Data Structure Toolkit in Python

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

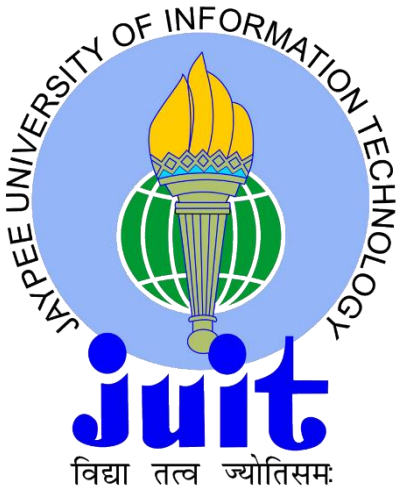
Kshitij Sharma (141268)

Navdeep Singh (141282)

Under the supervision of

Dr. Ravindara Bhatt

To



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan-173234, Himachal Pradesh

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this report entitled **"Data Structure Toolkit in Python"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science and Engineering**,** Jaypee University of Information Technology, Waknaghat, is an authentic record of our own work carried out over a period from August 2017 to December 2017 under the supervision of **Mr. Ravindara Bhatt**, Assistant Professor **(Senior Grade)** , Department of Computer Science and Engineering.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Kshitij Sharma (141268)
Navdeep Singh (141282)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr. Ravindara Bhatt
Assistant Professor(Senior Grade)
Department of Computer Science and Engineering
Dated:

ACKNOWLEDGEMENT

We are grateful and indebted to Mr. Ravindara Bhatt, Assistant Professor(Senior Grade), Department of Computer Science and Engineering, for his help and advice in the completion of this project report.

We also express our deep sense of gratitude and appreciation to our guide for his constant supervision, inspiration and encouragement right from the beginning of this project.

We also want to thank our parents and friends for their immense support and confidence upon us. We deem it a pleasant duty to place on record our sincere and heartfelt gratitude to our project guide for his long sightedness, wisdom and co-operation which helped us in tackling crucial aspects of the project in a very logical and practical way.

Kshitij Sharma (141268)

Navdeep Singh (141282)

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Expansive library of python language i.e. open source environment and tools for web frameworks, data analysis, and testing appliance create its ecosystem one of the biggest out of any programming community.

For entry level programmers Python is an accessible language because the community boasts many introductory resources. This programming language is widely studied in colleges and is used to work with pre-friendly devices such as Raspberry Pi.

This project aims to develop a Data Structure Toolkit in python so that the Data Structures can be easily incorporated during the production of other practical real-life projects. As the efficiency and success of a real-life project is somewhat dependent on the level and low-complexity of the algorithms incorporated in its code during the production, so a Data Structure toolkit in Python becomes a vital need for any project being developed on Python.

It will offer a combination of commonly used data structures and algorithms. It creates a programmer's basic "toolkit". Some algorithms or data structures in toolkit will provide a good solution for many issues. We focus on the structure and algorithm of the data that has become more profitable than that. This will be introduced to the interests of the trade agreement, whereas each data structure or algorithm costs and expenses to strengthen this concept.

CHAPTER 1
INTRODUCTION

1.1 Introduction

As more powerful computers are being developed, additional computing power is being used to tackle more complex problems, be it in the form of more advanced UIs, bigger problem sets, or new problems thought to be computationally infeasible. More complex problems are demanding more computation, making efficient programs even more necessary. However, as tasks are becoming more complex, they are tending become little like everyday experience. To train today's computer scientists, training must be trained properly to understand the principles behind effective design of the program, because their general life experiences are often not applicable when designing the programs.

Speaking in general sense, a data structure is defined as any data representation and the operations applied on that data. Even a floating point number or integer stored on the memory of computer can be viewed as a simple or plain data structure. Usually, the term "data structure" used by people to mean an organization or structuring for a collection of data items. Example of such a structuring can also be viewed as a sorted list of integers stored in an array.

Sufficient storage is given to store collection of *data sets*, it always allows to search for specified items which are stored in the collection, print or compute the data sets in any order which is desired, or make changes to the value of any particular data item. A clearer example can be considered as a row that contains everything in the unauthorized data. It is possible to perform all necessary actions on the unnecessary row. However, using the correct system will be running during a few seconds and will require more than one day, which will make a difference between the program. For example, sorting a level record in a hash table is faster than sorting in an unsaved row. Every data structure includes costs and benefits. In pursuit, it is often not true that a data structure is otherwise used otherwise. If a data structure or algorithm is related to one of the most respected, the deficit is usually forgotten for a long time. For nearly each organization and formula given during this report, we've written wherever it's the simplest alternative.

1.1 Problem Statement

A Data Structure Toolkit in Python similar to the ones that already exists in the market for old and common programming languages like C and C++.

Toolkits for most of the programming languages like C, C++ and Java are already available in the market. But it is not so in the case of Python. Due to the increased popularity of the Python programming language in the past two years it is vital that a Toolkit should be made, so that the Python programming language can be used extensively for the development of real-life working projects.
The Toolkit should aim to provide a large set of Data Structures and functionalities to choose from.

However, they have certain drawbacks too:

- Data Visualization is not provided by this toolkit. However the usefulness of the Toolkit would have increased two fold if it would have been able to transform information into graphical data.
- This Toolkit does not provide any documentation on selecting a particular Data Structure for a particular task. This Toolkit only provides the common use case examples of the Data Structures enlisted in this project report. However, to resolve a selected drawback it's crucial to keep this true statement in mind for choosing a suitable data structure. There will be hope of choosing the correct data structure for the task if goals for performance are kept in mind and technique of self analysing is being used. Analysis step is ignored by poor programmers and they apply a data structure that they're acquainted with however that is inappropriate to the problem. The result is typically a slow real-life working project.

1.2 Objectives

The objective of the design of a new Toolkit is that with the use of this toolkit the speed of writing codes can be accelerated for creating the complex and real life projects which is something different from the current procedure.

The proposed Toolkit will keep the well organised Data Structures in a common Directory. They can be incorporated in large projects so that these codes can be reused while writing the codes of the projects so that the code of the complex large project can remain clean and can relieve the programmer of the hassle of again and again writing the same basic code repeatedly. This would tremendously help in increasing the pace of the development of Python based real-life projects.

1.3 Methodology

With the enhancement in the complexity of the real-life projects, it becomes a challenging task to develop a toolkit having a long life span. Hence, mandatory for a toolkit developer to adopt the constantly changing market trend to gain the positive outcomes.

Nowadays, agile methodology is used by firms that is most expeditiously employed by software development firms useful in streamlining the important development method and create it quicker.

This method helps development team to style a final product exactly as per the expectation of their user category. The agile approach for developing a Toolkit in the main focuses of involvement of user, versatile designing ,risk management and continuous analysis.

The key steps involved in an agile development process include:

1. Project Initiation: The steps involved in initiation involves making the plan, specify the needs of business , locking the specifications, constructing team, and making ready initial or early architecture modelling.

2. Development Iterations: Active client participation involved in this repetitive stage, extremely thorough, careful and collaborative development by professionals, putting brand

new features upon requirement, confirmatory and investigative testing, and internal deployment of software.

3. Product Release: Testing of final system, acceptance by final user, and deployment of system into production and regular and updated delivery of software which meets the dynamical desires of the consumer. If needed ,Release phase of software system is reviewed by the end user and send the feedback additionally as request for a lot of options.

4. Production: Regular maintenance of the system is in this phase and defects are identified so that it system performance can be enhanced parallelly with stability.

Some benefits are mentioned below for Agile Software Development :

- Agile development methodology permits several scopes for amendments in the course of the whole life cycle of app development
- Cohesive work approach is utilized by it.
- For making the product seamless it needs testing at each stage and it enables to launch and work the product in a short frame of time.
- Customers ,developers and testers interacts constantly and this enhances the transparency and flexibility if the process.
- Comfortable work conditions are provided to team members.
- Quality and performance of the project can be increased by using the efficient and rapid application process.
- It enhances sustainability, design responsiveness in Toolkit usage and the experience of user.

CHAPTER 2
LITERATURE SURVEY

A literature review is a means to evaluate and interpret all available research relevant to a particular research question, or area. Its main aim is to present a fair evaluation of the research area of interest by conducting a rigorous and auditable methodology. The main purpose of our literature review is to find the relevant literature about the Data Structure Toolkit in Python for the purpose of background study, and to summarize the existing work and identify the gap in the current research.

2.1 Research Papers

2.1.1 The Research on Teaching Ideas of "Data Structure and Algorithm" in Non-computer Major

Authors: Zhao Wang1,2

For key issues and difficulties, non-computer professionals can face learning of data structure and algorithm courses, discussing this paper based on the experience of paper, data structure and algorithms. As the design methods use algorithms to introduce different types of algorithms, to manage course content as modules using data logical structure, students through examples of real life To improve interest, emphasis and application ability.

2.1.2 Why Python Rocks for Research

Authors: HOYT KOEPKE

This paper advocates why the pastor is a complete language for research. Various types of quality and quality features have been discussed in this paper. Many data stats include structural, class, customized functions, stabilizers, flexible function calling syntax, sink in a wide kitchen standard library, great scientific library,

And remaining papers during this paper have been provided. In addition, the fact is that the author allows someone to easily take pattern-oriented and functional design patterns. As soon

as there are different ways to think about various issues, in this way, different issues also say different programming paragraphs.

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 System Requirements

The following Software and system requirements are required to develop a Toolkit in Python:

Hardware requirements

- 0.5 GB RAM minimum, 1 GB RAM recommended
- 800*600 minimum screen resolution

System requirements

JRE 1.8 is bundled with support for all platforms.So one does not needs JAVA to run PyCharm on your machine.

Windows

- Minimum Microsoft Windows XP (incl.64-bit)
- Python 2.4 or higher, Jython, PyPy or IronPython.

macOS

- macOS 10.5 or higher.
- Only 64-bit macOS is supported.
- Python 2.4 or higher, Jython, PyPy or IronPython

Linux

- OS Linux 64 bit
- KDE, GNOME or Unity DE desktop
- Python 2.4 or higher, Jython, PyPy or IronPython

CHAPTER 4

ALGORITHMS

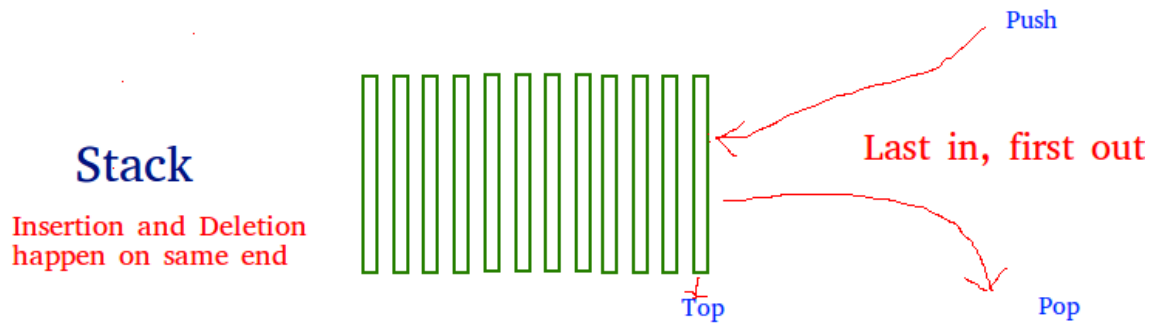
The Data Structures incorporated in the Toolkit are as follows:

4.1 STACK DATA STRUCTURE

Data structure Stack which is linear and it always follows a specific sequence in which the functions are being carried out. It follows the order which may be F.I.L.O.("First In Last Out") or L.I.F.O. ("Last In First Out").

Majorly these are the operations which are performed by this data structure:

- **Push:** This operation insert an item in the stack. Overflow condition appears ,if the stack is fully filled.
- **Pop:** This operates eliminates the data unit from stack. Reverse order is followed to pop the items in the order which they are pushed. Underflow condition arises when the stack is empty.
- **Top or Peek**: Top element is returned by stack.
- **isEmpty:** If it finds the stack empty it return the true value, otherwise it returns false value.



We can consider many actual life cases to understand stack. Let us consider the easy example of coins piled over one another. The coin which is present at the top is the first one to be

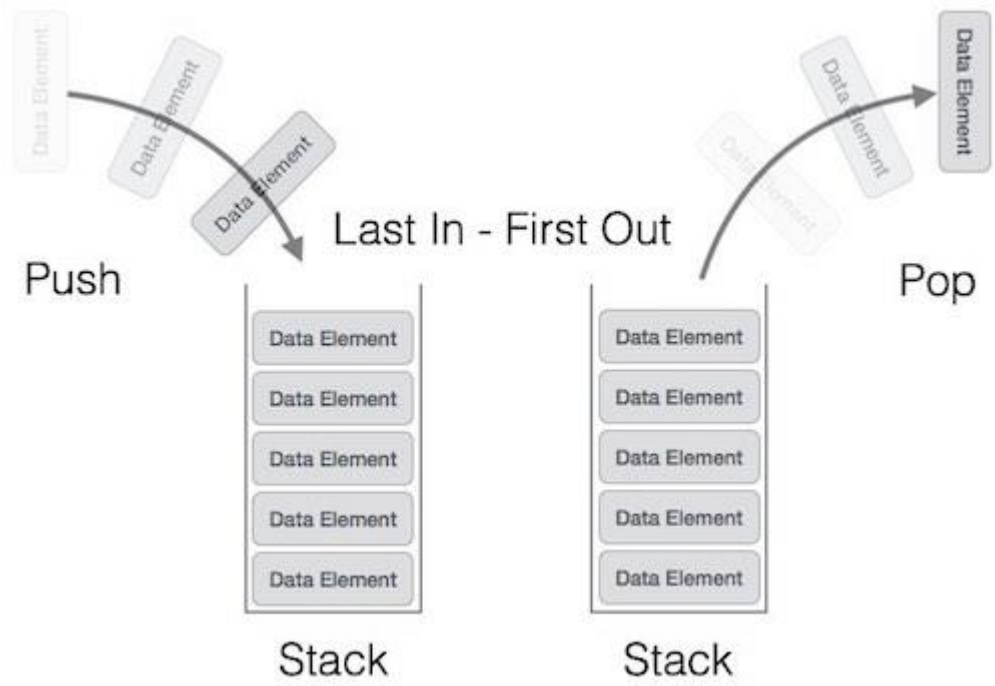
removed, i.e. the coin which is present at the base remains in the stack and spend the longest time at the bottom. Therefore, it can be noticed that it is to follow FILO/LIFO order.

Time Complexities for the operations applied on stack:

push(), pop(), isEmpty() and peek() all take O(1) time. These operations don't run any for loop.

STACK REPRESENTATION

The following diagram depicts a stack and its operations −



Implementation:

There are two ways in which a stack can be implemented -

- By using linked list
- By using array

ALGORITHM :

1) peek()

Algorithm of peek() function –

```
begin procedure peek

   return stack[top]

end procedure
```

2) isfull()

Algorithm of isfull() function –

```
begin procedure isfull

   if top equals to MAXSIZE
      return true
   else
      return false
   endif

end procedure
```

3) isempty()

Algorithm of isempty() function –

```
begin procedure isempty

   if top less than 1
      return true
   else
      return false
   endif
```

4) push()

```
begin procedure push: stack, data

   if stack is full
      return null
   endif

   top ← top + 1

   stack[top] ← data

end procedure
```

5)pop()

```
begin procedure pop: stack

   if stack is empty
      return null
   endif

   data ← stack[top]

   top ← top - 1

   return data

end procedure
```

Stack Data Structure Common Uses:

1) Expression evaluation stack : The foundational kind of stacks which were extensively delivered by special hardware were Expression evaluation stacks. As a arithmetic expression is interpreted an arithmetic expression, track must be kept of precedence of operations and intermediate stages which are using an evaluation stack. It keeps the two stacks in possession, if interpreted language is used by it.. One of two stacks contains the operations which are pending and it wait for the high precedence operation to be completed. The second stack contains the inputs which are intermediate and these are related to pending operations. In compiled language, a single expression evaluation stack is used by hardware so that it can hold intermediate results and track of pending operations is kept by compiler during its instruction generation.

2) The parameter stack : Stack is typically used in computing as a subroutine parameter stack. It must usually be provided a set of parameters on which it would compute when the stack is called. Parameters are passed by using registers in which the values are placed but it limits the number of parameters and it is the disadvantage of using registers. Another way of passing the pointer is that they are passes by copying them into list or pointers to them into a list and for this routine's memory is used. For this, recursion and redundancy may not be possible. The most workable method is ,performing a procedure call and just simply duplicate the elements onto a parameter stack. Both re-entrancy in programs and recursion is allowed in parameter stack.

3) Postfix Evaluation Algorithm

4) Backtracking : Some algorithms uses backtracking and these algorithms should be in which there are steps along some state from some initial point to some target. In all of these cases, it has to made some choices among the options which are present. But if we need to try some different possible way or we want to come back so it should keep the track of previous decision points. Lets consider the puzzle of choosing a right paths, a choice is made at some point and this choice leads to a dead end then we have to go back over to that faulty decision

point and try the next alternative .Again, for this solution stacks can be used. Recursion is also the another solution and it is implemented by a stack.

4.2 QUEUE DATA STRUCTURE

Queue and Stacks are similar in a moderate extent. Differently from stacks, queue is not closed at both its tips. One of two end is used to insert data which is called enqueuing and second is used to withdraw data which is called dequeuing. First-In-First-Out methodology is used by queue which is defined as the data unit stored first will be accessed first.

A real-world example of queue can be a ticket window, where the customer enters first, exits first. There can be more real-world examples for queues that are one way road and passengers waiting at bus-stop .

Queue Representation

In queue both ends are accessed for reasons which are apart from each other. This diagram will explain queue representation as data structure –

As similar to stack, we can implement queue using Linked-lists, Arrays, Structures, Pointers.

Basic Operations

Some queue operations are :

- Initializing the queue
- Using the queue
- Eliminating the queue from the memory.

The operations related to queues −

- **enqueue()** – Storing data into queue.
- **dequeue()** − Removing data item from the queue.

To make the queue operations efficient ,these mentioned below functions are also important.

- **peek()** – Getting the top element without popping it.
- **isfull()** –To check if the queue is completely filled to its capacity or not.
- **isempty()** –To check for an empty queue.

ALGORITHM:

1) peek():

```
begin procedure peek

   return queue[front]

end procedure
```

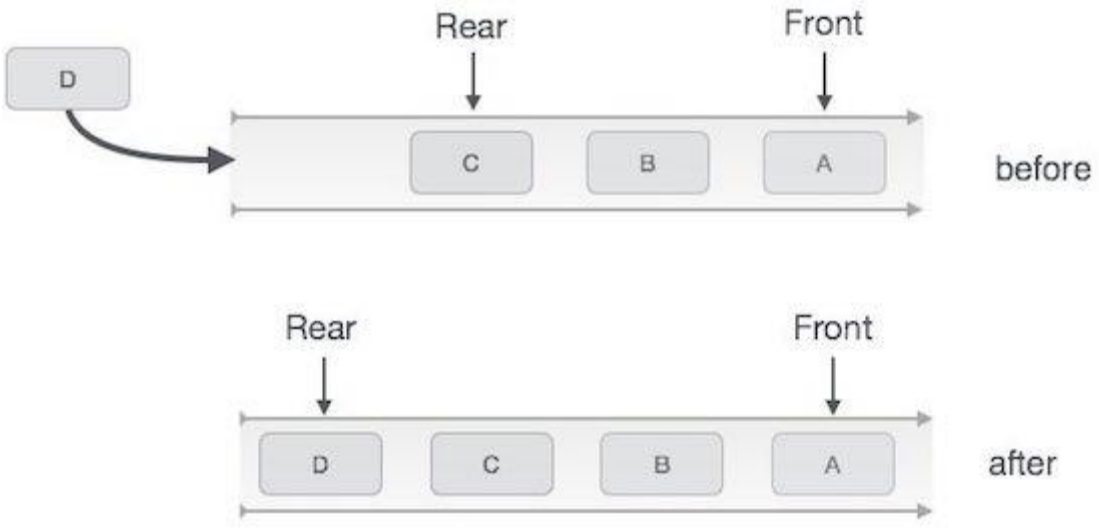
2) isfull():

```
begin procedure isfull

   if rear equals to MAXSIZE
      return true
   else
      return false
   endif

end procedure
```

3) isempty():

```
begin procedure isempty

   if front is less than MIN  OR front is greater than rear
      return true
   else
      return false
   endif

end procedure
```

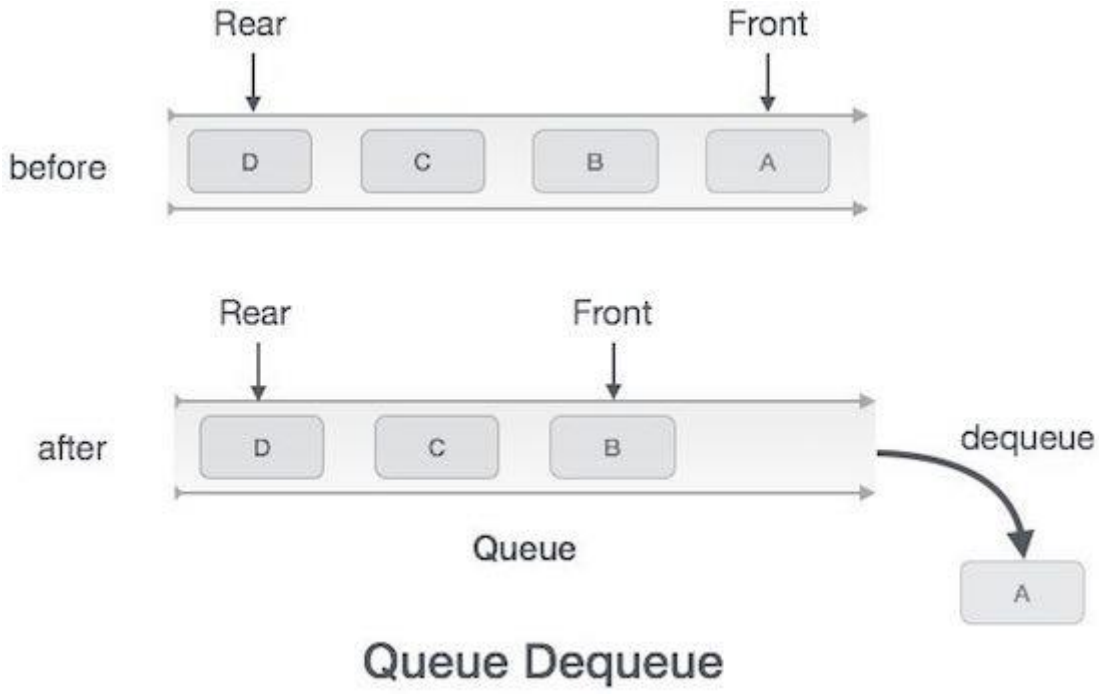
4)enqueue:



Queue Enqueue

```
procedure enqueue(data)
   if queue is full
      return overflow
   endif

   rear ← rear + 1

   queue[rear] ← data

   return true

end procedure
```

5)dequeue:



Queue Dequeue

```
procedure dequeue
   if queue is empty
      return underflow
   end if

   data = queue[front]
   front ← front + 1

   return true
end procedure
```

Common uses of Queue Data Structure:

Queue is used when things don't have to be processed immediately, but have to be processed in **F**irst **I**n **F**irst **O**ut order like Breadth First Search. This unique speciality of Queue data structure makes it also convenient in these scenarios.

1)Whenever multiple clients are sharing a same resource. Examples include Scheduling Disk and CPU scheduling.

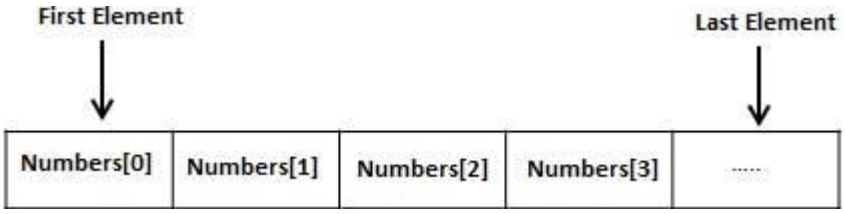
2) When data is transferred asynchronously (data not necessarily received at same rate as sent) among two different processes. Examples include IO Buffers, pipes and others.

4.3 ARRAY DATA STRUCTURE

Array is a data structure. Collection of data items is defined as array which all are of the same type and they are approached using a common name. An array can store the collection of data but it should be taken as a accumulation of variable which are of same type.

All elements which are present in array are accessed by its index position. In lieu to declare each and every variable separately such as data0 ,data1, data2, ……. and data99, we can declare one array variable as data and use this as data[0], data[1], and…..,data[99] so it can represent every individual variable.

Contiguous memory locations are allocated to arrays. The address which is lowest the position of first element and highest address shows the position of last element.



Accessing Array Elements

There may be element in array by naming row. The name of row element can be accessed after retaining the index element inside the square bracket.

For example -

Age = Age [4];

The following statement will take the 5th element from the row and the variable will be assigned the value of the age.

Two-dimensional Arrays

To define a simple form of multidimensional arrays we can consider a two dimensional array. A two-dimensional array is can be considered as a list of arrays which are one-dimensional . If we wish to declare a two dimensional array of integer of size[a][b],we have to write something in this order:

dtype arrName [a][b];

Here **arrName** can be a valid C identifier and **dtype** will be any valid C data type . We can take two-dimensional array as a table in which **a** will be number of rows and **b** will be number of columns.

Here is an example of two dimensional array with 3 rows and 4 columns.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Common uses of Array Data Structure:

If we want to practice and use matrices and vectors then arrays can be used .Arrays can be used to implement rectangular tables. A lot of databases consist of one-dimensional arrays whose data units or elements are records.

Many data structures such as heap ,hash tables ,queues ,strings can be implements using the arrays. To make data structure space efficient and simple we can use array based data structure. Little space overhead is required but it may have bad space complexity, particularly if we set it side by side with tree based data structures.

Particularly in memory pool allocation, one or more than one big arrays are once in a while used to emulate in-application dynamic memory allocation. To allocate "dynamic memory" portably this is the best way to allocate as of now and even historically.

To determine use of partial or complete control flow in codes ,we can use the arrays. To multiple IF statements ,it is a compact alternative. They are known on this context as control tables. They are used together on the grounds of building interpreter whose control flow is altered in step and whose values contained inside the array. Route of execution is executed by subroutine pointers.

4.4 LINKED LIST DATA STRUCTURE

Sequence of data structures is referred as linked list, they are connected together by way of links.

Linked List is a chain of links or it is a contains links which are part of sequence and data items are contained by it. Every link is connected to each other. After array, the most used data structure is Linked list.

- **Link** −Data is stored in link which is part of linked list and that link is called element.
- **Next** – Every single element (node) of a list is composed of two items – one of which is data and the other is address of next present node. Null reference is pointed by the last present node. Next always contains the address to next present node.
- **LinkedList** − First is the initial link of linked list.

Linked List Representation

Linked list is based on concept of chaining of nodes, where every node which is present in linked list points the next present node.

As reference to the above self explanatory diagram, these are some points to be considered-

- Next link links every link to next present link.
- End of list is marked by last node which always have a null value.

Linked List types

- **Simple Linked List** − Item can be navigated only ahead.
- **Doubly Linked List** − Forward and backward navigation can be there in case of doubly linked list.

- **Circular Linked List** – We can say that this type of Linked List i.e. circular is more complex linked list data structure. In this list data item can be inserted anywhere but in the array its not possible because of contiguous memory allocation . In this list the preceding element hoard the address of the next present element and last element hoard the address of the initial element. Circular chain is formed by pointing of elements in circular way. Because of the dynamic size of circular linked list memory can be allocated whenever required.

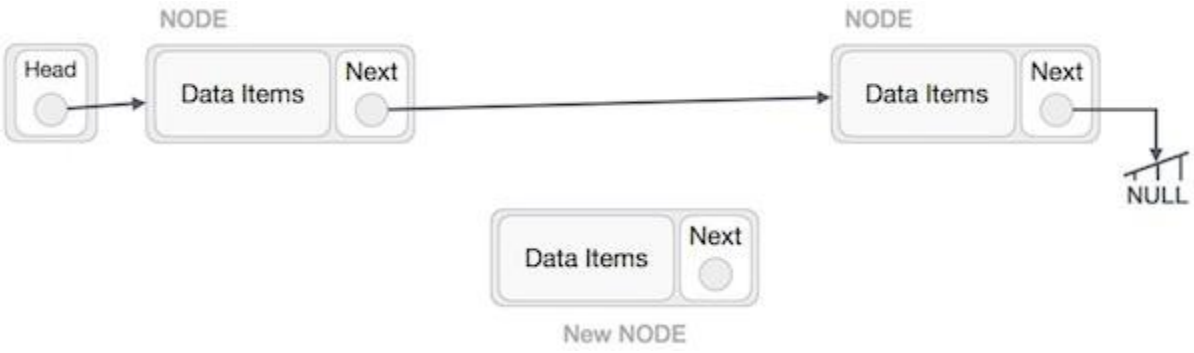
Basic Operations

Basic operations that a list can perform are.

- **Insertion** –Element added to the start of list.
- **Deletion** –Element deleted from start of list.
- **Display** –Complete linked list is displayed with this operation.
- **Delete** − Desired data item of user is eliminated by this operation.
- **Search** −It searches for the data unit in linked list using the given value.

Operation for Insertion

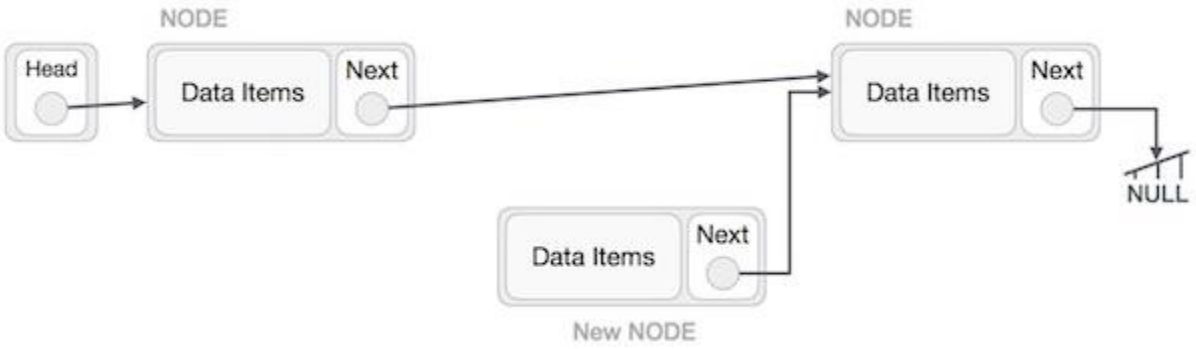
Here we will discuss the operation of adding new node to linked list and it is not a one step procedure. Pictorial representation will explain it clearly. Firstly, we have to construct a node which is of identical structure and then look for the position where it has to be placed.



Let us visualize that we need to insert a node **Y** (New Node), between **X**(LeftNode) (it is the first node)and **Z** (RightNode). Then point Y.next to Z−

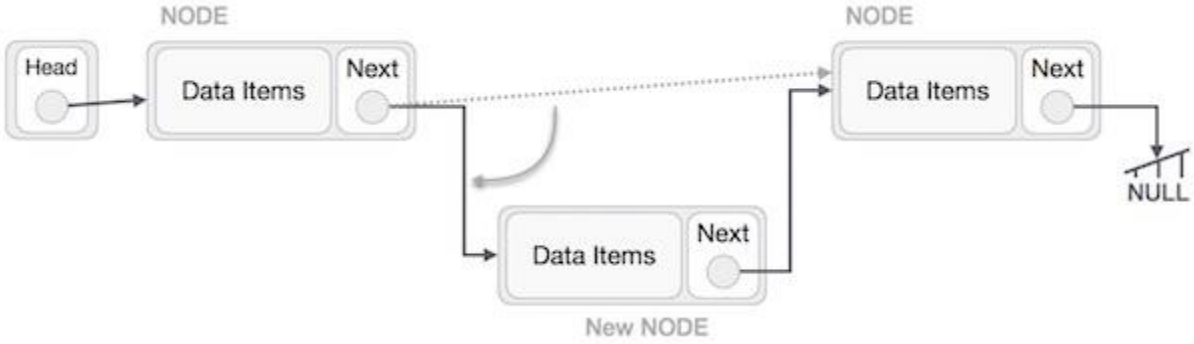
Operation will be " NewNode.next −> RightNode; "

Now the situation will become like this --



The next of the node at the left side should be pointing towards the new node.

Operation will be " LeftNode.next −> NewNode; "



This operation will add the newNODE in between the two nodes.

It can be seen as this --

This procedure should be followed to insert a node in the beginning of list. While at the time of inserting at the end, the node which is previous to last node of the list should be pointing to the new node and NULL is to be pointed by new node .

Deletion Operation

Deletion is also a long procedure rather than just a single step. We will use the search algorithm to locate the node which is to be eliminated from the list.

The node which is previous to the target node should be pointing to the node which is next to the victim node.

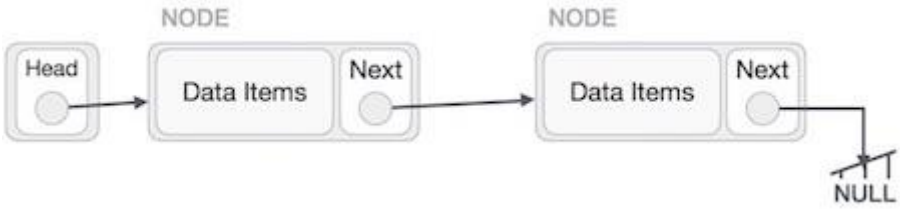
Operation will be " LeftNode.next −> TargetNode.next; "

This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

Now, our next objective is to eliminate the link where the target node is pointing.

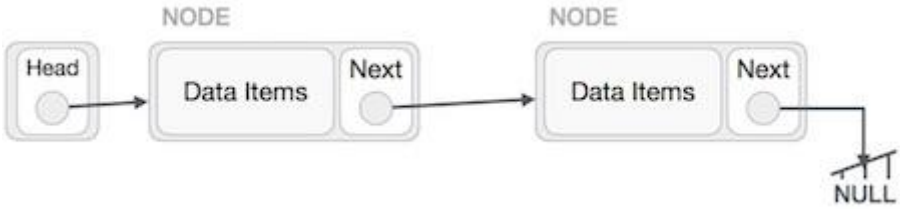
Operation which is used will be " TargetNode.next −> NULL; "

Now we have to use the node which is deleted. .If we want to keep the node memory we can do that or just deallocate memory used by it and thrash the victim node.

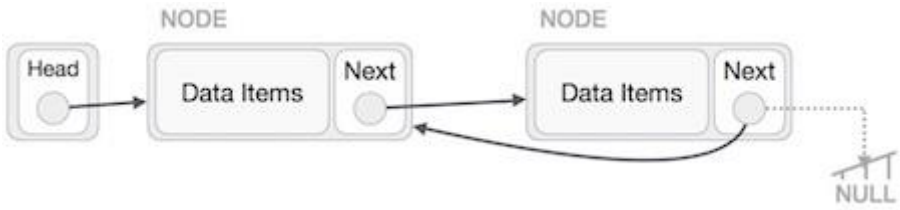


Reverse Operation

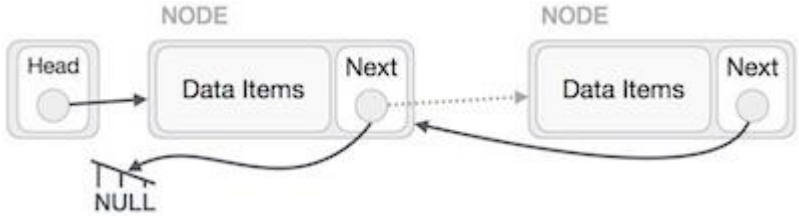
The whole list is reversed making the last node the first node and vice-versa. The size of the list and the address of the nodes remain the same just all the pointers are reversed.



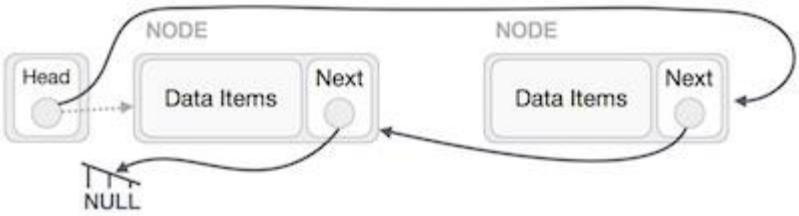
First, we traverse to the node that points to NULL because this is the end node. Now, we will make it point to its previous node−



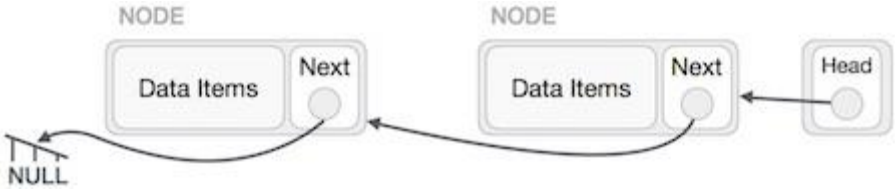
We will create a temporary node, which will be the new start node pointing to the last node. Now, we make all the pointers point to the previous of each node so that the list is reversed until we reach the old START node.



Now we will make the first node(old START) point to NULL because this will be the last node of the new reversed list.



We will use a temporary node to make the HEAD point to NULL.



Common uses of Linked Lists :

The integrated lists and the most common data are included in the linked list. Many other common abstracts can be used to apply data types, including lists, sticks, rows, association arrays, and S-expressions, even without applying any list without applying it. Directly the other data structure is not unusual to apply.

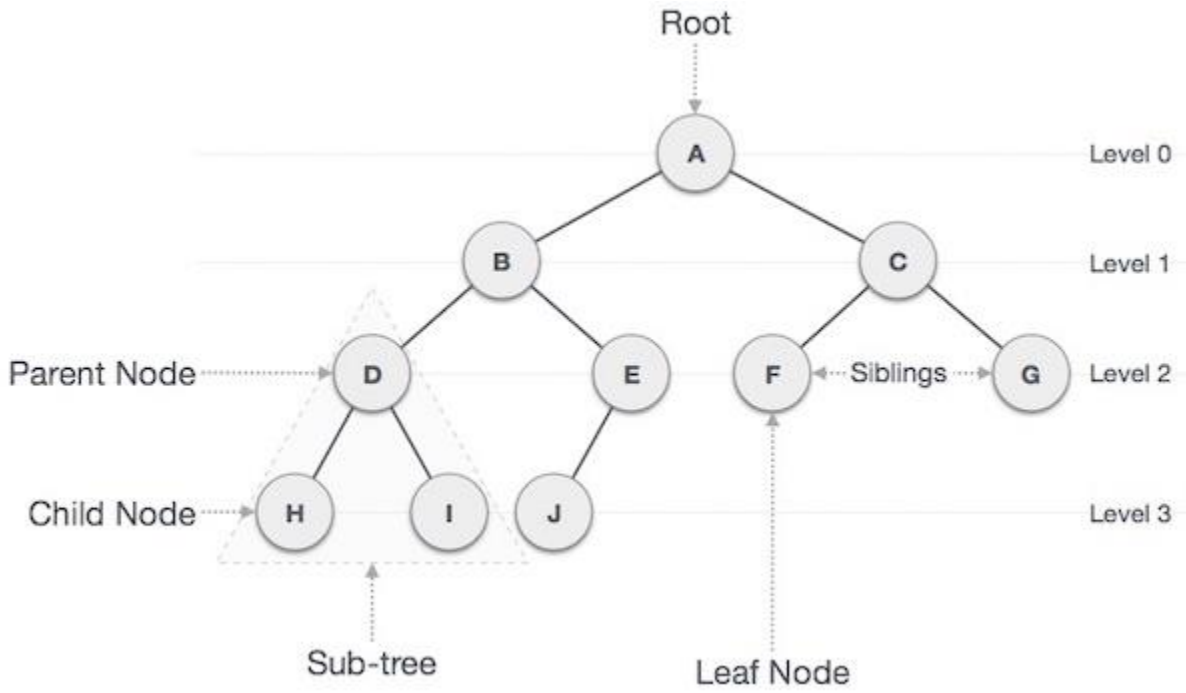
The principal advantage of a traditional end-linked list is that the list elements can easily be removed without the restructuring or re-layout of the entire structure, because the data item is not stored in a memory or disk contradictory. Yes, while there is a row before the program is set and running, the source code will be announced. Allows link lists to enter and remove nodes at any time in the list, and with this constant number of links you can add or remove this link.

On the other hand, simple attachment lists do not permit access to data, or random access to any form of effective indexing. In this way, many basic operations - such as obtaining the last node of the list (assume that the last node label does not remain as separate nod references in the structure), or find a given pad containing the given dots Yes, or find out where a new node can be inserted - perhaps the need for scanning of most or all list elements. The benefits and damages using linked lists are given below.

4.5 TREE DATA STRUCTURE

Tree is a hierarchical data structure that represents a directory like structure represented by nodes(memory locations) connected by edges(pointers).

Binary Tree is a data structure used for data storage and representation purposes. In a binary tree each node can have a maximum of two children. A binary tree has the advantages of both an array and a link list as the insertion and delete operations can be performed as fast as in link list and the search operation can be performed as fast as in an array.



Important Terms

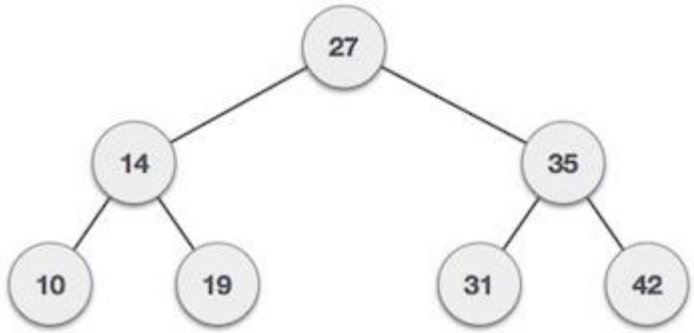
Tree Terminology:

- **Path** – Path is the sequence of nodes formed by pointers.
- **Root** – First node at level 0 of the tree is called the root of the tree. There can be only one root in tree and there exists a path from the root to every other node of the tree.
- **Parent** – The node just previous to a node in the traversal from the root to the node is called the parent of the node.
- **Child** – The node next in the traversal from root to a particular node is called the child of a node.
- **Leaf** – The node whose left and right pointers are null is called as the leaf node in the tree.

- **Subtree** − Subtree represents the descendants of a node.
- **Visiting** – Checking the value of a node by traversing to it.
- **Traversing** – Moving through the tree in a specific order.
- **Levels** – The distance of a node from the root node of the tree is called as the level of the node. The level of the root node is 0.
- **keys** – Represents he value stored at a node. On this value the search operation is carried out on the tree.

Binary Search Tree Representation

Binary Search tree exhibits a special behavior. A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.



Tree Node
ALGORITHM:

```
struct node {
   int data;
   struct node *leftChild;
   struct node *rightChild;
};
```

In a tree, all nodes share common construct.

BST Basic Operations

The basic operations that can be performed on a binary search tree data structure, are the following −

- **Insert** − Inserts an element in a tree/create a tree.

- **Search** − Searches an element in a tree.
- **Preorder Traversal** – Root -> Left -> Right.
- **Inorder Traversal** – Left -> Root -> Right.
- **Postorder Traversal** – Left -> Right -> Root.

Insert Operation

The first first entry creates the tree. Then, whenever the element is inserted, first locate your appropriate location. Start searching for root nodes, if the data is minimized, find the blank space in the left subtitle and enter the data. Otherwise, search for the subtitle in the right submission and enter the data.

Algorithm

```
If root is NULL
   then create root node
return

If root exists then
   compare the data with node.data

   while until insertion position is located

      If data is greater than node.data
         goto right subtree
      else
         goto left subtree

   endwhile

   insert data

end If
```

Search Operation

Whenever an element is to be searched, start searching from the root node, then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

Algorithm

```
If root.data is equal to search.data
   return root
else
   while data not found

      If data is greater than node.data
         goto right subtree
      else
         goto left subtree

      If data found
         return node


endwhile

   return data not found

end if
```

TREE TRAVERSAL:

Tree traversal is a process to visit all the nodes of a tree in a specified order. Because all the links(paths) in a tree start with the root node so we usually do start with the root node in the tree traversal process. There are 3 options of traversing a tree.

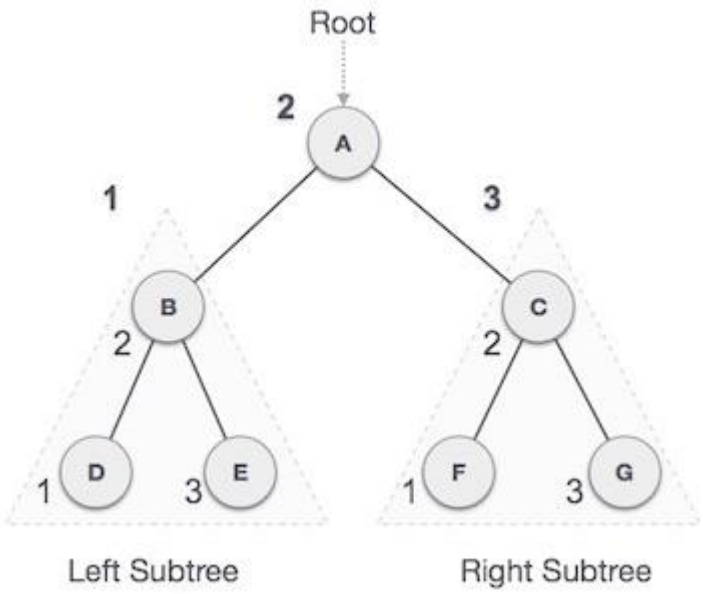
- In-order traversal
- Pre-order traversal
- Post-Order Traversal

In-order Traversal

In this traversal method, the left sub is seen first, then the root and the later sub-tree. We should always remember that every node can be offered by sub-submission itself.

If binary tree changes to the order, then the output will gradually generate key values in sequence.

If binary tree changes to the order, then the output will gradually generate key values in sequence.



We start from the node A then traversing in-order we move to the left sub-tree (i.e.B), now B is also traversed in the in-order way. This process is repeated until all the nodes of the tree are traversed. So therefore the in-order traversal of this tree will yield the result:

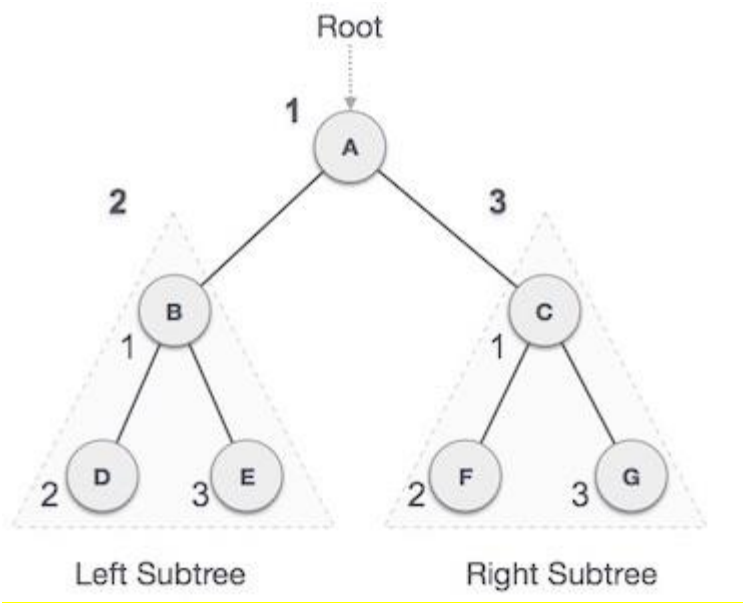
D -> B -> E -> A -> F -> C -> G

Algorithm

```
Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Visit root node.
Step 3 – Recursively traverse right subtree.
```

Pre-order Traversal

In this method the Root node is visited first then recursively the left sub-tree is traversed in the pre-order way and then the right sub-tree in the pre-order way.



We start from the node A then traversing pre-order we move to the left sub-tree (i.e.B), now B is also traversed in the pre-order way. This process is repeated until all the nodes of the tree are traversed. So therefore the in-order traversal of this tree will yield the result:

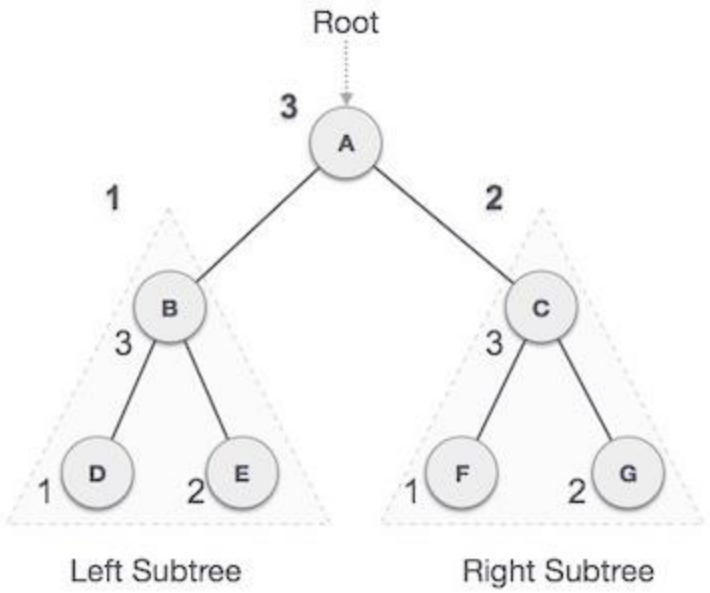
A*-> *B* -> *D* -> *E* -> *C* -> *F* -> *G

Algorithm

```
Until all nodes are traversed –
Step 1 – Visit root node.
Step 2 – Recursively traverse left subtree.
Step 3 – Recursively traverse right subtree.
```

Post-order Traversal

In this method the Left node is visited first then recursively the Root Node is traversed in the post-order way and then the right sub-tree in the post-order way.



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

D -> ***E*** -> ***B*** -> ***F*** -> ***G*** -> ***C*** -> ***A***

Algorithm

```
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.
```

Common applications of Trees :

- Binary Search Tree - Used in many search applications, where the data is continuously entering / leaving data, such as maps and items in libraries in many languages.
- Binary space distribution - Can be used to determine what is used in almost every 3D video game.
- Binary tries - Router tables are used in almost every high bandwidth router to store.

4.6 BINARY SEARCH TREE

A Binary Search Tree (BST) is a tree in which all the nodes follow the following properties −

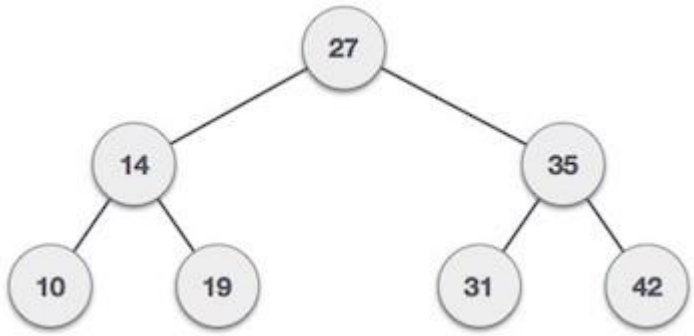
- All the keys to the left of a node are less than or equal to the node's value.
- All the keys to the right of a node are greater than or equal to the node's value.

Thus the BST divides the complete tree into two parts one left-sub tree and one right-sub tree with the following rule

$$\text{Left Node value} \leq \text{Root Node value} \leq \text{Right Node Value}$$

Representation

BST is a type of binary tree with the properties of a BST mentioned above. Every node has a key and a pointer. The values are put in a BST so that the properties of the BST are not disturbed. Here is a picture representation of a Binary Search Tree −



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

Basic Operations

Following are the basic operations of a tree −

- **Search** – Searching a node in the BST.
- **Insert** – Inserts a node in the BST.
- **Pre-order Traversal** – Traverse the tree in the root->left->right manner.
- **In-order Traversal** – Traverse the tree in the left->root->right manner.
- **Post-order Traversal** – Traverse the tree in the left->right->root manner.

Node

Define a node having some data, references to its left and right child nodes.

```
struct node {
   int data;
   struct node *leftChild;
   struct node *rightChild;
};
```

Search Operation

When we want to search for an element we start from the root node.If the element to be searched is smaller than the root we move to the left subtree otherwise left. We follow this recursively until we find the element to be searched. If we reach a leaf node following the above algorithm then the element is not present.

```
struct node* search(int data){
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data){

      if(current != NULL) {
         printf("%d ",current->data);

         //go to left tree
         if(current->data > data){
            current = current->leftChild;
         }//else go to right tree
         else {
            current = current->rightChild;
         }

         //not found
         if(current == NULL){
            return NULL;
         }
      }
   }
   return current;
}
```

Insert Operation

Whenever the element is entered, first locate the appropriate location. Start searching for root nodes, if the data is minimized, find the blank space in the left subtitle and enter the data. Otherwise, search for the subtitle in the right submission and enter the data.

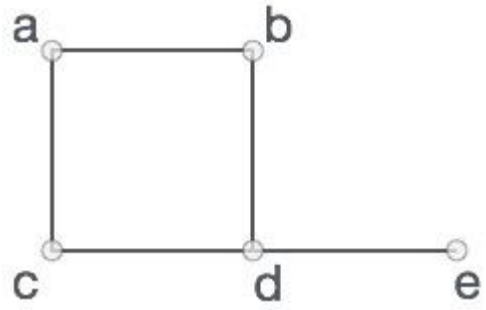
Algorithm

```
void insertinto(int key) {
   struct node1 *tempeNode = (struct node*) malloc(sizeof(struct node1));
   struct node1 *currentone;
   struct node1 *parentone;

   tempeNode->data = data;
   tempeNode->leftChild = NULL;
   tempeNode->rightChild = NULL;


   if(rootn == NULL) {
      rootn = tempeNode;
   }

   else {
      currentone = root;
      parentone = NULL;

      while(1) {
         parentone = currentone;


         if(key < parentone->key) {
            currentone = currentone->leftChild;

            if(currentone == NULL) {
               parentone->leftChild = tempeNode;
               return;
            }
         }
         else {
            currentone = currentone->rightChild;


            if(currentone == NULL) {
               parentone->rightChild = tempeNode;
               return;
            }
         }
      }
   }
}
```

4.7 GRAPH DATA STRUCTURE

The graph is a wonderful picture of a set of items where some pairs of items link to links. The points are represented by points represented by disconnected objects, and links to the circuit link are called links.

Formally, the graph is a pair set (V, E), where V is a set of vertical and e-edges are attached



to vertical pairs. Take a look at the following graphs:

In the above graph,

V = {A, B, C, D, E}

E = {AB, AC, BD, CD, DE}

Graph Data Structure

Mathematical graphs can be represented easily with the help of graph data structure. We can represent a graph using vertical and two-dimensional row of edges.

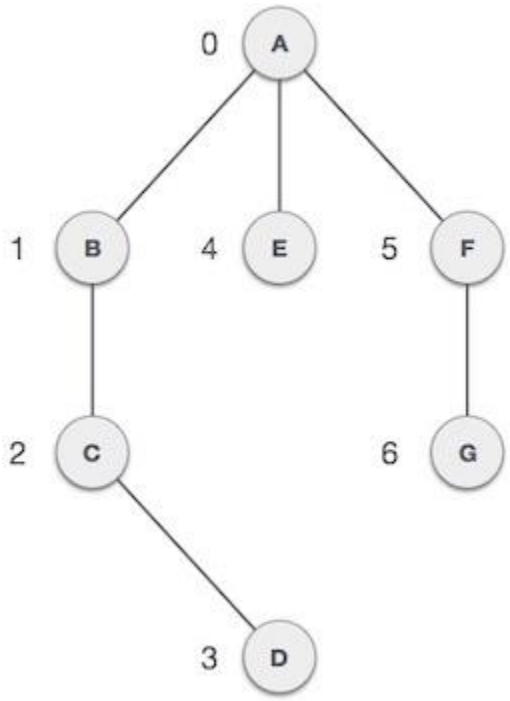
GRAPH TERMINOLOGY:

- **Vertex**- Each node of graph is presented vertically. In the following example, the labeled label presents vertically. Thus, A is Vertical. We are shown in the following picture as they can represent. Here A can be pointed out that index 0.B can be indicated which can be used index 1 and so on.
- **Edge -** Edge represents a line between a line between two vertical or two rows. The following example represents A to B, B to C, and similar lines of edges. The following image is shown as we can use two-dimensional rows to represent a row.

Here the array of AB can be presented as row 1, column 1, BC 1 as array 1, column 2 and so on, keep other combinations as 0.

Adjacency - There are two nodes or vertical attachments if they connect with each other. The following example is near B, similar to the CB, and so on.

• **The path :**represents the setting of the edges between the two vessels. In the



following example, ABCDA represents the path of D.

Basic Operations

Some basic operations that can be performed on Graph Data Structure:

- **Add Vertex** – Adding a new vertex to the Graph.
- **Add Edge** – Creating an edge between two vertices in the Graph.
- **Display Vertex** – Get a vertex of the Graph.

Breadth First Search (BFS)

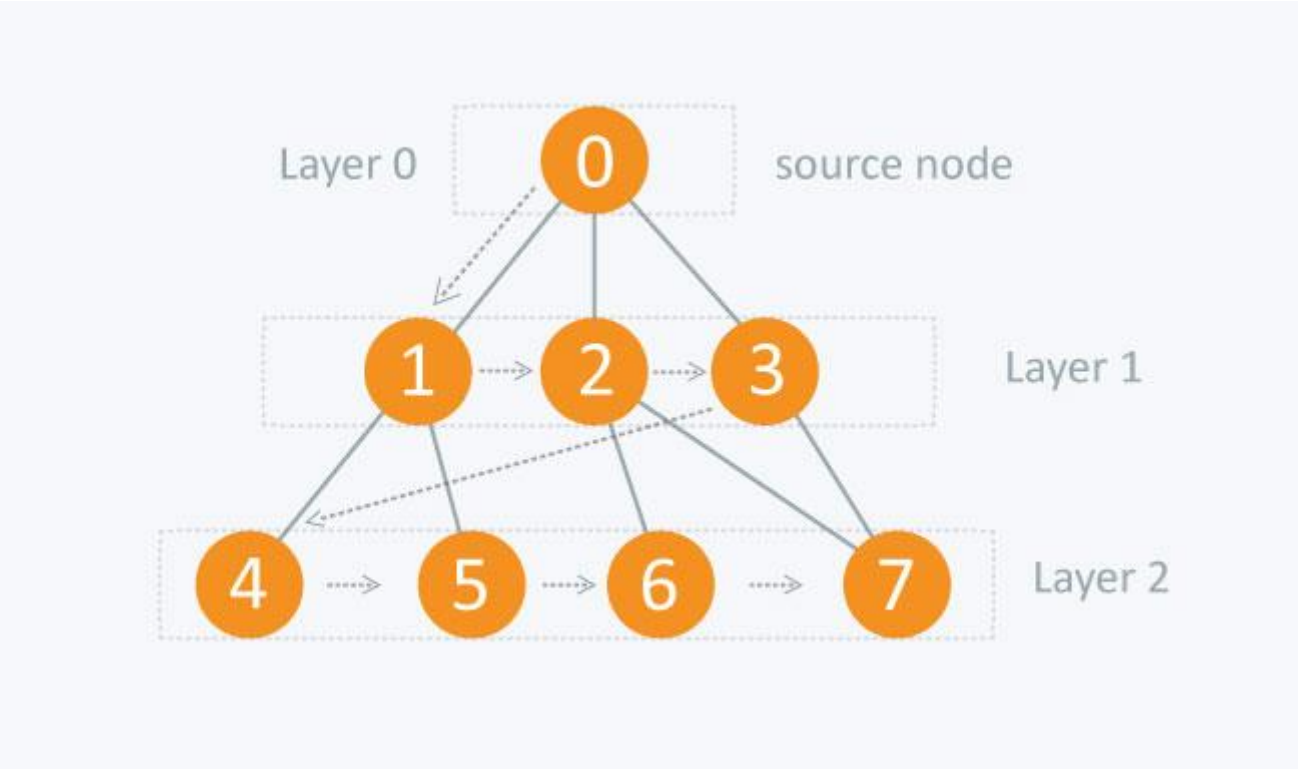
There are many ways to traverse the graphs. The BFS is the most used approach.

BFS is a traversal algorithm where you should start the process from the selected node (source or start) and thus lamp the graph in search of neighboring nodes (nodes that are directly linked to source source). You must move to the next level neighbor's nodes.

As BFS is known, you need to spread the above graphs.

1. First move horizontally and see all nodes of current layer

2. Move to the next layer



ALGORITHM:

```
BFS (G, s)

      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour
vertices are marked.

      mark s as visited.
      while ( Q is not empty)
           //Removing that vertex from queue,whose neighbour will be
visited now
           v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                        Q.enqueue( w )            //Stores w in Q to
further visit its neighbour
                        mark w as visited.
```

Depth First Search (DFS)

The DFS algorithm is a recursive algorithm which uses the backtracking idea. It includes exhaustive search of all nodes by moving ahead, if possible, and then by backtracking.

Here, the backtracking word is that when you are moving forward and do not have more nodes with the current route, you move Nodes to the same path. All nodes will be visited on the current route as long as all unexpected nodes have been replaced, which will be selected next.

ALGORITHM:

```
S-iterative (G, s):

       let S be stack
       S.push( s )            //Inserting s in stack
       mark s as visited.
       while ( S is not empty):
           //Pop a vertex from stack to visit next
           v  =  S.top( )
          S.pop( )
          //Push all the neighbours of v in stack that are not visited
         for all neighbours w of v in Graph G:
             if w is not visited :
                      S.push( w )
                     mark w as visited
```

```
DFS-recursive(G, s):
    mark s as visited
    for all neighbours w of s in Graph G:
        if w is not visited:
            DFS-recursive(G, w)
```

Common applications of Graphs :

Graphs can be used to deal with many kinds of relationships and processes in physical, biological, social and informative systems. Many practical issues can be represented by graphics. Strengthening their application on real-world systems, the term of the network sometimes means a graph in which attributes (eg names) are connected to nodes and / or edges.

Used in computer science, graphics communication, data organization, computing devices etc. etc. is used to represent flow flow. For example, a website's link structure can be represented by a graph representing the representation of vertical web pages. And represents links from one page to another. Therefore, the development of algorithm to handle graphics is of great interest in computer science. The graph change is often presented in a form of representation and represents graph rewriting system. Developed by graphic governance-based memory harassment, graph graphs for graph conversion systems are developed by drawing graphic secure, continuous storage and drawing questions.

4.8 HEAP DATA STRUCTURE

If we consider special case for the B.S.T. data structure then Heap is one of them. In this value of root node is equated with its child and then presented in order as per the result.

If **b** is child of **a−**

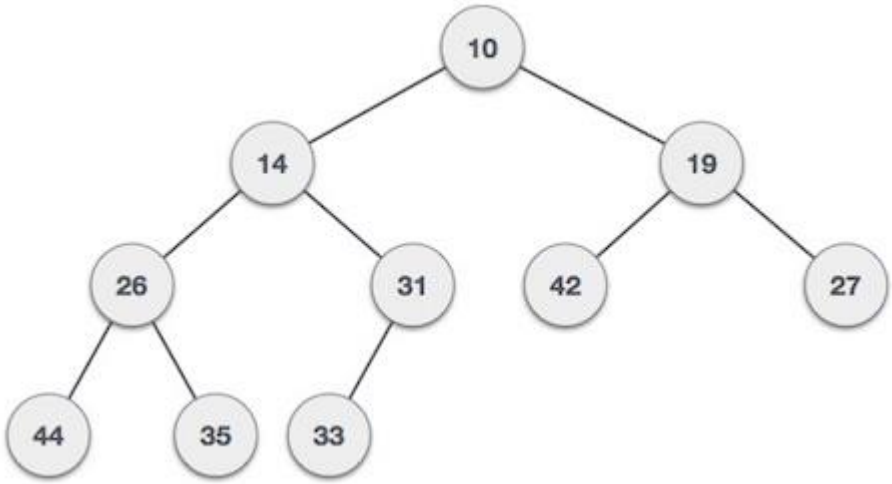
Value(a) ≥ Value(b)

Now the value of parent is more to the child , **Max Heap** is generated by this property.

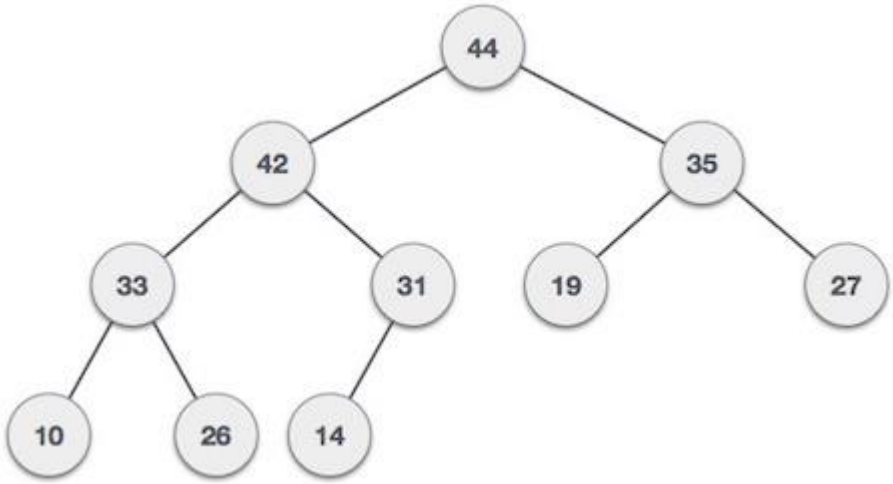
There are 2 ways that a heap be defined -

INPUT -> “35 33 42 10 14 19 27 44 26 31”

Min-Heap – The vaule of the parent nodes is greater than or equal to the value of their child.



Max-Heap − The vaule of the parent nodes is less than or equal to the value of their child..



Max Heap Construction Algorithm

Maximum and Minimum heap are generated in a same way but in this we look for min values rather than max values.

This unique property must be maintained by heap at any instance of time. At the time of inserting element, keep in mind that a node is being inserted to a tree which is already heapified.

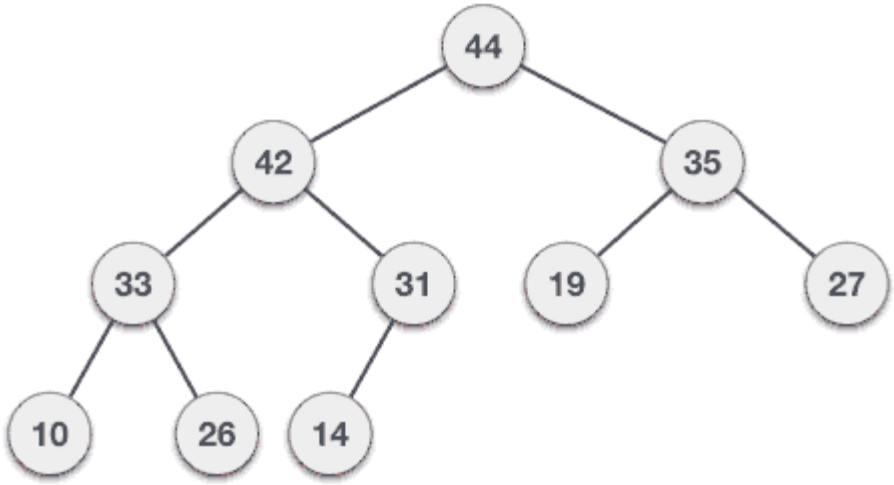
```
Step 1 - Create a new node at the end of heap.
Step 2 - Assign new value to the node.
Step 3 - Compare the value of this child node with its parent.
Step 4 - If value of parent is less than child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.
```

The value of parent node must be less than or equal to its children,if we are considering the min heap.

Max Heap Deletion Algorithm

Here is an algorithm to perform delete operation on a max heap. Always the root node is deleated in a heap and then the heap is rearranged to maintain the properties of the heap.

```
Step 1 - Remove root node.
Step 2 - Move the last element of last level to root.
Step 3 - Compare the value of this child node with its parent.
Step 4 - If value of parent is less than child, then swap them.
Step 5 - Repeat step 3 & 4 until Heap property holds.
```



Common applications of Heaps:

Some basic applications of Heap Data Structure are:

- Heapsort: Best sorting algorithm with no quadratic worst cases.
- Selection algorithms:Heap finds the minimum and maximum element in a set in constant time, and other search operations like middle or kth-element can be done in sub-linear time for the elements that are stored in a heap.
- Graph algorithms: By using heaps as internal traversal data structures, run time can be reduced by polynomial order. Examples of such problems are Prim's minimal-spanning-tree algorithm and Dijkstra's shortest-path algorithm.
- Priority Queue: A priority queue can be efficiently implemented with the help of heaps.
- Order statistics: The Heap data structure can be used to find smallest, largest, middle or k-th element in linear or sub-linear time.

CHAPTER 5

CONCLUSIONS

5.1 Conclusions

Data Structures is not just limited to common Data Structure like Stack, Queue, LinkedLists and Trees, but is quite a vast area. There are many more data structures which include Maps, Hash Tables, Graphs, Trees, etc. Each data structure has its own pros and cons so they should be used according to the needs of the application. Many high level and object oriented programming languages like C#, Java, Python come built in with many of these data structures.

The proposed Toolkit will keep the well organised Data Structures in a common Directory which can be incorporated in large projects so that these codes can be reused while writing the codes of the projects so that the code of the complex large project can remain clean and can relieve the programmer of the hassle of writing the same basic code again and again. This would tremendously help in speeding up the development of Python based real-life projects . Due to the increased popularity of the Python programming language in the past two years it the proposed Toolkit would make sure that the Python programming language can be used extensively for the development of real-life working projects.

5.2 Future Scope

In the coming time this Toolkit can be implemented to comprise these mentioned points :

- This Toolkit has no provision for data visualization. However the usefulness of the Toolkit would have increased two fold if it is transforming information into graphical data.
- There is no documentation that is provided by this toolkit to select a specific data structure for a specific task. This Toolkit just provide typical use case examples of Data Structures enlisted in this project report. However it is very important to keep the application in mind while we are choosing a particular *data structure* to find the solution for specific *problem*. Only by first analyzing the issue to determine the performance goals that must be achieved can there be any hope of choosing the suitable data structure for the job. Poor program designers ignore this analysis step.

REFERENCES

[1] Wang Z. (2012) The Research on Teaching Ideas of ”Data Structure and Algorithm” in Non-computer Major. In: Xie A., Huang X. (eds) Advances in Computer Science and Education. Advances in Intelligent and Soft Computing, vol 140. Springer, Berlin, Heidelberg

[2] Dive into python 3, Mark Pilgrim

[3] Think Python ,2nd Edition,Allen B. Downey

[4] Algorithm Design ,Jon Kleinberg and Eva Tardos Pearson(2013)

[5] https://www.stat.washington.edu/~hoytak/_static/papers/why-python.pdf

[6] “Introduction to Algorithms”, 3rd edition, Thomas H. Cormen et. Al.

[7] “15-121 Introduction to Data Structures”, Carnegie Mellon University-CORTINA

[8] https://www.python.org/doc/