单词接龙的规则是：

- 可用于接龙的单词首字母必须要与前一个单词的尾字母相同；
- 当可用于接龙的单词有多个时，则取长度最长的单词，如果长度也相等，则取字典序最小的单词，己经参与接龙的单词不能重复使用；
- 当无法接龙时（即没有可用于接龙的单词），若此时已经参与接龙的单词数量满足要求，则接龙成功，否则接龙失败。
- 题目要求输出最终接龙的字符串，若不能接龙则输出 error。

输入描述

- 输入的第一行为一个非负整数，表示将要输入的单词的数量 N，$0 <= N - 1$；
- 输入的第二行为一个非负整数，表示单词接龙的起始单词；
- 接下来的 N 行，分别是各个单词接龙的备选单词。

说明：

- 单词只可能由字母组成[1, 2]；
- 第一个单词及以后的接龙单词均为小写。

输出描述

- 输出一个字符串，表示按照规则的单词接龙。

用例

题目解析

本题可以用哈希表来解决。

我们将单词存放到一个哈希表中。

JavaScript算法源码

```
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let k, x;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length == 1) {
    [k, x] = lines.map(Number);
  }

  if (x && lines.length == x + 1) {
    const words = lines.slice(1);

    console.log(wordList(words, k));

    lines.length = 0;
  }
});

function wordList(words, k) {
  const chih = words.splice(k, 1);

  const prefix = {};
  words.forEach((word) => {
    const u = word[0];
    prefix[u] ? prefix[u].push(word) : (prefix[u] = [word]);
  });

  for (let key in prefix) {
    prefix[key].sort((a, b) =>
      a.length == b.length ? (a < b ? -1 : 1) : b.length - a.length
    );
  }

  while (true) {
    let tail = chih[chih.length - 1];

    if (prefix[tail]) {
      chih.push(prefix[tail].shift());
    } else {
      break;
    }
  }

  return chih.join("");
}
```

Java算法源码

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in).useDelimiter("\n");

    int k = sc.nextInt();
    int x = sc.nextInt();

    String[] words = new String[x];
    for (int i = 0; i < x; i++) words[i] = sc.next();

    System.out.println(getResult(k, x, words));
  }

  public static String getResult(int k, int x, String[] words) {
    ArrayList<String> chain = new ArrayList<>();
    chain.add(words[k]);

    words[k] = null;

    HashMap<Character, LinkedList<String>> prefix = new HashMap<>();
    for (String word : words) {
      if (word != null) {
        char c = word.charAt(0);
        prefix.putIfAbsent(c, new LinkedList<>());
        prefix.get(c).add(word);
      }
    }

    for (Character c : prefix.keySet()) {
      prefix
        .get(c)
        .sort((a, b) -> a.length() != b.length() ? b.length() - a.length() : a.compareTo(b));
    }

    while (true) {
      String tail = chain.get(chain.size() - 1);
      char c = tail.charAt(tail.length() - 1);

      if (prefix.containsKey(c)) {
        chain.add(prefix.get(c).removeFirst());
      } else {
        break;
      }
    }

    StringBuilder sb = new StringBuilder();
    for (String s : chain) sb.append(s);
    return sb.toString();
  }
}
```

Python算法源码

```
k = int(input())
x = int(input())
words = [input() for _ in range(x)]


# 算法
def getResult():
    chain = [words[k].pop(k)]

    prefix = {}
    for word in words:
        u = word[0]
        if prefix.get(u) is None:
            prefix[u] = []
        prefix[u].append(word)

    for u in prefix.keys():
        prefix[u].sort(key=lambda x: (-len(x), [ord(c) for c in x]))

    while True:
        tail = chain[-1][-1]

        if prefix.get(tail):
            chain.append(prefix[tail].pop(0))
        else:
            break

    return "".join(chain)


# 调用
print(getResult())
```