SNOWSCRIPT WINTER OF CODE (SWOC)
DOCUMENTATION ON DSA (LINKED LIST)

The Data Structure and Algorithm (DSA) forms the fundamental foundation of computer science and facilitates the efficient organization, storage and processing of data. This document will dive into one of the basic data structures: the Linked List. The study covers the theoretical knowledge, algorithms, syntax, and practical workings of linked lists, contributing to a thorough understanding of their application and effectiveness in various computing scenarios.

Introduction:

- Data structures are ways of organizing and storing data on a computer so that it can be accessed and modified efficiently. Different DS are suitable for different types of applications, and some are highly specialized for specific tasks. Among them, linked lists stand out due to their dynamic nature and flexibility in memory allocation.

Theoretical overview:

- A linked list is a linear collection of data elements, called nodes, each of which points to another node using a pointer. It is a data structure consisting of a collection of nodes that together represent a sequence.

Types of linked lists

- Singly Linked List: Consists of nodes that are unidirectionally linked.
- Doubly linked list: Nodes have two links, forward and backward.
- Circular Linked List: The last node points back to the first node.

An extended study on linked lists:

Singly linked list

- A singly linked list is a type of linked list that is one-way, that is, it can only be traversed in one direction from the head to the last node (end). Each element (node) contains data and a reference to the next node in the list.

Characteristics

- A simple structure with nodes connected in one direction.
- Memory efficient as it stores only one reference per node.

Example

- Consider a simply linked list with three nodes containing the values 1, 2, and 3.
    [1] -> [2] -> [3] -> null

Doubly linked list

- A doubly linked list is similar to a singly linked list, but with an additional link: each node contains a link to the next node and a link to the previous node.

Characteristics

- Nodes are connected in both directions, making it easy to pass back.
- It requires more memory per node (two references: next and previous).

Example

- A doubly linked list with nodes containing the values 1, 2, and 3:
  null <- [1] <-> [2] <-> [3] -> null

Circular linked list

- In a circular linked list, the last node does not point to null, but points back to the first node, forming a circle.

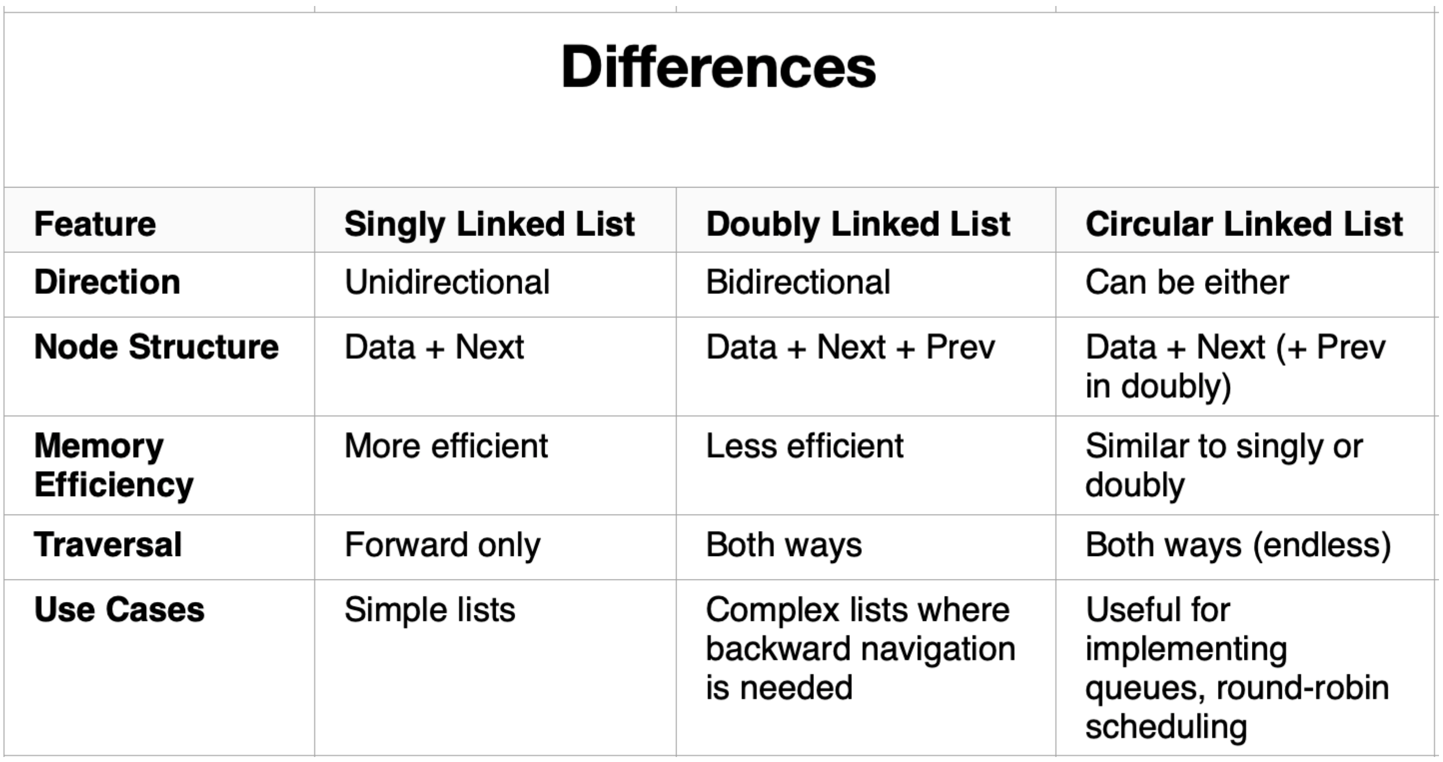
Characteristics

- They can be either simply circular or double circular linked lists.
- There is no "end" to the list because the last node points back to the first.

Example

- A circular singly linked list with three nodes:

[1] -> [2] -> [3] -> [1]



Differences

Feature	Singly Linked List	Doubly Linked List	Circular Linked List
Direction	Unidirectional	Bidirectional	Can be either
Node Structure	Data + Next	Data + Next + Prev	Data + Next (+ Prev in doubly)
Memory Efficiency	More efficient	Less efficient	Similar to singly or doubly
Traversal	Forward only	Both ways	Both ways (endless)
Use Cases	Simple lists	Complex lists where backward navigation is needed	Useful for implementing queues, round-robin scheduling

Work and applications

- In a linked list, each element is contained in a separate object called a "node". These nodes contain data as well as a link to the next node in the list. This structure allows efficient insertion and removal of elements from any position in the list.

Applications of linked lists include:

- Implementation of stacks and queues.
- Dynamic memory allocation.
- Implementation of graphs.

Conclusion

- Linked lists are fundamental to the study of data structures and provide a critical understanding of dynamic memory management. Each type of linked list offers unique benefits and is suitable for different scenarios. Singularly linked lists are simpler and more memory efficient, but lack flexibility in traversal. Doubly linked lists offer easier navigation at the cost of increased memory usage. Circular linked lists, on the other hand, are particularly useful in scenarios where you need a circular, infinite list structure, such as buffering data streams or implementing certain types of resource scheduling. Their flexibility and efficiency in insert and delete operations make them a valuable tool in a variety of programming scenarios.

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.
- Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley Professional.