Data Structures

DATAMATIKER-B

2020-11-30 LYNGBY

Agenda

Recap

Data Structures Intro

Hashmap

Linked Lists (+ Doubly Linked Lists & Circular Linked Lists)

Stacks

Queues

Graphs

Exercises

Terminology

Abstract Data Type (**ADT**)
Base Class Library (**BCL**) Types.

Big-O Notation:

- O(1) means in constant time - independent of the number of items.
- O(N) means in proportion to the number of items.
- O(log N) means a time proportional to log(N)

https://en.wikipedia.org/wiki/Big_O_notation

Data Structures

Data Structures

Basic Terminology

Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned

Data: Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

Group Items: Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

File: A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

Attribute and Entity: An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.

Field: Field is a single elementary unit of information representing the attribute of an entity.

Data Structures

Need of Data Structures

As applications are getting complexed and amount of data is increasing day by day, there may arrise the following problems:

Processor speed: To handle very large amout of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

Data Search: Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

Multiple requests: If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process

in order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

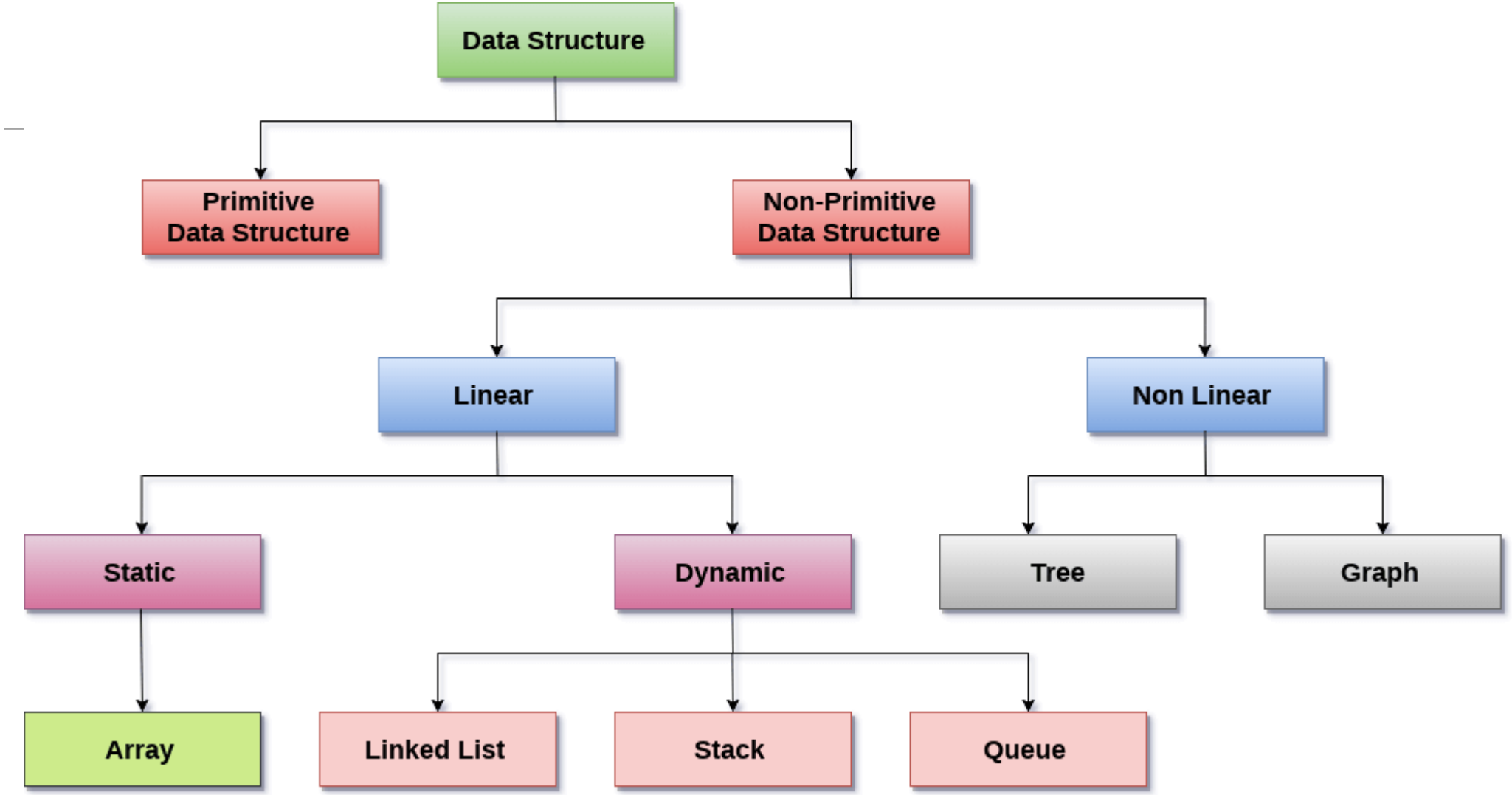
Data Structures

Advantages of Data Structures

Efficiency: Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data and we need to perform the search for a perticular record. In that case, if we organize our data in an array, we will have to search sequentially element by element. hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

Reusability: Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

Abstraction: Data structure is specified by the ADT which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.


Data Structure
Primitive Data Structure
Non-Primitive Data Structure
Linear
Non Linear
Static
Dynamic
Tree
Graph
Array
Linked List
Stack
Queue

Operations on data structure

1) **Traversing:** Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

Example: If we need to calculate the average of the marks obtained by a student in 6 different subject, we need to traverse the complete array of marks and calculate the total sum, then we will devide that sum by the number of subjects i.e. 6, in order to find the average.

2) **Insertion:** Insertion can be defined as the process of adding the elements to the data structure at any location.

If the size of data structure is **n** then we can only insert **n-1** data elements into it.

3) **Deletion:**The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.

If we try to delete an element from an empty data structure then **underflow** occurs.

4) **Searching:** The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search. We will discuss each one of them later in this tutorial.

5) **Sorting:** The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) **Merging:** When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging

Hvad er Hash?

An MD5 hash is NOT encryption. It is simply a fingerprint of the given input. However, it is a one-way transaction and as such it is almost impossible to reverse engineer an MD5 hash to retrieve the original string.

- https://www.md5hashgenerator.com/

- https://virustotal.com

HashSets & HashMaps

HashSet

A HashSet is a collection of items where every item is unique, and it is found in the java.util package:

Let’s try it!

Hashmaps

A HashMap, store items in "key/value" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values

Let’s try it!

https://www.javatpoint.com/java-hashmap

Linked Lists

Linked Lists

Types of Linked List

Following are the various types of linked list.

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists

Basic Operations

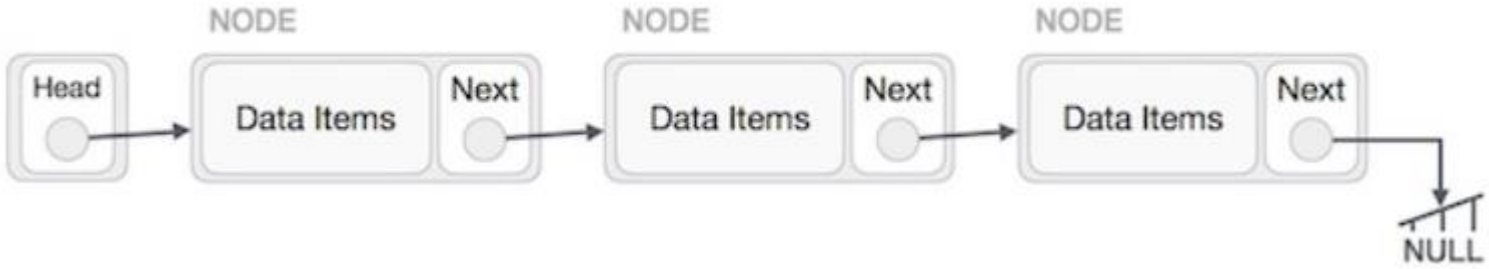
Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists

Linked list can be visualized as a chain of nodes, where every node points to the next node.

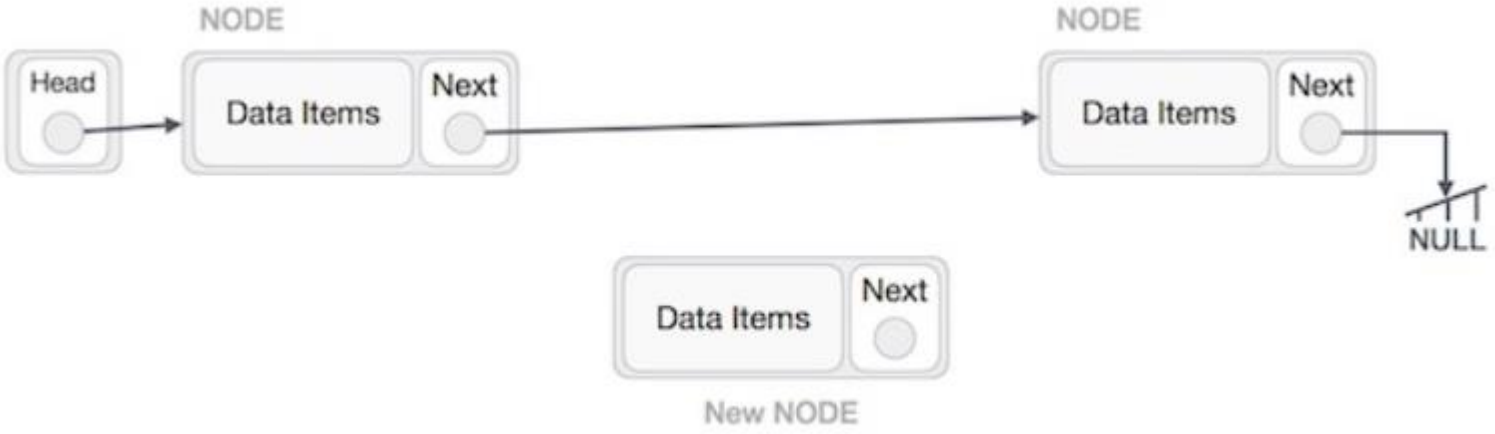


Summary `ArrayList` with `ArrayDeque` are preferable in *many* more use-cases than `LinkedList`. If you're not sure — just start with `ArrayList`.

https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

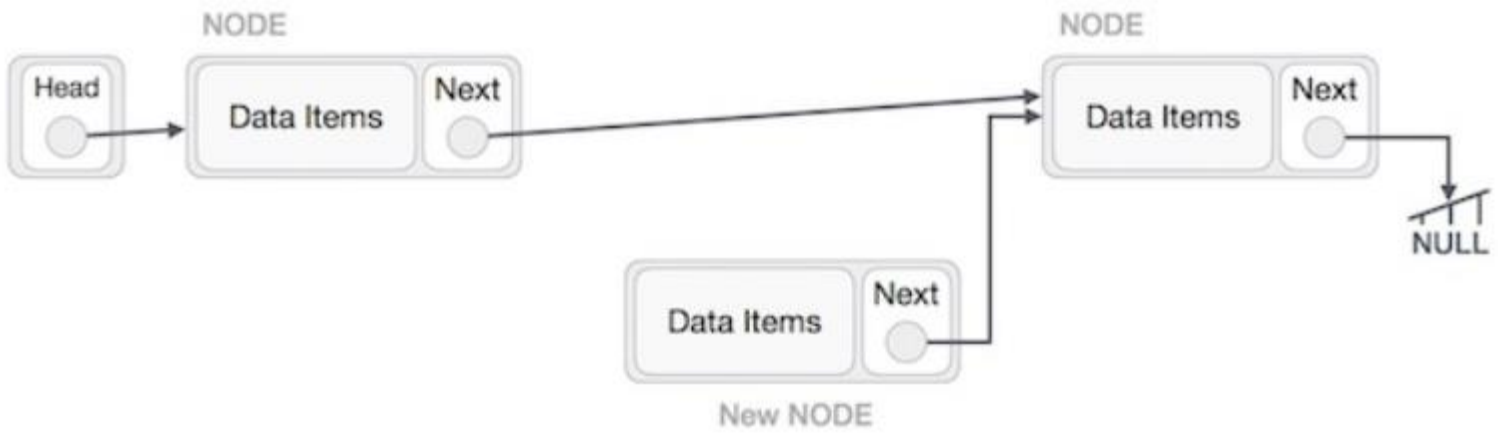
Linked Lists – Insert operation

Linked Lists - Insert



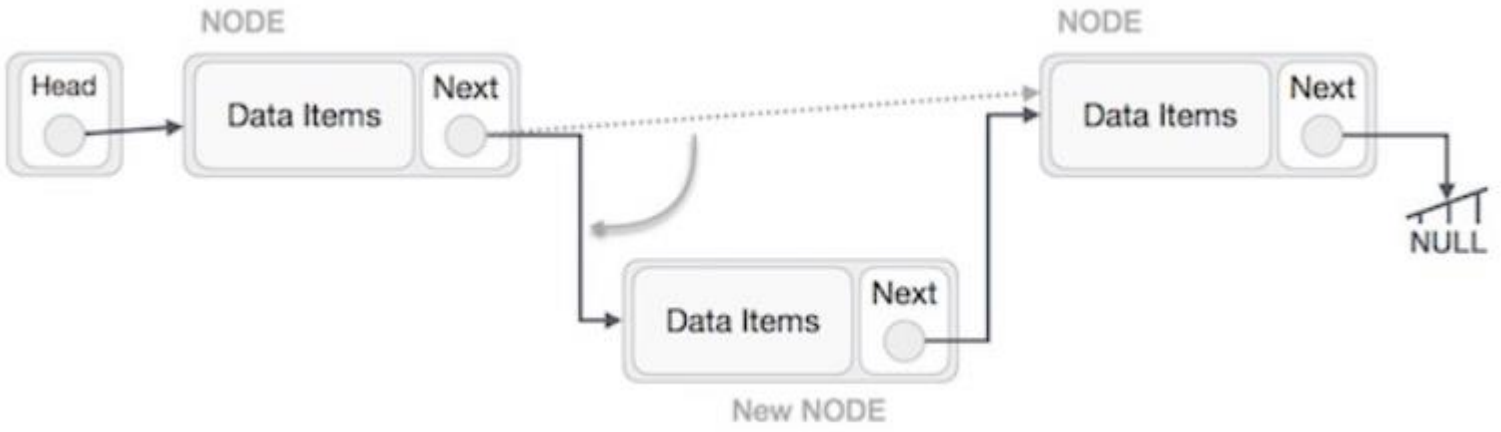
https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists - Insert



https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists - Insert



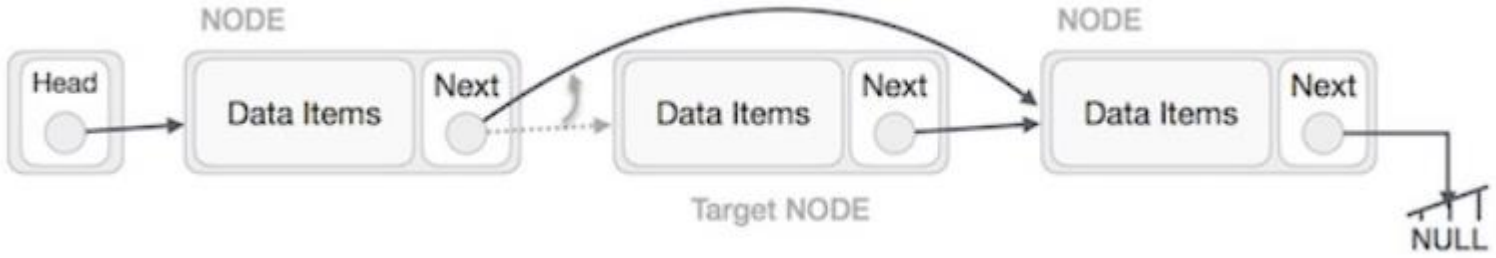
https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists – Delete operation

Linked Lists - Delete

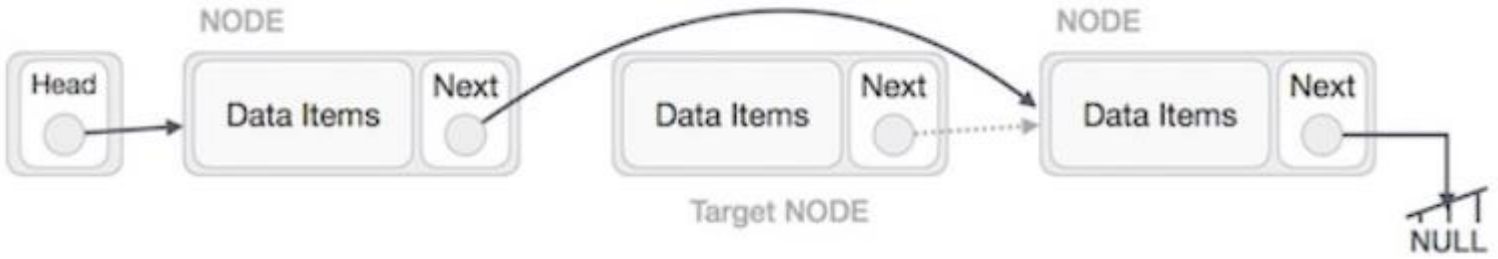
https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists - Delete



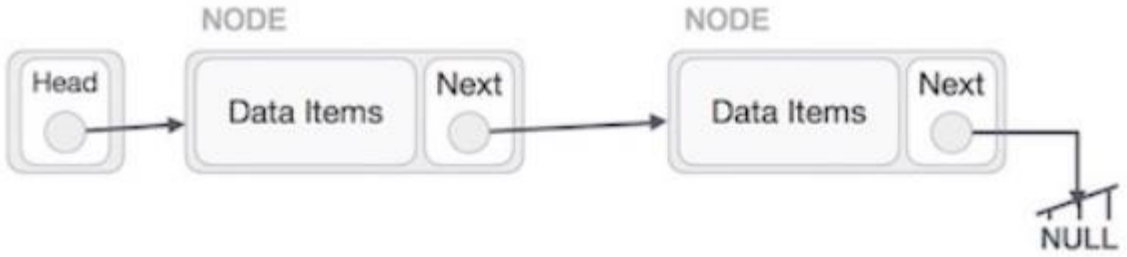
https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists - Delete



https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Linked Lists - Delete



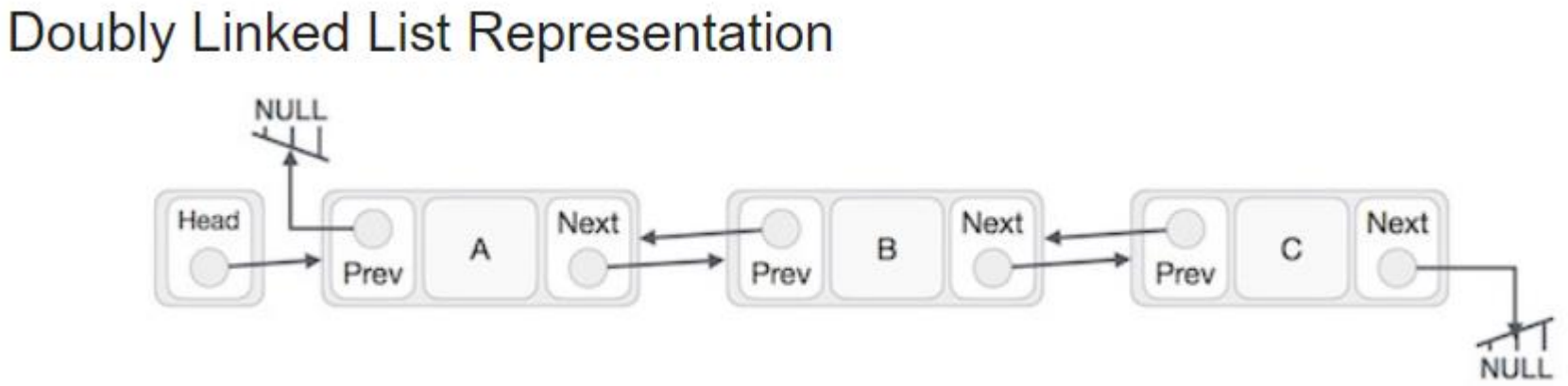
https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

Doubly Linked Lists

Doubly Linked Lists

- The browser cache which allows you to hit the BACK button (a linked list of URLs)

- Undo functionality in Photoshop or Word (a linked list of state)



https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm

Doubly Linked List

Basic Operations

Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Insert Last** − Adds an element at the end of the list.
- **Delete Last** − Deletes an element from the end of the list.
- **Insert After** − Adds an element after an item of the list.
- **Delete** − Deletes an element from the list using the key.
- **Display forward** − Displays the complete list in a forward manner.
- **Display backward** − Displays the complete list in a backward manner.

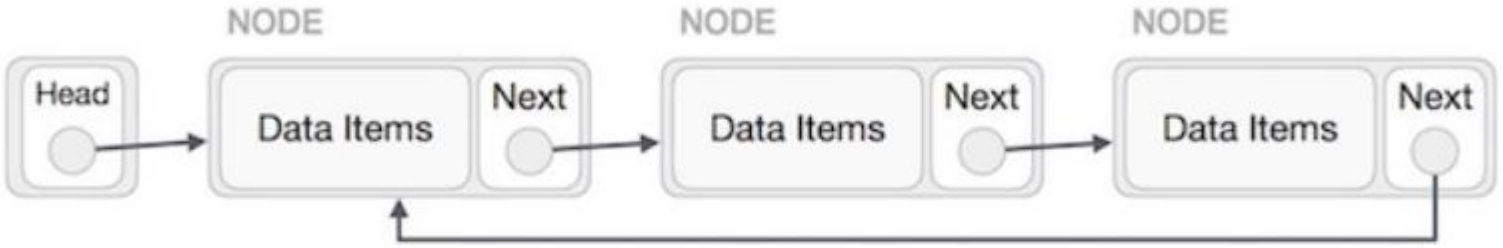
https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm

Circular Linked Lists

Circular Linked Lists

Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.



Doubly Linked List as Circular

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.

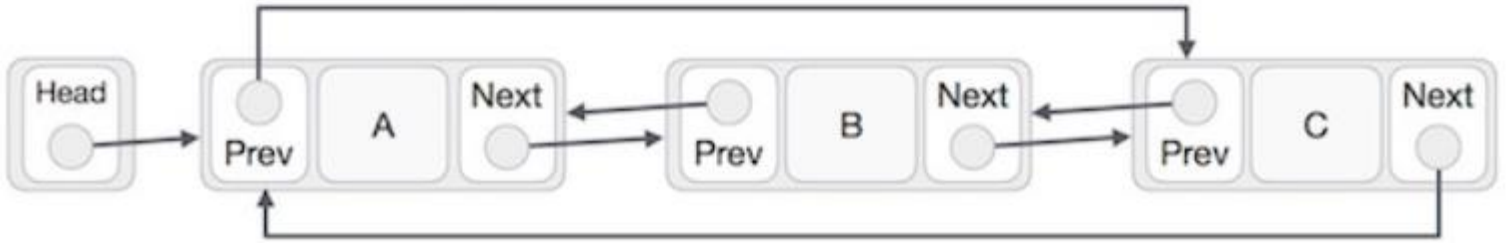


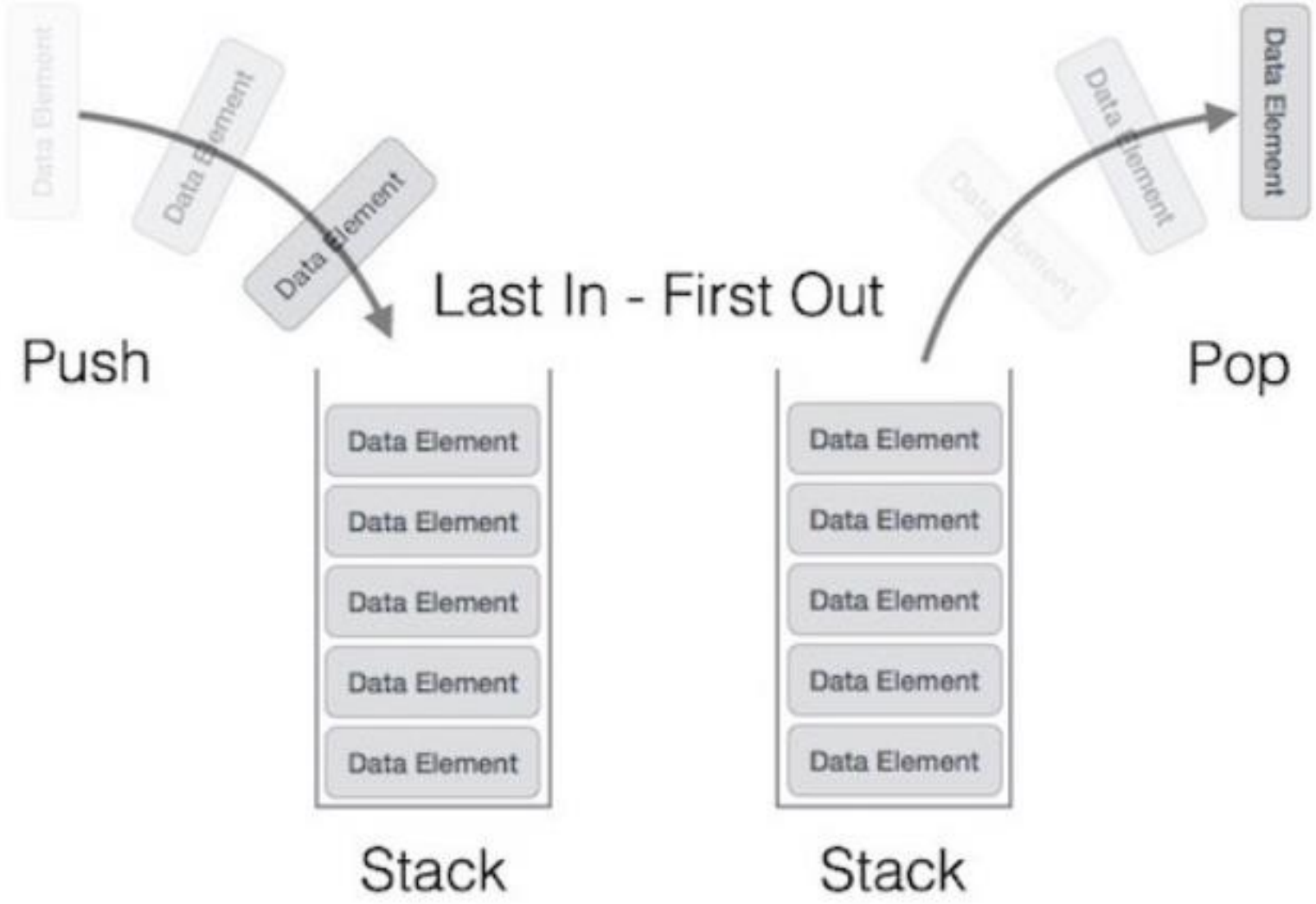
Image viewer, Music Player, other examples?

Pause

Stacks

Stack Representation

The following diagram depicts a stack and its operations −



https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Stacks

Applications of stack:

- Balancing of symbols
- Infix to Postfix /Prefix conversion
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- Backtracking is one of the algorithm designing technique .Some example of back tracking are Knight-Tour problem,N-Queen problem,find your way through maze and game like chess or checkers in all this problems we dive into someway if that way is not efficient we come back to the previous state and go into some another path. To get back from current state we need to store the previous state for that purpose we need stack.
- In Graph Algorithms like Topological Sorting and Strongly Connected Components
- In Memory management any modern computer uses stack as the primary-management for a running purpose.Each program that is running in a computer system has its own memory allocations
- String reversal is also a another application of stack.Here one by one each character get inserted into the stack.So the first character of string is on the bottom of the stack and the last element of string is on the top of stack. After Performing the pop operations on stack we get string in reverse order .

https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Stacks

Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Stacks

Implementation:

There are two ways to implement a stack:

- Using array
- Using linked list

Practice (requires login):

https://practice.geeksforgeeks.org/problems/implement-stack-using-array/1

Solution (please do practice first):

https://www.geeksforgeeks.org/stack-data-structure-introduction-program/

https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Queues

Queues

https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm

Queues

Basic Operations

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues −

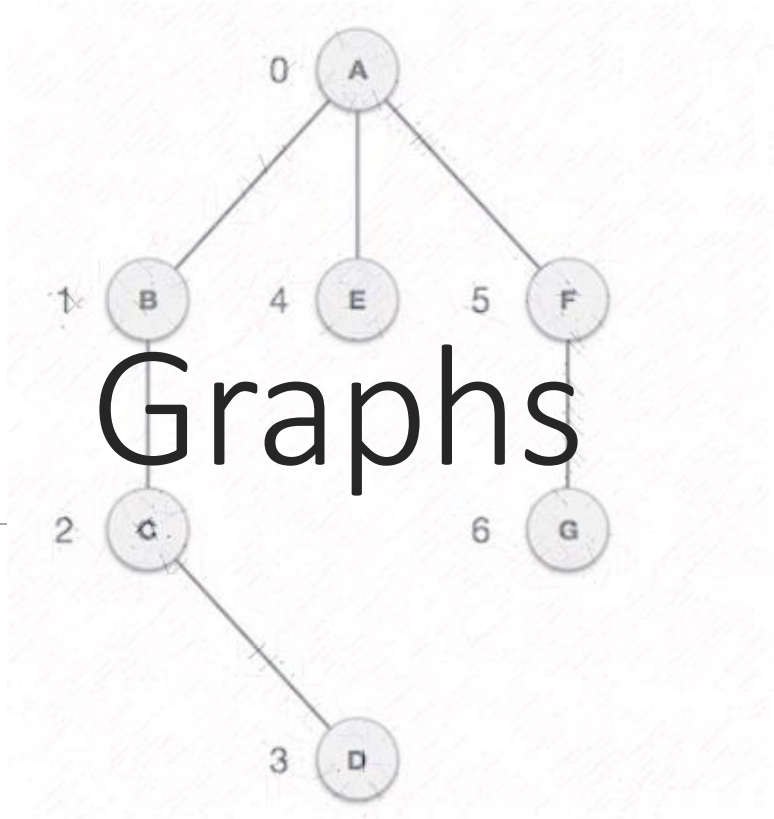
- **enqueue()** − add (store) an item to the queue.
- **dequeue()** − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are −

- **peek()** − Gets the element at the front of the queue without removing it.
- **isfull()** − Checks if the queue is full.
- **isempty()** − Checks if the queue is empty.

https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm

Graphs



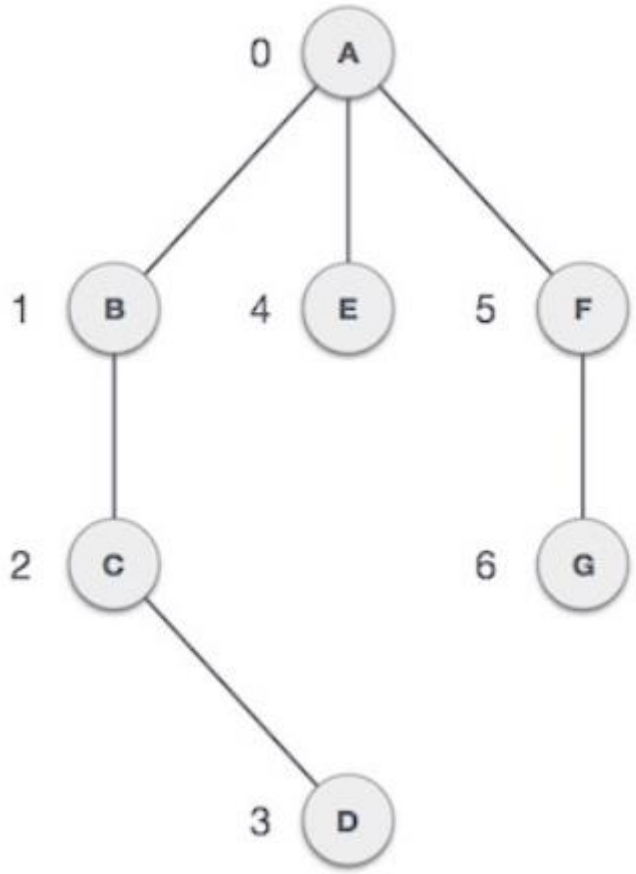
Graphs

Graph Data Structure

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms −

- **Vertex** − Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
- **Edge** − Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
- **Adjacency** − Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
- **Path** − Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.

Graphs



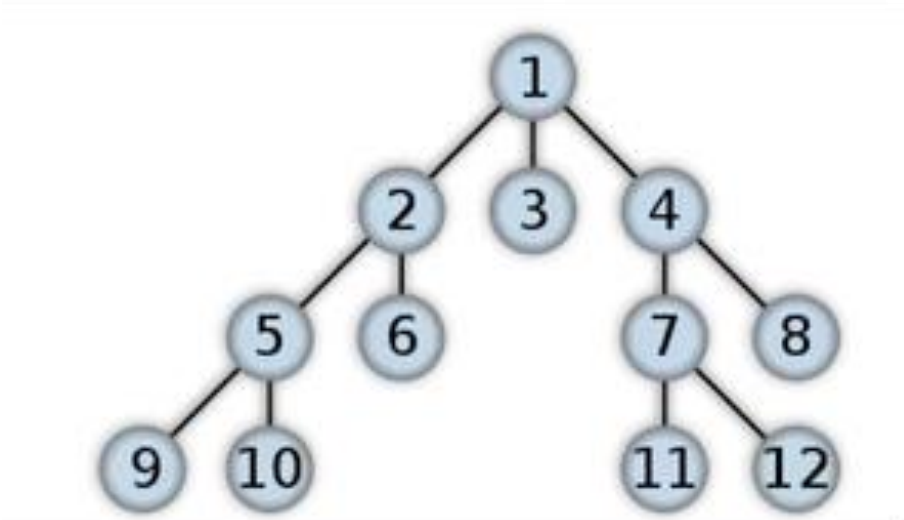
Basic Operations

Following are basic primary operations of a Graph −

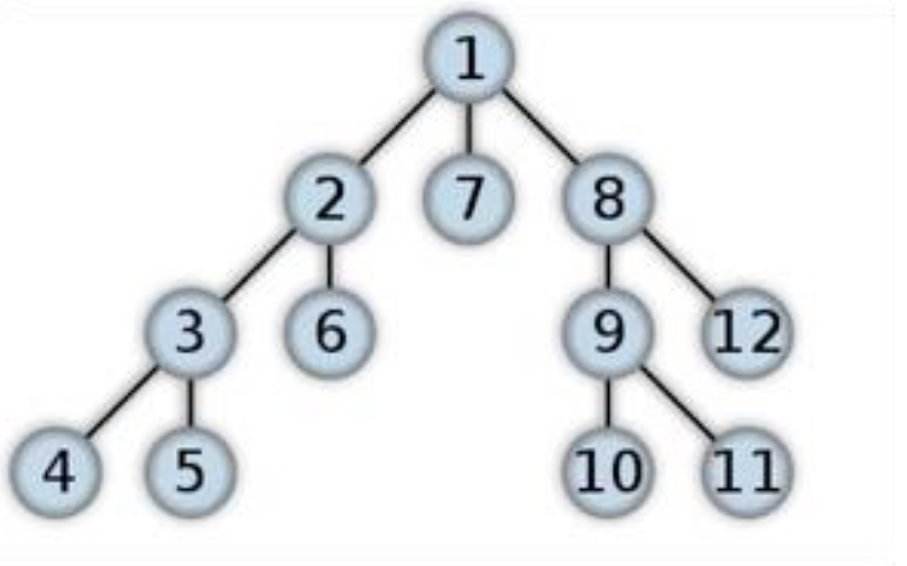
- **Add Vertex** − Adds a vertex to the graph.
- **Add Edge** − Adds an edge between the two vertices of the graph.
- **Display Vertex** − Displays a vertex of the graph.

Graphs –Traversal

BFS



DFS



Pause

Exercises

Exercises: Quick note

Alle svar kan findes på google. Lad være!

Det her er ikke noget I skal aflevere, så det er udelukkende for, at I bliver dygtigere.

Det er selvfølgelig ikke sjovt at sidde fast, men så snak hellere i gruppen om det eller kald på mig ☺

Exercises part 1

Implementér en hjemmelavet LinkedList. Print derefter alle elementer i den.

Du kan starte med at lave en Node.java som indeholder:

```
String data;
Node next;
```

Dernæst kan du lave en LinkedList.java, som har en addNewNode(String data) metode. Din LinkedList bør have en member variable der holder styr på den nuværende node.

Exercises part 2

Implementér nu en doubly linkedlist.

Du bør udvide din Node.java til også at have en reference til forrige Node:

```
Node prev;
```

Dernæst kan du lave en DoublyLinkedList.java, som gør præcis det samme som LinkedList.java, men også tager højde for previous node.

Exercises part 3

Her kan du vælge mellem følgende:

- en ny data struktur at implementere.
- Lav en musikafspiller ved brug af linkedlist og/eller queues
- Holde fri med god samvittighed, fordi du er pisse nice og nåede hertil.

Afrunding

I morgen fortsætter vi med Data Strukturer.

Vi samler op på hvad vi har været igennem i dag og kører derefter videre med Binary Trees.