Data Structures & Algorithms

Week 4

Arrays and Linked Structures

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 3

- Algorithm complexity is a measure which evaluates the order of the count of operations, performed by a given order algorithm as a function of the size of the input data.
- The asymptotic complexity measure does not give the exact number of operations of an algorithm, but it shows how that number grows with the size of the input.
- The Big-O notation, **O(g(n)),** is used to give an upper bound (worst-case) on a positive runtime function f(n) where n is the input size.
- The big omega gives the tighter lower bound of the given algorithm and we represent it as $f(n) = \Omega(g(n))$.
- Sequential search will work very nicely for modest-sized lists, but it will prove to very inefficient for large lists. It is O(n) complexity.
- Binary search is said to be **O(log n)**, which is **much faster** than **O(n)**.
- Selection sort is **$O(n^2)$** algorithm
- Merge sort is **O(n log n); t**his is far better than **$O(n^2)$**

Content

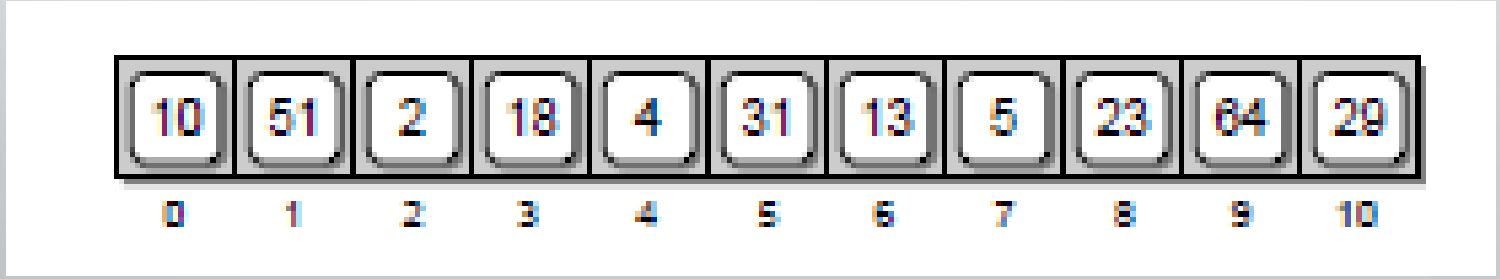
- Arrays
- Linked Structures

Part 1

Arrays

1.1 Introduction

- Arrays were introduced in lesson 2 of this course; some basic operations were discussed (using both Java and Python), as well as the advantages and disadvantages. It was also mentioned that arrays would be discussed in detail in a later lesson…remember? Well this is it.
- Let us have a quick paraphrased recap to introduce the content of this section of the lesson first.
- At the hardware level, most computer architectures provide a mechanism for creating and using one-dimensional arrays. A one-dimensional array, as illustrated below, is composed of multiple sequential elements stored in contiguous bytes of memory and allows for random access to the individual elements. (Necaise, 2011).
- The array consists of 11 elements, indexed from 0 to 10.



1.2 Array ADT

- The array structure is commonly found in most programming languages as a primitive type, but Python only provides the list structure for creating mutable sequences (lists will be discussed in lesson 9).
- We can define the Array ADT to represent a one-dimensional array for use in Python that works similarly to arrays found in other languages. (Necaise, 2011)
- Loosely a one dimensional array is a collection of items of similar type kept or stored in a linear arrangement. Necaise (2011) defines a one dimensional array as "a collection of contiguous elements in which individual elements are identified by a unique integer subscript starting with zero. Once an array is created, its size cannot be changed."
- Let us examine a list of operations (commands) available to the array ADT.

1.2.1 Array ADT operations

- Operations available for the array ADT are (Necaise, 2011):
- Array(size): Creates a one-dimensional array consisting of size elements with each element initially set to None. size must be greater than zero.
- length (): Returns the length or number of elements in the array.
- getitem (index): Returns the value stored in the array at element position index. The index argument must be within the valid range. Accessed using the subscript operator.
- setitem (index, value): Modifies the contents of the array element at position index to contain value. The index must be within the valid range. Accessed using the subscript operator.
- clearing(value): Clears the array by setting every element to value.
- iterator (): Creates and returns an iterator that can be used to traverse the elements of the array.

1.2.1 Array ADT operations

- Let us write a simple Python program that will create an array and read its contents.

```
# Fill a 1-Dimensional array with some random values, then print them, one per line.
# import the classes required to complete the operation
from array import Array
import random
# The constructor is called to create the array.
exampleList = Array( 20 )
# Fill the array with random floating-point values.
for i in range( len( exampleList ) ) :
    exampleList[ i ] = random.random()
# Print the values, one per line.
for n in exampleList :
    print( n)
```

1.2.1 Array ADT operations

- Let us examine another example provided by Necaise (2011):"
- Suppose you need to read the contents of a text file and count the number of letters occurring in the file with the results printed to the terminal. We know that characters are represented by the ASCII code, which consists of integer values. The letters of the alphabet, both upper- and lowercase, are part of what's known as the printable range of the ASCII code. This includes the ASCII values in the range [32 : : : 126] along with some of the codes with smaller values. The latter are known control characters and can include the tab, newline, and form-feed codes. Since all of the letters will have ASCII values less than 127, we can create an array of this size and let each element represent a counter for the corresponding ASCII value. After processing the file, we can traverse over the elements used as counters for the letters of the alphabet and ignore the others."
- The code that implements this is shown in slide 10; courtesy Necaise (2011):

```
#Count the number of occurrences of each
letter in a text file.

from array import Array

# Create an array for the counters and
initialize each element to 0.

theCounters = Array( 127 )

theCounters.clear( 0 )

# Open the text file for reading and extract
each line from the file

# and iterate over each character in the line.

theFile = open( 'atextfile.txt', 'r' )

for line in theFile :

    for letter in line :

        code = ord( letter )

        theCounters[code] += 1

# Close the file

theFile.close()

# Print the results. The uppercase letters have
ASCII values in the

# range 65..90 and the lowercase letters are in
the range 97..122.

for i in range( 26 ) :

    print( "%c - %4d %c - %4d" % \

        (chr(65+i), theCounters[65+i],
        chr(97+i), theCounters[97+i]) )
```

1.3 Implementing the Array

- An array (rather than the list) is the primary implementing structure in the collections of Python and many other programming languages (including C).
- Arrays, however, are more restrictive than lists (also mentioned as a disadvantage in lesson 2) in that "A programmer can access and replace an array's items at given positions, examine an array's length, and obtain its string representation—but that's all. The programmer cannot add or remove positions or make the length of the array larger or smaller. Typically, the length or capacity of an array is fixed when it is created." (Lambert, 2019).
- Python's array module does include an array class, which behaves more like a list but is limited to storing numbers.
- We therefore define a class called Array which will enable us to perform more operations on our array object. The class defines methods that allow clients to use the subscript operator [], the len function, the str function, and the for loop with array objects. The Array methods needed for these operations are listed in Table 1. The variable a in the left column refers to an Array object.

Table 1

Array class

User's Array Operation	Method in the Array Class
`a = Array(10)`	`__init__(capacity, fillValue = None)`
`len(a)`	`__len__()`
`str(a)`	`__str__()`
`for item in a:`	`__iter__()`
`a[index]`	`__getitem__(index)`
`a[index] = newitem`	`__setitem__(index, newItem)`

(From Lambert, 2019)

1.3 Implementing the Array

- When Python encounters an operation in the left column of Table 4-1, it automatically calls the corresponding method in the right column with the Array object. For example, Python automatically calls the Array object's __len_ method to find the length of the Array object. Note that the programmer must specify the capacity or the physical size of an array when it is created. The default fill value, None, can be overridden to provide another fill value if desired.
- The code that creates our Array class in provided in arrays.py (by Lambert, 2019)

```
"""
File: arrays.py
An Array is like a list, but the client can use
only [], len, iter, and str.
To instantiate, use
<variable> = Array(<capacity>, <optional fill value>)
The fill value is None by default.
"""
class Array(object):
"""Represents an array."""
def __init__(self, capacity, fillValue = None):
"""Capacity is the static size of the array.
        fillValue is placed at each position."""
        self.items = list()
        for count in range(capacity):
                self.items.append(fillValue)
def __len__(self):
"""-> The capacity of the array."""
        return len(self.items)
def __str__(self):
"""-> The string representation of the array."""
        return str(self.items)
def __iter__(self):
"""Supports traversal with a for loop."""
        return iter(self.items)
def __getitem__(self, index):
"""Subscript operator for access at index."""
        return self.items[index]
def __setitem__(self, index, newItem):
"""Subscript operator for replacement at index."""
        self.items[index] = newItem
```

1.3 Implementing the Array

- Let us write a small snippet (adapted from Lambert, 2019) that shows how our Array class is implemented (with output):

```
from arrays import Array
>>> a = Array(3)    # Create an array with 3 positions
>>> len(a)      # Show the number of positions
3
>>> print(a)   # Show the contents
[None, None, None]
>>> for i in range(len(a)):   # Replace contents with 1..3
            a[i] = i + 1
>>> a[0] # Access the first item
1
>>> for item in a: # traverse the array to print all
print(item)
1 2 3
```

1.3 Implementing the Array

- While Python does not provide the array structure as part of the language itself, it now includes the ctypes module as part of the Python Standard Library. This module provides access to the diverse set of data types available in the C language and the complete functionality provided by a wide range of C libraries.
- The availability of these modules is mostly due to the fact that Python is built using the C language. (C is also the language used to write most operating systems).
- The ctypes module provides the capability to create hardware-supported arrays just like the ones used to implement Python's string, list, tuple, and dictionary collection types. But the ctypes module is not meant for everyday use in Python programs as it was designed for use by module developers to aide in creating more portable Python modules by bridging the gap between Python and the C language. Much of the functionality provided by the ctypes module requires some knowledge of the C language. Thus, the technique provided by the module for creating an array should not typically be used directly within a Python program. But we can use it within our Array class to provide the functionality defined by the Array ADT since the details will be hidden within the class. (Necaise, 2011).

1.3 Implementing the Array

- Let us use ctypes to create an array to store 5 elements like we did using our Array class. Note that the ctypes module provides a technique for creating arrays that can store references to Python objects. (Necaise, 2011)
- **import** ctypes
- ArrayType = ctypes.py_object * 5
- slots = ArrayType()
- Once the array has been created we can access the elements of the array using Python's integer subscript notation.
- Next the array needs to be initialized for it to be used; this is done by assigning values to each element of the array (we can assign the value NONE just as well!):

```
For x in range(5):
    Slots (x) = None
```

1.3 Implementing the Array

- We can now store references to any type of Python object into our array *slots*. For example let us store 2 elements at different positions in the array.
- Slots[0] = 32 # index 0 is the first position in the array
- Slots[2] = 20 # index 2 is the third position in the array
- Slots[4] = 10 # index 4 is the fifth position in the array
- To remove an item from the array set the value to 'None'; for example to remove the value 20 we would write: slots[2] = None
- Also remember that the size of the array is fixed, it can't change; therefore removing an element in the array only does that; it does not change the size of the array. (if it is an array of 10 elements it will remain an array of 10 elements even if empty).
- Necaise (2011) provides the implementation of the Array ADT using a hardware-supported array created with the use of the ctypes module

```
# Implements the Array ADT using array capabilities of the ctypes module.
import ctypes

class Array :
# Creates an array with size elements.
def __init__( self, size ):
    assert size > 0, "Array size must be > 0"
    self._size = size
# Create the array structure using the ctypes module.
    PyArrayType = ctypes.py_object * size
    self._elements = PyArrayType()
# Initialize each element.
    self.clear( None )

# Returns the size of the array.
def __len__( self ):
    return self._size

# Gets the contents of the index element.
def __getitem__( self, index ):
    assert index >= 0 and index < len(self), "Array subscript out of range"
```

```
22    return self._elements[ index ]

24 # Puts the value in the array element at index position.

25 def __setitem__( self, index, value ):

26    assert index >= 0 and index < len(self), "Array subscript out of range"

27    self._elements[ index ] = value

29 # Clears the array by setting each element to the given value.

30 def clear( self, value ):

31    for i in range( len(self) ) :

32          self._elements[i] = value
```

```
34 # Returns the array's iterator for traversing the elements.

35 def __iter__( self ):

36    return _ArrayIterator( self._elements )

38 # An iterator for the Array ADT.

39 class _ArrayIterator :

40    def __init__( self, theArray ):

41          self._arrayRef = theArray

42          self._curNdx = 0

43

44    def __iter__( self ):

45          return self
```

```
47 def __next__( self ):
48   if self._curNdx < len( self._arrayRef ) :
49         entry = self._arrayRef[ self._curNdx ]
50         self._curNdx += 1
51         return entry
52   else :
53         raise StopIteration
```

- The clear() method is used to set each element of the array to a given value, which it does by iterating over the elements using an index variable. The len method, which returns the number of elements in the array, simply returns the value of size that was saved in the constructor. The iter method creates and returns an instance of the ArrayIterator private iterator class, which is provided in lines 39-53. as you might correctly guess the ArrayIterator class allows us to iterate over the array.

1.3 Implementing the Array

- Further, The __setitem__ operator method is used to set or change the contents of a specific element of the array. It takes two arguments: the array index of the element being modified and the new value that will be stored in that element. Before the element is modified, the precondition must be tested to verify the subscript is within the valid range. Python automatically calls the __setitem__ method when the subscript notation is used to assign a value to a specic element, x[i] = y. The index, i, specified in the subscript is passed as the first argument and the value to be assigned is passed as the second argument, __setitem__(i,y). (Necaise, 2011)
- Python also has the array module which can be used to create and manipulate arrays. In order to use this simply import it using the *import* command in Python.
- There are three ways of doing this (Lemonaki, 2022):
- Use import array at the top of the file; this will include the module array and you can then use array.array() to create an array

1.3 Implementing the Array

- Instead of array.array() all the time, use import array as arr; then create an array by simply typing arr.array(). Arr acts an alias in this case.
- Use 'from array import *'; this would import all functionalities available; you can then create an array using the array() constructor alone.
- Let us implement this in a few snippets of code:

```
import array as arr

numbers = arr.array('i',[50,10,3]) # array numbers contains integers 'i'

print(numbers)

#output the array('i', [10, 20, 30])
```

1.3 Implementing the Array

```
from array import *
#an array of floating point values
numbers = array('d',[10.0,20.0,30.0])
 print(numbers)
print (numbers[0]) # prints out the first element of the array
print (numbers[-1]) # prints out the last element of the array
print(numbers[-2]) #prints out the second last element of the array
 #search for the index of the value 10
print(numbers.index(10.0))
 #output
```

- In the above example we use the import * to import all functionality, therefore, we don't need to use arr as in the previous example.

1.3 Implementing the Array

Suppose you wish to slice the array in Python? Use the slicing operator : to do so. Let us demonstrate this using an example from (Lemonaki, 2022):

```
#original array
numbers = arr.array('i',[10,20,30])
#get the values 10 and 20 only
print(numbers[:2]) #first to second position
```

- When using the slicing operator and you only include one value, the counting starts from 0 by default. It gets the first item, and goes up to but not including the index number you specify.

1.3 Implementing the Array

- When passing two arguments in the program, you specify a range of numbers. In this case, the counting starts at the position of the first number in the range, and up to but not including the second one (Lemonaki, 2022):
- numbers = arr.array('i',[10,20,30])
- #get the values 20 and 30 only
- print(numbers[1:3]) #second to third position
- #output
- #array('i', [20, 30])

1.4 Memory

- In older programming languages arrays were static data structures. The length or capacity of the array was determined at compile time, so the programmer needed to specify this size with a constant. Because the programmer couldn't change the length of an array at run time, he needed to predict how much array memory would be needed by all applications of the program. (Lambart, 2019). We can already imagine how much of a headache this was for them.
- This problem can be handled using dynamic arrays as opposed to the traditional static arrays. A dynamic array is just like a normal array. The difference between the two is that the size of a dynamic array can be dynamically modified at runtime. We don't need to specify the size of the array beforehand. In a dynamic array, once the contagious memory is filled, a bigger chunk of memory is allocated. The contents of the original array are copied to this new space, and the available slots are filled continuously. (Sao, 2021). Python implements dynamic arrays using lists.
- Fortunately, Java or C++ programmers can specify the length of a dynamic array during instantiation. The Python Array class behaves in a similar manner.

1.4 Memory

- A programmer can change the length of an array to suit the data needs of an application through ways:
- Create the array with a big default size (if initial needs were 20 elements, create an array with say 50 elements).
- When the array is full (or getting full), create a new array, and transfer the elements of the old to the new one.
- When the array is not utilizing memory efficiently, reduce the size by transferring the elements to a smaller one (that you will have created).
- Let us examine some code that implements the above operations.

1.4 Memory

- To increase the size of the array we perform 3 steps: create a larger array, transfer elements from the old to the new array, point the old array to the new array. In Python this is implemented as follows:
- if oldSize == len(a):
- new = Array(len(a) + 1) # Create a new array
- for i in range(oldSize): # Copy data from the old
- new [i] = a[i] # array to the new array
- a = new # Reset the old array variable to the new array
- # the new array size can also be doubled by replacing +1, with *4 which quadruples its size

1.4 Memory

- To reduce the size of the array we repeat the process as before; only this time we create a smaller array instead of a bigger one:
- if oldSize <= len(a) //4 and len(a) >= DEFAULT_CAPACITY * 2:
- new = Array(len(a) //2 # Create a new array
- for i in range(oldSize): # Copy data from the old
- new [i] = a[i] # array to the new array
- a = new # Reset the old array variable to the new array
- There are many other operations that can be performed on a dynamic array in a similar manner such as inserting or removing an element from the array. The details of the code to do this are left for the learner to try on their own using the two provided examples here.
- Table 2 shows the complexity of different array operations.

Table 2

Complexity of array operations

Operation	Running Time
Access at ith position	O(1), best and worst cases
Replacement at ith position	O(1), best and worst cases
Insert at logical end	O(1), average case
Insert at ith position	O(n), average case
Remove from ith position	O(n), average case
Increase capacity	O(n), best and worst cases
Decrease capacity	O(n), best and worst cases
Remove from logical end	O(1), average case
Insert at ith position	O(n), average case
Remove from ith position	O(n), average case
Increase capacity	O(n), best and worst cases
Decrease Capacity	O(n), best and worst cases

(From Lambert, 2019)

1.5 Two dimensional arrays

- A two dimensional array can be viewed as an array of arrays so to speak. Elements are arranged similarly to a matrix in a row and column format.
- The elements of a two dimensional array are referred to in the RC (row column) format. For example (0,0) would refer to the first element in the first row (remember positions start at 0). Let us demonstrate this using Figure 1; the element at (2,3) is 120. Also the element at (3, 3) is 160; likewise 150 is at position (3,2).

Figure 1

Two dimension array

Row/Columns	0	1	2	3
0	10 (0,0)	20	30	40
1	50	60(1,1)	70	80
2	90	100	110 (2,2)	120
3	130	140	150	160(3,3)

1.5 Two dimensional arrays

- Operations that are performed on a two dimensional array are as follows:
- Array2D(nrows, ncols): Creates a two-dimensional array organized into rows and columns. The nrows and ncols arguments indicate the size of the table. The individual elements of the table are initialized to None.
- numRows(): Returns the number of rows in the 2-D array.
- numCols(): Returns the number of columns in the 2-D array.
- clear(value): Clears the array by setting each element to the given value.
- getitem(i1, i2): Returns the value stored in the 2-D array element at the position indicated by the 2-tuple (i1; i2), both of which must be within the valid range. Accessed using the subscript operator: y = x[1,2].
- setitem(i1, i2, value): Modifies the contents of the 2-D array element indicated by the 2-tuple (i1; i2) with the new value. Both indices must be within the valid range. Accessed using the subscript operator: x[0,3] = y.
- Let us perform a few operations on this array to understand how to use it in Python

1.5 Two dimensional arrays

```
from array import *
#code adapted from (Python - 2D Array - Tutorialspoint, n.d.)
newArray = [[9, 10, 11], [12, 13,14], [15, 16,17]]
print(newArray[0]) # print the first row
print(newArray[1][2]) #print element in second row, third position.
This will output [9,10,11] and 14 respectively
#to print out the entire array we use the for loop:
for r in newArray:
    for c in r:
        Print(c, end=«  « )
    Print()
```

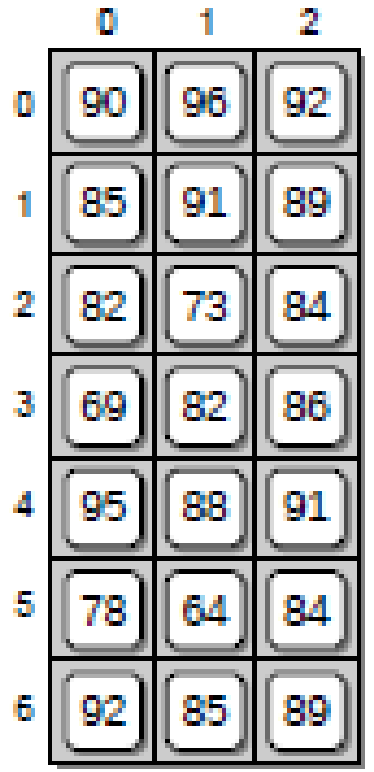
1.5 Two dimensional arrays

- Let us consider a practical day to day situation; the case of student grades.
- Supposing we wish to store the grades in a 2D array: the rows would represent individual students while the columns would represent the grades in different subjects.
- We wish to extract the grades from a text file and store them in the 2D array. This would make it easier to work with the different grades for each student. Figure 2 shows an example of how the grades would appear in text file and then when moved to the 2D array. Let us examine the Python code that would effect this change/movement.

Fig 2

Moving data from text file to array

```
7
3
90      96      92
85      91      89
82      73      84
69      82      86
95      88      91
78      64      84
92      85      89
```



(From Necaise, 2011)

```
from array import Array2D
# Open the text file for reading.
gradeFile = open( filename, "r" )
# Extract the first two values which indicate
the size of the array.
numExams = int( gradeFile.readline() )
numStudents = int( gradeFile.readline() )
# Create the 2-D array to store the grades.
examGrades = Array2D( numStudents,
numExams )
# Extract the grades from the remaining lines.
i = 0
for student in gradeFile :
    grades = student.split()
    for j in range( numExams ):
        examGrades[i,j] = int( grades[j] )
    i += 1
# Close the text file.
gradeFile.close()
```

1.5 Two dimensional arrays

- Suppose we want to compute and display each student's exam grade, which we can do with the following code (Necaise, 2011):

```
# Compute each student's average exam grade.
for i in range( numStudents ) :
# Tally the exam grades for the ith student.
total = 0
    for j in range( numExams ) :
    total += examGrades[i,j]
# Compute average for the ith student.
examAvg = total / numExams
print( "%2d: %6.2f" % (i+1, examAvg) )
```

Part 2

Linked structures

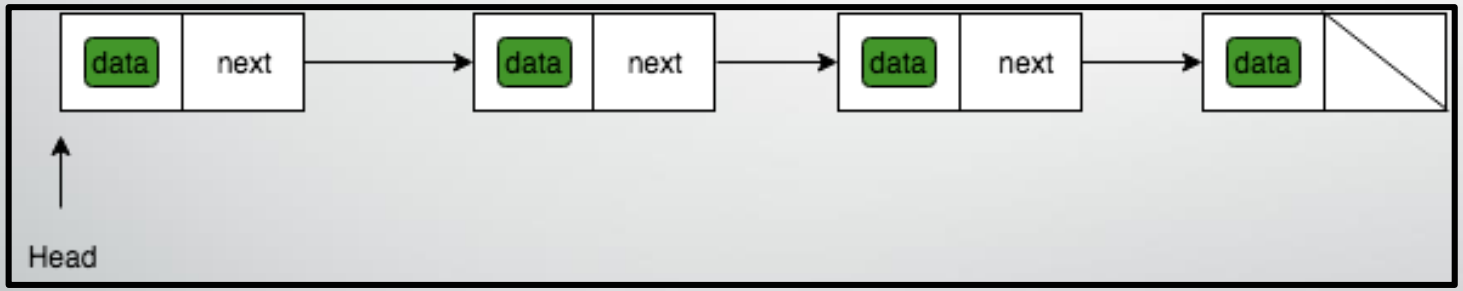
2.1 Introduction

- After arrays, linked structures are probably the most commonly used data structures in programs. Like an array, a linked structure is a concrete data type that implements many types of collections, including lists. A thorough examination of the use of linked structures in collections such as lists and binary trees appears (in later lessons in this course). (Lambert, 2019)
- In this section we wish to examine some key characteristics of linked structures regardless of the collection in which it is used.

2.1 Singly linked structures

- For purposes of explaining a singly linked structure we use a single linked list.
- A linked list is a linear collection of data items where each item points to a next item. Unlike an array, the items in the linked list do not need to be in a continuous memory location. They can be anywhere in the memory as long as they are connected by a link. (*Linked Lists*, n.d.) Figure 3 shows a singly linked list.

Figure 3
Linked list



(From Linked lists, n.d.)

2.1 Singly linked structures

- Linked lists (n.d.) describes the singly linked structure as follows:
- Description of parts:

  Each item in a linked list is called a Node. A node has two parts: the first part is called data and it holds the actual value of the item and the second part is called next and it holds the address of the next node in the sequence. Using the next part of the node, we can get to the next item in the linked list.
- Navigation:

  The first item in the list is pointed by a pointer called head. We can get to the second item by simply using the next part of the head. We can get to the third item using the next part of the second node and so on.

2.1 Singly linked structures

- Node variables are initialized to either the None value or a new Node object.
- Let us examine a code snippet to demonstrate this:

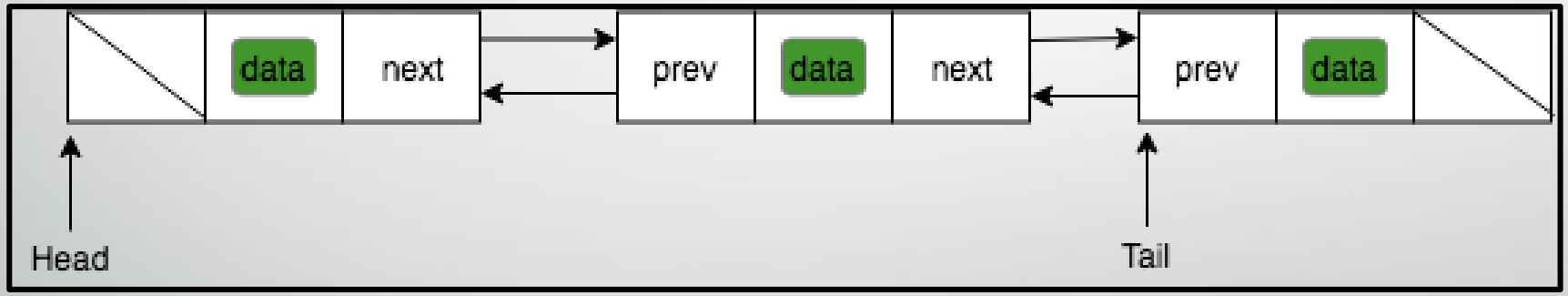
```
node1 = None
# A node containing data and an empty link
node2 = Node("X", None)
# A node containing data and a link to node2
node3 = Node("Z", node2)
```

2.2 Doubly linked structures

- It has a pointer to the next node in the list as well as a pointer to the previous node in the list. That means we can go in both forward and backward direction. Figure 4 shows an example of a doubly linked list.(Linked lists, n.d.)

Figure 4

Doubly linked structure



(From Linked lists, n.d.)

2.2 Doubly linked structures

- The following test program creates a doubly linked structure by adding items to the end. The program then displays the linked structure's contents by starting at the last item and working backward to the first item:

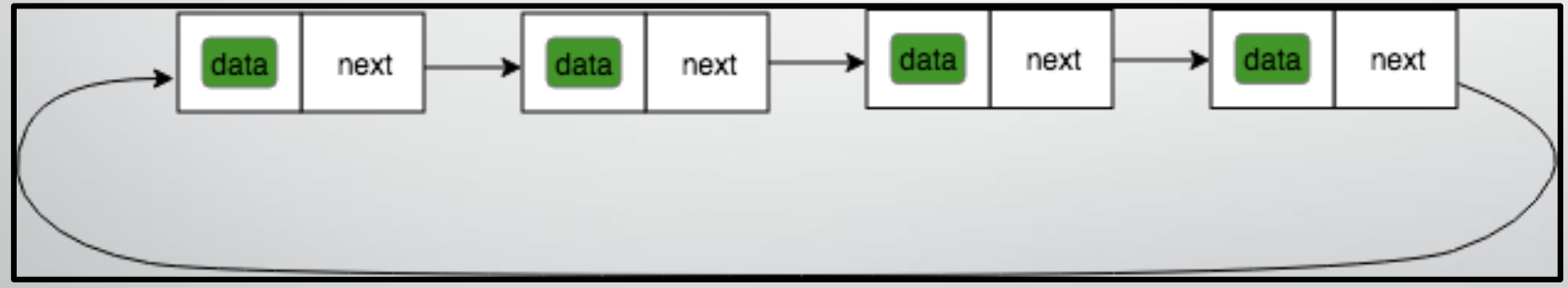
```
"""File: testtwowaynode.py

Tests the TwoWayNode class.

"""

from node import TwoWayNode

# Create a doubly linked structure with one node

head = TwoWayNode(1)

tail = head

# Add four nodes to the end of the doubly linked structure

for data in range(2, 6):

    tail.next = TwoWayNode(data, tail)

    tail = tail.next

# Print the contents of the linked structure in reverse order

probe = tail

while probe != None:

    print(probe.data)

    probe = probe.previous
```

2.3 Circular linked structures

- In a linear linked list, the last item points to nowhere. That means we can not go further in the forward direction from the last node. In a circular linked list, however, the last node points to the first node of the list making it a circular structure. Figure 5 is an example of a circular singly linked list. (Linked lists, n.d.)

Figure 5

Circular linked structure



(From Linked lists, n.d.)

Summary

- A one-dimensional array is composed of multiple sequential elements stored in contiguous bytes of memory and allows for random access to the individual elements.
- An array (rather than the list) is the primary implementing structure in the collections of Python and many other programming languages (including C).
- While Python does not provide the array structure as part of the language itself, it now includes the ctypes module as part of the Python Standard Library. This module provides access to the diverse set of data types available in the C language and the complete functionality provided by a wide range of C libraries.
- Python also has the array module which can be used to create and manipulate arrays. In order to use this simply import it using the *import* command in Python.
- A dynamic array is just like a normal array. The difference between the two is that the size of a dynamic array can be dynamically modified at runtime. We don't need to specify the size of the array beforehand. In a dynamic array, once the contagious memory is filled, a bigger chunk of memory is allocated.
- After arrays, linked structures are probably the most commonly used data structures in programs. Like an array, a linked structure is a concrete data type that implements many types of collections, including lists
- Three types of linked structures : singly linked, doubly linked, and circular linked structures.

References

- Lambert, K. (2019). *Fundamentals of Python: Data Structures*. Cengage Learning.
- Lemonaki, D. (2022, January 31). *Python Array Tutorial – Define, Index, Methods*. FreeCodeCamp.org. https://www.freecodecamp.org/news/python-array-tutorial-define-index-methods/
- *Linked Lists*. (n.d.). Algorithm Tutor; Algorithm Tutor. Retrieved September 29, 2023, from https://algorithmtutor.com/Data-Structures/Basic/Linked-Lists/
- Necaise, R. D. (2011). *Data structures and algorithms using Python*. John Wiley And Sons.
- *Python - 2D Array - Tutorialspoint*. (n.d.). Www.tutorialspoint.com. Retrieved September 29, 2023, from https://www.tutorialspoint.com/python_data_structure/python_2darray.htm
- Sao, P. (2021, January 24). *Python Dynamic Array: Implementation with Examples*. Python Pool. https://www.pythonpool.com/python-dynamic-array/