [insert](#) (ll data)
Function to insert a New Node into the Linked List.
- [SinglyLinkedListNode](#) * [find](#) (ll data)
Find Previous Node of a Node with a given data.
- bool [deleteVal](#) (ll data)
Delete the Node with given data.
- void [printer](#) (string sep=", ")
Function to print the Linked List with a given separation between the Data stored in each Node.
- void [reverse](#) ()
Reverse the Order of Elements in the Linked List.

Public Attributes

- [SinglyLinkedListNode](#) * **head**
Head Node of the Linked List.
- [SinglyLinkedListNode](#) * **tail**
Tail Node of the Linked List.

3.6.1 Detailed Description

Singly Linked List Data Structure.

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 SinglyLinkedList()

```
SinglyLinkedList::SinglyLinkedList ( ) [inline]
```

Construtor of a Singly Linked List.

Sets head and tail to NULL

3.6.3 Member Function Documentation

3.6.3.1 deleteVal()

```
bool SinglyLinkedList::deleteVal (
    ll data ) [inline]
```

Delete the Node with given data.

Parameters

in	<i>data</i>	- data stored in the Node which is to be deleted
----	-------------	--

Returns

- true - If data found in the Linked List and deleted
- false - If data given is not found in the Linked List

3.6.3.2 find()

```
SinglyLinkedListNode * SinglyLinkedList::find (
    ll data ) [inline]
```

Find Previous Node of a Node with a given data.

Parameters

in	<i>data</i>	- value of data to be found
----	-------------	-----------------------------

Returns

Previous Node of the Node with given data (if found), else returns NULL

3.6.3.3 insert()

```
void SinglyLinkedList::insert (
    ll data ) [inline]
```

Function to insert a New Node into the Linked List.

Parameters

in	data	- value of Data to be stored in the new Node Inserted
----	------	---

3.6.3.4 printer()

```
void SinglyLinkedList::printer (
    string sep = ", " ) [inline]
```

Function to print the Linked List with a given separation between the Data stored in each Node.

Parameters

in	sep	- Separation to be used between Datam of each node while printing the linked list
----	-----	---

3.6.3.5 reverse()

```
void SinglyLinkedList::reverse ( ) [inline]
```

Reverse the Order of Elements in the Linked List.

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

- [DSA.cpp](#)

3.7 SinglyLinkedListNode Class Reference

Node of a Singly Linked List. Stores Data stored in a Node and a pointer to its next Node.

Public Member Functions

- [SinglyLinkedListNode](#) ()
Default Constructor.
- [SinglyLinkedListNode](#) (ll val)
Constructor with a parameter.

Public Attributes

- **ll data**
Data stored in a Node.
- [SinglyLinkedListNode](#) * **next**
Pointer to next Node.

3.7.1 Detailed Description

Node of a Singly Linked List. Stores Data stored in a Node and a pointer to its next Node.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 [SinglyLinkedListNode](#)() [1/2]

```
SinglyLinkedListNode::SinglyLinkedListNode ( ) [inline]
```

Default Constructor.

Initializes "data" stored to -1 and "next" pointer to NULL

3.7.2.2 [SinglyLinkedListNode](#)() [2/2]

```
SinglyLinkedListNode::SinglyLinkedListNode (
    ll val ) [inline]
```

Constructor with a parameter.

Parameters

<i>val</i>	- value of data to be stored in the Node
------------	--

Initializes "data" stored to "val" and "next" pointer to NULL

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

- [DSA.cpp](#)

3.8 Trie Class Reference

The [Trie](#) Data Structure.

Public Member Functions

- **Trie ()**
Default Constructor to construct a new [Trie](#) object.
- bool **find** ([Trie](#) *T, char c)
Find a particular character in [Trie](#).
- void **insert** (string s)
Insert a new word into [Trie](#).
- bool **checkPrefix** (string s)
Function to check if a given prefix is present in the [Trie](#).
- **ll countPrefix** (string s)
Function to count number of Prefix in [Trie](#) of a string.

Public Attributes

- **ll count**
Number of total Nodes in [Trie](#).
- map< char, [Trie](#) * > **nodes**
Map of [Trie](#) Nodes and characters, in which a character is mapped to a [Trie](#) Pointer.

3.8.1 Detailed Description

The [Trie](#) Data Structure.

A [Trie](#) is used for text processing. It is an efficient information retrieval data structure which can store strings and search them optimally

3.8.2 Member Function Documentation

3.8.2.1 checkPrefix()

```
bool Trie::checkPrefix (
    string s ) [inline]
```

Function to check if a given prefix is present in the [Trie](#).

Parameters

s	- The prefix which is to be searched in the Trie
---	--

Returns

true - If the prefix is found
false - If the prefix is not found

3.8.2.2 countPrefix()

```
11 Trie::countPrefix (
    string s ) [inline]
```

Function to count number of Prefix in [Trie](#) of a string.

Parameters

<i>s</i>	- The string whose counts of prefix is to be counted
----------	--

Returns

The number of counts of Prefix

3.8.2.3 find()

```
bool Trie::find (
    Trie * T,
    char c ) [inline]
```

Find a particular character in [Trie](#).

Parameters

in	<i>T</i>	- the Trie Node starting from which it will start finding
in	<i>c</i>	- character which is to be found

Returns

true - If the character is found
false - If the character is not found

3.8.2.4 insert()

```
void Trie::insert (
    string s ) [inline]
```

Insert a new word into [Trie](#).

Parameters

s	- The string of word that is to be inserted
---	---

3.8.3 Member Data Documentation

3.8.3.1 count

```
11 Trie::count
```

Number of total Nodes in [Trie](#).

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

- [DSA.cpp](#)

Chapter 4

File Documentation

4.1 DSA.cpp File Reference

DSA Library consisting of implementation of several Data Structures including Binary Search Tree, Linked Lists (Single and Double) and [Trie](#).

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

Classes

- class [SinglyLinkedListNode](#)
Node of a Singly Linked List. Stores Data stored in a Node and a pointer to its next Node.
- class [SinglyLinkedList](#)
Singly Linked List Data Structure.
- class [DoublyLinkedListNode](#)
Node of a Doubly Linked List.
- class [DoublyLinkedList](#)
Doubly Linked List Data Structure.
- class [BSTNode](#)
Node of a Binary Search Tree.
- class [BinarySearchTree](#)
Binary Search Tree Data Structure.
- class [Trie](#)
The [Trie](#) Data Structure.
- class [Heap](#)
This class implements the [Heap](#) Data Structure.

Macros

- `#define ll long long int`
Defined as a long long integer Data Type.
- `#define vi vector<int>`
A Vector of integer variables.
- `#define vll vector<ll>`
A Vector of long long integer variables

Functions

- ostream & [operator<<](#) (ostream &out, const [SinglyLinkedListNode](#) &node)
Overloaded << operator to print the data of the Singly Linked List Node provided as argument into a file.
- [SinglyLinkedList](#) merge ([SinglyLinkedList](#) list1, [SinglyLinkedList](#) list2)
Function to Merge two Singly Linked Lists into one Singly Linked List.
- ostream & [operator<<](#) (ostream &out, const [DoublyLinkedListNode](#) &node)
Overloaded << operator to print the data of the Doubly Linked List Node provided as argument into a file.
- ostream & [operator<<](#) (ostream &out, const [BSTNode](#) &node)
Overloaded << operator to print the data of the Binary Search Tree Node provided as argument into a file.

4.1.1 Detailed Description

DSA Library consisting of implementation of several Data Structures including Binary Search Tree, Linked Lists (Single and Double) and [Trie](#).

Author

Atishay Jain (atishay@cse.iitb.ac.in)

Version

0.1

Date

2022-09-21

Copyright

Copyright (c) 2022

4.1.2 Function Documentation

4.1.2.1 merge()

```
SinglyLinkedList merge (
    SinglyLinkedList list1,
    SinglyLinkedList list2 )
```

Function to Merge two Singly Linked Lists into one Singly Linked List.

Parameters

<i>list1</i>	- First Linked List to be merged
<i>list2</i>	- Second Linked List to be merged

Returns

[SinglyLinkedList](#) - Merged Singly Linked List

4.1.2.2 operator<<() [1/3]

```
ostream & operator<< (
    ostream & out,
    const BSTNode & node )
```

Overloaded << operator to print the data of the Binary Search Tree Node provided as argument into a file.

Parameters

<i>out</i>	- File in which the Node data is printed
<i>node</i>	- The BSTNode whose data is to be printed

Returns

ostream& The File with the Node's data printed in it

4.1.2.3 operator<<() [2/3]

```
ostream & operator<< (
    ostream & out,
    const DoublyLinkedListNode & node )
```

Overloaded << operator to print the data of the Doubly Linked List Node provided as argument into a file.

Parameters

<i>out</i>	- File in which the Node data is printed
<i>node</i>	- The Doubly Linked List Node whose data is to be printed

Returns

ostream& The File with the Node's data printed in it

4.1.2.4 operator<<() [3/3]

```
ostream & operator<< (
    ostream & out,
    const SinglyLinkedListNode & node )
```

Overloaded << operator to print the data of the Singly Linked List Node provided as argument into a file.

Parameters

<i>in</i>	<i>out</i>	- File in which the Node data is printed
<i>in</i>	<i>node</i>	- The Singly Linked List Node whose data is to be printed

Returns

ostream& The File with the Node's data printed in it

Index

- BinarySearchTree, [5](#)
 - BinarySearchTree, [6](#)
 - height, [6](#)
 - IN, [6](#)
 - insert, [7](#)
 - order, [6](#)
 - POST, [6](#)
 - PRE, [6](#)
 - traverse, [7](#)
- BSTNode, [7](#)
 - BSTNode, [8](#)
- checkPrefix
 - Trie, [19](#)
- count
 - Trie, [21](#)
- countPrefix
 - Trie, [20](#)
- deleteMin
 - Heap, [13](#)
- deleteVal
 - SinglyLinkedList, [16](#)
- DoublyLinkedList, [8](#)
 - DoublyLinkedList, [9](#)
 - insert, [9](#)
 - printer, [9](#)
 - reverse, [10](#)
- DoublyLinkedListNode, [10](#)
 - DoublyLinkedListNode, [11](#)
- DSA.cpp, [23](#)
 - merge, [24](#)
 - operator<<, [25](#)
- find
 - SinglyLinkedList, [16](#)
 - Trie, [20](#)
- Heap, [11](#)
 - deleteMin, [13](#)
 - Heap, [12](#)
 - Heapify, [13](#)
 - insert, [13](#)
 - left, [13](#)
 - min, [14](#)
 - parent, [14](#)
 - right, [14](#)
- Heapify
 - Heap, [13](#)
- height
 - BinarySearchTree, [6](#)
- IN
 - BinarySearchTree, [6](#)
- insert
 - BinarySearchTree, [7](#)
 - DoublyLinkedList, [9](#)
 - Heap, [13](#)
 - SinglyLinkedList, [17](#)
 - Trie, [20](#)
- left
 - Heap, [13](#)
- merge
 - DSA.cpp, [24](#)
- min
 - Heap, [14](#)
- operator<<
 - DSA.cpp, [25](#)
- order
 - BinarySearchTree, [6](#)
- parent
 - Heap, [14](#)
- POST
 - BinarySearchTree, [6](#)
- PRE
 - BinarySearchTree, [6](#)
- printer
 - DoublyLinkedList, [9](#)
 - SinglyLinkedList, [17](#)
- reverse
 - DoublyLinkedList, [10](#)
 - SinglyLinkedList, [17](#)
- right
 - Heap, [14](#)
- SinglyLinkedList, [15](#)
 - deleteVal, [16](#)
 - find, [16](#)
 - insert, [17](#)
 - printer, [17](#)
 - reverse, [17](#)
 - SinglyLinkedList, [16](#)
- SinglyLinkedListNode, [17](#)
 - SinglyLinkedListNode, [18](#)
- traverse

BinarySearchTree, [7](#)
Trie, [19](#)
 checkPrefix, [19](#)
 count, [21](#)
 countPrefix, [20](#)
 find, [20](#)
 insert, [20](#)