1.Encrypted file name update

```
public static String getNextFileName(int n, int k, String s) {
    TreeSet<Character> allowedChars = new TreeSet<>();
    for (char ch : s.toCharArray()) {
        allowedChars.add(ch);
    }

    List<Character> sortedChars = new ArrayList<>(allowedChars);
    char minChar = sortedChars.get(0);

    if (k > n) {
        StringBuilder sb = new StringBuilder(s);
        for (int i = n; i < k; i++) {
            sb.append(minChar);
        }
        return sb.toString();
    }

    char[] ans = s.substring(0, k).toCharArray();

    for (int i = k - 1; i >= 0; i--) {
        char curr = ans[i];
        int index = Collections.binarySearch(sortedChars, curr);
        if (index < sortedChars.size() - 1) {
            ans[i] = sortedChars.get(index + 1);
            for (int j = i + 1; j < k; j++) {
                ans[j] = minChar;
            }
            return new String(ans);
        }
    }
```

```
        return ""; // no valid string
    }
```

2.Magic Square Transformation

```
public class MagicalSequence {

    public static int getMaxMagicalPower(String s) {
        int power = 0;
        int count = 0;

        for (char ch : s.toCharArray()) {
            if (ch == 'X') {
                count++;
            } else if (ch == 'Y') {
                if (count > 0) {
                    power++;
                    count--;
                }
            }
        }

        return power;
    }
}
```

3.Cloud Network Bandwidth Pricing

```
import java.util.*;

class CloudNetwork {

    static Map<Integer, Long> feeMap = new HashMap<>();
```

```
    // This function processes the list of events and returns a list of results for each transfer
    public static List<Long> processEvents(int[][] events) {
        List<Long> result = new ArrayList<>();

        for (int[] event : events) {
            if (event[0] == 1) {
                updateFee(event[1], event[2], event[3]);
            } else if (event[0] == 2) {
                result.add(calculateCost(event[1], event[2]));
            }
        }

        return result;
    }

    private static void updateFee(int a, int b, int x) {
        for (int node : getPath(a, b)) {
            feeMap.put(node, feeMap.getOrDefault(node, 0L) + x);
        }
    }

    private static long calculateCost(int a, int b) {
        long total = 0;
        for (int node : getPath(a, b)) {
            total += feeMap.getOrDefault(node, 0L);
        }
        return total;
    }

    private static List<Integer> getPath(int a, int b) {
        Set<Integer> pathA = new HashSet<>();
        List<Integer> fullPath = new ArrayList<>();
```

```
        int u = a, v = b;
        while (u != 0) {
            pathA.add(u);
            u /= 2;
        }

        while (!pathA.contains(v)) {
            v /= 2;
        }

        int lca = v;
        u = a;
        v = b;

        while (u != lca) {
            fullPath.add(u);
            u /= 2;
        }
        while (v != lca) {
            fullPath.add(v);
            v /= 2;
        }

        return fullPath;
    }
}
```

4.Logistic delivery problem

```
public int maxEqualParcelsAfterPermutation(int[] parcels) {
    Map<Integer, Integer> freqMap = new HashMap<>();
    int maxCount = 0;
```

```
    for (int i = 0; i < parcels.length; i++) {
        int key = parcels[i] - i;
        freqMap.put(key, freqMap.getOrDefault(key, 0) + 1);
        maxCount = Math.max(maxCount, freqMap.get(key));
    }

    return maxCount;
}
```

5.Kindom

```
public List<Integer> removeRebelliousNobles(int n, int[][] nobles) {
    List<Integer> result = new ArrayList<>();
    List<Integer>[] children = new ArrayList[n + 1];
    int[] parent = new int[n + 1];
    int[] respect = new int[n + 1];
    boolean[] removed = new boolean[n + 1];

    for (int i = 1; i <= n; i++) children[i] = new ArrayList<>();

    for (int i = 1; i <= n; i++) {
        int p = nobles[i - 1][0];
        int r = nobles[i - 1][1];
        parent[i] = p;
        respect[i] = r;

        if (p != -1) children[p].add(i);
    }

    boolean changed = true;
    while (changed) {
        changed = false;
```

```
        for (int i = 1; i <= n; i++) {
            if (removed[i] || parent[i] == -1 || respect[i] == 0) continue;

            boolean canRemove = true;
            for (int child : children[i]) {
                if (!removed[child] && respect[child] == 0) {
                    canRemove = false;
                    break;
                }
            }

            if (canRemove) {
                result.add(i);
                removed[i] = true;

                // reassign children to grandparent
                int par = parent[i];
                for (int child : children[i]) {
                    if (!removed[child]) {
                        parent[child] = par;
                        children[par].add(child);
                    }
                }

                children[i].clear();
                changed = true;
                break;  // Restart from smallest
            }
        }
    }

    return result.isEmpty() ? List.of(-1) : result;
}
```

6.City

```
public List<Integer> removeRebelliousNobles(int n, int[][] nobles) {
    List<Integer> result = new ArrayList<>();
    List<Integer>[] children = new ArrayList[n + 1];
    int[] parent = new int[n + 1];
    int[] respect = new int[n + 1];
    boolean[] removed = new boolean[n + 1];

    for (int i = 1; i <= n; i++) children[i] = new ArrayList<>();

    for (int i = 1; i <= n; i++) {
        int p = nobles[i - 1][0];
        int r = nobles[i - 1][1];
        parent[i] = p;
        respect[i] = r;

        if (p != -1) children[p].add(i);
    }

    boolean changed = true;
    while (changed) {
        changed = false;

        for (int i = 1; i <= n; i++) {
            if (removed[i] || parent[i] == -1 || respect[i] == 0) continue;

            boolean canRemove = true;
            for (int child : children[i]) {
                if (!removed[child] && respect[child] == 0) {
                    canRemove = false;
                    break;
                }
            }
```

```
            if (canRemove) {
                result.add(i);
                removed[i] = true;

                // reassign children to grandparent
                int par = parent[i];
                for (int child : children[i]) {
                    if (!removed[child]) {
                        parent[child] = par;
                        children[par].add(child);
                    }
                }

                children[i].clear();
                changed = true;
                break;  // Restart from smallest
            }
        }
    }

    return result.isEmpty() ? List.of(-1) : result;
}
```