

---

**itu.algs4**

**Jul 09, 2020**



---

## Contents:

---

<b>1</b>	<b>itu.algs4.fundamentals package</b>	<b>1</b>
1.1	Submodules	1
1.2	itu.algs4.fundamentals.bag module	1
1.3	itu.algs4.fundamentals.binary_search module	2
1.4	itu.algs4.fundamentals.evaluate module	2
1.5	itu.algs4.fundamentals.java_helper module	2
1.6	itu.algs4.fundamentals.queue module	2
1.7	itu.algs4.fundamentals.stack module	3
1.8	itu.algs4.fundamentals.three_sum module	4
1.9	itu.algs4.fundamentals.three_sum_fast module	4
1.10	itu.algs4.fundamentals.two_sum_fast module	4
1.11	itu.algs4.fundamentals.uf module	4
1.12	Module contents	7
<b>2</b>	<b>itu.algs4.graphs package</b>	<b>9</b>
2.1	Submodules	9
2.2	itu.algs4.graphs.Arbitrage module	9
2.3	itu.algs4.graphs.CPM module	9
2.4	itu.algs4.graphs.acyclic_lp module	9
2.5	itu.algs4.graphs.acyclic_sp module	10
2.6	itu.algs4.graphs.bellman_ford_sp module	10
2.7	itu.algs4.graphs.bipartite module	11
2.8	itu.algs4.graphs.breadth_first_paths module	11
2.9	itu.algs4.graphs.cc module	12
2.10	itu.algs4.graphs.cycle module	13
2.11	itu.algs4.graphs.degrees_of_separation module	14
2.12	itu.algs4.graphs.depth_first_order module	14
2.13	itu.algs4.graphs.depth_first_paths module	15
2.14	itu.algs4.graphs.depth_first_search module	15
2.15	itu.algs4.graphs.digraph module	15
2.16	itu.algs4.graphs.dijkstra_all_pairs_sp module	17
2.17	itu.algs4.graphs.dijkstra_sp module	18
2.18	itu.algs4.graphs.dijkstra_undirected_sp module	18
2.19	itu.algs4.graphs.directed_cycle module	19
2.20	itu.algs4.graphs.directed_dfs module	19
2.21	itu.algs4.graphs.directed_edge module	20

2.22	itu.algs4.graphs.edge module	20
2.23	itu.algs4.graphs.edge_weighted_digraph module	20
2.24	itu.algs4.graphs.edge_weighted_directed_cycle module	21
2.25	itu.algs4.graphs.edge_weighted_directed_cycle_anton module	22
2.26	itu.algs4.graphs.edge_weighted_graph module	22
2.27	itu.algs4.graphs.graph module	23
2.28	itu.algs4.graphs.kosaraju_sharir_scc module	24
2.29	itu.algs4.graphs.kruskal_mst module	25
2.30	itu.algs4.graphs.lazy_prim_mst module	26
2.31	itu.algs4.graphs.prim_mst module	26
2.32	itu.algs4.graphs.symbol_digraph module	27
2.33	itu.algs4.graphs.symbol_graph module	27
2.34	itu.algs4.graphs.topological module	28
2.35	itu.algs4.graphs.transitive_closure module	28
2.36	Module contents	29
<b>3</b>	<b>itu.algs4.searching package</b>	<b>31</b>
3.1	Submodules	31
3.2	itu.algs4.searching.binary_search_st module	31
3.3	itu.algs4.searching.bst module	34
3.4	itu.algs4.searching.file_index module	35
3.5	itu.algs4.searching.frequency_counter module	35
3.6	itu.algs4.searching.linear_probing_hst module	35
3.7	itu.algs4.searching.lookup_csv module	36
3.8	itu.algs4.searching.lookup_index module	36
3.9	itu.algs4.searching.red_black_bst module	36
3.10	itu.algs4.searching.seperate_chaining_hst module	38
3.11	itu.algs4.searching.sequential_search_st module	39
3.12	itu.algs4.searching.set module	40
3.13	itu.algs4.searching.sparse_vector module	41
3.14	itu.algs4.searching.st module	42
3.15	Module contents	42
<b>4</b>	<b>itu.algs4.sorting package</b>	<b>43</b>
4.1	Submodules	43
4.2	itu.algs4.sorting.heap module	43
4.3	itu.algs4.sorting.index_min_pq module	43
4.4	itu.algs4.sorting.insertion_sort module	44
4.5	itu.algs4.sorting.max_pq module	45
4.6	itu.algs4.sorting.merge module	45
4.7	itu.algs4.sorting.merge_bu module	46
4.8	itu.algs4.sorting.min_pq module	46
4.9	itu.algs4.sorting.quick3way module	46
4.10	itu.algs4.sorting.quicksort module	47
4.11	itu.algs4.sorting.selection module	47
4.12	itu.algs4.sorting.shellsort module	47
4.13	Module contents	48
<b>5</b>	<b>itu.algs4.stdlib package</b>	<b>49</b>
5.1	Submodules	49
5.2	itu.algs4.stdlib.binary_out module	49
5.3	itu.algs4.stdlib.binary_stdin module	49
5.4	itu.algs4.stdlib.binary_stdout module	50
5.5	itu.algs4.stdlib.color module	50

5.6	itu.algs4.stdlib.instream module . . . . .	51
5.7	itu.algs4.stdlib.outstream module . . . . .	52
5.8	itu.algs4.stdlib.picture module . . . . .	52
5.9	itu.algs4.stdlib.stdarray module . . . . .	53
5.10	itu.algs4.stdlib.stdaudio module . . . . .	54
5.11	itu.algs4.stdlib.stddraw module . . . . .	54
5.12	itu.algs4.stdlib.stdio module . . . . .	55
5.13	itu.algs4.stdlib.stdrandom module . . . . .	57
5.14	itu.algs4.stdlib.stdstats module . . . . .	57
5.15	Module contents . . . . .	58
<b>6</b>	<b>itu.algs4.strings package</b>	<b>59</b>
6.1	Submodules . . . . .	59
6.2	itu.algs4.strings.boyer_moore module . . . . .	59
6.3	itu.algs4.strings.huffman_compression module . . . . .	59
6.4	itu.algs4.strings.kmp module . . . . .	60
6.5	itu.algs4.strings.lsd module . . . . .	60
6.6	itu.algs4.strings.lzw module . . . . .	60
6.7	itu.algs4.strings.msd module . . . . .	61
6.8	itu.algs4.strings.nfa module . . . . .	61
6.9	itu.algs4.strings.quick3string module . . . . .	61
6.10	itu.algs4.strings.rabin_karp module . . . . .	62
6.11	itu.algs4.strings.trie_st module . . . . .	62
6.12	itu.algs4.strings.tst module . . . . .	63
6.13	Module contents . . . . .	63
<b>7</b>	<b>Indices and tables</b>	<b>65</b>
	<b>Python Module Index</b>	<b>67</b>
	<b>Index</b>	<b>69</b>



## 1.1 Submodules

## 1.2 itu.algs4.fundamentals.bag module

**class** `itu.algs4.fundamentals.bag.Bag`

Bases: `typing.Generic`

The Bag class represents a bag (or multiset) of generic items. It supports insertion and iterating over the items in arbitrary order.

This implementation uses a singly linked list with a static nested class Node. See `LinkedBag` for the version from the textbook that uses a non-static nested class.

The `add`, `is_empty`, and `size` operations take constant time. Iteration takes time proportional to the number of items.

**class** `Node`

Bases: `typing.Generic`

**add** (*item*: *T*) → None

Adds the item to this bag.

**Parameters** *item* – the item to add to this bag

**is\_empty** () → bool

Returns true if this bag is empty.

**Returns** true if this bag is empty false otherwise

**size** () → int

Returns the number of items in this bag.

**Returns** the number of items in this bag

## 1.3 itu.algs4.fundamentals.binary\_search module

`itu.algs4.fundamentals.binary_search.T = ~T`

The `binary_search` module provides a method for binary searching for an item in a sorted array. The `index_of` operation takes logarithmic time in the worst case.

`itu.algs4.fundamentals.binary_search.index_of(a: List[T], key: T)`

Returns the index of the specified key in the specified array. :param a: the array of items, must be sorted in ascending order :param key: the search key :return: index of key in array if present -1 otherwise

`itu.algs4.fundamentals.binary_search.main()`

Reads strings from first input file and sorts them Reads strings from second input file and prints every string not in first input file

## 1.4 itu.algs4.fundamentals.evaluate module

`itu.algs4.fundamentals.evaluate.evaluate()`

## 1.5 itu.algs4.fundamentals.java\_helper module

`itu.algs4.fundamentals.java_helper.java_string_hash(key)`

If key is a string, compute its java .hash() code. Taken from [http://garage.pimentech.net/libcommonPython\\_src\\_python\\_libcommon\\_javastringhashcode/](http://garage.pimentech.net/libcommonPython_src_python_libcommon_javastringhashcode/)

`itu.algs4.fundamentals.java_helper.trailing_zeros(i)`

## 1.6 itu.algs4.fundamentals.queue module

**class** `itu.algs4.fundamentals.queue.Node` (*item: T, next: Optional[Node[T]]*)

Bases: `typing.Generic`

**class** `itu.algs4.fundamentals.queue.Queue`

Bases: `typing.Generic`

The `Queue` class represents a first-in-first-out (FIFO) queue of generic items. It supports the usual enqueue and dequeue operations, along with methods for peeking at the first item, testing if the queue is empty, and iterating through the items in FIFO order This implementation uses a singly linked list of linked-list nodes The enqueue, dequeue, peek, size, and is\_empty operations all take constant time in the worst case

**dequeue** () → T

Removes and returns the item on this queue that was least recently added. :return: the item on this queue that was least recently added. :raises `NoSuchElementException`: if this queue is empty

**enqueue** (*item: T*) → None

Adds the item to this queue. :param item: the item to add

**is\_empty** () → bool

Returns true if this queue is empty. :return: True if this queue is empty otherwise False :rtype: bool

**peek** () → T

Returns the item least recently added to this queue. :return: the item least recently added to this queue :raises `NoSuchElementException`: if this queue is empty



**size()** → int

Returns the number of items in this queue. :return: the number of items in this queue :rtype: int

## 1.7 itu.algs4.fundamentals.stack module

**class** itu.algs4.fundamentals.stack.**FixedCapacityStack** (*capacity: int*)

Bases: typing.Generic

**is\_empty()** → bool

**pop()** → T

**push** (*item: T*)

**size()** → int

**class** itu.algs4.fundamentals.stack.**Node**

Bases: typing.Generic

**class** itu.algs4.fundamentals.stack.**ResizingArrayStack**

Bases: typing.Generic

**is\_empty()** → bool

**pop()** → T

**push** (*item: T*) → None

**resize** (*capacity: int*) → None

**size()** → int

**class** itu.algs4.fundamentals.stack.**Stack**

Bases: typing.Generic

The Stack class represents a last-in-first-out (LIFO) stack of generic items. It supports the usual push and pop operations, along with methods for peeking at the top item, testing if the stack is empty, and iterating through the items in LIFO order.

This implementation uses a singly linked list with a static nested class for linked-list nodes. See `LinkedStack` for the version from the textbook that uses a non-static nested class. See `ResizingArrayStack` for a version that uses a resizing array. The push, pop, peek, size, and is-empty operations all take constant time in the worst case.

**is\_empty()** → bool

Returns true if this stack is empty.

**Returns** true if this stack is empty false otherwise

**peek()** → T

Returns (but does not remove) the item most recently added to this stack.

**Returns** the item most recently added to this stack

**Raises** **ValueError** – if this stack is empty

**pop()** → T

Removes and returns the item most recently added to this stack.

**Returns** the item most recently added

**Raises** **ValueError** – if this stack is empty

**push** (*item*: *T*) → None

Adds the item to this stack.

**Parameters** *item* – the item to add

**size** () → int

Returns the number of items in this stack.

**Returns** the number of items in this stack

## 1.8 itu.algs4.fundamentals.three\_sum module

```
class itu.algs4.fundamentals.three_sum.ThreeSum
```

Bases: object

```
    static count (a)
```

## 1.9 itu.algs4.fundamentals.three\_sum\_fast module

```
class itu.algs4.fundamentals.three_sum_fast.ThreeSumFast
```

Bases: object

```
    static count (a)
```

## 1.10 itu.algs4.fundamentals.two\_sum\_fast module

```
class itu.algs4.fundamentals.two_sum_fast.TwoSumFast
```

Bases: object

```
    static count (a)
```

## 1.11 itu.algs4.fundamentals.uf module

The UF module implements several versions of the union-find data structure (also known as the disjoint-sets data type). It supports the union and find operations, along with a connected operation for determining whether two sites are in the same component and a count operation that returns the total number of components.

The union-find data type models connectivity among a set of *n* sites, named 0 through *n*-1. The is-connected-to relation must be an equivalence relation:

- Reflexive: *p* is connected to *p*.
- Symmetric: If *p* is connected to *q*, then *q* is connected to *p*.
- Transitive: If *p* is connected to *q* and *q* is connected to *r*, then *p* is connected to *r*.

```
class itu.algs4.fundamentals.uf.QuickFindUF (n: int)
```

Bases: object

This is an implementation of the union-find data structure - see module documentation for more info.

This implementation uses quick find. Initializing a data structure with *n* sites takes linear time. Afterwards, the find, connected, and count operations take constant time but the union operation takes linear time.

For additional documentation, see Section 1.5 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**connected** ( $p: \text{int}, q: \text{int}$ )  $\rightarrow \text{bool}$

Returns true if the two sites are in the same component.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**Returns** true if the two sites p and q are in the same component; false otherwise

**count** ()

**find** ( $p: \text{int}$ )  $\rightarrow \text{int}$

Returns the component identifier for the component containing site p.

**Parameters** **p** – the integer representing one site

**Returns** the component identifier for the component containing site p

**union** ( $p: \text{int}, q: \text{int}$ )  $\rightarrow \text{None}$

Merges the component containing site p with the component containing site q.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**class** `itu.algs4.fundamentals.uf.QuickUnionUF` ( $n: \text{int}$ )

Bases: `object`

This is an implementation of the union-find data structure - see module documentation for more info.

This implementation uses quick union. Initializing a data structure with n sites takes linear time. Afterwards, the union, find, and connected operations take linear time (in the worst case) and the count operation takes constant time. For alternate implementations of the same API, see `UF`, `QuickFindUF`, and `WeightedQuickUnionUF`.

For additional documentation, see Section 1.5 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**connected** ( $p: \text{int}, q: \text{int}$ )  $\rightarrow \text{bool}$

Returns true if the two sites are in the same component.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**Returns** true if the two sites p and q are in the same component; false otherwise

**count** ()  $\rightarrow \text{int}$

**find** ( $p: \text{int}$ )  $\rightarrow \text{int}$

Returns the component identifier for the component containing site p.

**Parameters** **p** – the integer representing one site

**Returns** the component identifier for the component containing site p

**union** ( $p: \text{int}, q: \text{int}$ )  $\rightarrow \text{None}$

Merges the component containing site p with the component containing site q.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**class** `itu.algs4.fundamentals.uf.UF` (*n: int*)

Bases: `object`

This is an implementation of the union-find data structure - see module documentation for more info.

This implementation uses weighted quick union by rank with path compression by halving. Initializing a data structure with *n* sites takes linear time. Afterwards, the union, find, and connected operations take logarithmic time (in the worst case) and the count operation takes constant time. Moreover, the amortized time per union, find, and connected operation has inverse Ackermann complexity.

For additional documentation, see Section 1.5 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**connected** (*p: int, q: int*) → bool

Returns true if the two sites are in the same component.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**Returns** true if the two sites *p* and *q* are in the same component; false otherwise

**count** () → int

**find** (*p: int*) → int

Returns the component identifier for the component containing site *p*.

**Parameters** **p** – the integer representing one site

**Returns** the component identifier for the component containing site *p*

**union** (*p: int, q: int*) → None

Merges the component containing site *p* with the component containing site *q*.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**class** `itu.algs4.fundamentals.uf.WeightedQuickUnionUF` (*n: int*)

Bases: `object`

This is an implementation of the union-find data structure - see module documentation for more info.

This implementation uses weighted quick union by size (without path compression). Initializing a data structure with *n* sites takes linear time. Afterwards, the union, find, and connected operations take logarithmic time (in the worst case) and the count operation takes constant time. For alternate implementations of the same API, see UF, QuickFindUF, and QuickUnionUF.

For additional documentation, see Section 1.5 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**connected** (*p: int, q: int*) → bool

Returns true if the two sites are in the same component.

**Parameters**

- **p** – the integer representing one site
- **q** – the integer representing the other site

**Returns** true if the two sites  $p$  and  $q$  are in the same component; false otherwise

**count**  $() \rightarrow \text{int}$

**find**  $(p: \text{int}) \rightarrow \text{int}$

Returns the component identifier for the component containing site  $p$ .

**Parameters**  $p$  – the integer representing one site

**Returns** the component identifier for the component containing site  $p$

**union**  $(p: \text{int}, q: \text{int}) \rightarrow \text{None}$

Merges the component containing site  $p$  with the component containing site  $q$ .

**Parameters**

- $p$  – the integer representing one site
- $q$  – the integer representing the other site

## 1.12 Module contents



## 2.1 Submodules

## 2.2 `itu.algs4.graphs.Arbitrage` module

## 2.3 `itu.algs4.graphs.CPM` module

## 2.4 `itu.algs4.graphs.acyclic_lp` module

**class** `itu.algs4.graphs.acyclic_lp.AcyclicLp`(*edge\_weighted\_digraph*, *s*)

Bases: `object`

The `AcyclicLp` class represents a data type for solving the single-source longest paths problem in edge-weighted directed acyclic graphs (DAGs). The edge weights can be positive, negative, or zero.

This implementation uses a topological-sort based algorithm. The constructor takes time proportional to  $V + E$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Each call to `distTo(int)` and `hasPathTo(int)` takes constant time; each call to `pathTo(int)` takes time proportional to the number of edges in the shortest path returned.

For additional documentation, see Section 4.4 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**dist\_to**(*v*)

Returns the length of a longest path from the source vertex *s* to vertex *v*.

**Parameters** *v* – the destination vertex

**Returns** the length of a longest path from the source vertex *s* to vertex *v*; negative infinity if no such path exists

**has\_path\_to**(*v*)

Is there a path from the source vertex *s* to vertex *v*?

**path\_to**(*v*)Returns a longest path from the source vertex *s* to vertex *v*.**Param** the destination vertex**Returns** a longest path from the source vertex *s* to vertex *v* as an iterable of edges, and None if no such path

## 2.5 itu.algs4.graphs.acyclic\_sp module

**class** itu.algs4.graphs.acyclic\_sp.**AcyclicSP**(*G, s*)

Bases: object

The AcyclicSP class represents a data type for solving the single-source shortest paths problem in edge-weighted directed acyclic graphs (DAGs). The edge weights can be positive, negative, or zero.

This implementation uses a topological-sort based algorithm. The constructor takes time proportional to  $V + E$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Each call to `distTo(int)` and `has_path_to(int)` takes constant time each call to `pathTo(int)` takes time proportional to the number of edges in the shortest path returned.

**dist\_to**(*v*)

Returns the length of a shortest path from the source vertex *s* to vertex *v*. :param *v*: the destination vertex :returns: the length of a shortest path from the source vertex *s* to vertex *v*

math.inf if no such path

**Raises ValueError** – unless  $0 \leq v < V$ **has\_path\_to**(*v*)

Is there a path from the source vertex *s* to vertex *v*? :param *v*: the destination vertex :returns: true if there is a path from the source vertex

*s* to vertex *v*, and false otherwise**Raises ValueError** – unless  $0 \leq v < V$ **path\_to**(*v*)

Returns a shortest path from the source vertex *s* to vertex *v*. :param *v*: the destination vertex :returns: a shortest path from the source vertex *s* to vertex *v*

as an iterable of edges, and None if no such path

**Raises ValueError** – unless  $0 \leq v < V$ 

## 2.6 itu.algs4.graphs.bellman\_ford\_sp module

**class** itu.algs4.graphs.bellman\_ford\_sp.**BellmanFordSP**(*G, s*)

Bases: object

**dist\_to**(*v*)**has\_negative\_cycle**()**has\_path\_to**(*v*)**negative\_cycle**()



```

    path_to(v)

itu.algs4.graphs.bellman_ford_sp.main(args)

```

## 2.7 itu.algs4.graphs.bipartite module

```
class itu.algs4.graphs.bipartite.Bipartite(G)
```

Bases: object

The Bipartite class represents a data type for determining whether an undirected graph is bipartite or whether it has an odd-length cycle. The isBipartite operation determines whether the graph is bipartite. If so, the color operation determines a bipartition if not, the oddCycle operation determines a cycle with an odd number of edges.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the isBipartite and color operations take constant time the oddCycle operation takes time proportional to the length of the cycle. See BipartiteX for a nonrecursive version that uses breadth-first search.

```
exception UnsupportedOperationException
```

Bases: Exception

```
color(v)
```

Returns the side of the bipartite that vertex  $v$  is on.

**Parameters**  $v$  – the vertex

**Returns** the side of the bipartition that vertex  $v$  is on two vertices are in the same side of the bipartition if and only if they have the same color

**Raises**

- **IllegalArgumentException** – unless  $0 \leq v < V$
- **UnsupportedOperationException** – if this method is called when the graph is not bipartite

```
is_bipartite()
```

Returns True if the graph is bipartite.

**Returns** True if the graph is bipartite False otherwise

```
odd_cycle()
```

Returns an odd-length cycle if the graph is not bipartite, and None otherwise.

**Returns** an odd-length cycle if the graph is not bipartite (and hence has an odd-length cycle), and None otherwise

## 2.8 itu.algs4.graphs.breadth\_first\_paths module

```
class itu.algs4.graphs.breadth_first_paths.BreadthFirstPaths(G, s)
```

Bases: object

The BreadthFirstPaths class represents a data type for finding shortest paths (number of edges) from a source vertex  $s$  (or a set of source vertices) to every other vertex in a directed or undirected graph.

This implementation uses breadth-first search. The constructor takes time proportional to  $V + E$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Each call to distTo(int) and hasPathTo(int) takes constant time

each call to `pathTo(int)` takes time proportional to the length of the path. It uses extra space (not including the graph) proportional to  $V$ .

**dist\_to**( $v$ )

Returns the number of edges in a shortest path between the source vertex  $s$  (or sources) and vertex  $v$ ?

**Parameters**  $v$  – the vertex

**Returns** the number of edges in a shortest path

**Raises** **ValueError** – unless  $0 \leq v < V$

**has\_path\_to**( $v$ )

Is there a path between the source vertex  $s$  (or sources) and vertex  $v$ ?

**Parameters**  $v$  – the vertex

**Returns** true if there is a path, and False otherwise

**Raises** **ValueError** – unless  $0 \leq v < V$

**path\_to**( $v$ )

Returns a shortest path between the source vertex  $s$  (or sources) and  $v$ , or null if no such path.

**Parameters**  $v$  – the vertex

**Returns** the sequence of vertices on a shortest path, as an Iterable

**Raises** **ValueError** – unless  $0 \leq v < V$

**class** `itu.algs4.graphs.breadth_first_paths.BreadthFirstPathsBook`( $G, s$ )

Bases: object

**has\_path\_to**( $v$ )

**path\_to**( $v$ )

## 2.9 itu.algs4.graphs.cc module

**class** `itu.algs4.graphs.cc.CC`( $G$ )

Bases: object

The CC class represents a data type for determining the connected components in an undirected graph. The `id` operation determines in which connected component a given vertex lies the `connected` operation determines whether two vertices are in the same connected component the `count` operation determines the number of connected components and the `size` operation determines the number of vertices in the connected component containing a given vertex.

The component identifier of a connected component is one of the vertices in the connected component: two vertices have the same component identifier if and only if they are in the same connected component.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `id`, `count`, `connected`, and `size` operations take constant time.

**connected**( $v, w$ )

Returns true if vertices  $v$  and  $w$  are in the same connected component.

**Parameters**

- $v$  – one vertex
- $w$  – the other vertex

**Returns** True if vertices  $v$  and  $w$  are in the same connected component; False otherwise

**Raises**

- **ValueError** – unless  $0 \leq v < V$
- **ValueError** – unless  $0 \leq w < V$

**count** ()

Returns the number of connected components in the graph  $G$ .

**Returns** the number of connected components in the graph  $G$

**id** ( $v$ )

Returns the component id of the connected component containing vertex  $v$ .

**Parameters**  $v$  – the vertex

**Returns** the component id of the connected component containing vertex  $v$

**Raises** **ValueError** – unless  $0 \leq v < V$

**size** ( $v$ )

Returns the number of vertices in the connected component containing vertex  $v$ .

**Parameters**  $v$  – the vertex

**Returns** the number of vertices in the connected component containing vertex  $v$

**Raises** **ValueError** – unless  $0 \leq v < V$

**class** `itu.algs4.graphs.cc.CCBook` ( $G$ )

Bases: `object`

**connected** ( $v, w$ )

**count** ()

**id** ( $v$ )

## 2.10 itu.algs4.graphs.cycle module

**class** `itu.algs4.graphs.cycle.Cycle` ( $G$ )

Bases: `object`

The Cycle class represents a data type for determining whether an undirected graph has a cycle. The `hasCycle` operation determines whether the graph has a cycle and, if so, the cycle operation returns one.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `hasCycle` operation takes constant time the cycle operation takes time proportional to the length of the cycle.

**cycle** ()

Returns a cycle in the graph  $G$ .

**Returns** a cycle if the graph  $G$  has a cycle, and null otherwise

**has\_cycle** ()

Returns true if the graph  $G$  has a cycle.

**Returns** true if the graph has a cycle false otherwise

## 2.11 itu.algs4.graphs.degrees\_of\_separation module

**class** `itu.algs4.graphs.degrees_of_separation.DegreesOfSeparation`

Bases: `object`

The `DegreesOfSeparation` class provides a client for finding the degree of separation between one distinguished individual and every other individual in a social network. As an example, if the social network consists of actors in which two actors are connected by a link if they appeared in the same movie, and Kevin Bacon is the distinguished individual, then the client computes the Kevin Bacon number of every actor in the network.

The running time is proportional to the number of individuals and connections in the network. If the connections are given implicitly, as in the movie network example (where every two actors are connected if they appear in the same movie), the efficiency of the algorithm is improved by allowing both movie and actor vertices and connecting each movie to all of the actors that appear in that movie.

**static main** (*args*)

Reads in a social network from a file, and then repeatedly reads in individuals from standard input and prints out their degrees of separation. Takes three command-line arguments: the name of a file, a delimiter, and the name of the distinguished individual. Each line in the file contains the name of a vertex, followed by a list of the names of the vertices adjacent to that vertex, separated by the delimiter.

**Parameters** *args* – the command-line arguments

## 2.12 itu.algs4.graphs.depth\_first\_order module

**class** `itu.algs4.graphs.depth_first_order.DepthFirstOrder` (*digraph*)

Bases: `object`

The `DepthFirstOrder` class represents a data type for determining depth-first search ordering of the vertices in a digraph or edge-weighted digraph, including preorder, postorder, and reverse postorder.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the preorder, postorder, and reverse postorder operation takes time proportional to  $V$ .

For additional documentation, see Section 4.2 of *Algorithms*, 4th Edition by Robert Sedgewick and Kevin Wayne.

**post** (*v=None*)

Either returns the postorder number of vertex *v* or, if *v* is `None`, returns the vertices in postorder.

**Parameters** *v* – `None`, or the vertex to return the postorder number of

**Returns** if *v* is `None`, the vertices in postorder, otherwise the postorder number of *v*

**pre** (*v=None*)

Either returns the preorder number of vertex *v* or, if *v* is `None`, returns the vertices in preorder.

**Parameters** *v* – `None`, or the vertex to return the preorder number of

**Returns** if *v* is `None`, the vertices in preorder, otherwise the preorder number of *v*

**reverse\_post** ()

Returns the vertices in reverse postorder.

**Returns** the vertices in reverse postorder, as an iterable of vertices

## 2.13 itu.algs4.graphs.depth\_first\_paths module

**class** `itu.algs4.graphs.depth_first_paths.DepthFirstPaths` ( $G, s$ )

Bases: `object`

The `DepthFirstPaths` class represents a data type for finding paths from a source vertex  $s$  to every other vertex in an undirected graph.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Each call to `hasPathTo(int)` takes constant time each call to `pathTo(int)` takes time proportional to the length of the path. It uses extra space (not including the graph) proportional to  $V$ .

**has\_path\_to** ( $v$ )

Is there a path between the source vertex  $s$  and vertex  $v$ ? :param  $v$ : the vertex :returns: true if there is a path, false otherwise :raises `ValueError`: unless  $0 \leq v < V$

**path\_to** ( $v$ )

Returns a path between the source vertex  $s$  and vertex  $v$ , or `None` if no such path. :param  $v$ : the vertex :returns: the sequence of vertices on a path between the source vertex

$s$  and vertex  $v$ , as an `Iterable`

**Raises** `ValueError` – unless  $0 \leq v < V$

## 2.14 itu.algs4.graphs.depth\_first\_search module

**class** `itu.algs4.graphs.depth_first_search.DepthFirstSearch` ( $G, s$ )

Bases: `object`

The `DepthFirstSearch` class represents a data type for determining the vertices connected to a given source vertex  $s$  in an undirected graph. For versions that find the paths, see `DepthFirstPaths` and `BreadthFirstPaths`.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. It uses extra space (not including the graph) proportional to  $V$ .

**count** ()

Returns the number of vertices connected to the source vertex  $s$ .

**Returns** the number of vertices connected to the source vertex  $s$

**marked** ( $v$ )

Is there a path between the source vertex  $s$  and vertex  $v$ ?

**Parameters**  $v$  – the vertex

**Returns** true if there is a path, false otherwise

**Raises** `ValueError` – unless  $0 \leq v < V$

## 2.15 itu.algs4.graphs.digraph module

This module implements the directed graph data structure described in *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne. For more information, see chapter 4.2 of the book.

```
class itu.algs4.graphs.digraph.Digraph(V)
    Bases: object
```

The Graph class represents an undirected graph of vertices named 0 through  $V - 1$ . It supports the following two primary operations: add an edge to the graph, iterate over all of the vertices adjacent to a vertex. It also provides methods for returning the number of vertices  $V$  and the number of edges  $E$ . Parallel edges and self-loops are permitted. By convention, a self-loop  $v-v$  appears in the adjacency list of  $v$  twice and contributes two to the degree of  $v$ .

This implementation uses an adjacency-lists representation, which is a vertex-indexed array of Bag objects. All operations take constant time (in the worst case) except iterating over the vertices adjacent to a given vertex, which takes time proportional to the number of such vertices.

```
E()
```

Returns the number of edges in this graph.

**Returns** the number of edges in this graph.

```
V()
```

Returns the number of vertices in this graph.

**Returns** the number of vertices in this graph.

```
add_edge(v, w)
```

Adds the undirected edge  $v-w$  to this graph.

**Parameters**

- **v** – one vertex in the edge
- **w** – the other vertex in the edge

**Raises** **ValueError** – unless both  $0 \leq v < V$  and  $0 \leq w < V$

```
adj(v)
```

Returns the vertices adjacent to vertex  $v$ .

**Parameters** **v** – the vertex

**Returns** the vertices adjacent to vertex  $v$ , as an iterable

**Raises** **ValueError** – unless  $0 \leq v < V$

```
degree(v)
```

Returns the degree of vertex  $v$ .

**Parameters** **v** – the vertex

**Returns** the degree of vertex  $v$

**Raises** **ValueError** – unless  $0 \leq v < V$

```
static from_graph(G)
```

Initializes a new graph that is a deep copy of  $G$

**Parameters** **G** – the graph to copy

**Returns** copy of  $G$

```
static from_stream(stream)
```

Initializes a graph from the specified input stream. The format is the number of vertices  $V$ , followed by the number of edges  $E$ , followed by  $E$  pairs of vertices, with each entry separated by whitespace.

**Parameters** **stream** – the input stream

**Returns** new graph from stream

**Raises**

- **ValueError** – if the endpoints of any edge are not in prescribed range
- **ValueError** – if the number of vertices or edges is negative
- **ValueError** – if the input stream is in the wrong format

**reverse()**

Returns the reverse of the digraph.

**Returns** the reverse of the digraph

## 2.16 itu.algs4.graphs.dijkstra\_all\_pairs\_sp module

This module implements a data type for solving the all-pairs shortest paths problem in edge-weighted digraphs where the edge weights are nonnegative.

**class** `itu.algs4.graphs.dijkstra_all_pairs_sp.DijkstraAllPairsSP` (*edge\_weighted\_digraph*)  
 Bases: `object`

This implementation runs Dijkstra’s algorithm from each vertex. The constructor takes time proportional to  $V(E \log V)$  and uses space proportional to  $V^2$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `dist()` and `hasPath()` methods take constant time and the `path()` method takes time proportional to the number of edges in the shortest path returned.

For additional documentation, see Section 4.4 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**dist** (*source*, *target*)

Returns the length of a shortest path from the source vertex to the target vertex.

**Parameters**

- **source** – the source vertex
- **target** – the target vertex

**Returns** the length of a shortest path from the source vertex to the target vertex; `float('inf')` if no such path

**has\_path** (*source*, *target*)

Is there a path from the source vertex to the target vertex?

**Parameters**

- **source** – the source vertex
- **target** – the target vertex

**Returns** True if there is a path from the source to the target, and False otherwise

**path** (*source*, *target*)

Returns a shortest path from source vertex to the target vertex.

**Parameters**

- **source** – the source vertex
- **target** – the destination vertex

**Returns** a shortest path from the source vertex to the target vertex as an iterable of edges, and None if no such path

## 2.17 itu.algs4.graphs.dijkstra\_sp module

**class** itu.algs4.graphs.dijkstra\_sp.DijkstraSP (*G*, *s*)

Bases: object

The DijkstraSP class represents a data type for solving the single-source shortest paths problem in edge-weighted digraphs where the edge weights are nonnegative. This implementation uses Dijkstra's algorithm with a binary heap. The constructor takes time proportional to  $E \log V$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Each call to `dist_to()` and `has_path_to()` takes constant time. Each call to `path_to()` takes time proportional to the number of edges in the shortest path returned.

**dist\_to** (*v*)

Returns the length of a shortest path from the source vertex *s* to vertex *v*. :param *v*: the destination vertex :return: the length of a shortest path from the source vertex *s* to vertex *v* :rtype: float :raises IllegalArgumentException: unless  $0 \leq v < V$

**has\_path\_to** (*v*)

Returns True if there is a path from the source vertex *s* to vertex *v*. :param *v*: the destination vertex :return: True if there is a path from the source vertex *s* to vertex *v*. Otherwise returns False :rtype: bool :raises IllegalArgumentException: unless  $0 \leq v < V$

**path\_to** (*v*)

Returns a shortest path from the source vertex *s* to vertex *v*. :param *v*: the destination vertex :return: a shortest path from the source vertex *s* to vertex *v* :rtype: collections.iterable[DirectedEdge] :raises IllegalArgumentException: unless  $0 \leq v < V$

itu.algs4.graphs.dijkstra\_sp.main()

Creates an EdgeWeightedDigraph from input file. Runs DijkstraSP on the graph with the given source vertex. Prints the shortest path from the source vertex to all other vertices.

## 2.18 itu.algs4.graphs.dijkstra\_undirected\_sp module

**class** itu.algs4.graphs.dijkstra\_undirected\_sp.DijkstraUndirectedSP (*G*, *s*)

Bases: object

The DijkstraSP class represents a data type for solving the single-source shortest paths problem in edge-weighted digraphs where the edge weights are nonnegative. This implementation uses Dijkstra's algorithm with a binary heap. The constructor takes time proportional to  $E \log V$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Each call to `dist_to()` and `has_path_to()` takes constant time each call to `path_to()` takes time proportional to the number of edges in the shortest path returned.

**dist\_to** (*v*)

Returns the length of a shortest path between the source vertex *s* and vertex *v*. :param *v*: the destination vertex :return: the length of a shortest path between the source vertex *s* and the vertex *v*. float('inf') is not such path :rtype: float :raises IllegalArgumentException: unless  $0 \leq v < V$

**has\_path\_to** (*v*)

Returns true if there is a path between the source vertex *s* and vertex *v*. :param *v*: the destination vertex :return: True if there is a path between the source vertex *s* to vertex *v*. False otherwise :rtype: bool

**path\_to** (*v*)

Returns a shortest path between the source vertex *s* and vertex *v*. :param *v*: the destination vertex :return: a shortest path between the source vertex *s* and vertex *v*. None if no such path :rtype: collections.iterable[Edge] :raises IllegalArgumentException: unless  $0 \leq v < V$

itu.algs4.graphs.dijkstra\_undirected\_sp.main()



## 2.19 itu.algs4.graphs.directed\_cycle module

This module implements the directed cycle algorithm described in Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne. This version works for both weighted and unweighted directed graphs, due to Python's duck-typing.

For more information, see chapter 4.2 of the book.

**class** `itu.algs4.graphs.directed_cycle.DirectedCycle` (*digraph*)

Bases: `object`

The `DirectedCycle` class represents a data type for determining whether a digraph has a directed cycle. The `hasCycle` operation determines whether the digraph has a directed cycle and, if so, the `cycle` operation returns one.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `hasCycle` operation takes constant time; the `cycle` operation takes time proportional to the length of the cycle.

See Topological to compute a topological order if the digraph is acyclic.

For additional documentation, see Section 4.2 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**cycle** ()

Returns a directed cycle if the digraph has a directed cycle, and null otherwise.

**Returns** a directed cycle (as an iterable) if the digraph has a directed cycle, and null otherwise

**has\_cycle** ()

Does the digraph have a directed cycle?

**Returns** true if there is a cycle, false otherwise

## 2.20 itu.algs4.graphs.directed\_dfs module

**class** `itu.algs4.graphs.directed_dfs.DirectedDFS` (*G, \*s*)

Bases: `object`

The `DirectedDFS` class represents a data type for determining the vertices reachable from a given source vertex *s* (or a set of source vertices) in a digraph. For versions that find the paths, see `DepthFirstDirectedPaths` and `BreadthFirstDirectedPaths`.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges.

For additional documentation, see Section 4.2 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**count** ()

Returns the number of vertices reachable from the source vertex (or source vertices) :returns: the number of vertices reachable from the source vertex (or source vertices)

**is\_marked** (*v*)

Is there a directed path from the source vertex and vertex *v*?

**Parameters** *v* – the vertex

**Returns** True if there is a directed

`itu.algs4.graphs.directed_dfs.main` ()

Unit tests the `DirectedDFS` data type.

## 2.21 itu.algs4.graphs.directed\_edge module

**class** `itu.algs4.graphs.directed_edge.DirectedEdge` (*v, w, weight*)

Bases: `object`

The `DirectedEdge` class represents a weighted edge in an `EdgeWeightedDigraph`. Each edge consists of two integers (naming the two vertices) and a real-value weight. The data type provides methods for accessing the two endpoints of the directed edge and the weight.

**from\_vertex** ()

Returns the tail vertex of the directed edge. :return: the tail vertex of the directed edge :rtype: `int`

**to\_vertex** ()

Returns the head vertex of the directed edge. :return: the head vertex of the directed edge :rtype: `int`

**weight** ()

Returns the weight of the directed edge. :return: the weight of the directed edge :rtype: `float`

`itu.algs4.graphs.directed_edge.main` ()

Creates a directed edge and prints it. :return:

## 2.22 itu.algs4.graphs.edge module

**class** `itu.algs4.graphs.edge.Edge` (*v, w, weight*)

Bases: `object`

The `Edge` class represents a weighted edge in an `EdgeWeightedGraph`. Each edge consists of two integers (naming the two vertices) and a real-value weight. The data type provides methods for accessing the two endpoints of the edge and the weight. The natural order for this data type is by ascending order of weight.

**either** ()

Returns either endpoint of this edge. :return: either endpoint of this edge :rtype: `int`

**other** (*vertex*)

Returns the endpoint of this edge that is different from the given vertex. :param *vertex*: one endpoint of this edge :return: the other endpoint of this edge :rtype: `int` :raises `IllegalArgumentException`: if the vertex is not one of the endpoints of this edge

**weight** ()

Returns the weight of this edge. :return: the weight of this edge :rtype: `float`

`itu.algs4.graphs.edge.main` ()

Creates an edge and prints it.

## 2.23 itu.algs4.graphs.edge\_weighted\_digraph module

**class** `itu.algs4.graphs.edge_weighted_digraph.EdgeWeightedDigraph` (*V*)

Bases: `object`

The `EdgeWeightedDigraph` class represents an edge-weighted digraph of vertices named 0 through *V*-1, where each directed edge is of type `DirectedEdge` and has a real-valued weight. It supports the following two primary operations: add a directed edge to the digraph and iterate over all edges incident from a given vertex. It also provides methods for returning the number of vertices *V* and the number of edges *E*. Parallel edges and self-loops are permitted. This implementation uses an adjacency-lists representation, which is a vertex-indexed array of `Bag` objects. All operations take constant time (in the worst case) except iterating over the edges incident from a given vertex, which takes time proportional to the number of such edges.

**E()**  
Returns the number of edges in this edge-weighted digraph. :return: the number of edges in this edge-weighted digraph :rtype: int

**V()**  
Returns the number of vertices in this edge-weighted digraph. :return: the number of vertices in this edge-weighted digraph :rtype: int

**add\_edge(e)**  
Adds the directed edge e to this edge-weighted digraph. :param e: the edge :raises `IllegalArgumentException`: unless endpoints of edge are between 0 and V-1

**adj(v)**  
Returns the directed edges incident from vertex v. :param v: the vertex :return: the directed edges incident from vertex v. :rtype: `collections.iterable[DirectedEdge]` :raises `IllegalArgumentException`: unless  $0 \leq v < V$

**edges()**  
Returns all directed edges in this edge-weighted digraph. :return: all edges in this edge-weighted digraph :rtype: `collections.iterable[DirectedEdge]`

**static from\_graph(G)**  
Initializes a new edge-weighted digraph that is a deep copy of G. :param G: the edge-weighted digraph to copy :return: a copy of graph G :rtype: `EdgeWeightedDigraph`

**static from\_stream(stream)**  
Initializes an edge-weighted digraph from the specified input stream. The format is the number of vertices V, followed by the number of edges E, followed by E pairs of vertices and edge weights, with each entry separated by whitespace :param stream: the input stream

**Raises**

- **`IllegalArgumentException`** – if the endpoints of any edge are not in prescribed range
- **`IllegalArgumentException`** – if the number of vertices or edges is negative

**Returns** the edge-weighted digraph

**Return type** *EdgeWeightedDigraph*

**indegree(v)**  
Returns the number of directed edges incident to vertex v. This is known as the indegree of vertex v. :param v: the vertex :return: the indegree of vertex v :rtype: int :raises `IllegalArgumentException`: unless  $0 \leq v < V$

**outdegree(v)**  
Returns the number of directed edges incident from vertex v. This is known as the outdegree of vertex v. :param v: the vertex :return: the outdegree of vertex v :rtype: int :raises `IllegalArgumentException`: unless  $0 \leq v < V$

`itu.algs4.graphs.edge_weighted_digraph.main()`  
Creates an edge-weighted digraph from the given input file and prints it.

## 2.24 itu.algs4.graphs.edge\_weighted\_directed\_cycle module

**class** `itu.algs4.graphs.edge_weighted_directed_cycle.EdgeWeightedDirectedCycle(G)`  
Bases: `object`

Determines whether the edge-weighted digraph  $G$  has a directed cycle and, if so, finds such a cycle. :param  $G$  the edge-weighted digraph

**cycle()**

**has\_cycle()**

`itu.algs4.graphs.edge_weighted_directed_cycle.main(args)`

## 2.25 itu.algs4.graphs.edge\_weighted\_directed\_cycle\_anton module

This module implements the directed cycle algorithm for `EdgeWeightedDigraphs` described in *Algorithms*, 4th Edition by Robert Sedgewick and Kevin Wayne. This version works for both weighted and unweighted directed graphs, due to Python's duck-typing.

For more information, see chapter 4.2 of the book.

**class** `itu.algs4.graphs.edge_weighted_directed_cycle_anton.EdgeWeightedDirectedCycle` (*edge\_weighted\_digraph*)  
Bases: `object`

The `EdgeWeightedDirectedCycle` class represents a data type for determining whether edge-weighted digraph has a directed cycle. The `hasCycle` operation determines whether the edge-weighted digraph has a directed cycle and, if so, the `cycle` operation returns one.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `hasCycle` operation takes constant time; the `cycle` operation takes time proportional to the length of the cycle.

See `Topological` to compute a topological order if the edge-weighted digraph is acyclic.

For additional documentation, see Section 4.4 of *Algorithms*, 4th Edition by Robert Sedgewick and Kevin Wayne.

**cycle()**

Returns a directed cycle if the edge weighted digraph has a directed cycle, and null otherwise.

**Returns** a directed cycle (as an iterable) if the digraph has a directed cycle, and null otherwise

**has\_cycle()**

Does the edge weighted digraph have a directed cycle?

**Returns** true if there is a cycle, false otherwise

## 2.26 itu.algs4.graphs.edge\_weighted\_graph module

**class** `itu.algs4.graphs.edge_weighted_graph.EdgeWeightedGraph` ( $V$ )  
Bases: `object`

The `EdgeWeightedGraph` class represents an edge-weighted graph of vertices named 0 through  $V-1$ , where each undirected edge is of type `Edge` and has a real-valued weight. It supports the following two primary operations: add an edge to the graph, iterate over all of the edges incident to a vertex. It also provides methods for returning the number of vertices  $V$  and the number of edges  $E$ . Parallel edges and self-loops are permitted. By convention, a self-loop  $v-v$  appears in the adjacency list of  $v$  twice and contributes two to the degree of  $v$ . This implementation uses an adjacency-list representation, which is a vertex-indexed array of `Bag` objects. All operations take constant time (in the worst case) except iterating over the edges incident to a given vertex, which takes time proportional to the number of such edges.

**E()**  
Returns the number of edges in this edge-weighted graph. :return: the number of edges in this edge-weighted graph :rtype: int

**V()**  
Returns the number of vertices in this edge-weighted graph. :return: the number of vertices in this edge-weighted graph :rtype: int

**add\_edge(e)**  
Adds the undirected edge e to this edge-weighted graph. :param e: the edge

**adj(v)**  
Returns the edges incident on vertex v. :param v: the vertex :return: the edges incident on vertex v :rtype: collections.iterable[Edge]

**degree(v)**  
Returns the degree of vertex v. :param v: the vertex :return: the degree of vertex v :rtype: int :raises IllegalArgumentException: unless  $0 \leq v < V$

**edges()**  
Returns all edges in this edge-weighted graph. :return: all edges in this edge-weighted graph

**static from\_graph(G)**  
Initializes a new edge-weighted graph that is a deep copy of G. :param G: the edge-weighted graph to copy :return: the copy of the graph edge-weighted graph G :rtype: EdgeWeightedGraph

**static from\_stream(stream)**  
Initializes an edge-weighted graph from an input stream. The format is the number of vertices V, followed by the number of edges E, followed by E pairs of vertices and edge weights, with each entry separated by whitespace. :param stream: the input stream :raises IllegalArgumentException: if the endpoints of any edge are not in prescribed range :raises IllegalArgumentException: if the number of vertices or edges is negative :return: the edge-weighted graph :rtype: EdgeWeightedGraph

`itu.algs4.graphs.edge_weighted_graph.main()`  
Creates an edge-weighted graph from the given input file and prints it.

## 2.27 itu.algs4.graphs.graph module

**class** `itu.algs4.graphs.graph.Graph(V)`  
Bases: object

The Graph class represents an undirected graph of vertices named 0 through  $V - 1$ . It supports the following two primary operations: add an edge to the graph, iterate over all of the vertices adjacent to a vertex. It also provides methods for returning the number of vertices  $V$  and the number of edges  $E$ . Parallel edges and self-loops are permitted. By convention, a self-loop  $v-v$  appears in the adjacency list of  $v$  twice and contributes two to the degree of  $v$ .

This implementation uses an adjacency-lists representation, which is a vertex-indexed array of Bag objects. All operations take constant time (in the worst case) except iterating over the vertices adjacent to a given vertex, which takes time proportional to the number of such vertices.

**E()**  
Returns the number of edges in this graph.  
**Returns** the number of edges in this graph.

**V()**  
Returns the number of vertices in this graph.

**Returns** the number of vertices in this graph.

**add\_edge** (*v*, *w*)

Adds the undirected edge *v-w* to this graph.

**Parameters**

- **v** – one vertex in the edge
- **w** – the other vertex in the edge

**Raises** **ValueError** – unless both  $0 \leq v < V$  and  $0 \leq w < V$

**adj** (*v*)

Returns the vertices adjacent to vertex *v*.

**Parameters** **v** – the vertex

**Returns** the vertices adjacent to vertex *v*, as an iterable

**Raises** **ValueError** – unless  $0 \leq v < V$

**degree** (*v*)

Returns the degree of vertex *v*.

**Parameters** **v** – the vertex

**Returns** the degree of vertex *v*

**Raises** **ValueError** – unless  $0 \leq v < V$

**static from\_graph** (*G*)

Initializes a new graph that is a deep copy of *G*

**Parameters** **G** – the graph to copy

**Returns** copy of *G*

**static from\_stream** (*stream*)

Initializes a graph from the specified input stream. The format is the number of vertices *V*, followed by the number of edges *E*, followed by *E* pairs of vertices, with each entry separated by whitespace.

**Parameters** **stream** – the input stream

**Returns** new graph from stream

**Raises**

- **ValueError** – if the endpoints of any edge are not in prescribed range
- **ValueError** – if the number of vertices or edges is negative
- **ValueError** – if the input stream is in the wrong format

## 2.28 itu.algs4.graphs.kosaraju\_sharir\_scc module

- Execution: `python kosaraju_sharir_scc.py filename.txt`
- Dependencies: Digraph TransitiveClosure InStream DepthFirstOrder
- Data files: <https://algs4.cs.princeton.edu/42digraph/tinyDG.txt>
- <https://algs4.cs.princeton.edu/42digraph/mediumDG.txt>
- <https://algs4.cs.princeton.edu/42digraph/largeDG.txt>

- 
- Compute the strongly-connected components of a digraph using the
- Kosaraju-Sharir algorithm.
- 
- Runs in  $O(E + V)$  time.
- 
- `% python kosaraju_sharir_scc.py tinyDG.txt`
- 5 strong components
- 1
- 0 2 3 4 5
- 9 10 11 12
- 6 8
- 7
- 

```
class itu.algs4.graphs.kosaraju_sharir_scc.KosarajuSharirSCC(G)
    Bases: object
```

- Computes the strong components of the digraph *G*.
- @param *G* the digraph

```
count()
```

```
id(v)
```

```
strongly_connected(v, w)
```

```
itu.algs4.graphs.kosaraju_sharir_scc.main(args)
```

## 2.29 itu.algs4.graphs.kruskal\_mst module

```
class itu.algs4.graphs.kruskal_mst.KruskalMST(G)
    Bases: object
```

The `KruskalMST` class represents a data type for computing a minimum spanning tree in an edge-weighted graph. The edge weights can be positive, zero, or negative and need not be distinct. If the graph is not connected, it computes a minimum spanning forest, which is the union of minimum spanning trees in each connected component. The `weight` method returns the weight of a minimum spanning tree and the `edges` method returns its edges. This implementation uses Kruskal's algorithm and the union-find data type. The constructor takes time proportional to  $E \log E$  and extra space (not including the graph) proportional to  $V$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `weight` method takes constant time and the `edges` method takes time proportional to  $V$ .

```
edges()
```

Returns the edges in a minimum spanning tree (or forest). :return: the edges in a minimum spanning tree (or forest)

```
weight()
```

Returns the sum of the edge weights in a minimum spanning tree (or forest). :return: the sum of the edge weights in a minimum spanning tree (or forest)

```
itu.algs4.graphs.kruskal_mst.main()
```

Creates an edge-weighted graph from an input file, runs Kruskal's algorithm on it, and prints the edges of the MST and the sum of the edge weights.

## 2.30 itu.algs4.graphs.lazy\_prim\_mst module

```
class itu.algs4.graphs.lazy_prim_mst.LazyPrimMST(G)
```

Bases: object

The LazyPrimMST class represents a data type for computing a minimum spanning tree in an edge-weighted graph. The edge weights can be positive, zero, or negative and need not be distinct. If the graph is not connected, it computes a minimum spanning forest, which is the union of minimum spanning trees in each connected component. The weight() method returns the weight of a minimum spanning tree and the edges() method returns its edges.

This implementation uses a lazy version of Prim's algorithm with a binary heap of edges. The constructor takes time proportional to  $E \log E$  and extra space (not including the graph) proportional to  $E$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the weight() method takes constant time and the edges() method takes time proportional to  $V$ .

**FLOATING\_POINT\_EPSILON = 1e-12**

**edges()**

Returns the edges in a minimum spanning tree (or forest). :returns: the edges in a minimum spanning tree (or forest) as

an iterable of edges

**weight()**

Returns the sum of the edge weights in a minimum spanning tree (or forest). :returns: the sum of the edge weights in a minimum spanning tree (or forest)

## 2.31 itu.algs4.graphs.prim\_mst module

```
class itu.algs4.graphs.prim_mst.PrimMST(G)
```

Bases: object

The PrimMST class represents a data type for computing a minimum spanning tree in an edge-weighted graph. The edge weights can be positive, zero, or negative and need not be distinct. If the graph is not connected, it computes a minimum spanning forest, which is the union of minimum spanning trees in each connected component. The weight() method returns the weight of a minimum spanning tree and the edges() method returns its edges.

This implementation uses Prim's algorithm with an indexed binary heap. The constructor takes time proportional to  $E \log V$  and extra space (not including the graph) proportional to  $V$ , where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the weight() method takes constant time and the edges() method takes time proportional to  $V$ .

**FLOATING\_POINT\_EPSILON = 1e-12**

**edges()**

Returns the edges in a minimum spanning tree (or forest). :returns: the edges in a minimum spanning tree (or forest) as

an iterable of edges



**weight ()**

Returns the sum of the edge weights in a minimum spanning tree (or forest). :returns: the sum of the edge weights in a minimum spanning tree (or forest)

## 2.32 itu.algs4.graphs.symbol\_digraph module

**class** `itu.algs4.graphs.symbol_digraph.SymbolDigraph (filename, delimiter)`

Bases: `object`

The SymbolDigraph class represents a digraph, where the vertex names are arbitrary strings. By providing mappings between vertex names and integers, it serves as a wrapper around the Digraph data type, which assumes the vertex names are integers between 0 and  $V - 1$ . It also supports initializing a symbol digraph from a file.

This implementation uses an ST to map from strings to integers, an array to map from integers to strings, and a Digraph to store the underlying graph. The `index_of` and `contains` operations take time proportional to  $\log V$ , where  $V$  is the number of vertices. The `name_of` operation takes constant time.

**contains (s)**

Does the graph contain the vertex named `s`?

**Parameters** `s` – the name of a vertex

:return: `s` true if `s` is the name of a vertex, and false otherwise

**digraph ()****index\_of (s)**

Returns the integer associated with the vertex named `s`. :param `s`: the name of a vertex :returns: the integer (between 0 and  $V - 1$ ) associated with the vertex named `s`

**name\_of (v)**

Returns the name of the vertex associated with the integer `v`. @param `v` the integer corresponding to a vertex (between 0 and  $V - 1$ ) @throws `IllegalArgumentException` unless  $0 \leq v < V$  @return the name of the vertex associated with the integer `v`

## 2.33 itu.algs4.graphs.symbol\_graph module

**class** `itu.algs4.graphs.symbol_graph.SymbolGraph (filename, delimiter)`

Bases: `object`

The SymbolGraph class represents an undirected graph, where the vertex names are arbitrary strings. By providing mappings between vertex names and integers, it serves as a wrapper around the Graph data type, which assumes the vertex names are integers between 0 and  $V - 1$ . It also supports initializing a symbol graph from a file.

This implementation uses an ST to map from strings to integers, an array to map from integers to strings, and a Graph to store the underlying graph. The `index_of` and `contains` operations take time proportional to  $\log V$ , where  $V$  is the number of vertices. The `name_of` operation takes constant time.

**contains (s)**

Does the graph contain the vertex named `s`?

**Parameters** `s` – the name of a vertex

:return: `s` true if `s` is the name of a vertex, and false otherwise

**graph ()**

**index\_of**(*s*)

Returns the integer associated with the vertex named *s*. :param *s*: the name of a vertex :returns: the integer (between 0 and  $V - 1$ ) associated with the vertex named *s*

**name\_of**(*v*)

Returns the name of the vertex associated with the integer *v*. @param *v* the integer corresponding to a vertex (between 0 and  $V - 1$ ) @throws `IllegalArgumentException` unless  $0 \leq v < V$  @return the name of the vertex associated with the integer *v*

## 2.34 itu.algs4.graphs.topological module

This module implements the topological order algorithm described in Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne. For more information, see chapter 4.2 of the book.

**class** `itu.algs4.graphs.topological.Topological`(*digraph*)

Bases: `object`

The `Topological` class represents a data type for determining a topological order of a directed acyclic graph (DAG). Recall, a digraph has a topological order if and only if it is a DAG. The `hasOrder` operation determines whether the digraph has a topological order, and if so, the `order` operation returns one.

This implementation uses depth-first search. The constructor takes time proportional to  $V + E$  (in the worst case), where  $V$  is the number of vertices and  $E$  is the number of edges. Afterwards, the `hasOrder` and `rank` operations takes constant time the `order` operation takes time proportional to  $V$ .

See `DirectedCycle`, `DirectedCycleX`, and `EdgeWeightedDirectedCycle` to compute a directed cycle if the digraph is not a DAG. See `TopologicalX` for a nonrecursive queue-based algorithm to compute a topological order of a DAG.

For additional documentation, see Section 4.2 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**has\_order**()

Does the digraph have a topological order?

**Returns** True if the digraph has a topological order (or equivalently, if the digraph is a DAG), and False otherwise

**order**()

Returns a topological order if the digraph has a topological order, and None otherwise.

**Returns** a topological order of the vertices (as an iterable) if the digraph has a topological order (or equivalently, if the digraph is a DAG), and None otherwise

**rank**(*v*)

The the rank of vertex *v* in the topological order -1 if the digraph is not a DAG

**Parameters** *v* – the vertex

**Returns** the position of vertex *v* in a topological order of the digraph -1 if the digraph is not a DAG

## 2.35 itu.algs4.graphs.transitive\_closure module

- Execution: `python transitive_closure.py filename.txt`
- Dependencies: `Digraph` `DirectedDFS`

- Data files: <https://algs4.cs.princeton.edu/42digraph/tinyDG.txt>
- 
- Compute transitive closure of a digraph and support
- reachability queries.
- 
- Preprocessing time:  $O(V(E + V))$  time.
- Query time:  $O(1)$ .
- Space:  $O(V^2)$ .
- 
- `% python transitive_closure.py tinyDG.txt`
- `0 1 2 3 4 5 6 7 8 9 10 11 12`
- 
- `0: T T T T T T`
- `1: T`
- `2: T T T T T T`
- `3: T T T T T T`
- `4: T T T T T T`
- `5: T T T T T T`
- `6: T T T T T T T T T T`
- `7: T T T T T T T T T T T T`
- `8: T T T T T T T T T T T T`
- `9: T T T T T T T T T T`
- `10: T T T T T T T T T T`
- `11: T T T T T T T T T T`
- `12: T T T T T T T T T T`
- 

```

class itu.algs4.graphs.transitive_closure.TransitiveClosure(G)
    Bases: object
        • Computes the transitive closure of the digraph G.
        • @param G the digraph

    reachable(v, w)

itu.algs4.graphs.transitive_closure.main(args)

```

## 2.36 Module contents



## 3.1 Submodules

## 3.2 itu.algs4.searching.binary\_search\_st module

**class** `itu.algs4.searching.binary_search_st.BinarySearchST` (*capacity=2*)  
Bases: `object`

The BST class represents an ordered symbol table of generic key-value pairs. It supports the usual put, get, contains, delete, size, and is-empty methods. It also provides ordered methods for finding the minimum, maximum, floor, select, and ceiling. It also provides a keys method for iterating over all of the keys. A symbol table implements the associative array abstraction: when associating a value with a key that is already in the symbol table, the convention is to replace the old value with the new value. Unlike `java.util.Map`, this class uses the convention that values cannot be `None`—setting the value associated with a key to `None` is equivalent to deleting the key from the symbol table.

This implementation uses a sorted array. It requires that the key type implements the `Comparable` interface and calls the `compareTo()` method to compare two keys. It does not call either `equals()` or `hashCode()`. The put and remove operations each take linear time in the worst case the contains, ceiling, floor, and rank operations take logarithmic time the size, is-empty, minimum, maximum, and select operations take constant time. Construction takes constant time.

**ceiling** (*key*)

Returns the smallest key in this symbol table greater than or equal to key.

**Parameters** *key* – the key

**Returns** the smallest key in this symbol table greater than or equal to key

**Raises**

- **ValueError** – if there is no such key
- **ValueError** – if key is `None`

**contains** (*key*)

Does this symbol table contain the given key?

**Parameters** **key** – the key

**Returns** True if this symbol table contains key and False otherwise

**Raises** **ValueError** – if key is None

**delete** (*key*)

Removes the specified key and associated value from this symbol table (if the key is in the symbol table).

**Parameters** **key** – the key

**Raises** **ValueError** – if key is None

**deleteMax** ()

Removes the largest key and associated value from this symbol table.

**Raises** **ValueError** – if the symbol table is empty

**deleteMin** ()

Removes the smallest key and associated value from this symbol table.

**Raises** **ValueError** – if the symbol table is empty

**floor** (*key*)

Returns the largest key in this symbol table less than or equal to key.

**Parameters** **key** – the key

**Returns** the largest key in this symbol table less than or equal to key

**Raises**

- **ValueError** – if there is no such key
- **ValueError** – if key is None

**get** (*key*)

Returns the value associated with the given key in this symbol table.

**Parameters** **key** – the key

**Returns** the value associated with the given key if the key is in the symbol table and None if the key is not in the symbol table

**Raises** **ValueError** – if key is None

**is\_empty** ()

Returns True if this symbol table is empty.

**Returns** True if this symbol table is empty False otherwise

**keys** ()

Returns all keys in this symbol table as an Iterable. To iterate over all of the keys in the symbol table named st, use the foreach notation: for (Key key : st.keys()).

**Returns** all keys in this symbol table

**keys\_between** (*lo*, *hi*)

Returns all keys in this symbol table in the given range, as an Iterable.

**Parameters**

- **lo** – minimum endpoint
- **hi** – maximum endpoint

**Returns** all keys in this symbol table between lo (inclusive) and hi (inclusive)

**Raises ValueError** – if either lo or hi are None

**max()**

Returns the largest key in this symbol table.

**Returns** the largest key in this symbol table

**Raises ValueError** – if this symbol table is empty

**min()**

Returns the smallest key in this symbol table.

**Returns** the smallest key in this symbol table

**Raises ValueError** – if this symbol table is empty

**put(key, val)**

Inserts the specified key-value pair into the symbol table, overwriting the old value with the new value if the symbol table already contains the specified key. Deletes the specified key (and its associated value) from this symbol table if the specified value is None.

**Parameters**

- **key** – the key
- **val** – the value

**Raises ValueError** – if key is None

**rank(key)**

Returns the number of keys in this symbol table strictly less than key.

**Parameters key** – the key

**Returns** the number of keys in the symbol table strictly less than key

**Raises ValueError** – if key is None

**select(k)**

Return the kth smallest key in this symbol table.

**Parameters k** – the order statistic

**Returns** the kth smallest key in this symbol table

**Raises ValueError** – unless k is between 0 and n-1

**size()**

Returns the number of key-value pairs in this symbol table.

**Returns** the number of key-value pairs in this symbol table

**size\_between(lo, hi)**

Returns the number of keys in this symbol table in the specified range.

**Parameters**

- **lo** – minimum endpoint
- **hi** – maximum endpoint

**Returns** the number of keys in this symbol table between lo (inclusive) and hi (inclusive)

**Raises ValueError** – if either lo or hi is None

### 3.3 itu.algs4.searching.bst module

**class** `itu.algs4.searching.bst.BST`

Bases: `typing.Generic`

**ceiling** (*key: Key*) → *Key*

Returns the smallest key in the symbol table greater than or equal to *key*. Raises `NoSuchElementException` if no such key exists.

**contains** (*key: Key*) → *bool*

Does this symbol table contain the given key? :param *key*: the key to search for :return *boolean*: true if symbol table contains *key*, false otherwise

**delete** (*key: Key*) → *None*

Removes the specified key and its associated value from this symbol table (if the key is in this symbol table)

**delete\_max** () → *None*

Removes the largest key and associated value from the symbol table

**delete\_min** () → *None*

Removes the smallest key and associated value from the symbol table TODO exception?

**floor** (*key: Key*) → *Key*

Returns the largest key in the symbol table less than or equal to *key*. Raises `NoSuchElementException` if no such key exists.

**get** (*key: Key*) → *Optional[Val]*

Returns the value associated with the given key

**Parameters** *key* – The key whose value is returned

**Returns** the value associated with the given key if the key is in the symbol table, *None* otherwise

**height** () → *int*

Returns the height of the BST (for debugging)

**is\_empty** () → *bool*

Returns true if this symbol table is empty

**keys** () → `itu.algs4.fundamentals.queue.Queue[~Key][Key]`

Returns all keys in the symbol table as a list.

**level\_order** () → `itu.algs4.fundamentals.queue.Queue[~Key][Key]`

Returns the keys in the BST in level order (for debugging)

**max** () → *Key*

Returns the largest key in the symbol table

**min** () → *Key*

Returns the smallest key in the BST

**put** (*key: Key, value: Optional[Val]*) → *None*

Inserts the specified key-value pair into the symbol table, overwriting the old value with the new value if the symbol table already contains the specified key. Deletes the specified key (and its associated value) from this symbol table if the specified value is *None*.

**Parameters** *value* (*key*,) – the key-value pair to be inserted

**range\_keys** (*lo: Key, hi: Key*) → `itu.algs4.fundamentals.queue.Queue[~Key][Key]`

returns all keys in the symbol table in the given range as a list



**Parameters**

- **lo** – minimum endpoint
- **hi** – maximum endpoint

**Returns** all keys in symbol table between lo (inclusive) and hi (inclusive)

**rank** (*key: Key*) → int

Returns the number of keys in the symbol table strictly less than key. :param key: the key :return: the number of keys in the symbol table strictly less than key :rtype: int :raises IllegalArgumentException: if key is None

**select** (*k: int*) → Key

Return the kth smallest key in the symbol table. :param k: the order statistic :return: the kth smallest key in the symbol table :raises IllegalArgumentException: unless k is between 0 and n-1

**size** () → int

Returns the number of key-value pairs in this symbol table

**size\_range** (*lo: Key, hi: Key*) → int

Returns the number of keys in the symbol table in the given range. :param lo: minimum endpoint :param hi: maximum endpoint :return: the number of keys in the symbol table between lo (inclusive) and hi (inclusive) :rtype: int :raises IllegalArgumentException: if either lo or hi is None

**class** itu.algs4.searching.bst.Comparable (\*args, \*\*kwargs)

Bases: typing\_extensions.Protocol

**class** itu.algs4.searching.bst.Node (key: Key, value: Optional[Val], size: int)

Bases: typing.Generic

## 3.4 itu.algs4.searching.file\_index module

## 3.5 itu.algs4.searching.frequency\_counter module

## 3.6 itu.algs4.searching.linear\_probing\_hst module

**class** itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST (capacity=4)

Bases: object

The LinearProbingHashST class represents a symbol table of dynamic key-value pairs. It supports the usual put, get, contains, delete, size, and is-empty methods. It also provides a key\_list method for iterating over all of the keys. A symbol table implements the associative array abstraction: when associating a value with a key that is already in the symbol table, the convention is to replace the old value with the new value. Unlike the Map-class in Java, this class uses the convention that values cannot be null/None. Setting the value associated with a key to None is equivalent to deleting the key from the symbol table.

This implementation uses a linear probing hash table. It requires that the key type overrides the \_\_eq\_\_ and \_\_hash\_\_ methods. The expected time per put, contains, or remove operation is constant, subject to the uniform hashing assumption. The size, and is-empty operations take constant time. Construction takes constant time.

**contains** (*key*)

Returns True if this symbol table contains the specified key.

**Parameters** **key** – the key

**Returns** True if this symbol table contains the key; False otherwise

**Raises `ValueError`** – if key is None

**`delete (key)`**

Removes the specified key and its associated value from this symbol table (if the key is in this symbol table).

**Parameters** **key** – the key

**Raises `ValueError`** – if key is None

**`get (key)`**

Returns the value associated with the specified key.

**Parameters** **key** – the key

**Returns** the value associated with the key in the symbol table; None if no such value

**Raises `ValueError`** – if key is None

**`is_empty ()`**

Returns True if this symbol table is empty.

**Returns** True if this symbol table is empty; False otherwise

**`key_list ()`**

Returns the keys in the symbol table as an iterable :returns: A list containing all keys

**`put (key, value)`**

Inserts the specified key-value pair into the symbol table, overwriting the old value with the new value if the symbol table already contains the specified key. Deletes the specified key (and its associated value) from this symbol table if the specified value is None.

**Parameters**

- **key** – the key
- **value** – the value

**Raises `ValueError`** – if key is None.

**`size ()`**

Returns the number of key-value pairs in this symbol table.

**Returns** the number of key-value pairs in this symbol table.

`itu.algs4.searching.linear_probing_hst.main ()`

Unit tests the LinearProbingHashST data type.

## 3.7 itu.algs4.searching.lookup\_csv module

## 3.8 itu.algs4.searching.lookup\_index module

## 3.9 itu.algs4.searching.red\_black\_bst module

**`class`** `itu.algs4.searching.red_black_bst.Comparable` (*\*args, \*\*kwargs*)

Bases: `typing_extensions.Protocol`

**`class`** `itu.algs4.searching.red_black_bst.Node` (*key: Key, val: Val, color: bool, size: int*)

Bases: `typing.Generic`

RedBlackBST helper node data type.

```
class itu.algs4.searching.red_black_bst.RedBlackBST
```

```
    Bases: typing.Generic
```

The RedBlackBST class represents an ordered symbol table of generic key-value pairs. It supports the usual put, get, contains, delete, size, and is-empty methods. It also provides ordered methods for finding the minimum, maximum, floor, and ceiling. It also provides a keys method for iterating over all the keys. A symbol table implements the associative array abstraction: when associating a value with a key that is already in the symbol table, the convention is to replace the old value with the new value. This class uses the convention that values cannot be None—setting the value associated with a key to None is equivalent to deleting the key from the symbol table. This implementation uses a left-leaning red-black BST. It requires that the keys are all of the same type and that they can be compared. The put, contains, remove, minimum, maximum, ceiling, and floor operations each take logarithmic time in the worst case, if the tree becomes unbalanced. The size, and is-empty operations take constant time. Construction takes constant time.

```
BLACK = False
```

```
RED = True
```

```
ceiling (key: Key) → Key
```

Returns the smallest key in the symbol table greater than or equal to key. :param key: the key :return: the smallest key in the symbol table greater than or equal to key :raises IllegalArgumentException: if key is None :raises NoSuchElementException: if there is no such key

```
contains (key: Key) → bool
```

Does this symbol table contain the given key? :param key: the key :return: True if this symbol table contains key and False otherwise

```
delete (key: Key) → None
```

Removes the specified key and its associated value from this symbol table (if the key is in this symbol table). :param key: the key :raises IllegalArgumentException: if key is None

```
delete_max () → None
```

Removes the largest key and associated value from the symbol table. :raises NoSuchElementException: if the symbol table is empty

```
delete_min () → None
```

Removes the smallest key and associated value from the symbol table. :raises NoSuchElementException: if the symbol table is empty

```
floor (key: Key) → Key
```

Returns the largest key in the symbol table less than or equal to key. :param key: the key :return: the largest key in the symbol table less than or equal to key :raises IllegalArgumentException: if key is None :raises NoSuchElementException: if there is no such key

```
get (key: Key) → Optional[Val]
```

Returns the value associated with the given key. :param key: the key :return: the value associated with the given key if the key is in the symbol table and None if the key is not in the symbol table :raises IllegalArgumentException: if key is None

```
height () → int
```

Returns the height of the RedBlackBST :return: the height of the RedBlackBST (a 1-node tree has height 0)

```
is_empty () → bool
```

Is this symbol table empty? :return: True if this symbol table is empty and False otherwise

```
keys () → itu.algs4.fundamentals.queue.Queue[~Key][Key]
```

Returns all keys in the symbol table. :return: all keys in the symbol table

```
keys_range (lo: Key, hi: Key) → itu.algs4.fundamentals.queue.Queue[~Key][Key]
```

Returns all keys in the symbol table in the given range. :param lo: minimum endpoint :param hi: maximum

endpoint :return: all keys in the symbol table between lo (inclusive) and hi (inclusive) :raises `IllegalArgumentException`: if either lo or hi is None

**max** () → Key  
Returns the largest key in the symbol table. :return: the largest key in the symbol table :raises `NoSuchElementException`: if the symbol table is empty

**min** () → Key  
Returns the smallest key in the symbol table. :return: the smallest key in the symbol table :raises `NoSuchElementException`: if the symbol table is empty

**put** (key: Key, val: Val) → None  
Inserts the specified key-value pair into the symbol table, overwriting the old value with the new value if the symbol table already contains the specified key. Deletes the specified key (and its associated value) from this symbol table if the specified value is None. :param key: the key :param val: the value :raises `IllegalArgumentException`: if key is None

**rank** (key: Key) → int  
Returns the number of keys in the symbol table strictly less than key. :param key: the key :return: the number of keys in the symbol table strictly less than key :raises `IllegalArgumentException`: if key is None

**select** (k: int) → Key  
Return the kth smallest key in the symbol table. :param k: the order statistic :return: the kth smallest key in the symbol table :raises `IllegalArgumentException`: unless k is between 0 and n-1

**size** () → int  
Return the number of key-value pairs in this symbol table. :return: the number of key-value pairs in this symbol table

**size\_range** (lo: Key, hi: Key) → int  
Returns the number of keys in the symbol table in the given range. :param lo: minimum endpoint :param hi: maximum endpoint :return: the number of keys in the symbol table between lo (inclusive) and hi (inclusive) :raises `IllegalArgumentException`: if either lo or hi is None

## 3.10 itu.algs4.searching.seperate\_chaining\_hst module

**class** `itu.algs4.searching.seperate_chaining_hst.SeparateChainingHashST` ( $M=997$ )  
Bases: `object`

The `SeparateChainingHashST` class represents a symbol table of dynamic key-value pairs. It supports the usual `put`, `get`, `contains`, `delete`, `size`, and `is-empty` methods. It also provides a `keys` method for iterating over all of the keys. A symbol table implements the associative array abstraction: when associating a value with a key that is already in the symbol table, the convention is to replace the old value with the new value. Unlike the `Map`-class in Java, this class uses the convention that values cannot be `null/None`. Setting the value associated with a key to `None` is equivalent to deleting the key from the symbol table.

This implementation uses a separate chaining hash table. It requires that the key type overrides the `__eq__` and `__hash__` methods. The expected time per `put`, `contains`, or `remove` operation is constant, subject to the uniform hashing assumption. The `size`, and `is-empty` operations take constant time. Construction takes constant time.

**contains** (key)  
Returns true if this symbol table contains the specified key.

**Parameters** **key** – the key

**Returns** True if this symbol table contains the key; False otherwise.

**Raises** **ValueError** – if key is None.

**delete** (*key*)

Removes the specified key and its associated value from this symbol table (if the key is in this symbol table).

**Parameters** **key** – the key

**Raises** **ValueError** – if key is None

**get** (*key*)

Returns the value associated with the specified key.

**Parameters** **key** – the key

**Returns** the value associated with the key in the symbol table; None if no such value

**Raises** **ValueError** – if key is None

**is\_empty** ()

Returns true if the symbol table is empty.

**Returns** True if this symbol table is empty; False otherwise.

**keys** ()

Returns the keys in the symbol table as an iterable :returns: A list containing all keys

**put** (*key*, *value*)

Inserts the specified key-value pair into the symbol table, overwriting the old value with the new value if the symbol table already contains the specified key. Deletes the specified key (and its associated value) from this symbol table if the specified value is None.

**Parameters**

- **key** – the key
- **value** – the value

**Raises** **ValueError** – if key is None.

**size** ()

Returns the number of key-value pairs in this symbol table.

**Returns** the number of key-value pairs in this symbol table.

`itu.algs4.searching.separate_chaining_hst.main()`

Unit tests the SeparateChainingHashST data type.

## 3.11 itu.algs4.searching.sequential\_search\_st module

**class** `itu.algs4.searching.sequential_search_st.SequentialSearchST`

Bases: `object`

The `SequentialSearchST` class represents an (unordered) symbol table of generic key-value pairs. It supports the usual `put`, `get`, `contains`, `delete`, `size`, and `is-empty` methods. It also provides a `keys` method for iterating over all of the keys. A symbol table implements the associative array abstraction: when associating a value with a key that is already in the symbol table, the convention is to replace the old value with the new value. The class also uses the convention that values cannot be `None`. Setting the value associated with a key to `None` is equivalent to deleting the key from the symbol table.

This implementation uses a singly-linked list and sequential search. It relies on the `equals()` method to test whether two keys are equal. It does not call either the `compareTo()` or `hashCode()` method. The `put` and `delete` operations take linear time the `get` and `contains` operations takes linear time in the worst case. The `size`, and `is-empty` operations take constant time. Construction takes constant time.

```
class Node (key, val, next)
    Bases: object

contains (key)
    """ Returns true if this symbol table contains the specified key.

    :param key the key :returns: true if this symbol table contains key
        false otherwise

    Raises ValueError – if key is None

delete (key)
    Removes the specified key and its associated value from this symbol table (if the key is in this symbol
    table).

    :param key the key :raises ValueError: if key is None

get (key)
    Returns the value associated with the given key in this symbol table.

    Parameters key – the key

    Returns the value associated with the given key if the key is in the symbol table and None if the
        key is not in the symbol table

    Raises ValueError – if key is None

is_empty ()
    Returns true if this symbol table is empty.

    Returns true if this symbol table is empty false otherwise

keys ()
    Returns all keys in the symbol table as an Iterable. To iterate over all of the keys in the symbol table named
    st, use the foreach notation: for Key key in st.keys().

    Returns all keys in the symbol table

put (key, val)
    Inserts the specified key-value pair into the symbol table, overwriting the old value with the new value if
    the symbol table already contains the specified key. Deletes the specified key (and its associated value)
    from this symbol table if the specified value is None.

    Parameters

    • key – the key
    • val – the value

    Raises ValueError – if key is None

size ()
    Returns the number of key-value pairs in this symbol table.

    Returns the number of key-value pairs in this symbol table
```

## 3.12 itu.algs4.searching.set module

```
class itu.algs4.searching.set.SET (x=None)
    Bases: object
```

```

    add(key)
    ceiling(key)
    contains(key)
    delete(key)
    floor(key)
    hashCode()
    intersects(that)
    is_empty()
    max()
    min()
    size()
    union(that)

itu.algs4.searching.set.main()

```

### 3.13 itu.algs4.searching.sparse\_vector module

```
class itu.algs4.searching.sparse_vector.SparseVector(d)
```

Bases: object

The SparseVector class represents a d-dimensional mathematical vector. Vectors are mutable: their values can be changed after they are created. It includes methods for addition, subtraction, dot product, scalar product, unit vector and Euclidean norm.

The implementation is a symbol table of indices and values for which the vector coordinates are nonzero. This makes it efficient when most of the vector coordinates are zero.

**dimension()**

Returns the dimension of this vector.

**Returns** the dimension of this vector.

**dot(that)**

Returns the inner product of this vector with the specified vector.

**Parameters that** – the other vector

**Returns** the dot product between this vector and that vector

**Raises ValueError** – if the lengths of the two vectors are not equal

**get(i)**

Returns the ith coordinate of this vector.

**Parameters i** – the index

**Returns** the value of the ith coordinate of this vector

**Raises ValueError** – unless i is between 0 and d-1

**magnitude()**

Returns the magnitude of this vector. This is also known as the L2 norm or the Euclidean norm.

**Returns** the magnitude of this vector

**nnz()**

Returns the number of nonzero entries in this vector.

**Returns** the number of nonzero entries in this vector.

**plus(that)**

Returns the sum of this vector and the specified vector.

**Parameters** **that** – the vector to add to this vector

**Returns** the sum of this vector and that vector

**Raises** **ValueError** – if the dimension of the two vectors are not equal

**put(i, value)**

Sets the *i*th coordinate of this vector to the specified value.

**Parameters**

- **i** – the index
- **value** – the new value

**Raises** **ValueError** – unless *i* is between 0 and *d*-1

**scale(alpha)**

Returns the scalar-vector product of this vector with the specified scalar.

**Parameters** **alpha** – the scalar

**Returns** the scalar-vector product of this vector with the specified scalar

`itu.algs4.searching.sparse_vector.main()`

Unit tests the SparseVector data type.

## 3.14 itu.algs4.searching.st module

**class** `itu.algs4.searching.st.ST`

Bases: `object`

**ceiling**(*key*)**contains**(*key*)**delete**(*key*)**floor**(*key*)**get**(*key*)**is\_empty**()**keys**()**max**()**min**()**put**(*key, val*)**size**()

## 3.15 Module contents



## 4.1 Submodules

## 4.2 `itu.algs4.sorting.heap` module

`itu.algs4.sorting.heap.main()`  
Reads in a sequence of strings from stdin heapsorts them, and prints the result in ascending order.

`itu.algs4.sorting.heap.sort(pq)`  
Rearranges the array in ascending order, using the natural order. :param pq: the array to be sorted

## 4.3 `itu.algs4.sorting.index_min_pq` module

**class** `itu.algs4.sorting.index_min_pq.IndexMinPQ(max_n)`  
Bases: `object`

The `IndexMinPQ` class represents an indexed priority queue of generic keys. It supports the usual insert and delete-the-minimum operations, along with delete and change-the-key methods. In order to let the client refer to the keys on the priority queue, an integer between 0 and `maxN - 1` is associated with each key-the client uses this integer to specify which key to delete or change. It also supports methods for peeking at the minimum key, testing if the priority queue is empty, and iterating through the keys. This implementation uses a binary heap along with an array to associate keys with integers, in the given range. The insert, delete-the-minimum, delete, change-key, decrease-key, and increase-key operations take logarithmic time. The is-empty, size, min-index, min-key, and key-of operations take constant time. Construction takes time proportional to the specified capacity.

**change\_key** (*i, key*)

Change the key associated with index *i* to the specified value. :param *i*: the index of the key to change  
:param *key*: change the key associated with index *i* to this key :raises `IllegalArgumentException`: unless  $0 \leq i < \text{max\_n}$  :raises `NoSuchElementException`: if no key is associated with index *i*

**contains (i)**

Is i an index on this priority queue? :param i: an index :return: True if i is an index on this priority queue  
False otherwise :rtype: bool :raises IllegalArgumentException: unless  $0 \leq i < \text{max\_n}$

**decrease\_key (i, key)**

Decrease the key associated with index i to the specified value. :param i: the index of the key to decrease  
:param key: decrease the key associated with index i to this key :raises IllegalArgumentException: unless  
 $0 \leq i < \text{max\_n}$  :raises IllegalArgumentException: if  $\text{key} \geq \text{key\_of}(i)$  :raises NoSuchElementException:  
if no key is associated with index i

**del\_min ()**

Removes a minimum key and returns its associated index. :return: an index associated with a minimum  
key :raises NoSuchElementException: if this priority queue is empty :rtype: int

**delete (i)**

Remove the key associated with index i :param i: the index of the key to remove :raises IllegalArgu-  
mentException: unless  $0 \leq i < \text{max\_n}$  :raises NoSuchElementException: if no key is associated with  
index i

**increase\_key (i, key)**

Increase the key associated with index i to the specified value. :param i: the index of the key to increase  
:param key: increase the key associated with index i to this key :raises IllegalArgumentException: unless  
 $0 \leq i < \text{max\_n}$  :raises IllegalArgumentException: if  $\text{key} \leq \text{key\_of}(i)$  :raises NoSuchElementException:  
if no key is associated with index i

**insert (i, key)**

Associates key with index i :param i: an index :param key: the key to associate with index i :raises  
IllegalArgumentException: unless  $0 \leq i < \text{max\_n}$  :raises IllegalArgumentException: if there already is  
an item associated with index i

**is\_empty ()**

Returns True if this priority queue is empty. :return: True if this priority queue is empty False otherwise  
:rtype: bool

**key\_of (i)**

Returns the key associated with index i. :param i: the index of the key to return :return: the key associated  
with index i :raises IllegalArgumentException: unless  $0 \leq i < \text{max\_n}$  :raises NoSuchElementException:  
if no key is associated with index i

**min\_index ()**

Returns an index associated with a minimum key. :return: an index associated with a minimum key :rtype:  
int :raises NoSuchElementException: if this priority queue is empty

**min\_key ()**

Returns a minimum key. :return: a minimum key :raises NoSuchElementException: if this priority queue  
is empty

**size ()**

Returns the number of keys on this priority queue. :return: the number of keys on this priority queue  
:rtype: int

**itu.algs4.sorting.index\_min\_pq.main ()**

Inserts a bunch of strings to an indexed priority queue, deletes and prints them, inserts them again, and prints  
them using an iterator.

## 4.4 itu.algs4.sorting.insertion\_sort module

The Insertion module provides static methods for sorting an array using insertion sort.

This implementation makes  $\sim 1/2 n^2$  compares and exchanges in the worst case, so it is not suitable for sorting large arbitrary arrays. More precisely, the number of exchanges is exactly equal to the number of inversions. So, for example, it sorts a partially-sorted array in linear time. The sorting algorithm is stable and uses  $O(1)$  extra memory.

`itu.algs4.sorting.insertion_sort.is_sorted(a: List[T])`

Returns true if a is sorted. :param a: the array to be checked. :returns: True if a is sorted.

`itu.algs4.sorting.insertion_sort.main()`

Reads in a sequence of strings from standard input; Shellsorts them; and prints them to standard output in ascending order.

`itu.algs4.sorting.insertion_sort.sort(a: List[T])`

Rearranges the array in ascending order, using the natural order. :param a: the array to be sorted.

## 4.5 itu.algs4.sorting.max\_pq module

**class** `itu.algs4.sorting.max_pq.MaxPQ(_max: int = 1)`

Bases: `typing.Generic`

The MaxPQ class represents a priority queue of generic keys. It supports the usual insert and delete-the-maximum operations, along with methods for peeking at the maximum key, testing if the priority queue is empty, and iterating through the keys. This implementation uses a binary heap. The insert and delete-the-maximum operations take logarithmic amortized time. The max, size and is\_empty operations take constant time. Construction takes time proportional to the specified capacity.

**del\_max()** → `Key`

Removes and returns a largest key on this priority queue. :return: a largest key on this priority queue :raises NoSuchElementException: if this priority queue is empty

**insert(x: Key)** → `None`

Adds a new key to this priority queue. :param x: the new key to add to this priority queue

**is\_empty()** → `bool`

Returns True if this priority queue is empty. :return: True if this priority queue is empty otherwise False :rtype: bool

**max()** → `Key`

Returns a largest key on this priority queue. :return: a largest key on the priority queue :raises NoSuchElementException: if this priority queue is empty

**size()** → `int`

Returns the number of keys on this priority queue. :return: the number of keys on this priority queue :rtype: int

`itu.algs4.sorting.max_pq.main()`

Reads strings from stdin and adds them to a priority queue. When reading a '-' it removes a maximum item on the priority queue and prints it to stdout. Prints the amount of items left on the priority queue

## 4.6 itu.algs4.sorting.merge module

`itu.algs4.sorting.merge.sort(a: List[T])`

Rearranges the array in ascending order, using the natural order.

**Parameters** **a** – the array to be sorted

## 4.7 itu.algs4.sorting.merge\_bu module

This module provides functions for sorting an array using bottom-up mergesort.

For additional documentation, see Section 2.1 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

`itu.algs4.sorting.merge_bu.sort(a)`

Rearranges the array in ascending order, using the natural order.

**Parameters** `a` – the array to be sorted

## 4.8 itu.algs4.sorting.min\_pq module

**class** `itu.algs4.sorting.min_pq.MinPQ(_max: int = 1)`

Bases: `typing.Generic`

The MinPQ class represents a priority queue of generic keys. It supports the usual insert and delete-the-minimum operations, along with methods for peeking at the minimum key, testing if the priority queue is empty, and iterating through the keys. This implementation uses a binary heap. The insert and delete-the-minimum operations take logarithmic amortized time. The min, size and is-empty operations take constant time. Construction takes time proportional to the specified capacity.

**del\_min()** → `Key`

Removes and returns a smallest key on this priority queue. :return: a smallest key on this priority queue  
:raises `NoSuchElementException`: if this priority queue is empty

**insert(x: Key)** → `None`

Adds a new key to this priority queue. :param x: the new key to add to this priority queue

**is\_empty()** → `bool`

Returns True if this priority queue is empty. :return: True if this priority queue is empty otherwise False  
:rtype: bool

**min()** → `Key`

Returns a smallest key on this priority queue. :return: a smallest key on the priority queue :raises `NoSuchElementException`: if this priority queue is empty

**size()** → `int`

Returns the number of keys on this priority queue. :return: the number of keys on this priority queue  
:rtype: int

`itu.algs4.sorting.min_pq.main()`

Reads strings from stdin and adds them to a minimum priority queue. When reading a '-' it removes the minimum element and prints it to stdout.

## 4.9 itu.algs4.sorting.quick3way module

The Quick3Way module provides static methods for sorting an array using quicksort with 3-way partitioning.

`itu.algs4.sorting.quick3way.is_sorted(a)`

Returns true if a is sorted. :param a: the array to be checked. :returns: True if a is sorted.

`itu.algs4.sorting.quick3way.main()`

Reads in a sequence of strings from standard input; Shellsorts them; and prints them to standard output in ascending order.

```
itu.algs4.sorting.quick3way.sort (a)
```

Rearranges the array in ascending order using the natural order. :param a: the array to be sorted.

## 4.10 itu.algs4.sorting.quicksort module

The quicksort module provides methods for sorting an array and selecting the *i*th smallest element in an array using quicksort.

For additional documentation, see Section 2.3 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

**original author** Robert Sedgewick and Kevin Wayne

**original java code** <https://algs4.cs.princeton.edu/23quicksort/Quick.java.html>

```
itu.algs4.sorting.quicksort.is_sorted (array)
```

```
itu.algs4.sorting.quicksort.select (array, k)
```

Rearranges the array so that array[k] contains the kth smallest key; array[0] through array[k-1] are less than (or equal to) array[k]; and array[k+1] through array[n-1] are greater than (or equal to) array[k]

### Parameters

- **array** – the array
- **k** – the rank of the key

**Returns** the key of rank k

```
itu.algs4.sorting.quicksort.show (array)
```

```
itu.algs4.sorting.quicksort.sort (array)
```

Rearranges the array in ascending order, using the natural order

## 4.11 itu.algs4.sorting.selection module

```
itu.algs4.sorting.selection.main ()
```

Reads strings from stdin, sorts them, and prints the result to stdout.

```
itu.algs4.sorting.selection.sort (a)
```

Rearranges the array in ascending order, using the natural order. :param a: the array to be sorted

## 4.12 itu.algs4.sorting.shellsort module

The Shellsort module provides static methods for sorting an array using shellsort with Knuth's increment sequence (1, 4, 13, 40, ...).

```
itu.algs4.sorting.shellsort.is_sorted (a)
```

Returns true if a is sorted. :param a: the array to be checked. :returns: True if a is sorted.

```
itu.algs4.sorting.shellsort.main ()
```

Reads in a sequence of strings from standard input; Shellsorts them; and prints them to standard output in ascending order.

```
itu.algs4.sorting.shellsort.sort (a)
```

Rearranges the array in ascending order using the natural order. :param a: the array to be sorted.

## 4.13 Module contents

## 5.1 Submodules

## 5.2 `itu.algs4.stdlib.binary_out` module

```
class itu.algs4.stdlib.binary_out.BinaryOut (os=<_io.TextIOWrapper name='<stdout>'  
                                             mode='w' encoding='UTF-8'>)  
    Bases: object  
    close()  
    flush()  
    write_bool(x)  
    write_byte(x)  
    write_char(x)  
    write_int(x)  
    write_string(s)  
itu.algs4.stdlib.binary_out.main()
```

## 5.3 `itu.algs4.stdlib.binary_stdin` module

```
class itu.algs4.stdlib.binary_stdin.BinaryStdIn  
    Bases: object  
    EOF = -1  
    buffer_ = 0
```

```
static close()
    Close this input stream and release any associated system resources

ins = <_io.BufferedReader name=0>

static is_empty()

is_init = False

n = 0

static read_bool()

static read_char()

static read_int(r=32)

static read_string()

itu.algs4.stdlib.binary_stdin.main()
```

## 5.4 itu.algs4.stdlib.binary\_stdout module

```
class itu.algs4.stdlib.binary_stdout.BinaryStdOut
    Bases: object

    buffer_ = 0

    static close()

    static flush()

    is_init = False

    n = 0

    out = <_io.BufferedWriter name='<stdout>'>

    static write_bool(x)

    static write_byte(x)

    static write_char(x, r=8)

    static write_int(x, r=32)

    write_string(r=8)

itu.algs4.stdlib.binary_stdout.main()
```

## 5.5 itu.algs4.stdlib.color module

color.py

The color module defines the Color class and some popular Color objects.

```
class itu.algs4.stdlib.color.Color(r=0, g=0, b=0)
    Bases: object

    A Color object models an RGB color.

    getBlue()
        Return the blue component of self.
```



**getGreen()**  
Return the green component of self.

**getRed()**  
Return the red component of self.

## 5.6 itu.algs4.stdlib.instream module

instream.py

The instream module defines the InStream class.

**class** itu.algs4.stdlib.instream.InStream(*fileOrUrl=None*)

Bases: object

An InStream object wraps around a text file or sys.stdin, and supports reading from that stream.

Note: Usually it's a bad idea to mix these three sets of methods:

- isEmpty(), readInt(), readFloat(), readBool(), readString()
- hasNextLine(), readLine()
- readAll(), readAllInts(), readAllFloats(), readAllBools(), readAllStrings(), readAllLines()

Usually it's better to use one set exclusively.

**hasNextLine()**  
Return True iff the stream wrapped by self has a next line.

**isEmpty()**  
Return True iff no non-whitespace characters remain in the stream wrapped by self.

**readAll()**  
Read and return as a string all remaining lines of the stream wrapped by self.

**readAllBools()**  
Read all remaining strings from the stream wrapped by self, convert each to a bool, and return those bools in an array. Raise a ValueError if any of the strings cannot be converted to a bool.

**readAllFloats()**  
Read all remaining strings from the stream wrapped by self, convert each to a float, and return those floats in an array. Raise a ValueError if any of the strings cannot be converted to a float.

**readAllInts()**  
Read all remaining strings from the stream wrapped by self, convert each to an int, and return those ints in an array. Raise a ValueError if any of the strings cannot be converted to an int.

**readAllLines()**  
Read all remaining lines from the stream wrapped by self, and return them as strings in an array.

**readAllStrings()**  
Read all remaining strings from the stream wrapped by self, and return them in an array.

**readBool()**  
Discard leading white space characters from the stream wrapped by self. Then read from the stream a sequence of characters comprising a bool. Convert the sequence of characters to an bool, and return the bool. Raise an EOFError if no non-whitespace characters remain in the stream. Raise a ValueError if the next characters to be read from the stream cannot comprise an bool.

**readFloat ()**

Discard leading white space characters from the stream wrapped by self. Then read from the stream a sequence of characters comprising a float. Convert the sequence of characters to an float, and return the float. Raise an EOFError if no non-whitespace characters remain in the stream. Raise a ValueError if the next characters to be read from the stream cannot comprise a float.

**readInt ()**

Discard leading white space characters from the stream wrapped by self. Then read from the stream a sequence of characters comprising an integer. Convert the sequence of characters to an integer, and return the integer. Raise an EOFError if no non-whitespace characters remain in the stream. Raise a ValueError if the next characters to be read from the stream cannot comprise an integer.

**readLine ()**

Read and return as a string the next line of the stream wrapped by self. Raise an EOFError if there is no next line.

**readString ()**

Discard leading white space characters from the stream wrapped by self. Then read from the stream a sequence of characters comprising a string, and return the string. Raise an EOFError if no non-whitespace characters remain in the stream.

## 5.7 itu.algs4.stdlib.outstream module

outstream.py

The outstream module defines the OutStream class.

```
class itu.algs4.stdlib.outstream.OutStream (f=None)
    Bases: object
```

An OutStream object wraps around a text file or sys.stdout, and supports writing to that stream.

**write (x=)**

Write x to the stream wrapped by self.

**writef (fmt, \*args)**

Write each element of args to the stream wrapped by self. Use the format specified by string fmt.

**writeln (x=)**

Write x and an end-of-line mark to the stream wrapped by self.

## 5.8 itu.algs4.stdlib.picture module

picture.py

The picture module defines the Picture class.

```
class itu.algs4.stdlib.picture.Picture (arg1=None, arg2=None)
    Bases: object
```

A Picture object models an image. It is initialized such that it has a given width and height and contains all black pixels. Subsequently you can load an image from a given JPG or PNG file.

**get (x, y)**

Return the color of self at location (x, y).

**height ()**

Return the height of self.

**save** (*f*)  
 Save self to the file whose name is *f*.

**set** (*x*, *y*, *c*)  
 Set the color of self at location (*x*, *y*) to *c*.

**width** ()  
 Return the width of self.

## 5.9 itu.algs4.stdlib.stdarray module

stdarray.py

The stdarray module defines functions related to creating, reading, and writing one- and two-dimensional arrays.

`itu.algs4.stdlib.stdarray.create1D` (*length*, *value=None*)  
 Create and return a 1D array containing *length* elements, each initialized to *value*.

`itu.algs4.stdlib.stdarray.create2D` (*rowCount*, *colCount*, *value=None*)  
 Create and return a 2D array having *rowCount* rows and *colCount* columns, with each element initialized to *value*.

`itu.algs4.stdlib.stdarray.readBool1D` ()  
 Read from `sys.stdin` and return an array of booleans. An integer at the beginning of `sys.stdin` defines the array's length.

`itu.algs4.stdlib.stdarray.readBool2D` ()  
 Read from `sys.stdin` and return a two-dimensional array of booleans. Two integers at the beginning of `sys.stdin` define the array's dimensions.

`itu.algs4.stdlib.stdarray.readFloat1D` ()  
 Read from `sys.stdin` and return an array of floats. An integer at the beginning of `sys.stdin` defines the array's length.

`itu.algs4.stdlib.stdarray.readFloat2D` ()  
 Read from `sys.stdin` and return a two-dimensional array of floats. Two integers at the beginning of `sys.stdin` define the array's dimensions.

`itu.algs4.stdlib.stdarray.readInt1D` ()  
 Read from `sys.stdin` and return an array of integers. An integer at the beginning of `sys.stdin` defines the array's length.

`itu.algs4.stdlib.stdarray.readInt2D` ()  
 Read from `sys.stdin` and return a two-dimensional array of integers. Two integers at the beginning of `sys.stdin` define the array's dimensions.

`itu.algs4.stdlib.stdarray.write1D` (*a*)  
 Write array *a* to `sys.stdout`. First write its length. bool objects are written as 0 and 1, not False and True.

`itu.algs4.stdlib.stdarray.write2D` (*a*)  
 Write two-dimensional array *a* to `sys.stdout`. First write its dimensions. bool objects are written as 0 and 1, not False and True.

## 5.10 itu.algs4.stdlib.stdaudio module

## 5.11 itu.algs4.stdlib.stddraw module

stddraw.py

The stddraw module defines functions that allow the user to create a drawing. A drawing appears on the canvas. The canvas appears in the window. As a convenience, the module also imports the commonly used Color objects defined in the color module.

`itu.algs4.stdlib.stddraw.circle(x, y, r)`  
Draw on the background canvas a circle of radius *r* centered on (*x*, *y*).

`itu.algs4.stdlib.stddraw.clear(c=<itu.algs4.stdlib.color.Color object>)`  
Clear the background canvas to color *c*, where *c* is an object of class `color.Color`. *c* defaults to `std draw.WHITE`.

`itu.algs4.stdlib.stddraw.filledCircle(x, y, r)`  
Draw on the background canvas a filled circle of radius *r* centered on (*x*, *y*).

`itu.algs4.stdlib.stddraw.filledPolygon(x, y)`  
Draw on the background canvas a filled polygon with coordinates (*x*[*i*], *y*[*i*]).

`itu.algs4.stdlib.stddraw.filledRectangle(x, y, w, h)`  
Draw on the background canvas a filled rectangle of width *w* and height *h* whose lower left point is (*x*, *y*).

`itu.algs4.stdlib.stddraw.filledSquare(x, y, r)`  
Draw on the background canvas a filled square whose sides are of length *2r*, centered on (*x*, *y*).

`itu.algs4.stdlib.stddraw.hasNextKeyTyped()`  
Return True if the queue of keys the user typed is not empty. Otherwise return False.

`itu.algs4.stdlib.stddraw.line(x0, y0, x1, y1)`  
Draw on the background canvas a line from (*x*0, *y*0) to (*x*1, *y*1).

`itu.algs4.stdlib.stddraw.mousePressed()`  
Return True if the mouse has been left-clicked since the last time `mousePressed` was called, and False otherwise.

`itu.algs4.stdlib.stddraw.mouseX()`  
Return the *x* coordinate in user space of the location at which the mouse was most recently left-clicked. If a left-click hasn't happened yet, raise an exception, since `mouseX()` shouldn't be called until `mousePressed()` returns True.

`itu.algs4.stdlib.stddraw.mouseY()`  
Return the *y* coordinate in user space of the location at which the mouse was most recently left-clicked. If a left-click hasn't happened yet, raise an exception, since `mouseY()` shouldn't be called until `mousePressed()` returns True.

`itu.algs4.stdlib.stddraw.nextKeyTyped()`  
Remove the first key from the queue of keys that the the user typed, and return that key.

`itu.algs4.stdlib.stddraw.picture(pic, x=None, y=None)`  
Draw *pic* on the background canvas centered at (*x*, *y*). *pic* is an object of class `picture.Picture`. *x* and *y* default to the midpoint of the background canvas.

`itu.algs4.stdlib.stddraw.point(x, y)`  
Draw on the background canvas a point at (*x*, *y*).

`itu.algs4.stdlib.stddraw.polygon(x, y)`  
Draw on the background canvas a polygon with coordinates (*x*[*i*], *y*[*i*]).

`itu.algs4.stdlib.stddraw.rectangle(x, y, w, h)`  
 Draw on the background canvas a rectangle of width *w* and height *h* whose lower left point is (*x*, *y*).

`itu.algs4.stdlib.stddraw.save(f)`  
 Save the window canvas to file *f*.

`itu.algs4.stdlib.stddraw.setCanvasSize(w=512, h=512)`  
 Set the size of the canvas to *w* pixels wide and *h* pixels high. Calling this function is optional. If you call it, you must do so before calling any drawing function.

`itu.algs4.stdlib.stddraw.setFontFamily(f='Helvetica')`  
 Set the font family to *f* (e.g. 'Helvetica' or 'Courier').

`itu.algs4.stdlib.stddraw.setFontSize(s=12)`  
 Set the font size to *s* (e.g. 12 or 16).

`itu.algs4.stdlib.stddraw.setPenColor(c=<itu.algs4.stdlib.color.Color object>)`  
 Set the pen color to *c*, where *c* is an object of class `color.Color`. *c* defaults to `stdraw.BLACK`.

`itu.algs4.stdlib.stddraw.setPenRadius(r=0.005)`  
 Set the pen radius to *r*, thus affecting the subsequent drawing of points and lines. If *r* is 0.0, then points will be drawn with the minimum possible radius and lines with the minimum possible width.

`itu.algs4.stdlib.stddraw.setXscale(min=0.0, max=1.0)`  
 Set the x-scale of the canvas such that the minimum *x* value is *min* and the maximum *x* value is *max*.

`itu.algs4.stdlib.stddraw.setYscale(min=0.0, max=1.0)`  
 Set the y-scale of the canvas such that the minimum *y* value is *min* and the maximum *y* value is *max*.

`itu.algs4.stdlib.stddraw.show(msec=inf)`  
 Copy the background canvas to the window canvas, and then wait for *msec* milliseconds. *msec* defaults to infinity.

`itu.algs4.stdlib.stddraw.square(x, y, r)`  
 Draw on the background canvas a square whose sides are of length *2r*, centered on (*x*, *y*).

`itu.algs4.stdlib.stddraw.text(x, y, s)`  
 Draw string *s* on the background canvas centered at (*x*, *y*).

## 5.12 itu.algs4.stdlib.stdio module

`stdio.py`

The `stdio` module supports reading from standard input and writing to `sys.stdout`.

Note: Usually it's a bad idea to mix these three sets of reading functions:

- `isEmpty()`, `readInt()`, `readFloat()`, `readBool()`, `readString()`
- `hasNextLine()`, `readLine()`
- **`readAll()`, `readAllInts()`, `readAllFloats()`, `readAllBools()`, `readAllStrings()`, `readAllLines()`**

Usually it's better to use one set exclusively.

`itu.algs4.stdlib.stdio.eprint(*args, **kwargs)`

`itu.algs4.stdlib.stdio.hasNextLine()`  
 Return True if standard input has a next line. Otherwise return False.

`itu.algs4.stdlib.stdio.isEmpty()`  
 Return True if no non-whitespace characters remain in standard input. Otherwise return False.

`itu.algs4.stdlib.stdio.readAll()`  
Read and return as a string all remaining lines of standard input.

`itu.algs4.stdlib.stdio.readAllBools()`  
Read all remaining strings from standard input, convert each to a bool, and return those bools in an array. Raise a `ValueError` if any of the strings cannot be converted to a bool.

`itu.algs4.stdlib.stdio.readAllFloats()`  
Read all remaining strings from standard input, convert each to a float, and return those floats in an array. Raise a `ValueError` if any of the strings cannot be converted to a float.

`itu.algs4.stdlib.stdio.readAllInts()`  
Read all remaining strings from standard input, convert each to an int, and return those ints in an array. Raise a `ValueError` if any of the strings cannot be converted to an int.

`itu.algs4.stdlib.stdio.readAllLines()`  
Read all remaining lines from standard input, and return them as strings in an array.

`itu.algs4.stdlib.stdio.readAllStrings()`  
Read all remaining strings from standard input, and return them in an array.

`itu.algs4.stdlib.stdio.readBool()`  
Discard leading white space characters from standard input. Then read from standard input a sequence of characters comprising a bool. Convert the sequence of characters to a bool, and return the bool. Raise an `EOFError` if no non-whitespace characters remain in standard input. Raise a `ValueError` if the next characters to be read from standard input cannot comprise a bool.

These character sequences can comprise a bool: – True – False – 1 (means true) – 0 (means false)

`itu.algs4.stdlib.stdio.readFloat()`  
Discard leading white space characters from standard input. Then read from standard input a sequence of characters comprising a float. Convert the sequence of characters to a float, and return the float. Raise an `EOFError` if no non-whitespace characters remain in standard input. Raise a `ValueError` if the next characters to be read from standard input cannot comprise a float.

`itu.algs4.stdlib.stdio.readInt()`  
Discard leading white space characters from standard input. Then read from standard input a sequence of characters comprising an integer. Convert the sequence of characters to an integer, and return the integer. Raise an `EOFError` if no non-whitespace characters remain in standard input. Raise a `ValueError` if the next characters to be read from standard input cannot comprise an integer.

`itu.algs4.stdlib.stdio.readLine()`  
Read and return as a string the next line of standard input. Raise an `EOFError` if there is no next line.

`itu.algs4.stdlib.stdio.readString()`  
Discard leading white space characters from standard input. Then read from standard input a sequence of characters comprising a string, and return the string. Raise an `EOFError` if no non-whitespace characters remain in standard input.

`itu.algs4.stdlib.stdio.write(x="")`  
Write `x` to standard output.

`itu.algs4.stdlib.stdio.writef(fmt, *args)`  
Write each element of `args` to standard output. Use the format specified by string `fmt`.

`itu.algs4.stdlib.stdio.writeln(x="")`  
Write `x` and an end-of-line mark to standard output.

## 5.13 itu.algs4.stdlib.stdrandom module

stdrandom.py

The stdrandom module defines functions related to pseudo-random numbers.

`itu.algs4.stdlib.stdrandom.bernoulli` (*p=0.5*)

Return True with probability *p*.

`itu.algs4.stdlib.stdrandom.binomial` (*n, p=0.5*)

Return the number of heads in *n* coin flips, each of which is heads with probability *p*.

`itu.algs4.stdlib.stdrandom.discrete` (*a*)

Return a float from a discrete distribution: *i* with probability *a*[*i*]. Precondition: the elements of array *a* sum to 1.

`itu.algs4.stdlib.stdrandom.exp` (*lambda*)

Return a float from an exponential distribution with rate *lambda*.

`itu.algs4.stdlib.stdrandom.gaussian` (*mean=0.0, stddev=1.0*)

Return a float according to a standard Gaussian distribution with the given mean (*mean*) and standard deviation (*stddev*).

`itu.algs4.stdlib.stdrandom.seed` (*i=None*)

Seed the random number generator as `hash(i)`, where *i* is an int. If *i* is None, then seed using the current time or, quoting the help page for `random.seed()`, “an operating system specific randomness source if available.”

`itu.algs4.stdlib.stdrandom.shuffle` (*a*)

Shuffle array *a*.

`itu.algs4.stdlib.stdrandom.uniform` (*hi*)

Return an integer chosen uniformly from the range [0, *hi*).

`itu.algs4.stdlib.stdrandom.uniformFloat` (*lo, hi*)

Return a number chosen uniformly from the range [*lo*, *hi*).

`itu.algs4.stdlib.stdrandom.uniformInt` (*lo, hi*)

Return an integer chosen uniformly from the range [*lo*, *hi*).

## 5.14 itu.algs4.stdlib.stdstats module

stdstats.py

The stdstats module defines functions related to statistical analysis and graphical data display.

`itu.algs4.stdlib.stdstats.mean` (*a*)

Return the average of the elements of array *a*.

`itu.algs4.stdlib.stdstats.median` (*a*)

Return the median of the elements of array *a*.

`itu.algs4.stdlib.stdstats.plotBars` (*a*)

Plot the elements of array *a* as bars.

`itu.algs4.stdlib.stdstats.plotLines` (*a*)

Plot the elements of array *a* as line end-points.

`itu.algs4.stdlib.stdstats.plotPoints` (*a*)

Plot the elements of array *a* as points.

`itu.algs4.stdlib.stdstats.stddev(a)`

Return the standard deviation of the elements of array a.

`itu.algs4.stdlib.stdstats.var(a)`

Return the sample variance of the elements of array a.

## 5.15 Module contents

This module is based on the code at <https://introcs.cs.princeton.edu/python/code/> written by Robert Sedgewick, Kevin Wayne, and Robert Dondero



## 6.1 Submodules

## 6.2 `itu.algs4.strings.boyer_moore` module

**class** `itu.algs4.strings.boyer_moore.BoyerMoore` (*pat*)  
Bases: `object`

The `BoyerMoore` class finds the first occurrence of a pattern string in a text string. This implementation uses the Boyer-Moore algorithm (with the bad-character rule, but not the strong good suffix rule).

**search** (*txt*)

Returns the index of the first occurrence of the pattern string in the text string. :param *txt*: the text string  
:return: the index of the first occurrence of the pattern string in the text string; N if no such match

`itu.algs4.strings.boyer_moore.main()`

Takes a pattern string and an input string as command-line arguments; searches for the pattern string in the text string; and prints the first occurrence of the pattern string in the text string. Will print the pattern after the end of the string if no match is found.

## 6.3 `itu.algs4.strings.huffman_compression` module

The Huffman compression module provides static methods for compressing and expanding a binary input using Huffman codes over the 8-bit extended ASCII alphabet

For additional documentation, see Section 5.5 of *Algorithms*, 4th edition by Robert Sedgewick and Kevin Wayne.

`itu.algs4.strings.huffman_compression.compress()`

Reads a sequence of 8-bit bytes from standard input; compresses them using Huffman codes with an 8-bit alphabet; and writes the results to standard input.

`itu.algs4.strings.huffman_compression.expand()`  
Reads a sequence of bits that represents a Huffman-compressed message from standard input; expands them; and writes the results to standard output.

`itu.algs4.strings.huffman_compression.main()`  
Sample client that calls `compress()` if the command-line argument is “-“, and `expand()` if it is “+”.

## 6.4 itu.algs4.strings.kmp module

**class** `itu.algs4.strings.kmp.KMP` (*pat*)  
Bases: `object`

**search** (*txt*)  
Returns the index of the first occurrence of the pattern string  
in the text string. :param *txt*: the text string :return: the index of the first occurrence of the pattern string in the text string; N if no such match

`itu.algs4.strings.kmp.main()`  
Takes a pattern string and an input string as command-line arguments;  
searches for the pattern string in the text string; and prints the first occurrence of the pattern string in the text string. Will print the pattern after the end of the string if no match is found.

## 6.5 itu.algs4.strings.lsd module

This module provides functions for sorting arrays of strings using `lsd` sort.

For additional documentation, see Section 5.1 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

`itu.algs4.strings.lsd.sort` (*a*, *w*, *radix*=256)  
Rearranges the array of *w*-character strings in ascending order.

### Parameters

- **a** – the array to be sorted
- **w** – the number of characters per string
- **radix** – an optional number specifying the size of the alphabet to sort

## 6.6 itu.algs4.strings.lzw module

The LZW module provides static methods for compressing and expanding a binary input using LZW over the 8-bit extended ASCII alphabet with 12-bit codewords.

For additional documentation see Section 5.5 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

`itu.algs4.strings.lzw.compress()`  
Reads a sequence of 8-bit bytes from standard input; compresses them using LZW compression with 12-bit codewords; and writes the results to standard output.

`itu.algs4.strings.lzw.expand()`  
Reads a sequence of bit encoded using LZW compression with 12-bit codewords from standard input; expands them; and writes the results to standard output.

```
itu.algs4.strings.lzw.main()
```

Sample client that calls `compress()` if the command-line argument is “-“, and `expand()` if it is “+”.

Example: `echo huhu | python3 algs4/strings/lzw.py - | python3 algs4/strings/lzw.py +`

## 6.7 itu.algs4.strings.msd module

This module provides functions for sorting arrays of strings using msd sort.

For additional documentation, see Section 5.1 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

```
itu.algs4.strings.msd.sort(a, radix=256)
```

Rearranges the array of 32-bit integers in ascending order. Currently assumes that the integers are nonnegative.

**Parameters** `a` – the array to be sorted

## 6.8 itu.algs4.strings.nfa module

```
class itu.algs4.strings.nfa.NFA(regex)
```

Bases: `object`

The NFA class provides a data type for creating a nondeterministic finite state automaton (NFA) from a regular expression and testing whether a given string is matched by that regular expression. It supports the following operations: concatenation, closure, binary or, and parentheses, metacharacters (either in the text or pattern), capturing capabilities, greedy or reluctant/lazy modifiers, and other features in industrial-strength implementations such as Java’s Pattern and Matcher.

This implementation builds the NFA using a digraph and a stack and simulates the NFA using digraph search (see the textbook for details). The constructor takes time proportional to  $m$ , where  $m$  is the number of characters in the regular expression. The `recognizes()` method takes time proportional to  $m \cdot n$ , where  $n$  is the number of characters in the text.

For additional documentation, see section 5.4 of Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne.

```
recognizes(txt)
```

Returns True if the text is matched by the regular expression.

**Parameters** `txt` – the text

**Returns** True if the text is matched by the regular expression; False otherwise

```
itu.algs4.strings.nfa.main()
```

Unit tests the NFA data type

## 6.9 itu.algs4.strings.quick3string module

The Quick3String module provides functions for sorting an array of strings using 3-way radix quicksort

```
itu.algs4.strings.quick3string.is_sorted(a)
```

Returns true if `a` is sorted. :param `a`: the array to be checked. :returns: True if `a` is sorted.

```
itu.algs4.strings.quick3string.main()
```

Reads in a sequence of fixed-length strings from standard input; 3-way radix quicksorts them; and prints them to standard output in ascending order.

```
itu.algs4.strings.quick3string.sort(a)
```

Rearranges the array of strings in ascending order. :param a: the array to be sorted.

## 6.10 itu.algs4.strings.rabin\_karp module

```
class itu.algs4.strings.rabin_karp.RabinKarp(pat)
```

Bases: object

The RabinKarp class finds the first occurrence of a pattern string in a text string. This implementation uses the Monte Carlo version of the Rabin-Karp algorithm.

```
search(txt)
```

Returns the index of the first occurrence of the pattern string in the text string. :param txt: the text string  
:return: the index of the first occurrence of the pattern string in the text string; N if no such match

```
itu.algs4.strings.rabin_karp.long_random_prime(k)
```

Generates a random prime. :param k: the desired bit length of the prime :returns: a random prime of bit length k

```
itu.algs4.strings.rabin_karp.main()
```

Takes a pattern string and an input string as command-line arguments;

searches for the pattern string in the text string; and prints the first occurrence of the pattern string in the text string. Will print the pattern after the end of the string if no match is found.

## 6.11 itu.algs4.strings.trie\_st module

```
class itu.algs4.strings.trie_st.TrieST
```

Bases: object

```
class Node
```

Bases: object

```
R = 256
```

```
R = 256
```

```
contains(key)
```

```
delete(key)
```

```
get(key)
```

```
is_empty()
```

```
keys()
```

```
keys_that_match(pattern)
```

```
keys_with_prefix(prefix)
```

```
longest_prefix_of(query)
```

```
put(key, val)
```

```
size()
```

```
itu.algs4.strings.trie_st.test()
```

## 6.12 itu.algs4.strings.tst module

```
class itu.algs4.strings.tst.TST
    Bases: object

    class Node
        Bases: object

        contains (key)
        get (key)
        keys ()
        keys_that_match (pattern)
        keys_with_prefix (prefix)
        longest_prefix_of (query)
        put (key, val)
        size ()

    itu.algs4.strings.tst.test ()
```

## 6.13 Module contents



## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### i

itu.algs4.fundamentals, 7  
itu.algs4.fundamentals.bag, 1  
itu.algs4.fundamentals.binary\_search, 2  
itu.algs4.fundamentals.evaluate, 2  
itu.algs4.fundamentals.java\_helper, 2  
itu.algs4.fundamentals.queue, 2  
itu.algs4.fundamentals.stack, 3  
itu.algs4.fundamentals.three\_sum, 4  
itu.algs4.fundamentals.three\_sum\_fast, 4  
itu.algs4.fundamentals.two\_sum\_fast, 4  
itu.algs4.fundamentals.uf, 4  
itu.algs4.graphs, 29  
itu.algs4.graphs.acyclic\_lp, 9  
itu.algs4.graphs.acyclic\_sp, 10  
itu.algs4.graphs.Arbitrage, 9  
itu.algs4.graphs.bellman\_ford\_sp, 10  
itu.algs4.graphs.bipartite, 11  
itu.algs4.graphs.breadth\_first\_paths, 11  
itu.algs4.graphs.cc, 12  
itu.algs4.graphs.CPM, 9  
itu.algs4.graphs.cycle, 13  
itu.algs4.graphs.degrees\_of\_separation, 14  
itu.algs4.graphs.depth\_first\_order, 14  
itu.algs4.graphs.depth\_first\_paths, 15  
itu.algs4.graphs.depth\_first\_search, 15  
itu.algs4.graphs.digraph, 15  
itu.algs4.graphs.dijkstra\_all\_pairs\_sp, 17  
itu.algs4.graphs.dijkstra\_sp, 18  
itu.algs4.graphs.dijkstra\_undirected\_sp, 18  
itu.algs4.graphs.directed\_cycle, 19  
itu.algs4.graphs.directed\_dfs, 19  
itu.algs4.graphs.directed\_edge, 20  
itu.algs4.graphs.edge, 20  
itu.algs4.graphs.edge\_weighted\_digraph, 20  
itu.algs4.graphs.edge\_weighted\_directed\_cycle, 21  
itu.algs4.graphs.edge\_weighted\_directed\_cycle\_antonov, 22  
itu.algs4.graphs.edge\_weighted\_graph, 22  
itu.algs4.graphs.graph, 23  
itu.algs4.graphs.kosaraju\_sharir\_scc, 24  
itu.algs4.graphs.kruskal\_mst, 25  
itu.algs4.graphs.lazy\_prim\_mst, 26  
itu.algs4.graphs.prim\_mst, 26  
itu.algs4.graphs.symbol\_digraph, 27  
itu.algs4.graphs.symbol\_graph, 27  
itu.algs4.graphs.topological, 28  
itu.algs4.graphs.transitive\_closure, 28  
itu.algs4.searching, 42  
itu.algs4.searching.binary\_search\_st, 31  
itu.algs4.searching.bst, 34  
itu.algs4.searching.file\_index, 35  
itu.algs4.searching.frequency\_counter, 35  
itu.algs4.searching.linear\_probing\_hst, 35  
itu.algs4.searching.lookup\_csv, 36  
itu.algs4.searching.lookup\_index, 36  
itu.algs4.searching.red\_black\_bst, 36  
itu.algs4.searching.seperate\_chaining\_hst, 38  
itu.algs4.searching.sequential\_search\_st, 39  
itu.algs4.searching.set, 40  
itu.algs4.searching.sparse\_vector, 41  
itu.algs4.searching.st, 42  
itu.algs4.sorting, 48  
itu.algs4.sorting.heap, 43  
itu.algs4.sorting.index\_min\_pq, 43

itu.algs4.sorting.insertion\_sort, 44  
itu.algs4.sorting.max\_pq, 45  
itu.algs4.sorting.merge, 45  
itu.algs4.sorting.merge\_bu, 46  
itu.algs4.sorting.min\_pq, 46  
itu.algs4.sorting.quick3way, 46  
itu.algs4.sorting.quicksort, 47  
itu.algs4.sorting.selection, 47  
itu.algs4.sorting.shellsort, 47  
itu.algs4.stdlib, 58  
itu.algs4.stdlib.binary\_out, 49  
itu.algs4.stdlib.binary\_stdin, 49  
itu.algs4.stdlib.binary\_stdout, 50  
itu.algs4.stdlib.color, 50  
itu.algs4.stdlib.instream, 51  
itu.algs4.stdlib.outstream, 52  
itu.algs4.stdlib.picture, 52  
itu.algs4.stdlib.stdarray, 53  
itu.algs4.stdlib.stddraw, 54  
itu.algs4.stdlib.stdio, 55  
itu.algs4.stdlib.stdrandom, 57  
itu.algs4.stdlib.stdstats, 57  
itu.algs4.strings, 63  
itu.algs4.strings.boyer\_moore, 59  
itu.algs4.strings.huffman\_compression,  
    59  
itu.algs4.strings.kmp, 60  
itu.algs4.strings.lsd, 60  
itu.algs4.strings.lzw, 60  
itu.algs4.strings.msd, 61  
itu.algs4.strings.nfa, 61  
itu.algs4.strings.quick3string, 61  
itu.algs4.strings.rabin\_karp, 62  
itu.algs4.strings.trie\_st, 62  
itu.algs4.strings.tst, 63

## A

AcyclicLp (class in *itu.algs4.graphs.acyclic\_lp*), 9  
 AcyclicSP (class in *itu.algs4.graphs.acyclic\_sp*), 10  
 add() (*itu.algs4.fundamentals.bag.Bag* method), 1  
 add() (*itu.algs4.searching.set.SET* method), 40  
 add\_edge() (*itu.algs4.graphs.digraph.Digraph* method), 16  
 add\_edge() (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* method), 21  
 add\_edge() (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* method), 23  
 add\_edge() (*itu.algs4.graphs.graph.Graph* method), 24  
 adj() (*itu.algs4.graphs.digraph.Digraph* method), 16  
 adj() (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* method), 21  
 adj() (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* method), 23  
 adj() (*itu.algs4.graphs.graph.Graph* method), 24

## B

Bag (class in *itu.algs4.fundamentals.bag*), 1  
 Bag.Node (class in *itu.algs4.fundamentals.bag*), 1  
 BellmanFordSP (class in *itu.algs4.graphs.bellman\_ford\_sp*), 10  
 bernoulli() (in module *itu.algs4.stdlib.stdrandom*), 57  
 BinaryOut (class in *itu.algs4.stdlib.binary\_out*), 49  
 BinarySearchST (class in *itu.algs4.searching.binary\_search\_st*), 31  
 BinaryStdIn (class in *itu.algs4.stdlib.binary\_stdin*), 49  
 BinaryStdOut (class in *itu.algs4.stdlib.binary\_stdout*), 50  
 binomial() (in module *itu.algs4.stdlib.stdrandom*), 57  
 Bipartite (class in *itu.algs4.graphs.bipartite*), 11  
 Bipartite.UnsupportedOperationException, 11  
 BLACK (*itu.algs4.searching.red\_black\_bst.RedBlackBST*

attribute), 37  
 BoyerMoore (class in *itu.algs4.strings.boyer\_moore*), 59  
 BreadthFirstPaths (class in *itu.algs4.graphs.breadth\_first\_paths*), 11  
 BreadthFirstPathsBook (class in *itu.algs4.graphs.breadth\_first\_paths*), 12  
 BST (class in *itu.algs4.searching.bst*), 34  
 buffer\_ (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* attribute), 49  
 buffer\_ (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* attribute), 50

## C

CC (class in *itu.algs4.graphs.cc*), 12  
 CBook (class in *itu.algs4.graphs.cc*), 13  
 ceiling() (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 31  
 ceiling() (*itu.algs4.searching.bst.BST* method), 34  
 ceiling() (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
 ceiling() (*itu.algs4.searching.set.SET* method), 41  
 ceiling() (*itu.algs4.searching.st.ST* method), 42  
 change\_key() (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 43  
 circle() (in module *itu.algs4.stdlib.stddraw*), 54  
 clear() (in module *itu.algs4.stdlib.stddraw*), 54  
 close() (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49  
 close() (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* static method), 49  
 close() (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* static method), 50  
 Color (class in *itu.algs4.stdlib.color*), 50  
 color() (*itu.algs4.graphs.bipartite.Bipartite* method), 11  
 Comparable (class in *itu.algs4.searching.bst*), 35  
 Comparable (class in *itu.algs4.searching.red\_black\_bst*), 36

compress () (in module *itu.algs4.strings.huffman\_compression*), 59  
 compress () (in module *itu.algs4.strings.lzw*), 60  
 connected () (*itu.algs4.fundamentals.uf.QuickFindUF* method), 5  
 connected () (*itu.algs4.fundamentals.uf.QuickUnionUF* method), 5  
 connected () (*itu.algs4.fundamentals.uf.UF* method), 6  
 connected () (*itu.algs4.fundamentals.uf.WeightedQuickUnionUF* method), 6  
 connected () (*itu.algs4.graphs.cc.CC* method), 12  
 connected () (*itu.algs4.graphs.cc.CCBook* method), 13  
 contains () (*itu.algs4.graphs.symbol\_digraph.SymbolDigraph* method), 27  
 contains () (*itu.algs4.graphs.symbol\_graph.SymbolGraph* method), 27  
 contains () (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 31  
 contains () (*itu.algs4.searching.bst.BST* method), 34  
 contains () (*itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST* method), 35  
 contains () (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
 contains () (*itu.algs4.searching.seperate\_chaining\_hst.SeparateChainingHashST* method), 38  
 contains () (*itu.algs4.searching.sequential\_search\_st.SequentialSearchST* method), 40  
 contains () (*itu.algs4.searching.set.SET* method), 41  
 contains () (*itu.algs4.searching.st.ST* method), 42  
 contains () (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 43  
 contains () (*itu.algs4.strings.trie\_st.TrieST* method), 62  
 contains () (*itu.algs4.strings.tst.TST* method), 63  
 count () (*itu.algs4.fundamentals.three\_sum.ThreeSum* static method), 4  
 count () (*itu.algs4.fundamentals.three\_sum\_fast.ThreeSumFast* static method), 4  
 count () (*itu.algs4.fundamentals.two\_sum\_fast.TwoSumFast* static method), 4  
 count () (*itu.algs4.fundamentals.uf.QuickFindUF* method), 5  
 count () (*itu.algs4.fundamentals.uf.QuickUnionUF* method), 5  
 count () (*itu.algs4.fundamentals.uf.UF* method), 6  
 count () (*itu.algs4.fundamentals.uf.WeightedQuickUnionUF* method), 7  
 count () (*itu.algs4.graphs.cc.CC* method), 13  
 count () (*itu.algs4.graphs.cc.CCBook* method), 13  
 count () (*itu.algs4.graphs.depth\_first\_search.DepthFirstSearch* method), 15  
 count () (*itu.algs4.graphs.directed\_dfs.DirectedDFS* method), 19  
 count () (*itu.algs4.graphs.kosaraju\_sharir\_scc.KosarajuSharirSCC* method), 25  
 create1D () (in module *itu.algs4.stdlib.stdarray*), 53  
 create2D () (in module *itu.algs4.stdlib.stdarray*), 53  
 Cycle (class in *itu.algs4.graphs.cycle*), 13  
 cycle () (*itu.algs4.graphs.cycle.Cycle* method), 13  
 cycle () (*itu.algs4.graphs.directed\_cycle.DirectedCycle* method), 19  
 cycle () (*itu.algs4.graphs.edge\_weighted\_directed\_cycle.EdgeWeightedDirectedCycle* method), 22  
 cycle () (*itu.algs4.graphs.edge\_weighted\_directed\_cycle\_anton.EdgeWeightedDirectedCycleAnton* method), 22  
 decrease\_key () (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 44  
 degree () (*itu.algs4.graphs.digraph.Digraph* method), 16  
 degree () (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* method), 23  
 degrees\_of\_separation () (*itu.algs4.graphs.graph.Graph* method), 24  
 DegreesOfSeparation (class in *itu.algs4.graphs.degrees\_of\_separation*), 14  
 delete\_max () (*itu.algs4.sorting.max\_pq.MaxPQ* method), 45  
 delete\_min () (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 44  
 del\_min () (*itu.algs4.sorting.min\_pq.MinPQ* method), 46  
 delete () (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 32  
 delete () (*itu.algs4.searching.bst.BST* method), 34  
 delete () (*itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST* method), 36  
 delete () (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
 delete () (*itu.algs4.searching.seperate\_chaining\_hst.SeparateChainingHashST* method), 38  
 delete () (*itu.algs4.searching.sequential\_search\_st.SequentialSearchST* method), 40  
 delete () (*itu.algs4.searching.set.SET* method), 41  
 delete () (*itu.algs4.searching.st.ST* method), 42  
 delete () (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 44  
 delete () (*itu.algs4.strings.trie\_st.TrieST* method), 62  
 delete\_max () (*itu.algs4.searching.bst.BST* method), 34  
 delete\_max () (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
 delete\_min () (*itu.algs4.searching.bst.BST* method), 34

`delete_min()` (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
`delete_min()` (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* method), 22  
`deleteMax()` (*itu.algs4.searching.binary\_search\_st.BinarySearchTree* method), 23  
`deleteMax()` (*itu.algs4.graphs.graph.Graph* method), 23  
`deleteMin()` (*itu.algs4.searching.binary\_search\_st.BinarySearchTree* method), 20  
`deleteMin()` (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* method), 21  
`DepthFirstOrder` (class in *itu.algs4.graphs.depth\_first\_order*), 14  
`DepthFirstOrder` (*edges()* (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* method), 23  
`DepthFirstPaths` (class in *itu.algs4.graphs.depth\_first\_paths*), 15  
`DepthFirstPaths` (*edges()* (*itu.algs4.graphs.kruskal\_mst.KruskalMST* method), 25  
`DepthFirstSearch` (class in *itu.algs4.graphs.depth\_first\_search*), 15  
`DepthFirstSearch` (*edges()* (*itu.algs4.graphs.lazy\_prim\_mst.LazyPrimMST* method), 26  
`dequeue()` (*itu.algs4.fundamentals.queue.Queue* method), 2  
`dequeue()` (*edges()* (*itu.algs4.graphs.prim\_mst.PrimMST* method), 26  
`Digraph` (class in *itu.algs4.graphs.digraph*), 15  
`Digraph` (*EdgeWeightedDigraph* (class in *itu.algs4.graphs.edge\_weighted\_digraph*), 20  
`digraph()` (*itu.algs4.graphs.symbol\_digraph.SymbolDigraph* method), 27  
`DijkstraAllPairsSP` (class in *itu.algs4.graphs.dijkstra\_all\_pairs\_sp*), 17  
`DijkstraAllPairsSP` (*EdgeWeightedDirectedCycle* (class in *itu.algs4.graphs.edge\_weighted\_directed\_cycle*), 21  
`DijkstraSP` (class in *itu.algs4.graphs.dijkstra\_sp*), 18  
`DijkstraSP` (*EdgeWeightedDirectedCycle* (class in *itu.algs4.graphs.edge\_weighted\_directed\_cycle\_anton*), 22  
`DijkstraUndirectedSP` (class in *itu.algs4.graphs.dijkstra\_undirected\_sp*), 18  
`dimension()` (*itu.algs4.searching.sparse\_vector.SparseVector* method), 41  
`dimension()` (*EdgeWeightedGraph* (class in *itu.algs4.graphs.edge\_weighted\_graph*), 22  
`DirectedCycle` (class in *itu.algs4.graphs.directed\_cycle*), 19  
`DirectedCycle` (*either()* (*itu.algs4.graphs.edge.Edge* method), 20  
`DirectedDFS` (class in *itu.algs4.graphs.directed\_dfs*), 19  
`DirectedDFS` (*enqueue()* (*itu.algs4.fundamentals.queue.Queue* method), 2  
`DirectedEdge` (class in *itu.algs4.graphs.directed\_edge*), 20  
`DirectedEdge` (*EOF* (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* attribute), 49  
`discrete()` (in module *itu.algs4.stdlib.stdrandom*), 57  
`discrete()` (*eprint()* (in module *itu.algs4.stdlib.stdio*), 55  
`dist()` (*itu.algs4.graphs.dijkstra\_all\_pairs\_sp.DijkstraAllPairsSP* method), 17  
`dist()` (*evaluate()* (in module *itu.algs4.fundamentals.evaluate*), 2  
`dist_to()` (*itu.algs4.graphs.acyclic\_lp.AcyclicLp* method), 9  
`dist_to()` (*exp()* (in module *itu.algs4.stdlib.stdrandom*), 57  
`dist_to()` (*itu.algs4.graphs.acyclic\_sp.AcyclicSP* method), 10  
`dist_to()` (*expand()* (in module *itu.algs4.strings.huffman\_compression*), 59  
`dist_to()` (*itu.algs4.graphs.bellman\_ford\_sp.BellmanFordSP* method), 10  
`dist_to()` (*expand()* (in module *itu.algs4.strings.lzw*), 60  
`dist_to()` (*itu.algs4.graphs.breadth\_first\_paths.BreadthFirstPaths* method), 12  
`dist_to()` (*fillCircle()* (in module *itu.algs4.stdlib.stddraw*), 54  
`dist_to()` (*itu.algs4.graphs.dijkstra\_sp.DijkstraSP* method), 18  
`dist_to()` (*filledPolygon()* (in module *itu.algs4.stdlib.stddraw*), 54  
`dist_to()` (*itu.algs4.graphs.dijkstra\_undirected\_sp.DijkstraUndirectedSP* method), 18  
`dist_to()` (*fillRectangle()* (in module *itu.algs4.stdlib.stddraw*), 54  
`dot()` (*itu.algs4.searching.sparse\_vector.SparseVector* method), 41  
`dot()` (*filledSquare()* (in module *itu.algs4.stdlib.stddraw*), 54  
**E**  
`E()` (*itu.algs4.graphs.digraph.Digraph* method), 16  
`E()` (*find()* (*itu.algs4.fundamentals.uf.QuickFindUF* method), 5  
`E()` (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* method), 5  
`E()` (*find()* (*itu.algs4.fundamentals.uf.QuickUnionUF* method), 5  
`E()` (*find()* (*itu.algs4.fundamentals.uf.UF* method), 6

`find()` (*itu.algs4.fundamentals.uf.WeightedQuickUnionUF* method), 7  
`FixedCapacityStack` (class in *itu.algs4.fundamentals.stack*), 3  
`FLOATING_POINT_EPSILON` (*itu.algs4.graphs.lazy\_prim\_mst.LazyPrimMST* attribute), 26  
`FLOATING_POINT_EPSILON` (*itu.algs4.graphs.prim\_mst.PrimMST* attribute), 26  
`floor()` (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 32  
`floor()` (*itu.algs4.searching.bst.BST* method), 34  
`floor()` (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
`floor()` (*itu.algs4.searching.set.SET* method), 41  
`floor()` (*itu.algs4.searching.st.ST* method), 42  
`flush()` (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49  
`flush()` (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* static method), 50  
`from_graph()` (*itu.algs4.graphs.digraph.Digraph* static method), 16  
`from_graph()` (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* static method), 21  
`from_graph()` (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* static method), 23  
`from_graph()` (*itu.algs4.graphs.graph.Graph* static method), 24  
`from_stream()` (*itu.algs4.graphs.digraph.Digraph* static method), 16  
`from_stream()` (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* static method), 21  
`from_stream()` (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* static method), 23  
`from_stream()` (*itu.algs4.graphs.graph.Graph* static method), 24  
`from_vertex()` (*itu.algs4.graphs.directed\_edge.DirectedEdge* method), 20  
  
**G**  
`gaussian()` (in module *itu.algs4.stdlib.stdrandom*), 57  
`get()` (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 32  
`get()` (*itu.algs4.searching.bst.BST* method), 34  
`get()` (*itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST* method), 36  
`get()` (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37  
`get()` (*itu.algs4.searching.seperate\_chaining\_hst.SeparateChainingHashST* method), 39  
`get()` (*itu.algs4.searching.sequential\_search\_st.SequentialSearchST* method), 40  
`get()` (*itu.algs4.searching.sparse\_vector.SparseVector* method), 41  
`get()` (*itu.algs4.searching.st.ST* method), 42  
`get()` (*itu.algs4.stdlib.picture.Picture* method), 52  
`get()` (*itu.algs4.strings.trie\_st.TrieST* method), 62  
`get()` (*itu.algs4.strings.tst.TST* method), 63  
`getBlue()` (*itu.algs4.stdlib.color.Color* method), 50  
`getGreen()` (*itu.algs4.stdlib.color.Color* method), 50  
`getRed()` (*itu.algs4.stdlib.color.Color* method), 51  
`Graph` (class in *itu.algs4.graphs.graph*), 23  
`graph()` (*itu.algs4.graphs.symbol\_graph.SymbolGraph* method), 27  
  
**H**  
`has_cycle()` (*itu.algs4.graphs.cycle.Cycle* method), 13  
`has_cycle()` (*itu.algs4.graphs.directed\_cycle.DirectedCycle* method), 19  
`has_cycle()` (*itu.algs4.graphs.edge\_weighted\_directed\_cycle.EdgeWeightedDirectedCycle* method), 22  
`has_cycle()` (*itu.algs4.graphs.edge\_weighted\_directed\_cycle\_anton.EdgeWeightedDirectedCycleAnton* method), 22  
`has_negative_cycle()` (*itu.algs4.graphs.bellman\_ford\_sp.BellmanFordSP* method), 10  
`has_negative_cycle()` (*itu.algs4.graphs.topological.Topological* method), 28  
`has_path()` (*itu.algs4.graphs.dijkstra\_all\_pairs\_sp.DijkstraAllPairsSP* method), 17  
`has_path_to()` (*itu.algs4.graphs.acyclic\_lp.AcyclicLp* method), 9  
`has_path_to()` (*itu.algs4.graphs.acyclic\_sp.AcyclicSP* method), 10  
`has_path_to()` (*itu.algs4.graphs.bellman\_ford\_sp.BellmanFordSP* method), 10  
`has_path_to()` (*itu.algs4.graphs.breadth\_first\_paths.BreadthFirstPaths* method), 12  
`has_path_to()` (*itu.algs4.graphs.breadth\_first\_paths.BreadthFirstPaths* method), 12  
`has_path_to()` (*itu.algs4.graphs.depth\_first\_paths.DepthFirstPaths* method), 15  
`has_path_to()` (*itu.algs4.graphs.dijkstra\_sp.DijkstraSP* method), 18  
`has_path_to()` (*itu.algs4.graphs.dijkstra\_undirected\_sp.DijkstraUndirectedSP* method), 18  
`hasNext()` (*itu.algs4.searching.set.SET* method), 41  
`hasNextKeyTyped()` (in module *itu.algs4.stdlib.stddraw*), 54  
`hasNextLine()` (in module *itu.algs4.stdlib.stdio*), 55  
`hasNextLine()` (*itu.algs4.stdlib.instream.InStream* method), 51  
`height()` (*itu.algs4.searching.bst.BST* method), 34  
`height()` (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 37



height() (itu.algs4.stdlib.picture.Picture method), 52

**I**

id() (itu.algs4.graphs.cc.CC method), 13

id() (itu.algs4.graphs.cc.CCBook method), 13

id() (itu.algs4.graphs.kosaraju\_sharir\_scc.KosarajuSharirSCC method), 25

increase\_key() (itu.algs4.sorting.index\_min\_pq.IndexMinPQ method), 44

indegree() (itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph method), 21

index\_of() (in module itu.algs4.fundamentals.binary\_search), 2

index\_of() (itu.algs4.graphs.symbol\_digraph.SymbolDigraph method), 27

index\_of() (itu.algs4.graphs.symbol\_graph.SymbolGraph method), 28

IndexMinPQ (class in itu.algs4.sorting.index\_min\_pq), 43

ins (itu.algs4.stdlib.binary\_stdin.BinaryStdIn attribute), 50

insert() (itu.algs4.sorting.index\_min\_pq.IndexMinPQ method), 44

insert() (itu.algs4.sorting.max\_pq.MaxPQ method), 45

insert() (itu.algs4.sorting.min\_pq.MinPQ method), 46

InStream (class in itu.algs4.stdlib.instream), 51

intersects() (itu.algs4.searching.set.SET method), 41

is\_bipartite() (itu.algs4.graphs.bipartite.Bipartite method), 11

is\_empty() (itu.algs4.fundamentals.bag.Bag method), 1

is\_empty() (itu.algs4.fundamentals.queue.Queue method), 2

is\_empty() (itu.algs4.fundamentals.stack.FixedCapacityStack method), 3

is\_empty() (itu.algs4.fundamentals.stack.ResizingArrayStack method), 3

is\_empty() (itu.algs4.fundamentals.stack.Stack method), 3

is\_empty() (itu.algs4.searching.binary\_search\_st.BinarySearchST method), 32

is\_empty() (itu.algs4.searching.bst.BST method), 34

is\_empty() (itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST method), 36

is\_empty() (itu.algs4.searching.red\_black\_bst.RedBlackBST method), 37

is\_empty() (itu.algs4.searching.seperate\_chaining\_hst.SeperateChainingHashST method), 39

is\_empty() (itu.algs4.searching.sequential\_search\_st.SequentialSearchST method), 40

is\_empty() (itu.algs4.searching.set.SET method), 41

is\_empty() (itu.algs4.searching.st.ST method), 42

is\_empty() (itu.algs4.sorting.index\_min\_pq.IndexMinPQ method), 44

is\_empty() (itu.algs4.sorting.max\_pq.MaxPQ method), 45

is\_empty() (itu.algs4.sorting.min\_pq.MinPQ method), 46

is\_empty() (itu.algs4.stdlib.binary\_stdin.BinaryStdIn static method), 50

is\_swapped() (itu.algs4.strings.trie\_st.TrieST method), 62

is\_init (itu.algs4.stdlib.binary\_stdin.BinaryStdIn attribute), 50

is\_init (itu.algs4.stdlib.binary\_stdout.BinaryStdOut attribute), 50

is\_marked() (itu.algs4.graphs.directed\_dfs.DirectedDFS method), 19

is\_sorted() (in module itu.algs4.sorting.insertion\_sort), 45

is\_sorted() (in module itu.algs4.sorting.quick3way), 46

is\_sorted() (in module itu.algs4.sorting.quicksort), 47

is\_sorted() (in module itu.algs4.sorting.shellsort), 47

is\_sorted() (in module itu.algs4.strings.quick3string), 61

isEmpty() (in module itu.algs4.stdlib.stdio), 55

isEmpty() (itu.algs4.stdlib.instream.InStream method), 51

itu.algs4.fundamentals (module), 7

itu.algs4.fundamentals.bag (module), 1

itu.algs4.fundamentals.binary\_search (module), 2

itu.algs4.fundamentals.evaluate (module), 2

itu.algs4.fundamentals.java\_helper (module), 2

itu.algs4.fundamentals.queue (module), 2

itu.algs4.fundamentals.stack (module), 3

itu.algs4.fundamentals.three\_sum (module), 4

itu.algs4.fundamentals.three\_sum\_fast (module), 4

itu.algs4.fundamentals.two\_sum\_fast (module), 4

itu.algs4.fundamentals.uf (module), 4

itu.algs4.graphs (module), 29

itu.algs4.graphs.acyclic\_lp (module), 9

itu.algs4.graphs.acyclic\_sp (module), 10

itu.algs4.graphs.Arbitrage (module), 9

itu.algs4.graphs.bellman\_ford\_sp (module), 10

itu.algs4.graphs.bipartite (module), 11

itu.algs4.graphs.breadth\_first\_paths (module), 11  
itu.algs4.graphs.cc (module), 12  
itu.algs4.graphs.CPM (module), 9  
itu.algs4.graphs.cycle (module), 13  
itu.algs4.graphs.degrees\_of\_separation (module), 14  
itu.algs4.graphs.depth\_first\_order (module), 14  
itu.algs4.graphs.depth\_first\_paths (module), 15  
itu.algs4.graphs.depth\_first\_search (module), 15  
itu.algs4.graphs.digraph (module), 15  
itu.algs4.graphs.dijkstra\_all\_pairs\_sp (module), 17  
itu.algs4.graphs.dijkstra\_sp (module), 18  
itu.algs4.graphs.dijkstra\_undirected\_sp (module), 18  
itu.algs4.graphs.directed\_cycle (module), 19  
itu.algs4.graphs.directed\_dfs (module), 19  
itu.algs4.graphs.directed\_edge (module), 20  
itu.algs4.graphs.edge (module), 20  
itu.algs4.graphs.edge\_weighted\_digraph (module), 20  
itu.algs4.graphs.edge\_weighted\_directed\_cycle (module), 21  
itu.algs4.graphs.edge\_weighted\_directed\_cycle\_graph (module), 22  
itu.algs4.graphs.edge\_weighted\_graph (module), 22  
itu.algs4.graphs.graph (module), 23  
itu.algs4.graphs.kosaraju\_sharir\_scc (module), 24  
itu.algs4.graphs.kruskal\_mst (module), 25  
itu.algs4.graphs.lazy\_prim\_mst (module), 26  
itu.algs4.graphs.prim\_mst (module), 26  
itu.algs4.graphs.symbol\_digraph (module), 27  
itu.algs4.graphs.symbol\_graph (module), 27  
itu.algs4.graphs.topological (module), 28  
itu.algs4.graphs.transitive\_closure (module), 28  
itu.algs4.searching (module), 42  
itu.algs4.searching.binary\_search\_st (module), 31  
itu.algs4.searching.bst (module), 34  
itu.algs4.searching.file\_index (module), 35  
itu.algs4.searching.frequency\_counter (module), 35  
itu.algs4.searching.linear\_probing\_hst (module), 35  
itu.algs4.searching.lookup\_csv (module), 36  
itu.algs4.searching.lookup\_index (module), 36  
itu.algs4.searching.red\_black\_bst (module), 36  
itu.algs4.searching.separate\_chaining\_hst (module), 38  
itu.algs4.searching.sequential\_search\_st (module), 39  
itu.algs4.searching.set (module), 40  
itu.algs4.searching.sparse\_vector (module), 41  
itu.algs4.searching.st (module), 42  
itu.algs4.sorting (module), 48  
itu.algs4.sorting.heap (module), 43  
itu.algs4.sorting.index\_min\_pq (module), 43  
itu.algs4.sorting.insertion\_sort (module), 44  
itu.algs4.sorting.max\_pq (module), 45  
itu.algs4.sorting.merge (module), 45  
itu.algs4.sorting.merge\_bu (module), 46  
itu.algs4.sorting.min\_pq (module), 46  
itu.algs4.sorting.quick3way (module), 46  
itu.algs4.sorting.quicksort (module), 47  
itu.algs4.sorting.selection (module), 47  
itu.algs4.sorting.shell\_sort (module), 47  
itu.algs4.sorting.shell\_sort (module), 47  
itu.algs4.stdlib (module), 58  
itu.algs4.stdlib.binary\_out (module), 49  
itu.algs4.stdlib.binary\_stdin (module), 49  
itu.algs4.stdlib.binary\_stdout (module), 50  
itu.algs4.stdlib.color (module), 50  
itu.algs4.stdlib.instream (module), 51  
itu.algs4.stdlib.outstream (module), 52  
itu.algs4.stdlib.picture (module), 52  
itu.algs4.stdlib.stdarray (module), 53  
itu.algs4.stdlib.stddraw (module), 54  
itu.algs4.stdlib.stdio (module), 55  
itu.algs4.stdlib.stdrandom (module), 57  
itu.algs4.stdlib.stdstats (module), 57  
itu.algs4.strings (module), 63  
itu.algs4.strings.boyer\_moore (module), 59  
itu.algs4.strings.huffman\_compression (module), 59  
itu.algs4.strings.kmp (module), 60  
itu.algs4.strings.lsd (module), 60  
itu.algs4.strings.lzw (module), 60  
itu.algs4.strings.msds (module), 61  
itu.algs4.strings.nfa (module), 61



itu.algs4.strings.quick3string (module), 61  
 itu.algs4.strings.rabin\_karp (module), 62  
 itu.algs4.strings.trie\_st (module), 62  
 itu.algs4.strings.tst (module), 63

## J

java\_string\_hash() (in module itu.algs4.fundamentals.java\_helper), 2

## K

key\_list() (itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST method), 36  
 key\_of() (itu.algs4.sorting.index\_min\_pq.IndexMinPQ method), 44  
 keys() (itu.algs4.searching.binary\_search\_st.BinarySearchST method), 32  
 keys() (itu.algs4.searching.bst.BST method), 34  
 keys() (itu.algs4.searching.red\_black\_bst.RedBlackBST method), 37  
 keys() (itu.algs4.searching.seperate\_chaining\_hst.SeparateChainingHashST method), 39  
 keys() (itu.algs4.searching.sequential\_search\_st.SequentialSearchST method), 40  
 keys() (itu.algs4.searching.st.ST method), 42  
 keys() (itu.algs4.strings.trie\_st.TrieST method), 62  
 keys() (itu.algs4.strings.tst.TST method), 63  
 keys\_between() (itu.algs4.searching.binary\_search\_st.BinarySearchST method), 32  
 keys\_range() (itu.algs4.searching.red\_black\_bst.RedBlackBST method), 37  
 keys\_that\_match() (itu.algs4.strings.trie\_st.TrieST method), 62  
 keys\_that\_match() (itu.algs4.strings.tst.TST method), 63  
 keys\_with\_prefix() (itu.algs4.strings.trie\_st.TrieST method), 62  
 keys\_with\_prefix() (itu.algs4.strings.tst.TST method), 63  
 KMP (class in itu.algs4.strings.kmp), 60  
 KosarajuSharirSCC (class in itu.algs4.graphs.kosaraju\_sharir\_scc), 25  
 KruskalMST (class in itu.algs4.graphs.kruskal\_mst), 25

## L

LazyPrimMST (class in itu.algs4.graphs.lazy\_prim\_mst), 26  
 level\_order() (itu.algs4.searching.bst.BST method), 34  
 line() (in module itu.algs4.stdlib.stddraw), 54

LinearProbingHashST (class in itu.algs4.searching.linear\_probing\_hst), 35  
 long\_random\_prime() (in module itu.algs4.strings.rabin\_karp), 62  
 longest\_prefix\_of() (itu.algs4.strings.trie\_st.TrieST method), 62  
 longest\_prefix\_of() (itu.algs4.strings.tst.TST method), 63

## M

magnitude() (itu.algs4.searching.sparse\_vector.SparseVector method), 41  
 main() (in module itu.algs4.fundamentals.binary\_search), 2  
 main() (in module itu.algs4.graphs.bellman\_ford\_sp), 11  
 main() (in module itu.algs4.graphs.dijkstra\_sp), 18  
 main() (in module itu.algs4.graphs.dijkstra\_undirected\_sp), 18  
 main() (in module itu.algs4.graphs.directed\_dfs), 19  
 main() (in module itu.algs4.graphs.directed\_edge), 20  
 main() (in module itu.algs4.graphs.edge), 20  
 main() (in module itu.algs4.graphs.edge\_weighted\_digraph), 21  
 main() (in module itu.algs4.graphs.edge\_weighted\_directed\_cycle), 22  
 main() (in module itu.algs4.graphs.edge\_weighted\_graph), 23  
 main() (in module itu.algs4.graphs.kosaraju\_sharir\_scc), 25  
 main() (in module itu.algs4.graphs.kruskal\_mst), 25  
 main() (in module itu.algs4.graphs.transitive\_closure), 29  
 main() (in module itu.algs4.searching.linear\_probing\_hst), 36  
 main() (in module itu.algs4.searching.seperate\_chaining\_hst), 39  
 main() (in module itu.algs4.searching.set), 41  
 main() (in module itu.algs4.searching.sparse\_vector), 42  
 main() (in module itu.algs4.sorting.heap), 43  
 main() (in module itu.algs4.sorting.index\_min\_pq), 44  
 main() (in module itu.algs4.sorting.insertion\_sort), 45  
 main() (in module itu.algs4.sorting.max\_pq), 45  
 main() (in module itu.algs4.sorting.min\_pq), 46  
 main() (in module itu.algs4.sorting.quick3way), 46  
 main() (in module itu.algs4.sorting.selection), 47  
 main() (in module itu.algs4.sorting.shellsort), 47  
 main() (in module itu.algs4.stdlib.binary\_out), 49  
 main() (in module itu.algs4.stdlib.binary\_stdin), 50  
 main() (in module itu.algs4.stdlib.binary\_stdout), 50  
 main() (in module itu.algs4.strings.boyer\_moore), 59

main() (in module *itu.algs4.strings.huffman\_compression*), 60  
 main() (in module *itu.algs4.strings.kmp*), 60  
 main() (in module *itu.algs4.strings.lzw*), 60  
 main() (in module *itu.algs4.strings.nfa*), 61  
 main() (in module *itu.algs4.strings.quick3string*), 61  
 main() (in module *itu.algs4.strings.rabin\_karp*), 62  
 main() (*itu.algs4.graphs.degrees\_of\_separation.DegreesOfSeparation* static method), 14  
 marked() (*itu.algs4.graphs.depth\_first\_search.DepthFirstSearch* method), 15  
 max() (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33  
 max() (*itu.algs4.searching.bst.BST* method), 34  
 max() (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38  
 max() (*itu.algs4.searching.set.SET* method), 41  
 max() (*itu.algs4.searching.st.ST* method), 42  
 max() (*itu.algs4.sorting.max\_pq.MaxPQ* method), 45  
 MaxPQ (class in *itu.algs4.sorting.max\_pq*), 45  
 mean() (in module *itu.algs4.stdlib.stdstats*), 57  
 median() (in module *itu.algs4.stdlib.stdstats*), 57  
 min() (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33  
 min() (*itu.algs4.searching.bst.BST* method), 34  
 min() (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38  
 min() (*itu.algs4.searching.set.SET* method), 41  
 min() (*itu.algs4.searching.st.ST* method), 42  
 min() (*itu.algs4.sorting.min\_pq.MinPQ* method), 46  
 min\_index() (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 44  
 min\_key() (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 44  
 MinPQ (class in *itu.algs4.sorting.min\_pq*), 46  
 mousePressed() (in module *itu.algs4.stdlib.stddraw*), 54  
 mouseX() (in module *itu.algs4.stdlib.stddraw*), 54  
 mouseY() (in module *itu.algs4.stdlib.stddraw*), 54  
**N**  
 n (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* attribute), 50  
 n (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* attribute), 50  
 name\_of() (*itu.algs4.graphs.symbol\_digraph.SymbolDigraph* method), 27  
 name\_of() (*itu.algs4.graphs.symbol\_graph.SymbolGraph* method), 28  
 negative\_cycle() (*itu.algs4.graphs.bellman\_ford\_sp.BellmanFordSP* method), 10  
 nextKeyTyped() (in module *itu.algs4.stdlib.stddraw*), 54  
 NFA (class in *itu.algs4.strings.nfa*), 61  
 nnz() (*itu.algs4.searching.sparse\_vector.SparseVector* method), 41  
 Node (class in *itu.algs4.fundamentals.queue*), 2  
 Node (class in *itu.algs4.fundamentals.stack*), 3  
 Node (class in *itu.algs4.searching.bst*), 35  
 Node (class in *itu.algs4.searching.red\_black\_bst*), 36  
**O**  
 odd\_cycle() (*itu.algs4.graphs.bipartite.Bipartite* method), 11  
 order() (*itu.algs4.graphs.topological.Topological* method), 28  
 other() (*itu.algs4.graphs.edge.Edge* method), 20  
 out (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* attribute), 50  
 outdegree() (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* method), 21  
 OutputStream (class in *itu.algs4.stdlib.outstream*), 52  
**P**  
 path() (*itu.algs4.graphs.dijkstra\_all\_pairs\_sp.DijkstraAllPairsSP* method), 17  
 path\_to() (*itu.algs4.graphs.acyclic\_lp.AcyclicLp* method), 9  
 path\_to() (*itu.algs4.graphs.acyclic\_sp.AcyclicSP* method), 10  
 path\_to() (*itu.algs4.graphs.bellman\_ford\_sp.BellmanFordSP* method), 11  
 path\_to() (*itu.algs4.graphs.breadth\_first\_paths.BreadthFirstPaths* method), 12  
 path\_to() (*itu.algs4.graphs.breadth\_first\_paths.BreadthFirstPathsBook* method), 12  
 path\_to() (*itu.algs4.graphs.depth\_first\_paths.DepthFirstPaths* method), 15  
 path\_to() (*itu.algs4.graphs.dijkstra\_sp.DijkstraSP* method), 18  
 path\_to() (*itu.algs4.graphs.dijkstra\_undirected\_sp.DijkstraUndirectedSP* method), 18  
 peek() (*itu.algs4.fundamentals.queue.Queue* method), 2  
 peek() (*itu.algs4.fundamentals.stack.Stack* method), 3  
 Picture (class in *itu.algs4.stdlib.picture*), 52  
 picture() (in module *itu.algs4.stdlib.stddraw*), 54  
 plotBars() (in module *itu.algs4.stdlib.stdstats*), 57  
 plotLines() (in module *itu.algs4.stdlib.stdstats*), 57  
 plotPoints() (in module *itu.algs4.stdlib.stdstats*), 57  
 plus() (*itu.algs4.searching.sparse\_vector.SparseVector* method), 42  
 pop() (*itu.algs4.fundamentals.stack.FixedCapacityStack* method), 3  
 pop() (*itu.algs4.fundamentals.stack.ResizingArrayStack* method), 3

pop () (*itu.algs4.fundamentals.stack.Stack* method), 3  
 post () (*itu.algs4.graphs.depth\_first\_order.DepthFirstOrder* method), 14  
 pre () (*itu.algs4.graphs.depth\_first\_order.DepthFirstOrder* method), 14  
 PrimMST (*class in itu.algs4.graphs.prim\_mst*), 26  
 push () (*itu.algs4.fundamentals.stack.FixedCapacityStack* method), 3  
 push () (*itu.algs4.fundamentals.stack.ResizingArrayStack* method), 3  
 push () (*itu.algs4.fundamentals.stack.Stack* method), 3  
 put () (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33  
 put () (*itu.algs4.searching.bst.BST* method), 34  
 put () (*itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST* method), 36  
 put () (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38  
 put () (*itu.algs4.searching.seperate\_chaining\_hst.SeparateChainingHashST* method), 39  
 put () (*itu.algs4.searching.sequential\_search\_st.SequentialSearchST* method), 40  
 put () (*itu.algs4.searching.sparse\_vector.SparseVector* method), 42  
 put () (*itu.algs4.searching.st.ST* method), 42  
 put () (*itu.algs4.strings.trie\_st.TrieST* method), 62  
 put () (*itu.algs4.strings.tst.TST* method), 63

**Q**

Queue (*class in itu.algs4.fundamentals.queue*), 2  
 QuickFindUF (*class in itu.algs4.fundamentals.uf*), 4  
 QuickUnionUF (*class in itu.algs4.fundamentals.uf*), 5

**R**

R (*itu.algs4.strings.trie\_st.TrieST* attribute), 62  
 R (*itu.algs4.strings.trie\_st.TrieST.Node* attribute), 62  
 RabinKarp (*class in itu.algs4.strings.rabin\_karp*), 62  
 range\_keys () (*itu.algs4.searching.bst.BST* method), 34  
 rank () (*itu.algs4.graphs.topological.Topological* method), 28  
 rank () (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33  
 rank () (*itu.algs4.searching.bst.BST* method), 35  
 rank () (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38  
 reachable () (*itu.algs4.graphs.transitive\_closure.TransitiveClosure* method), 29  
 read\_bool () (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* static method), 50  
 read\_char () (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* static method), 50  
 read\_int () (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* static method), 50  
 read\_string () (*itu.algs4.stdlib.binary\_stdin.BinaryStdIn* static method), 50  
 readAll () (*in module itu.algs4.stdlib.stdio*), 55  
 readAll () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readAllBools () (*in module itu.algs4.stdlib.stdio*), 56  
 readAllBools () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readAllFloats () (*in module itu.algs4.stdlib.stdio*), 56  
 readAllFloats () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readAllInts () (*in module itu.algs4.stdlib.stdio*), 56  
 readAllInts () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readAllLines () (*in module itu.algs4.stdlib.stdio*), 56  
 readAllLines () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readAllStrings () (*in module itu.algs4.stdlib.stdio*), 56  
 readAllStrings () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readBool () (*in module itu.algs4.stdlib.stdio*), 56  
 readBool () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readBool1D () (*in module itu.algs4.stdlib.stdarray*), 53  
 readBool2D () (*in module itu.algs4.stdlib.stdarray*), 53  
 readFloat () (*in module itu.algs4.stdlib.stdio*), 56  
 readFloat () (*itu.algs4.stdlib.instream.InStream* method), 51  
 readFloat1D () (*in module itu.algs4.stdlib.stdarray*), 53  
 readFloat2D () (*in module itu.algs4.stdlib.stdarray*), 53  
 readInt () (*in module itu.algs4.stdlib.stdio*), 56  
 readInt () (*itu.algs4.stdlib.instream.InStream* method), 52  
 readInt1D () (*in module itu.algs4.stdlib.stdarray*), 53  
 readInt2D () (*in module itu.algs4.stdlib.stdarray*), 53  
 readLine () (*in module itu.algs4.stdlib.stdio*), 56  
 readLine () (*itu.algs4.stdlib.instream.InStream* method), 52  
 readString () (*in module itu.algs4.stdlib.stdio*), 56  
 readString () (*itu.algs4.stdlib.instream.InStream* method), 52  
 recognizes () (*itu.algs4.strings.nfa.NFA* method), 61  
 rectangle () (*in module itu.algs4.stdlib.stddraw*), 54  
 RED (*itu.algs4.searching.red\_black\_bst.RedBlackBST* attribute), 37  
 RedBlackBST (*class in itu.algs4.searching.red\_black\_bst*), 37  
 resize () (*itu.algs4.fundamentals.stack.ResizingArrayStack*

- method), 3
- ResizingArrayStack (class in *itu.algs4.fundamentals.stack*), 3
- reverse() (*itu.algs4.graphs.digraph.Digraph* method), 17
- reverse\_post() (*itu.algs4.graphs.depth\_first\_order.DepthFirstOrder* method), 14
- ## S
- save() (in module *itu.algs4.stdlib.stddraw*), 55
- save() (*itu.algs4.stdlib.picture.Picture* method), 53
- scale() (*itu.algs4.searching.sparse\_vector.SparseVector* method), 42
- search() (*itu.algs4.strings.boyer\_moore.BoyerMoore* method), 59
- search() (*itu.algs4.strings.kmp.KMP* method), 60
- search() (*itu.algs4.strings.rabin\_karp.RabinKarp* method), 62
- seed() (in module *itu.algs4.stdlib.stdrandom*), 57
- select() (in module *itu.algs4.sorting.quicksort*), 47
- select() (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33
- select() (*itu.algs4.searching.bst.BST* method), 35
- select() (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38
- SeparateChainingHashST (class in *itu.algs4.searching.seperate\_chaining\_hst*), 38
- SequentialSearchST (class in *itu.algs4.searching.sequential\_search\_st*), 39
- SequentialSearchST.Node (class in *itu.algs4.searching.sequential\_search\_st*), 39
- SET (class in *itu.algs4.searching.set*), 40
- set() (*itu.algs4.stdlib.picture.Picture* method), 53
- setCanvasSize() (in module *itu.algs4.stdlib.stddraw*), 55
- setFontFamily() (in module *itu.algs4.stdlib.stddraw*), 55
- setFontSize() (in module *itu.algs4.stdlib.stddraw*), 55
- setPenColor() (in module *itu.algs4.stdlib.stddraw*), 55
- setPenRadius() (in module *itu.algs4.stdlib.stddraw*), 55
- setXscale() (in module *itu.algs4.stdlib.stddraw*), 55
- setYscale() (in module *itu.algs4.stdlib.stddraw*), 55
- show() (in module *itu.algs4.sorting.quicksort*), 47
- show() (in module *itu.algs4.stdlib.stddraw*), 55
- shuffle() (in module *itu.algs4.stdlib.stdrandom*), 57
- size() (*itu.algs4.fundamentals.bag.Bag* method), 1
- size() (*itu.algs4.fundamentals.queue.Queue* method), 2
- size() (*itu.algs4.fundamentals.stack.FixedCapacityStack* method), 3
- size() (*itu.algs4.fundamentals.stack.ResizingArrayStack* method), 3
- size() (*itu.algs4.fundamentals.stack.Stack* method), 4
- size() (*itu.algs4.graphs.cc.CC* method), 13
- size() (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33
- size() (*itu.algs4.searching.bst.BST* method), 35
- size() (*itu.algs4.searching.linear\_probing\_hst.LinearProbingHashST* method), 36
- size() (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38
- size() (*itu.algs4.searching.seperate\_chaining\_hst.SeparateChainingHashST* method), 39
- size() (*itu.algs4.searching.sequential\_search\_st.SequentialSearchST* method), 40
- size() (*itu.algs4.searching.set.SET* method), 41
- size() (*itu.algs4.searching.st.ST* method), 42
- size() (*itu.algs4.sorting.index\_min\_pq.IndexMinPQ* method), 44
- size() (*itu.algs4.sorting.max\_pq.MaxPQ* method), 45
- size() (*itu.algs4.sorting.min\_pq.MinPQ* method), 46
- size() (*itu.algs4.strings.trie\_st.TrieST* method), 62
- size() (*itu.algs4.strings.tst.TST* method), 63
- size\_between() (*itu.algs4.searching.binary\_search\_st.BinarySearchST* method), 33
- size\_range() (*itu.algs4.searching.bst.BST* method), 35
- size\_range() (*itu.algs4.searching.red\_black\_bst.RedBlackBST* method), 38
- sort() (in module *itu.algs4.sorting.heap*), 43
- sort() (in module *itu.algs4.sorting.insertion\_sort*), 45
- sort() (in module *itu.algs4.sorting.merge*), 45
- sort() (in module *itu.algs4.sorting.merge\_bu*), 46
- sort() (in module *itu.algs4.sorting.quick3way*), 46
- sort() (in module *itu.algs4.sorting.quicksort*), 47
- sort() (in module *itu.algs4.sorting.selection*), 47
- sort() (in module *itu.algs4.sorting.shellsort*), 47
- sort() (in module *itu.algs4.strings.lsd*), 60
- sort() (in module *itu.algs4.strings.msd*), 61
- sort() (in module *itu.algs4.strings.quick3string*), 61
- SparseVector (class in *itu.algs4.searching.sparse\_vector*), 41
- square() (in module *itu.algs4.stdlib.stddraw*), 55
- ST (class in *itu.algs4.searching.st*), 42
- Stack (class in *itu.algs4.fundamentals.stack*), 3
- stddev() (in module *itu.algs4.stdlib.stdstats*), 57
- strongly\_connected() (*itu.algs4.graphs.kosaraju\_sharir\_scc.KosarajuSharirSCC* method), 25
- SymbolDigraph (class in *itu.algs4.graphs.symbol\_digraph*), 27
- SymbolGraph (class in *itu.algs4.graphs.symbol\_graph*), 27



*itu.algs4.graphs.symbol\_graph*), 27

## T

*T* (in module *itu.algs4.fundamentals.binary\_search*), 2

*test()* (in module *itu.algs4.strings.trie\_st*), 62

*test()* (in module *itu.algs4.strings.tst*), 63

*text()* (in module *itu.algs4.stdlib.stddraw*), 55

*ThreeSum* (class in *itu.algs4.fundamentals.three\_sum*), 4

*ThreeSumFast* (class in *itu.algs4.fundamentals.three\_sum\_fast*), 4

*to\_vertex()* (*itu.algs4.graphs.directed\_edge.DirectedEdge* method), 20

*Topological* (class in *itu.algs4.graphs.topological*), 28

*trailing\_zeros()* (in module *itu.algs4.fundamentals.java\_helper*), 2

*TransitiveClosure* (class in *itu.algs4.graphs.transitive\_closure*), 29

*Triest* (class in *itu.algs4.strings.trie\_st*), 62

*Triest.Node* (class in *itu.algs4.strings.trie\_st*), 62

*TST* (class in *itu.algs4.strings.tst*), 63

*TST.Node* (class in *itu.algs4.strings.tst*), 63

*TwoSumFast* (class in *itu.algs4.fundamentals.two\_sum\_fast*), 4

## U

*UF* (class in *itu.algs4.fundamentals.uf*), 6

*uniform()* (in module *itu.algs4.stdlib.stdrandom*), 57

*uniformFloat()* (in module *itu.algs4.stdlib.stdrandom*), 57

*uniformInt()* (in module *itu.algs4.stdlib.stdrandom*), 57

*union()* (*itu.algs4.fundamentals.uf.QuickFindUF* method), 5

*union()* (*itu.algs4.fundamentals.uf.QuickUnionUF* method), 5

*union()* (*itu.algs4.fundamentals.uf.UF* method), 6

*union()* (*itu.algs4.fundamentals.uf.WeightedQuickUnionUF* method), 7

*union()* (*itu.algs4.searching.set.SET* method), 41

*weight()* (*itu.algs4.graphs.edge.Edge* method), 20

*weight()* (*itu.algs4.graphs.kruskal\_mst.KruskalMST* method), 25

*weight()* (*itu.algs4.graphs.lazy\_prim\_mst.LazyPrimMST* method), 26

*weight()* (*itu.algs4.graphs.prim\_mst.PrimMST* method), 26

*WeightedQuickUnionUF* (class in *itu.algs4.fundamentals.uf*), 6

*width()* (*itu.algs4.stdlib.picture.Picture* method), 53

*write()* (in module *itu.algs4.stdlib.stdio*), 56

*write()* (*itu.algs4.stdlib.outstream.OutStream* method), 52

*write1D()* (in module *itu.algs4.stdlib.stdarray*), 53

*write2D()* (in module *itu.algs4.stdlib.stdarray*), 53

*write\_bool()* (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49

*write\_bool()* (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* static method), 50

*write\_byte()* (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49

*write\_byte()* (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* static method), 50

*write\_char()* (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49

*write\_char()* (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* static method), 50

*write\_int()* (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49

*write\_int()* (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* static method), 50

*write\_string()* (*itu.algs4.stdlib.binary\_out.BinaryOut* method), 49

*write\_string()* (*itu.algs4.stdlib.binary\_stdout.BinaryStdOut* method), 50

*writeln()* (in module *itu.algs4.stdlib.stdio*), 56

*writeln()* (*itu.algs4.stdlib.outstream.OutStream* method), 52

*writeln()* (in module *itu.algs4.stdlib.stdio*), 56

*writeln()* (*itu.algs4.stdlib.outstream.OutStream* method), 52

## V

*V()* (*itu.algs4.graphs.digraph.Digraph* method), 16

*V()* (*itu.algs4.graphs.edge\_weighted\_digraph.EdgeWeightedDigraph* method), 21

*V()* (*itu.algs4.graphs.edge\_weighted\_graph.EdgeWeightedGraph* method), 23

*V()* (*itu.algs4.graphs.graph.Graph* method), 23

*var()* (in module *itu.algs4.stdlib.stdstats*), 58

## W

*weight()* (*itu.algs4.graphs.directed\_edge.DirectedEdge* method), 20