```java
import java.util.*;

public class IX {
    public static void main(String... args) {
        int n = 7;
        ArrayList<Vertex> graph = createHasseGraph(n);
        System.out.println("DFS: ");
        dfs(graph.get(0));
        System.out.println("BFS: ");
        for (Vertex u: graph) {
            u.visited = false;
        }
        bfs(graph.get(0));
    }

    public static void dfs(Vertex u) {  // u is the starting vertex
        System.out.println(
            "id: " + u.nodeId + "; partition: " + Arrays.toString(u.partition)
        );  // print the node
        u.visited = true;  // mark u as visited
        for (Vertex v: u.adjacents) {  // for each vertex v that are adjacent to u
            if (!v.visited) {  // if v is not visited
                dfs(v);  // recursively call dfs making v that starting vertex
            }
        }
    }

    public static void bfs(Vertex u) {  // u is the starting vertex
        LinkedList<Vertex> queue = new LinkedList<>();  // create an empty queue
        u.visited = true;  // mark u as visited
        queue.add(u);  // enqueue u to the queue
        while (queue.size() != 0) {  // while the queue is not empty
            u = (Vertex) queue.remove();  // dequeue an item from the queue and put it in u
            System.out.println(
                "nodeId: " + u.nodeId + "; partition: " + Arrays.toString(u.partition)
            );  // print the node
            for (Vertex v: u.adjacents) {  // for each vertex v that are adjacent to u
                if (!v.visited) {  // if v is not visited
                    v.visited = true;  // mark v as visisted
                    queue.add(v);  // add v to the queue
                }
            }
        }
    }

    public static ArrayList<Vertex> createHasseGraph(int n) {
        int id = 1; // the id of the starting vertex
        int[] partition = new int[]{n}; // the partition of the starting vertex
        Vertex start = new Vertex(id, partition);  // create the starting vertex
        LinkedList<Vertex> queue = new LinkedList<>(); // create an empty queue
        HashMap<String, Vertex> map = new HashMap<>();  // create a hashmap with a string
        representing the partition as a key and corresponding vertex as the associated value
        map.put(Arrays.toString(partition), start); // put the start node in the hashmap
        queue.add(start);  // enqueue the strat node to the queue
        ArrayList<Vertex> graph = new ArrayList<>(); // create an empty grapgh
        while (queue.size() != 0) { // while the queue is not empty
            Vertex u = (Vertex) queue.remove(); // dequeue an item from the queue and put it
            in u
            graph.add(u);  // add u to the grapgh
            for (int i = 0; i < u.partition.length; i++) { //for each part of u.partition
                for (int j = 1; j <= u.partition[i] / 2; j++) {  // for each j in {1..m}
                where m = the half of the current part
                    partition = new int[u.partition.length + 1]; // create a partition
                    for (int k = 0; k < i; k++) { // copy the first (i-1) items
                        partition[k] = u.partition[k];
                    }
                    partition[i] = j; // put j as the ith part
                    partition[i + 1] = u.partition[i] - j; // put the current part of u - j
                    as the (i+1)th part
                    for (int k = i + 2; k < partition.length; k++) { // copy the reamining
                    items
```

```
                        partition[k] = u.partition[k - 1];
                    }
                    Arrays.sort(partition); // sort the partition
                    for (int k = 0; k < partition.length / 2; k++) { // reverse the sorted
                    partition
                        int t = partition[k];
                        partition[k] = partition[partition.length - k - 1];
                        partition[partition.length - k - 1] = t;
                    } // creating partion ends here
                    String key = Arrays.toString(partition); // create a key (String)
                    represnting the partition
                    Vertex v = map.get(key); //  get the vertex correspong to the generated
                    key from the hashmap
                    if (v == null) {  // if there is no such  vertex
                        v = new Vertex(++id, partition); // create a new vertex
                        map.put(key, v);  // add the vertex to the hashmap
                        queue.add(v);  // add the vertex to the queue
                    }
                    u.adjacents.add(v);  // add v in the set of adjacents of u
                    v.adjacents.add(u);  // add u in the set of adjacents of v
                }
            }
        }
        return graph;
    }
}

class Vertex {
    int nodeId = 0;
    int[] partition = null;
    ArrayList<Vertex> adjacents = new ArrayList<>();
    boolean visited = false;
    Vertex(int nodeId, int[] partition) {
        this.nodeId = nodeId;
        this.partition = partition;
    }
}
```