1) What is Graph Traversal?

Graph traversal is a method used to search nodes and edges in a graph, such as a tree or a network. It is also used to decide the order of vertices visited in the search process, without creating loops. There are different ways to order the vertices, such as depth-first search (DFS) or breadth-first search (BFS). DFS starts at the root node and explores as far as possible along each branch before backtracking. BFS starts at the root node and explores all the neighboring nodes first, before moving to the next level of nodes.

2) What are graph traversal algorithms?

1. **Breadth-First Search (BFS):** is an algorithm that is used to traverse a finite graph, either for searching or graph data processing. The algorithm begins at the root node and visits all the nodes at the same level before moving to the next level. The algorithm also uses a queue to store the nodes that are waiting to be visited. One of the applications of BFS is to find the shortest path between two nodes in a graph.

   A real-life example of **BFS** is the **"six degrees of Kevin Bacon"** game, which is based on the idea that any two actors can be connected by a chain of co-stars in six steps or less2. For example, to find the Bacon number of Tom Cruise, we can use BFS to search the graph of actors and movies, starting from Kevin Bacon as the root node. We can find that Tom Cruise has a Bacon number of 2, because he co-starred with Dustin Hoffman in Rain Man, and Dustin Hoffman co-starred with Kevin Bacon in Sleepers.

2. **Depth-first search (DFS):** is a recursive algorithm that is used to traverse a finite graph, either for searching or graph data processing. The algorithm begins at the root node and visits the child nodes first, going deeper into the graph before exploring the sibling nodes. The algorithm also uses a stack, either explicitly or implicitly through recursion, to store the nodes that are waiting to be visited.

   A real-life example of **DFS** is maze generation and solving. DFS can be used to create a random maze by starting from a grid of cells and randomly choosing a neighboring cell to connect with a passage. DFS can also be used to find a path from the entrance to the exit of a maze by marking the visited cells and backtracking when reaching a dead end.

3. **Tricolor algorithm:** is an abstract algorithm that can be used to express graph traversal or search. The algorithm assigns one of three colors to each node in the graph: white, gray, or black. The color of a node can change over time, depending on its state in the traversal.

   - **White** nodes are undiscovered nodes that have not been seen yet in the current traversal and may even be unreachable.
   - **Gray** nodes are discovered nodes that have been seen but not fully explored yet. They may have some white neighbors that need to be visited.
   - **Black** nodes are finished nodes that have been fully explored and have no white neighbors left. They will not be visited again in the current traversal.

A real-life example of **Tricolor algorithm** is garbage collection in some programming languages. Garbage collection is the process of reclaiming memory that is no longer used by the program. The tricolor algorithm can be used to mark the reachable objects in memory as gray or black, and collect the unreachable objects that are white.

4. **Topological sort:** is an algorithm that is used to order the vertices of a directed acyclic graph (DAG) such that for every directed edge u -> v, vertex u comes before v in the ordering. A DAG is a graph that has no cycles, meaning that there is no path from a vertex to itself. A topological sort is possible only if the graph is a DAG.

   A real-life example of **Topological sort** is course scheduling in a university. Each course may have some prerequisites that need to be completed before taking the course. The courses and their prerequisites can be represented as a DAG, where each vertex is a course, and each edge is a prerequisite relation. A topological sort of this graph can give a valid order of taking the courses without violating the prerequisites.

5. **Dijkstra's algorithm:** is an algorithm that is used to find the shortest path from a source node to all other nodes in a weighted graph. The algorithm works by maintaining a set of nodes whose shortest distance from the source is already known, and a set of nodes whose distance is still unknown. The algorithm repeatedly selects the node with the smallest distance from the unknown set and updates the distances of its neighbors by adding the edge weight. The algorithm stops when all nodes are visited or the smallest distance in the unknown set is infinity.

   A real-life example of **Dijkstra's algorithm** is digital mapping services such as Google Maps. When we want to find the shortest route from one location to another, the algorithm can be used to calculate the distance and time based on the traffic, road conditions, and speed limits. The locations and roads can be represented as a graph, where each node is a location, and each edge is a road with a weight corresponding to the travel time.

3) If a hash function assigns the same location to two elements, what is the solution?

To understand this problem, we need to understand what a HashMap is, a HashMap is a data structure that is used to store and retrieve values based on keys. A key is a unique identifier for a value, and a value can be any data type. A HashMap uses a hash function to map keys to indices in an array, where the values are stored. A hash function is a mathematical function that converts a key into a number, which is then used as an index in the array. A HashMap has some advantages.

Now this problem is calling Collision: Collision occurs when two different keys have the same hash value, and map to the same index in the array. This can cause data loss or incorrect results. Collision can be resolved by using different techniques, such as chaining or linear probing.

To resolve collisions, there are different techniques, such as

- Chaining: Chaining is a technique where each index in the array stores a linked list of key-value pairs with the same hash value. When a new key-value pair is inserted, it is added to the end of the linked list at that index. When a key is searched, it is compared with all the keys in the linked list at that index until a match is found or the list ends.

- Linear probing: Linear probing is a technique where each index in the array stores only one key-value pair. When a new key-value pair is inserted, it is placed at the first available index after its hash value. If that index is occupied, it moves to the next index until an empty slot is found. When a key is searched, it is compared with the key at its hash value. If they do not match, it moves to the next index until a match is found or an empty slot is encountered.

4) What is a lambda expression, provide an example?

A lambda expression in Python is a short and anonymous function that can be defined inline and used as a parameter or a value in other functions. A lambda expression in Python has the following syntax:

- lambda: A lambda expression starts with the keyword lambda, which indicates that it is a function definition.
- Parameters: A lambda expression can have zero, one or more parameters, which are separated by commas and enclosed in parentheses. The parameter types are not specified.
- Colon: A lambda expression uses the: operator to separate the parameters from the body.
- Body: A lambda expression can have only one expression as its body, which is evaluated and returned as the result of the function.

Some examples:

- A lambda expression that takes one parameter num and returns “Even number” if num is even, or “Odd number” if num is odd:
  lambda num: “Even number” if num % 2 == 0 else “Odd number”

- A lambda expression that takes one parameter x and returns a list of its factors:
  lambda x: [i for i in range(1, x + 1) if x % i == 0]

- A lambda expression that takes one parameter s and returns True if s is a palindrome, or False otherwise: lambda s: s == s[::-1]

5) What is code refactoring?

Code refactoring is a process of modifying the source code of a software system to improve its quality, readability, and maintainability, without changing its functionality or behavior. Code refactoring can have many benefits for the software development process, such as enhancing the readability and clarity of the code, reducing the number of bugs and errors, increasing the testability and reliability of the code, and facilitating the extensibility and scalability of the code. Code refactoring can be done for various reasons, such as following coding standards and conventions, eliminating code duplication, and improving code reusability, simplifying complex or nested code, reorganizing the structure or hierarchy of the code, and optimizing the performance or efficiency of the code. Code refactoring can be done manually or automatically using tools that can analyze, transform, and rewrite the source code, such as DMS Software Reengineering Toolkit, Eclipse, or Visual Studio Code