Data Structures

Abstract Data Types for Linear Lists

November 12, 2016

Outline

DataLab
Data Science Community

Outline

DataLab
Data Science Community

What is a data structure?

Intuition

A data object is a set or collection of instances!!!

What is a data structure?

Intuition

A data object is a set or collection of instances!!!

Examples

- integer = $\{0, +1, -1, +2, -2, +3, -3, \ldots\}$
- daysOfWeek = $\{S, M, T, W, Th, F, Sa\}$

What is a data structure?

Intuition

A data object is a set or collection of instances!!!

Examples

- integer $= \{0, +1, -1, +2, -2, +3, -3, ...\}$
- daysOfWeek $= \{S,M,T,W,Th,F,Sa\}$

Those instance may be related!!!

Something quite basic

Instances may or may not be related.

Example

myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Thus

Data Structure ≈ Data object + relationships that exist among instances and elements that comprise an instance.

Those instance may be related!!!

Something quite basic

Instances may or may not be related.

Example

myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Thus

Data Structure $\approx$ Data object + relationships that exist among instances and elements that comprise an instance.

Those instance may be related!!!

Something quite basic

Instances may or may not be related.

Example

myDataObject $= \{$apple, chair, 2, 5.2, red, green, Jack$\}$

Thus

Data Structure $\approx$ Data object $+$ relationships that exist among instances and elements that comprise an instance.

Examples

Among instances of integers

- $369 < 370$
- $280 + 4 = 284$

Thus

The relationships are usually specified by specifying operations on one or more instances.

Examples

add, subtract, predecessor, multiply

Examples

Among instances of integers

- $369 < 370$
- $280 + 4 = 284$

Thus

The relationships are usually specified by specifying operations on one or more instances.

Examples

add, subtract, predecessor, multiply

Examples

Among instances of integers

- $369 < 370$
- $280 + 4 = 284$

Thus

The relationships are usually specified by specifying operations on one or more instances.

Examples

add, subtract, predecessor, multiply

Outline

DataLab
Data Science Community

Abstract Data Type

Data Type

A data type such as **int** or **double** is a group of values and operations on those values that is defined within a specific programming language.

Abstract Data Type

An Abstract Data Type, or ADT, is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

Thus

A data structure is an implementation of an ADT within a programming language.

Abstract Data Type

Data Type

A data type such as **int** or **double** is a group of values and operations on those values that is defined within a specific programming language.

Abstract Data Type

An Abstract Data Type, or ADT, is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

Thus

A data structure is an implementation of an ADT within a programming language.

Abstract Data Type

Data Type

A data type such as **int** or **double** is a group of values and operations on those values that is defined within a specific programming language.

Abstract Data Type

An Abstract Data Type, or ADT, is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

Thus

A data structure is an implementation of an ADT within a programming language.

DataLab
Data Science Community

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

- An abstract object may be passed as a parameter to a procedure.
- An abstract object may be assigned to a variable.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

DataLab
Data Science Community

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

For More

We have the following

"PROGRAMMING WITH ABSTRACT DATA TYPES" by Barbara Liskov and Stephen Zilles

Outline

DataLab
Data Science Community

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

- $n \geq 0$ is finite
- List size is n

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Properties

- Where e_i denotes a list element
- $n \geq 0$ is finite
- List size is n

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Properties

- Where e_i denotes a list element
- $n \geq 0$ is finite
- List size is n

DataLab
Data Science Community

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Properties

- Where e_i denotes a list element
- $n \geq 0$ is finite
- List size is n

DataLab
Data Science Community

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

Relationships

- e_0 is the zero'th (or front) element
- e_{n-1} is the last element
- e_i immediately precedes e_{i+1}

DataLab
Data Science Community

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

Relationships

- e_0 is the zero'th (or front) element
- e_{n-1} is the last element
- e_i immediately precedes e_{i+1}

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

Relationships

- e_0 is the zero'th (or front) element
- e_{n-1} is the last element
- e_i immediately precedes e_{i+1}

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

DataLab
Data Science Community

Outline

Which operations must be supported?

We need the size of a linear list

Hey, we need to know how many elements the linear list has.

Determine list size for

$L = (a, b, c, d, e)$

So, we need to have a operation that returns the size

- size(L)=5

Which operations must be supported?

We need the size of a linear list

Hey, we need to know how many elements the linear list has.

Determine list size for

$L = (a, b, c, d, e)$

So, we need to have a operation that returns the size

- size(L)=5

DataLab
Data Science Community

Which operations must be supported?

We need the size of a linear list

Hey, we need to know how many elements the linear list has.

Determine list size for

$L = (a, b, c, d, e)$

So, we need to have a operation that returns the size

- size(L)=5

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

DataLab
Data Science Community

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) $=$ a
- get(2) $=$ c
- get(4) $=$ e
- get(-1) $=$ error
- get(9) $=$ error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) $=$ a
- get(2) $=$ c
- get(4) $=$ e
- get(-1) $=$ error
- get(9) $=$ error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) $=$ a
- get(2) $=$ c
- get(4) $=$ e
- get(-1) $=$ error
- get(9) $=$ error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) $=$ a
- get(2) $=$ c
- get(4) $=$ e
- get(-1) $=$ error
- get(9) $=$ error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

- indexOf(a) = 0
- indexOf(z) = -1

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) $=$ 2
- indexOf(a) $=$ 0
- indexOf(z) $=$ -1

DataLab
Data Science Community

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) $=$ 2
- indexOf(a) $=$ 0
- indexOf(z) $=$ -1

DataLab
Data Science Community

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) $= 2$
- indexOf(a) $= 0$
- indexOf(z) $= -1$

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) $=$ 2
- indexOf(a) $=$ 0
- indexOf(z) $=$ -1

Linear List Operations: add(theIndex, theElement)

Definition

Add an element so that the new element has a specified index.

If we have

$L = (a, b, c, d, e, f, g)$

- Thus, index of a, b, c, d, e, f, g increases by 1.

Linear List Operations: add(theIndex, theElement)

Definition

Add an element so that the new element has a specified index.

If we have

$L = (a, b, c, d, e, f, g)$

Thus

- **add(0,h)** $\Longrightarrow L = (h, a, b, c, d, e, f, g)$
  - Thus index of a, b, c, d, e, f, g increase by 1

Linear List Operations: add(theIndex, theElement)

Definition

Add an element so that the new element has a specified index.

If we have

$L = (a, b, c, d, e, f, g)$

Thus

- **add(0,h)** $\Longrightarrow L = (h, a, b, c, d, e, f, g)$
  - Thus, index of a, b, c, d, e, f, g increases by 1.

DataLab
Data Science Community

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

What about out of the bound

- **add(10,h)** $\Longrightarrow$ error
- **add(-6,h)** $\Longrightarrow$ error

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

What about out of the bound

- **add(10,h)** $\Longrightarrow$ error
- **add(-6,h)** $\Longrightarrow$ error

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

What about out of the bound

- **add(10,h)** $\Longrightarrow$ error
- **add(-6,h)** $\Longrightarrow$ error

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

For example

$L = (a, b, c, d, e, f, g)$

- Index of e, f, g decreases by 1

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

For example

$L = (a, b, c, d, e, f, g)$

Example

- **Remove(2)** returns c and L becomes (a, b, d, e, f, g)
  - Index of d, e, f, g decreases by 1

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

For example

$L = (a, b, c, d, e, f, g)$

Example

- **Remove(2)** returns c and L becomes (a, b, d, e, f, g)
  - Index of d, e, f, g decreases by 1

What about the Error

For

$L = (a, b, c, d, e, f, g)$

Thus

- remove(-1) $\Longrightarrow$ error
- remove(20) $\Longrightarrow$ error

What about the Error

For

$L = (a, b, c, d, e, f, g)$

Thus

- remove(-1) $\Longrightarrow$ error
- remove(20) $\Longrightarrow$ error

The Final Abstract Data Type for Linear List

Linear List

AbstractDataType LinearList
{
instances
ordered finite collections of zero or more elements
operations
isEmpty(): return true iff the list is empty, false otherwise
size(): return the list size (i.e., number of elements in the list)
get(index): return the element with "index" index
indexOf(x): return the index of the first occurrence of x in the list, return -1 if x is not in the list
remove(index): remove and return the indexth element, elements with higher index have their index reduced by 1
add(theIndex, x): insert x as the index of th element, elements with theIndex $\geq$ index have their index increased by 1
output(): output the list elements from left to right
}

Outline

DataLab
Data Science Community

CPython

We have

1. typedef struct {
2. PyObject_VAR_HEAD;
3. PyObject **ob_item;
4. Py_ssize_t allocated;
5. } PyListObject;

Outline

DataLab
Data Science Community

How do we implement the ADT List?

In our first representation will use an array

Use a one-dimensional array **element**[]:

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

The previous array

A representation of $L = (a, b, c, d, e)$ using position i in **element**[i].

How do we implement the ADT List?

In our first representation will use an array

Use a one-dimensional array **element**[]:

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

The previous array

A representation of $L = (a, b, c, d, e)$ using position i in **element**$[i]$.

Where to map in the array

Right To Left Mapping

-	-	-	-	e	d	c	b	a

Mapping That Skips Every Other Position

a	-	b	-	c	-	d	-	e	-	-

Wrap Around Mapping

d	e	-	-	-	-	-	-	a	b	c

Where to map in the array

Right To Left Mapping

-	-	-	-	e	d	c	b	a

Mapping That Skips Every Other Position

a	-	b	-	c	-	d	-	e	-	-

Wrap Around Mapping

d	e	-	-	-	-	-	-	a	b	c

DataLab
Data Science Community

Where to map in the array

Right To Left Mapping

-	-	-	-	e	d	c	b	a

Mapping That Skips Every Other Position

a	-	b	-	c	-	d	-	e	-	-

Wrap Around Mapping

d	e	-	-	-	-	-	-	a	b	c

DataLab
Data Science Community

Outline

Representation Used

Something Notable

a	b	c	d	e	-	-	-	-
0	1	2	3	4	5	6	7	8

Size=5

Outline

Now, we have a common problem

Because

We do not know how many elements will be stored at the list.

We use

An initial length and dynamically increase the size as needed.

Now, we have a common problem

Because

We do not know how many elements will be stored at the list.

We use

An initial length and dynamically increase the size as needed.

Example

Example

Length of array element[] is 6:

a	b	c	d	e	f

First create a new and larger array

NewArray = (Item[]) new Object[12]

-	-	-	-	-	-	-	-	-	-	-	-

Now copy the new elements into the new array!!!

System.arraycopy(element, 0, newArray, 0, element.length);

a	b	c	d	e	f	-	-	-	-	-	-

Example

Example

Length of array element[] is 6:

a	b	c	d	e	f

First create a new and larger array

NewArray = (Item[]) new Object[12]

-	-	-	-	-	-	-	-	-	-	-	-

Now copy the new elements into the new array!!!

System.arraycopy(element, 0, newArray, 0, element.length);

a	b	c	d	e	f	-	-	-	-	-	-

Example

Example

Length of array element[] is 6:

a	b	c	d	e	f

First create a new and larger array

NewArray = (Item[]) new Object[12]

-	-	-	-	-	-	-	-	-	-	-	-

Now copy the new elements into the new array!!!

System.arraycopy(element, 0, newArray, 0, element.length);

a	b	c	d	e	f	-	-	-	-	-	-

Example

Finally, rename new array

- element = NewArray;

element[0] ⟶	a	b	c	d	e	f	-	-	-	-	-	-

Outline

DataLab
Data Science Community

First Attempt

What if you use the following policy?

At least 1 more than current array length.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

First Attempt

What if you use the following policy?

At least 1 more than current array length.

Thus

What would be the cost of all operations?

- Generate a new array.
- Copy all the items to it.
- insert the new element at the end.

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

Ok

Not a good idea!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

Ok

Not a good idea!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O\left(n^2\right) \tag{1}$$

Ok

Not a good idea!!!

What if we do n insertions?

We finish with something like this

1. First Insertion: Creation of the List $\Rightarrow$ Cost $= 1$.
2. Second Insertion Cost $= 2$.
3. Third Insertion Cost $= 3$
4. etc!!!

Thus, we have

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O\left(n^2\right) \tag{1}$$

Ok

Not a good idea!!!

Outline

DataLab
Data Science Community

A better strategy

Dynamic Array

To avoid incurring the cost of re-sizing many times, dynamic arrays re-size by an amount a.

In our example we double the size, $a = 2$

A better strategy

Dynamic Array

To avoid incurring the cost of re-sizing many times, dynamic arrays re-size by an amount a.

In our example we double the size, $a = 2$

```
Item NewArray[];
if (this.size == element.lenght){
    // Resize the capacity
    NewArray = (Item[]) new Object[2*this.size]
    for(int i=0; i < size ; i++){
        NewArray[i]=element[i];
    }
    element = NewArray;
}
```

Space Complexity

Every time an insertion triggers a doubling of the array

Thus, space wasted in the new array:

$$\text{Space Wasted } = \text{Old Lenght } - 1 \tag{2}$$

a	b	c	d	e	f	-	-	-	-

Wasted

Remember: We double the array and insert!!!

Thus, the average space wasted is

$$\Theta(n) \tag{3}$$

Space Complexity

Every time an insertion triggers a doubling of the array

Thus, space wasted in the new array:

$$\text{Space Wasted } = \text{Old Lenght } - 1 \tag{2}$$

a	b	c	d	e	f	-	-	-	-

Wasted

Remember: We double the array and insert!!!

Thus, the average space wasted is

$$\Theta\left(n\right) \tag{3}$$

For example, we have the following

A trade-off between time and space

- You have and average time for insertion is $\frac{2}{2-1}$.
- In addition, an upper bound for the wasted cells in the array is $(2-1)\,n-1=n-1$.

Actually a more general term of expansion, a

Thus, we have:

- Average time for insertion is $\frac{a}{a-1}$.
- An upper bound for the wasted cells in the array is $(a-1)\,n-1=an-n-1$.

Different languages use different values

- Java, $a=\frac{3}{2}$.
- Python, $a=\frac{9}{8}$

For example, we have the following

A trade-off between time and space

- You have and average time for insertion is $\frac{2}{2-1}$.
- In addition, an upper bound for the wasted cells in the array is $(2-1)\,n-1=n-1$.

Actually a more general term of expansion, a

Thus, we have:

- Average time for insertion is $\frac{a}{a-1}$.
- An upper bound for the wasted cells in the array is $(a-1)\,n-1=an-n-1$.

Different languages use different values

- Java, $a=\frac{3}{2}$.
- Python, $a=\frac{9}{8}$

For example, we have the following

A trade-off between time and space

- You have and average time for insertion is $\frac{2}{2-1}$.
- In addition, an upper bound for the wasted cells in the array is $(2-1)\,n-1=n-1$.

Actually a more general term of expansion, a

Thus, we have:

- Average time for insertion is $\frac{a}{a-1}$.
- An upper bound for the wasted cells in the array is $(a-1)\,n-1=an-n-1$.

Different languages use different values

- Java, $a=\frac{3}{2}$.
- Python, $a=\frac{9}{8}$

Outline

DataLab
Data Science Community

We have the following final analysis

Amortized Analysis Vs. Classic

	Array Classic Analysis	Dynamic Array Amortized Analysis
Indexing	$O(1)$	$O(1)$
Search	$O(n)$	$O(n)$
Add/Remove	$O(n)$	$O(n)$
Space Complexity	$O(n)$	$O(n)$

DataLab
Data Science Community