DS’s Applications

Contents

[Linked List 2](#_Toc475629119)

[Circular Linked List 2](#_Toc475629120)

[Queue 2](#_Toc475629121)

[Deque or Double Ended Queue 2](#_Toc475629122)

[Hash Table 2](#_Toc475629123)

[Tree 2](#_Toc475629124)

[Depth First Search 2](#_Toc475629125)

[Breadth First Traversal 2](#_Toc475629126)

[Heap 2](#_Toc475629127)

# Linked List

**Advantages over arrays**

1. Dynamic size
2. Ease of insertion/deletion

**Drawbacks:**

1. Random access is not allowed. We have to access elements sequentially starting from the first node. Can’t do binary search.
2. Extra memory space for a pointer is required with each element of the list.

Application:

1. Useful for implementation of queue. Two pointers rear and front are maintained for queue ends.

## Circular Linked List

1. Useful for implementation of queue. Only one pointer (last inserted node) is required for rear, head of queue will be rear next.
2. Circular lists are useful in applications to **repeatedly go around the list**. For example, when **multiple applications are running on a PC**, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
3. Used to make observer pattern.
4. Used for implementation of advanced data structures like [Fibonacci Heap](http://en.wikipedia.org/wiki/Fibonacci_heap).

# Stack

LIFO (**Last In First Out**) or FILO (**First In Last Out**)

**Array** is best choice for implementation.

Basic operations:

1. **Push**: Adds an item in the stack.
2. **Pop**: Removes an item from the stack.
3. **Peek**: Get the topmost item.

Application:

1. **Arithmetic expression evaluation**
2. **Infix notation:**X + Y – Operators are written in-between their operands. This is the usual way we **write expressions**.

A \* ( B + C ) / D

1. **Postfix notation (also known as “Reverse Polish notation”):**X Y + Operators are written after their operands.

A B C + \* D/

1. **Prefix notation (also known as “Polish notation”):**+ X Y Operators are written before their operands

/ \* A + B C D

1. **Tower of Hanoi**
2. **Function-call abstraction**
3. **Load balancing. Simulate the process of assigning n items to a set of m servers. For each item, it chooses a sample of s servers and assigns the item to the server that has the fewest current items.**
4. **Parentheses Balanced: as IDE have**

# Queue

First In First Out **(FIFO)**

**Linked Lists** are best choice for implementation.

Basic operations:

1. **Enqueue:** Adds an item to the queue.
2. **Dequeue**: Removes an item from the queue.
3. **Front:** Get the front item from queue.
4. **Rear**: Get the last item from queue.

Application:

This property of Queue makes it also useful in following kind of scenarios

1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

## Deque or Double Ended Queue

Deque supports both stack and queue operations, it can be used as both.

**Doubly linked list** or **circular array** is best way to implement.

Basic operations: O(1)

1. **insetFront**: Adds an item at the front of Deque.
2. **insertLast**: Adds an item at the rear of Deque.
3. **deleteFront**: Deletes an item from front of Deque.
4. **deleteLast**: Deletes an item from rear of Deque.
5. **getFront**: Gets the front item from queue.
6. **getRear**: Gets the last item from queue.
7. **isEmpty**: Checks whether Deque is empty or not.
8. **isFull**: Checks whether Deque is full or not.

C++ STL [std::deque](http://www.cplusplus.com/reference/deque/deque/) and Java provides [Deque interface](http://docs.oracle.com/javase/7/docs/api/java/util/Deque.html" \t "_blank).

# Hash Table

# Tree

http://www.geeksforgeeks.org/applications-of-tree-data-structure/

## Depth First Search

http://www.geeksforgeeks.org/applications-of-depth-first-search/

## Breadth First Traversal

http://www.geeksforgeeks.org/applications-of-breadth-first-traversal/

# Heap

# Graph