**2.1.5**

# What is a priority queue?

A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.

The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.

For example, suppose we have some values like 1, 3, 4, 8, 14, 22 inserted in a priority queue with an ordering imposed on the values is from least to the greatest. Therefore, the 1 number would be having the highest priority while 22 will be having the lowest priority.

### Characteristics of a Priority queue

A priority queue is an extension of a queue that contains the following characteristics:

* Every element in a priority queue has some priority associated with it.
* An element with the higher priority will be deleted before the deletion of the lesser priority.
* If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle.

**Let's understand the priority queue through an example.**

We have a priority queue that contains the following values:

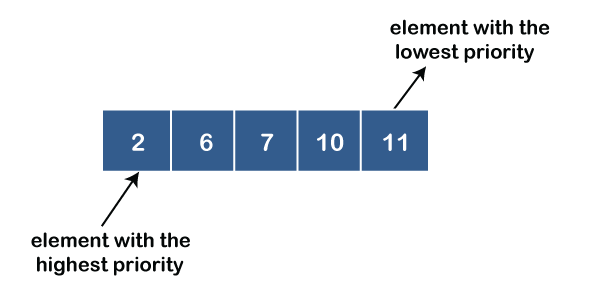
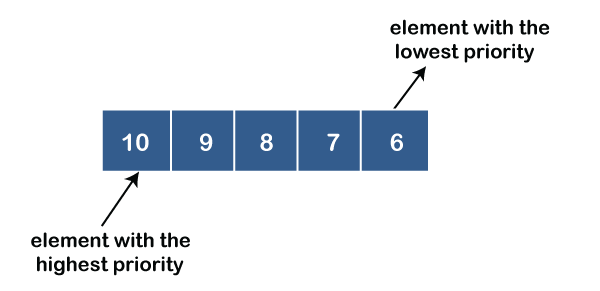
**1, 3, 4, 8, 14, 22**

All the values are arranged in ascending order. Now, we will observe how the priority queue will look after performing the following operations:

* **poll():** This function will remove the highest priority element from the priority queue. In the above priority queue, the '1' element has the highest priority, so it will be removed from the priority queue.
* **add(2):** This function will insert '2' element in a priority queue. As 2 is the smallest element among all the numbers so it will obtain the highest priority.
* **poll():** It will remove '2' element from the priority queue as it has the highest priority queue.
* **add(5):** It will insert 5 element after 4 as 5 is larger than 4 and lesser than 8, so it will obtain the third highest priority in a priority queue.

### Types of Priority Queue

**There are two types of priority queue:**

* **Ascending order priority queue:** In ascending order priority queue, a lower priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in an ascending order like 1,2,3,4,5; therefore, the smallest number, i.e., 1 is given as the highest priority in a priority queue.  
  
* **Descending order priority queue:** In descending order priority queue, a higher priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in descending order like 5, 4, 3, 2, 1; therefore, the largest number, i.e., 5 is given as the highest priority in a priority queue.
* 

### Representation of priority queue

Now, we will see how to represent the priority queue through a one-way list.

We will create the priority queue by using the list given below in which **INFO** list contains the data elements, **PRN** list contains the priority numbers of each data element available in the **INFO** list, and LINK basically contains the address of the next node.



**Let's create the priority queue step by step.**

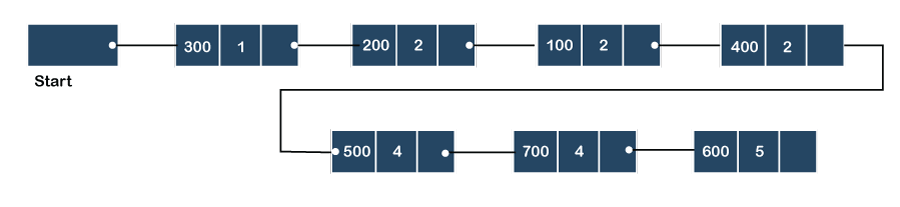
**In the case of priority queue, lower priority number is considered the higher priority, i.e.,** lower priority number = higher priority.

**Step 1:** In the list, lower priority number is 1, whose data value is 333, so it will be inserted in the list as shown in the below diagram:

**Step 2:** After inserting 333, priority number 2 is having a higher priority, and data values associated with this priority are 222 and 111. So, this data will be inserted based on the FIFO principle; therefore 222 will be added first and then 111.

**Step 3:** After inserting the elements of priority 2, the next higher priority number is 4 and data elements associated with 4 priority numbers are 444, 555, 777. In this case, elements would be inserted based on the FIFO principle; therefore, 444 will be added first, then 555, and then 777.

**Step 4:** After inserting the elements of priority 4, the next higher priority number is 5, and the value associated with priority 5 is 666, so it will be inserted at the end of the queue.



### Implementation of Priority Queue

The priority queue can be implemented in four ways that include arrays, linked list, heap data structure and binary search tree. The heap data structure is the most efficient way of implementing the priority queue, so we will implement the priority queue using a heap data structure in this topic. Now, first we understand the reason why heap is the most efficient way among all the other data structures.

**What is a Graph?**

A graph is a **unique data structure** in programming that consists of finite sets of nodes or vertices and a set of edges that connect these vertices to them. At this moment, adjacent vertices can be called those vertices that are connected to the same edge with each other. In simple terms, a graph is a visual representation of vertices and edges sharing some connection or relationship. Although there are plenty of graph algorithms that you might have been familiar with, only some of them are put to use. The reason for this is simple as the standard graph algorithms are designed in such a way to solve millions of problems with just a few lines of logically coded technique. To some extent, one perfect algorithm is solely optimized to achieve such efficient results.

**Types of Graphs**

There are various types of graph algorithms that you would be looking at in this article but before that, let's look at some types of terms to imply the fundamental variations between them.

**Order:** Order defines the total number of vertices present in the graph.

**Size:** Size defines the number of edges present in the graph.

**Self-loop:** It is the edges that are connected from a vertex to itself.

**Isolated vertex:** It is the vertex that is not connected to any other vertices in the graph.

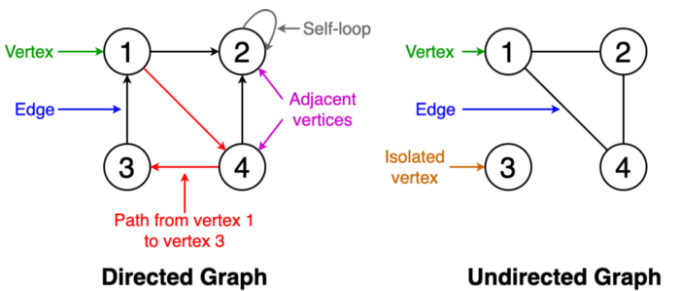
**Vertex degree:** It is defined as the number of edges incident to a vertex in a graph.

**Weighted graph:** A graph having value or weight of vertices.

**Unweighted graph:** A graph having no value or weight of vertices.

**Directed graph:** A graph having a direction indicator.

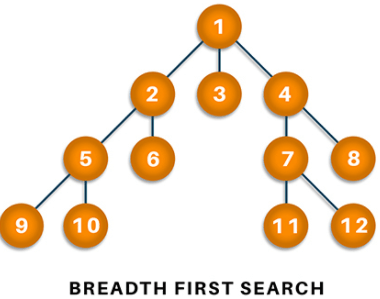
**Undirected graph:** A graph where no directions are defined.



# Different types of graph algorithms

### Breadth-First Search

Traversing or searching is one of the most used operations that are undertaken while working on graphs. Therefore, in **breadth-first-search** (BFS), you start at a particular vertex, and the algorithm tries to visit all the neighbors at the given depth before moving on to the next level of traversal of vertices. Unlike trees, graphs may contain cyclic paths where the first and last vertices are remarkably the same always. Thus, in BFS, you need to keep note of all the track of the vertices you are visiting. To implement such an order, you use a queue data structure which First-in, First-out approach. To understand this, see the image given below.



**Algorithm**

1. Start putting anyone vertices from the graph at the back of the queue.
2. First, move the front queue item and add it to the list of the visited node.
3. Next, create nodes of the adjacent vertex of that list and add them which have not been visited yet.
4. Keep repeating steps two and three until the queue is found to be empty.

**Pseudocode**

1. Set all nodes to "not visited";
2. q = new Queue();
3. q.enqueue(initial node);
4. while ( q ? empty ) do
5. {
6. x = q.dequeue();
7. if ( x has not been visited )
8. {
9. visited[x] = true;         // Visit node x !
11. for ( every edge (x, y)  /\* we are using all edges ! \*/ )
12. if ( y has not been visited )
13. q.enqueue(y);       // Use the edge (x,y) !!!
14. }
15. }

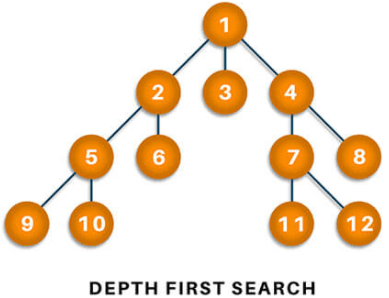
**Complexity: 0(V+E)** where V is vertices and E is edges.

**Applications**

BFS algorithm has various applications. For example, it is used to determine the **shortest path** and **minimum spanning tree.** It is also used in web crawlers to creates web page indexes. It is also used as powering search engines on social media networks and helps to find out peer-to-peer networks in BitTorrent.

### Depth-first search

In depth-first-search (DFS), you start by particularly from the vertex and explore as much as you along all the branches before backtracking. In DFS, it is essential to keep note of the tracks of visited nodes, and for this, you use stack data structure.



**Algorithm**

1. Start by putting one of the vertexes of the graph on the stack's top.
2. Put the top item of the stack and add it to the visited vertex list.
3. Create a list of all the adjacent nodes of the vertex and then add those nodes to the unvisited at the top of the stack.
4. Keep repeating steps 2 and 3, and the stack becomes empty.

**Pseudocode**

1. DFS(G,v)   ( v is the vertex where the search starts )
2. Stack S := {};   ( start with an empty stack )
3. for each vertex u, set visited[u] := false;
4. push S, v;
5. while (S is not empty) do
6. u := pop S;
7. if (not visited[u]) then
8. visited[u] := true;
9. for each unvisited neighbour w of uu
10. push S, w;
11. end if
12. end while
13. END DFS()

**Applications**

DFS finds its application when it comes to finding paths between two vertices and detecting cycles. Also, topological sorting can be done using the DFS algorithm easily. DFS is also used for one-solution puzzles.

### Dijkstra's shortest path algorithm

Dijkstra's shortest path algorithm works to find the minor path from one vertex to another. The sum of the vertex should be such that their sum of weights that have been traveled should output minimum. The shortest path algorithm is a highly curated algorithm that works on the concept of receiving efficiency as much as possible.

### Minimum Spanning Tree

A minimum spanning is defined as a subset of edges of a graph having no cycles and is well connected with all the vertices so that the minimum sum is availed through the edge weights. It solely depends on the cost of the spanning tree and the minimum span or least distance the vertex covers. There can be many minimum spanning trees depending on the edge weight and various other factors.

