Q1: Traversal Graph Algorithms

Traversal algorithms are essential techniques in graph theory and computer science for navigating and exploring the nodes and edges of a graph. They are used to visit all or some of the nodes in a graph, and they serve as the basis for various graph-related problems and applications. There are two main types of traversal algorithms: depth-first traversal and breadth-first traversal.

BFS: In breadth-first traversal, the algorithm explores all the neighbours of a node before moving to their respective neighbours. This type of traversal is commonly implemented using a queue.

Sample Code for BFS:

```
def bfs(graph, node):

    visited = []

    queue = []

    visited.append(node)

    queue.append(node)


    while queue:

        m = queue.pop(0)

        print(m)

        for i in graph[m]:

            if i not in visited:

                visited.append(i)

                queue.append(i)


graph = {

    'A': ['B','C'],

    'B': ['D','E'],

    'C': ['F'],

    'D': [],

    'E': ['F'],

    'F': []

}


print("BFS")

bfs(graph, 'A')
```

DFS: is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It's often used for searching paths, cycle detection, and component connectivity.

Sample code for DFS:

```
def dfs(graph, node):
    S=[]
    visited=[]
    S.append(node)
    while(S):
        x=S.pop()
        if(x not in visited):
            visited.append(x)
            for i in graph[x]:
                S.append(i)
    print(visited)


graph = {
    'A': ['B','C'],
    'B': ['D','E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
print("DFS")
dfs(graph, 'A')
```

Question 2: What happens if HashMap function maps to same memory with 2 different data? What should be done to fix it?

If a hash map function maps two different pieces of data to the same memory location, this situation is known as a "hash collision." Hash collisions can occur due to the nature of hash functions and the finite number of memory locations available. Hash functions aim to distribute data uniformly across the available memory locations, but sometimes different data values can produce the same hash value, leading to collisions.

When a hash collision occurs, most hash map implementations employ strategies to handle and resolve these collisions. Here are some common collision resolution techniques:

1. Chaining: In this method, each bucket (memory location) of the hash map contains a linked list or another data structure to hold multiple key-value pairs that share the same hash value. When a collision happens, the new key-value pair is simply appended to the list in that bucket.

2. Open Addressing: In open addressing, when a collision occurs, the algorithm searches for the next available slot in the hash map and stores the key-value pair there. This is often done through various probing techniques, such as linear probing (moving to the next slot), quadratic probing (using quadratic intervals), or double hashing (applying a second hash function to determine the next slot).

3. Robin Hood Hashing: This is a variant of open addressing that tries to minimize the variance in the probe sequence. If a new key-value pair must be inserted and the desired bucket is occupied, Robin Hood Hashing may swap the pair with the pair already present in the bucket if the newly arrived pair has a shorter probe distance.

Question 3: What is Refactor?

Refactoring is a term used in software development to describe the process of restructuring and modifying existing code or software systems to improve its design, readability, maintainability, and sometimes even performance, without changing its external behaviour. The primary goal of refactoring is to make the codebase easier to understand, modify, and extend while preserving its functionality. Refactoring involves making a series of small, focused changes to the code, often driven by well-defined patterns and best practices. These changes can include:

1. Simplifying Code: Removing unnecessary complexity, redundant code, or overly convoluted logic.

2. Improving Naming: Choosing more descriptive and meaningful names for variables, functions, and classes to enhance code readability.

3. Breaking Down Functions/Methods: Splitting large functions or methods into smaller, more focused ones that perform specific tasks.

4. Extracting Reusable Components: Identifying common patterns and extracting them into separate functions or classes that can be reused across the codebase.

5. Reducing Duplication: Eliminating code duplication by abstracting common functionality into shared components.

6. Eliminating Magic Numbers/Strings: Replacing hard-coded values with named constants or variables to enhance code maintainability.

7. Applying Design Patterns: Introducing well-known design patterns to improve the overall architecture and structure of the code.

8. Improving Performance: Identifying bottlenecks and areas of inefficiency and optimizing code to run faster.

9. Enhancing Testability: Modifying code to make it easier to write unit tests and improve overall test coverage.

10. Consolidating Code: Combining similar code segments to reduce redundancy and improve maintainability.

Question4: What is Lambda expression?

In Python, a lambda expression (also known as a lambda function) is a concise way to create small, anonymous functions. Lambda functions are often used for short, simple operations that can be defined in a single line of code. They are defined using the lambda keyword, followed by the input arguments and the expression to be evaluated. The result of the expression is automatically returned.

Lambda arguments: expression

```
Def add(x,y):
        Return x+y
```

Equivalent to

Add_lambda=lambda x,y: x+y