```
import java.util.HashMap;
import java.io.PrintStream;
import java.io.BufferedReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Collection;
import java.util.Map;
import java.util.List;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.TreeMap;
import java.util.ArrayList;
import java.util.Set;
import java.util.StringTokenizer;
import java.io.InputStream;

/**
 * Built using CHelper plug-in
 * Actual solution is at the top
 */
public class Main {
	public static void main(String[] args) {
		InputStream inputStream = System.in;
		OutputStream outputStream = System.out;
		InputReader in = new InputReader(inputStream);
		PrintWriter out = new PrintWriter(outputStream);
		SimilarSequencesDPGen solver = new SimilarSequencesDPGen();
		solver.solve(1, in, out);
		out.close();
	}
}

class SimilarSequencesDPGen {
    static final int n = 6;
    static final int m = 2;
    static final int interesting = 3;

    static class State {
        String a;
        String b;
        State representative;

        State(String a, String b) {
            this.a = a;
            this.b = b;
        }

        public String toString() {
            return a + "$" + b;
        }
    }

    boolean areSimilar(String a, String b, int maxRemovals) {
        if (a.length() != b.length()) throw new RuntimeException();
        int[][] maxCommon = new int[a.length() + 1][b.length() + 1];
        for (int pa = 0; pa < a.length(); ++pa)
            for (int pb = 0; pb < b.length(); ++pb)
```

```
                if (a.charAt(pa) == b.charAt(pb)) {
                    maxCommon[pa + 1][pb + 1] = 1 + maxCommon[pa][pb];
                } else {
                    maxCommon[pa + 1][pb + 1] = Math.max(maxCommon[pa][pb + 1], maxCommon[pa +
1][pb]);
                }
        return maxCommon[a.length()][b.length()] >= a.length() - maxRemovals;
    }

    public void solve(int testNumber, InputReader in, PrintWriter out) {
        List<String> allStrings = new ArrayList<String>();
        for (int len = 0; len <= n; ++len) {
            for (int index = 0; index < Math.pow(m, len); ++index) {
                StringBuilder builder = new StringBuilder();
                int tmp = index;
                for (int pos = 0; pos < len; ++pos) {
                    builder.append((char) ('a' + tmp % m));
                    tmp /= m;
                }
                allStrings.add(builder.toString());
            }
        }
        Map<String, State> allStates = new HashMap<String, State>();
        for (String a : allStrings) {
            for (String b : allStrings) {
                if (a.length() == b.length()) {
                    State s = new State(a, b);
                    allStates.put(s.toString(), s);
                }
            }
        }
        for (int len = n; len >= 0; --len) {
            Map<String, State> representatives = new HashMap<String, State>();
            for (State state : allStates.values()) {
                if (state.a.length() != len) continue;
                String description;
                if (len == n) {
                    description = areSimilar(state.a, state.b, 2) ? "Y" : "N";
                } else {
                    Map<String, Integer> counts = new TreeMap<String, Integer>();
                    for (int ca = 0; ca < m; ++ca) {
                        String na = state.a + (char) ('a' + ca);
                        for (int cb = 0; cb < m; ++cb) {
                            String nb = state.b + (char) ('a' + cb);
                            State next = allStates.get(new State(na, nb).toString());
                            String nextDesc = next.representative.toString();
                            Integer old = counts.get(nextDesc);
                            if (old == null) {
                                counts.put(nextDesc, 1);
                            } else {
                                counts.put(nextDesc, old + 1);
                            }
                        }
                    }
                    StringBuilder descriptionBuilder = new StringBuilder();
                    for (Map.Entry<String, Integer> entry : counts.entrySet()) {
```

```
                        descriptionBuilder.append("|" + entry.getKey() + "|" +
entry.getValue());
                    }
                    description = descriptionBuilder.toString();
                }
                state.representative = representatives.get(description);
                if (state.representative == null) {
                    state.representative = state;
                    representatives.put(description, state);
                }
            }
            System.out.println("For len " + len + " there are " + representatives.size() + "
representatives.");
        }
        for (State s : allStates.values()) {
            if (s.representative == s && s.a.length() == interesting) {
                for (State other : allStates.values()) {
                    if (other.representative == s) {
                        System.out.print(other.toString() + " ");
                    }
                }
                System.out.println();
            }
        }
    }
}

class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream), 32768);
        tokenizer = null;
    }

}
```