Software Design Laboratory Laboratory Report #3: Data Structures (PYTHON) Content:

1. **PreLab**

* **Readings**
  1. **Python DS**:
     + Chapters 7 ,8 ,10 ,12 : Ref Lab3 Fundamentals Of Python Data Structures , Kenneth A. Lambert

Write your own Observation and Conclusion from what you have read.

Observation:

The stack, a widely used collection in computer science, is explained in this chapter 7,8,10, and 12. The stack, as discussed earlier in the chapter, is the easiest collection to learn and use. It does, however, have a number of intriguing uses. Arrays and linked structures are two popular implementations covered in this chapter. The chapter concludes with a case study in which stacks play a crucial role: arithmetic expression translation and evaluation.

Linear collections include both stacks and queues. With queues, insertions are limited to one end, the back, while deletions are limited to the other end, the front. The first-in, first-out (FIFO) protocol is thus supported by a queue. People or objects are arranged in a line for service or processing on a first-come, first-served basis, resulting in queues. To name a few, businesses, highway tollbooths, and airport baggage check-in lines all have lines.

In the linear data structures explored thus far, all items except the first have a separate predecessor, and all items except the last have a distinct successor. In a tree, the concepts of predecessor and successor are substituted by parent and child.

The graph begins by defining several key terms related to graphs. The two most common graph representations are the adjacency matrix representation and the adjacency list representation. The book then goes into a number of well-known and widely used graph-based algorithms. The algorithms of primary relevance include graph traversals, minimal spanning trees, topological sorting, and shortest-path issues. The chapter concludes with a case study and the introduction of a graph class.

Conclusion:

Finally, a data structure is a data format for arranging and storing data so that any user may easily access and deal with relevant data to run a program efficiently. Data structure is the method of logically or mathematically organizing computer memory data.

* 1. **Python DS**: Chapters 7, 10 : Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al

Write your own Observation and Conclusion from what you have read.

Observation:

Hash tables are notable for their constant time complexity of O(1), which means they scale effectively when employed in algorithms. The linear time complexity of searching through a data structure like an array is O. (n)

A skip list is a data structure that is based on probability. With a linked list, the skip list is used to hold a sorted list of elements or data. It enables the elements or data to be viewed efficiently. It skips numerous components of the entire list in a single step, which is why it's called a skip list. Hashmaps are the most often used implementation of the map notion. They enable the association of arbitrary items with other arbitrary objects. The constant time complexity of hash tables is O(1), which means they scale effectively when employed in algorithms. A linear time complexity of O exists when searching across a data structure like an array (n)

A probabilistic data structure is a skip list. A linked list can use the skip list to store a sorted list of elements or data. It makes it possible to efficiently see the elements or data. It is called as a skip list because it skips certain components of the full list in a single step.Hashmaps are the most common way to represent a map. They enable the association of any thing with any other object.

Conclusion:

In conclusion primitive data structures, non-primitive data types of not only store values, but a collection of values in different formats.

**Answers to Questions**

Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al

Answer : R-7.1 to R-7.4 page 294

R-7.1

from typing import Any from toydata . Linked Lists import Singlellist def Second\_To\_Last(Single\_Linked\_List : Singlellist ) - > Any : Pointer = Single\_Linked\_List.head while Pointer.next.next is not None : Pointer = Pointer.next return Pointer.value Single\_Linked\_List = Singlellist ( [ 1 , 2 , 3 ] ) print ( Second\_To\_Last( Single\_Linked\_List ) ) R - 7.3

R-7.2

To combine two singly linked lists L and M into a single list L' that contains everything, traverse the first list to get to the nodes of L, then the nodes of M, and finally the nodes of M.

The node of the first list has been discovered. Set the next attribute of the final node to the top of the second list.

R-7.3

Iterative C ++ program to find length / / or count of nodes in a linked list #include < bits/stdc++.h> using namespace std;

R-7.4

//Swapping in singly Linked List

void swappingNodes(Node \*\*head, Node\* x, Node\* y)

{

//if both nodes are same

if (x == y) return;

// Search for y (keep track of previous of node y and Current node y )

Node \*prevY = NULL, \*currentY = \*head;

while (currentY && currentY!= y)

{

prevY = currentY;

currentY = currentY->next;

}

// Search for x (keep track of previous of node x and Current node x )

Node \*prevX = NULL, \*currentX = \*head;

while (currentX && currentX != x)

{

prevX = currentX;

currentX = currentX->next;

}

// If either x or y is not present in the list

if (currentX == NULL || currentY == NULL)

return;

// If x is not head of linked list

if (prevX != NULL) prevX->next = currentY;

else // Else make y as new head

\*head = currentY;

// If y is not head of linked list

if (prevY != NULL)

prevY->next = currentX;

else // Else make x as new head

\*head = currentX;

// Swap next pointers

Node \*temp = currentY->next;

currentY->next = currentX->next;

currentX->next = temp;

}

//swapping nodes in doubly linked list

void swapingNodes(struct Node\* x, struct Node\* y){

if ( x->prev != NULL){

x->prev->next = y;

}

else {

head = y;

}

if ( y->next !=NULL){

y->next->prev = x;

}

x->next = y->next;

y->prev = x->prev;

y->next = x;

x->prev = y;

}

swapping node in singly linked list takes more time.

1. **InLab**

* To know organizing, managing, and storing data to enables easier access and efficient modifications.
* To show Data Structures that shows how to organize data Implementation of Single / Circular / Doubly Linked List

1. Write Python programs for the following operations on Single / Circular / Doubly Linked List.
   1. Creation (ii) insertion (iii) deletion (iv) traversal

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Discussion:

A linked list's elements are sorted in ascending order using Bubble Sort. Create a new node index for later use using head as the current node. Return if head is nil. Otherwise, loop until you reach the last node. Index the current node's next node. Check if the current node's data is greater than the next node's. Swap current and index if it's higher.

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

1. To store a polynomial expression in memory using Single / Circular / Doubly Linked List.

Discussion:

The program generates a circular doubly linked list and displays a menu for the user to do various actions on it. The variable first refers to the circular doubly linked list's initial member. Get node, insert after, insert before, insert at begin, insert at end, remove, and display are all methods to define. get node accepts an index as a parameter and returns the node at that index by traversing the list from the first node that many times. If it hits the initial node again, it will come to a halt.

Implementation of Stack and Queue

1. Design and implement Stack and its operations using List.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

The list data structure in Python can be utilized as a stack. Instead of pushing elements to the top of the stack, append() adds them to the bottom, while pop() removes them in LIFO order. Unfortunately, there are a few flaws in the list. The main difficulty is that as it grows, it may experience speed issues. The elements in the list are stored in memory next to each other; if the stack gets larger than the block of memory that currently holds it, Python must allocate additional memory. As a result, some add() calls may take significantly longer than others. In Python, a stack can be implemented in a number of different ways.

1. Design and implement Queue and its operations using List

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

Before, the introduced the queue and explored array implementationis introduced . The implementation of linked lists is now discussed. The following two major operations must be carried out successfully. There is a need to keep two pointers, front and back, in a Queue data structure. The front indicates the first item in the queue, while the back indicates the last item. enQueue() Adds a new node after rear and transfers rear to the next node with this operation. deQueue() removes the front node and moves it to the next node in the queue.

Implementation of Binary Search Tree

1. Create a binary search tree.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Discussion:

If there had been a sorted array, a binary search might have been conducted to find a value. If there is a need to find a number in an array, it should start by identifying the entire list as the search space; the number can only exist within the search space. Now compare the number to be searched or the element to be searched with the middle element (median) of the search space, and if the record to be searched is less than the middle element, search in the left half, otherwise, search in the right half; if the two elements are equal, the element has been found.

In binary search, start with 'n' elements in search space and if the mid element is not the element that is being sought, reduce the search space to 'n/2'. Continue reducing the search space until it reaches only one element in search space or the record that is being sought, and then stop.

1. Traverse the above binary search tree recursively in pre-order, post-order and in-order.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

1. Count the number of nodes in the binary search tree.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

Push root node to Queue and create an empty Queue Node.

While nodeQeue is not empty, do the following. Take something from the queue and process it. If the node is full, increment count++. If available, push the left child of the popped item to the Queue. If available, push the right child of the popped item to the Queue . Note that leaves should not be handled because both of their children are NULL.

Implementation of Traversal Algorithm for Breadth first traversal

1. Create a traversal algorithm for Breadth first traversal

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

A graph's Breadth-First Traversal (or Search) is comparable to a tree's Breadth-First Traversal. The one caveat is that, unlike trees, graphs can have cycles, which means it might end up at the same node twice. Uutilize a boolean visited array to prevent processing a node multiple times. Assume that all vertices are reachable from the starting vertex for the sake of simplicity.

Start traversing from vertex 2 in the graph below, for example. Look for all of the vertices that are adjacent to vertex 0. Additionally, 2 is a neighboring vertex of 0. If no visited vertices are marked, 2 will be processed again, and the process will become non-terminating. The following graph has a Breadth-First Traversal of 2, 0, 3, 1.

Implementation of Traversal Algorithm for Breadth first traversal

1. Create a program that implements the concept of hashing using separate chaining.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Discussion:

The hashing function may cause a collision if two or more keys are assigned to the same value when hashing. Collisions are avoided by chain hashing. Each cell in a hash table should point to a linked list of records with the same hash function value. When a collision happens in Linear Probing, we probe to the next empty slot. Whenever a collision occurs in Quadratic Probing, we probe for the i2nd slot in the ith iteration and continuing probing until an empty slot in the hashtable is located.

Write your own Observation and Conclusion from what you have executed/read.

Observation:

It is critical to understand different algorithms. An algorithm is a step-by-step technique that outlines a collection of instructions that must be followed in a specific order to get the intended result. Algorithms are often written without regard to the underlying programming languages; that is, an algorithm can be written in more than one computer language.

Python is a great language for beginners since it allows them to create a wide range of programs, from simple text processing to web browsers and games. One of the primary issues that many of us have faced when developing in Python is the lack of pointers.

Conclusion:

Data Structures and Algorithms expertise is used to help programmers. Learning its different uses and how it is implemented can assist programmers in doing their project better. Thus, it is important to study the usage of data structures and algorithms.

1. **PostLab**

* **Project**

Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al: Chapter 7 Projects page 297

Do Projects P-7.44, P-7.45, P-7.46

P-7.44

* **Debugging and Sample Run** (with screenshots and Discussion)

P-7.45

Step-by-step instructions

An Array List ADT stores a succession of arbitrary objects in an array. The index of an element can be used to access, insert, or remove it. An exception occurs when an index is entered incorrectly.

The basic methods of an Array List ADT are get(), set(), add(), delete(), size(), and isEmpty ( ). A list is a dynamic ordered tuple of homogenous elements A0,A1,...A m-1, where Ai denotes the list's i-th element. The majority of the items in an array are null. A list L can be used to efficiently implement such an array, A.

We can store an entry (i,e) in L for each non-null cell A[i], where e is the element recorded at A[i]. This A stands for O(m), where m refers to the number of non-null items in A.

The array list ADT has the following methods:

1) Create an m-dimensional array A. By returning A[i), the operation get(i) is implemented in O(1) time.

2) By doing t=A[i], A[i]=0, and returning t, the operation set(i,0) is implemented in O(1) time.

3) In the add(i,0) operation, we must create a new element by pushing forward then n-i elements.

A[i],.....A[m-1]. It takes O(m) time in the worst scenario (i=0).

4) In the remove(i) operation, we must fill the gap left by the removed element by shifting backward then n-i-1 elements A[i+],......A[m-1]. It takes O(m) time in the worst scenario (i=0).

The following are the most efficient approaches for performing the array list ADT procedures:

1) The data structure takes up O space (m).

2) The size, isEmpty, get, and set functions take O(1) time to execute.

3) In the worst-case scenario, adding and removing items takes O(n) time.

4) If the array is used in a circular fashion, the operations add(0,x) and remove(0,x) take O(1) time.

5) Instead of throwing an exception when an array is full, we can replace it with a larger one in an add operation.

P-7.46

One basic way to using an array to create a list is to store the list items in the array's elements 0..n-1, where n is the current list length. We'd also need to keep track of the current position and number of items in the list, so we'd create instance variables for each of these in our class, which may look something like this.

public class ListViaNaiveAry { private Object[] contents; // array in which list items are stored private int locOfLast; // points to array element holding last item private int crrntPos; // indicates item previous to current position

**Observers:**

lengthOf() : return locOfLast + 1; isEmpty() : return lengthOf() == 0; atFront() : return crrntPos == -1; atRear() : return crrntPos == locOfLast; prevItem() : if (atFront()) { throw an exception; } else { return contents[crrntPos]; } nextItem() : if (atRear()) { throw an exception; } else { return contents[crrntPos+1]; }

**Navigation Mutators:**

moveToFront() : crrntPos = -1; moveToRear() : crrntPos = locOfLast; moveForward() : if (atRear()) { throw an exception; } else { crrntPos = crrntPos + 1; } moveBackward() : if (atFront()) { throw an exception; } else { crrntPos = crrntPos - 1; }

**So far, everything appears to be fine. However, we will find that the insertion and deletion operations are troublesome in the sense that there appears to be no way to make them run quicker than linearly.**

**Node Mutators:**

replacePrev(x) : if (atFront()) { throw an exception; } else { contents[crrntPos] = x; } replaceNext(x) : if (atRear()) { throw an exception; } else { contents[crrntPos+1] = x; } insertRear(x) : if ( lengthOf() == contents.length ) { code to increase length of contents[] } contents[locOfLast+1] = x; locOfLast = locOfLast + 1;

removePrev() : if (atFront()) { throw an exception; } else { // shift contents[crrntPos+1..locOfLast] to left one place for (int i = crrntPos; i != locOfLast; i = i+1) { contents[i] = contents[i+1]; } locOfLast = locOfLast - 1; } insertFront(x) : // shift contents[0..locOfLast] to right one place for (int i = locOfLast+1; i != 0; i = i-1) { contents[i] = contents[i-1]; } locOfLast = locOfLast + 1; contents[0] = x;

The values in the segment contents[crrntPos+1..locOfLast] are shifted to the left one place to perform removePrev(). We should expect around half of the items in the list to be relocated on average, making this a linear time operation.

InsertFront() is slightly worse (but still linear in time), because every value in the segment contents[0..locOfLast] must be relocated one place to the right to make room (at location 0) for the inserted value.

Both insertBefore() and insertAfter() are similar to insertFront(), with the main difference being that the array segment that needs to be relocated to the right starts at crrntPos (in insertBefore()) or crrntPos+1 (in insertAfter()).

Given the limits of our chosen representation scheme, there appears to be no way to avoid the insertion and deletion processes taking time proportional to the amount of elements in the list. This makes us question if there is an alternate array-based representation technique that might allow for faster versions of these operations.

Write your own Observation and Conclusion from what you have executed/read.

Python is a general-purpose programming language that may be used to create a text editor, debug with pycharm, and store items with various data types. Because of the number of libraries, versatility, and simplicity of structure, it is one of the most popular programming languages among developers.

Conclusion:

Data structures and algorithms are essential in Python.

When working with data and a need to improve processing, these principles are essential.

While data structures aid in the organization of information, algorithms provide guidance for solving data analysis problems.

They provide a method for computer scientists to process information provided as input data.