题目描述：

单词接龙的规则是：可用于接龙的单词首字母必须要前一个单词的尾字母相同；当存在多个首字母相同的单词时，取长度最长的单词，如果长度也相等，则取字典序最小的单词；已经参与接龙的单词不能重复使用。
现给定一组全部由小写字母组成单词数组，并指定其中的一个单词作为起始单词，进行单词接龙，请输出最长的单词串，单词串是单词拼接而成，中间没有空格。

输入描述：

输入的第一行为一个非负整数，表示起始单词在数组中的索引 K，0 <= K < N ；

输入的第二行为一个非负整数，表示单词的个数 N；

接下来的 N 行，分别表示单词数组中的单词。

输出描述：

输出一个字符串，表示最终拼接的单词串。

补充说明：

单词个数 N 的取值范围为[1, 20]；

单个单词的长度的取值范围为[1, 30]；

示例 1

输入：

```
0

6

word

dd

da

dc

dword

d
```

输出：

```
worddwordda
```

说明：

先确定起始单词 word，再接以 d 开头的且长度最长的单词 dword，剩余以 d 开头且长度最长的有 dd、da、dc，则取字典序最小的 da，所以最后输出 worddwordda。

示例 2

输入：

```
4
6
word
dd
da
dc
dword
d
```

输出：

```
dwordda
```

说明：

先确定起始单词 dword，剩余以 d 开头且长度最长的有 dd、da、dc，则取字典序最小的 da，所以最后输出 dwordda。

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int index = sc.nextInt();
        sc.nextLine();
        int n = sc.nextInt();
        sc.nextLine();
        String[] arr = new String[n];
        boolean[] visited = new boolean[n];
        visited[index] = true;
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextLine();
```

```
            }
            System.out.println(arr[index] + dfs(arr, arr[index].charAt(arr[index].length() - 1),
visited));
        }

        public static String dfs(String[] arr, char ch, boolean[] visited) {
            int index = -1;
            int len = 0;
            String res = "";
            for (int i = 0; i < arr.length; i++) {
                if (!visited[i]) {
                    if (arr[i].charAt(0) == ch) {
                        if (arr[i].length() > len) {
                            if (index != -1) {
                                visited[index] = false;
                            }
                            len = arr[i].length();
                            index = i;
                            visited[i] = true;
                            res = arr[i];
                        } else if (arr[i].length() == len) {
                            if (res.compareTo(arr[i]) > 0) {
                                if (index != -1) {
                                    visited[index] = false;
                                }
                                index = i;
                                visited[i] = true;
                                res = arr[i];
                            }
                        }
                    }
                }
            }
            if (!res.equals("")) {
                res += dfs(arr, arr[index].charAt(arr[index].length() - 1), visited);
            }
            return res;
        }
}
```