Docker and Containerization

Difference between Docker Images and Containers

Docker Image	Docker Container
It is a blueprint of the Container.	It is an instance of the Image.
Image is a logical entity.	The container is a real-world entity.
Images are created only once.	Containers are created any number of times using an image.
Images are immutable. One cannot attach volumes and networks.	Containers change only if the old image is deleted and a new one is used to build the container. One can attach volumes, networks, etc.
Images do not require computing resources to work.	Containers require computing resources to run as they run with a Docker Virtual Machine.
To make a docker image, you have to write a script in a Dockerfile.	To make a container from an image, you have to run the “docker run <image>” command
Docker Images are used to package up applications and pre-configured server environments.	Containers use server information and a file system provided by an image in order to operate.
Images can be shared on Docker Hub.	It makes no sense in sharing a running entity, always docker images are shared.
There is no such thing as a running state of a Docker Image.	Containers use RAM when created and in a running state.
An image must not refer to any state to remove the image.	A container must be in a running state to remove it.
One cannot connect to the images as these images are like snapshots.	In this, one cannot connect them and execute the commands.
Sharing of Docker Images is possible.	Sharing of containers is not possible directly.
It has multiple read-only layers.	It has a single writable layer.
These image templates can exist in isolation.	These containers cannot exist without images.

Adjacency Matrix

Characteristics of the adjacency matrix are:

- **The size of the matrix is determined by the number of vertices (or nodes) in a graph.**
- **The edges in the graph are represented as values in the matrix. In case of unweighted graphs, the values are 0 or 1. In case of weighted graphs, the values are weights of the edges if edges are present, else 0.**
- **If the graph has few edges, the matrix will be sparse.**

Advantages of using Adjacency Matrix:

- **An adjacency matrix is simple and easy to understand.**
- **Adding or removing edges from a graph is quick and easy.**
- **It allows constant time access to any edge in the graph.**

Disadvantages of using Adjacency Matrix:

- **It is inefficient in terms of space utilization for sparse graphs because it takes up O(N2) space.**
- **Computing all neighbors of a vertex takes O(N) time.**

Inverse Matrix

$$M = \begin{bmatrix} 3 & 0 & 2 \\ 2 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix} ; \quad M^{-1} = ?$$

$$\left[\begin{array}{ccc|ccc} 3 & 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & -2 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc} 3 & 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & -2 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$\cdots \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0.2 & 0.2 & 0 \\ 0 & 1 & 0 & -0.2 & 0.3 & 1 \\ 0 & 0 & 1 & 0.2 & -0.3 & 0 \end{array}\right]$$

Cython Code

```
def primes(int nb_primes):
 2     cdef int n, i, len_p
 3     cdef int[1000] p
 4
 5     if nb_primes > 1000:
 6         nb_primes = 1000
 7
 8
 9
10
11     len_p = 0  # The current number of elements in p.
12     n = 2
13     while len_p < nb_primes:
14         # Is n prime?
15         for i in p[:len_p]:
16             if n % i == 0:
17                 break
18
19         # If no break occurred in the loop, we have a
prime.
20         else:
21             p[len_p] = n
22             len_p += 1
23         n += 1
24
25     # Let's copy the result into a Python list:
26     result_as_list = [prime for prime in p[:len_p]]
27     return result_as_list
```