1.
```java
// Constructor
DirectedGraph(int V) {
    this.V = V;
    adjListArray = new LinkedList[V];
    for (int i = 0; i < V; i++) {
        adjListArray[i] = new LinkedList<>();
    }
}
```

2.
```java
  // Method untuk membuat simpul baru
  public void addEdge(char src, char dest) {
      adjListArray[src - 'A'].add(dest);
  }
```

3.
```java
// Method untuk mencetak graph
public void printGraph() {
    for (int i = 0; i < V; i++) {
        if (!adjListArray[i].isEmpty()) {
            char vertex = (char) (i + 'A');
            System.out.print("Vertex " + vertex + " is connected to: ");
            for (int j = 0; j < adjListArray[i].size(); j++) {
                System.out.print(adjListArray[i].get(j) + " ");
            }
            System.out.println();
        }
    }
}
```

4.

```java
public void BFS(char start) {
    boolean visited[] = new boolean[V];
    Queue<Character> queue = new LinkedList<>();
    visited[start - 'A'] = true;
    queue.add(start);

    // Memulai loop untuk terus berjalan selama antrian kosong (masih ada simpul
    // yang perlu dijelajahi)
    while (!queue.isEmpty()) {
        char vertex = queue.poll(); // Membuat simpul atau node untuk diproses
        System.out.print(vertex + " "); // Mencetak node yang sedang diproses
        // Loop untuk melalui setiap simpul yang sedang diproses
        for (Character adj : adjListArray[vertex - 'A']) {
            // Untuk memeriksa tetangga berikutnya apakah sudah dikunjungi atau belum
            if (!visited[adj - 'A']) {
                visited[adj - 'A'] = true;
                queue.add(adj); // Menambahkan tetangga yang belum dikunjungi untuk masuk ke proses
            }
        }
    }
}
```

5.

```java
public void DFS(char start) { // Fungsi untuk DFS
    boolean visited[] = new boolean[V]; // Untuk Mengecek apakah sudah dikunjungi
    Stack<Character> stack = new Stack<>();
    stack.push(start);

    while (!stack.isEmpty()) { // Loop untuk menjelajahi semua simpul sampai tidak ada yang tersisa
        char vertex = stack.pop();
        if (!visited[vertex - 'A']) { // Untuk mengecek apakah sudah dikunjungi atau belum
            visited[vertex - 'A'] = true;
            System.out.print(vertex + " ");
            for (Character adj : adjListArray[vertex - 'A']) {
                // Mengecek apakah tetangga yang akan dikunjungi sudah pernah atau blm
                if (!visited[adj - 'A']) {
                    stack.push(adj);
                }
            }
        }
    }
}
```