/\*\* A linked list of character data objects.

 \*  (Actually, a list of Node objects, each holding a reference to a character data object.

 \*  However, users of this class are not aware of the Node objects. As far as they are concerned,

 \*  the class represents a list of CharData objects. Likwise, the API of the class does not

 \*  mention the existence of the Node objects). \*/

public class List {

    // Points to the first node in this list

    private Node first;

    // The number of elements in this list

    private int size;

    /\*\* Constructs an empty list. \*/

    public List() {

        first = null;

        size = 0;

    }

    /\*\* Returns the number of elements in this list. \*/

    public int getSize() {

          return size;

    }

    /\*\* Returns the first element in the list \*/

    public CharData getFirst() {

        return first.cp;

    }

    /\*\* GIVE Adds a CharData object with the given character to the beginning of this list. \*/

    public void addFirst(char chr) {

        Node tempfirst = first;

        first = new Node(new CharData(chr), tempfirst);

        size++;

    }

    /\*\* GIVE Textual representation of this list. \*/

    public String toString() {

         String str = "(";

         ListIterator iterator = listIterator(0);

         while (iterator.hasNext()) {

            str = str + iterator.current.cp.toString() + " ";

            iterator.next();

         }

         str = str.substring(0, str.length() - 1);

         str = str + ")";

         return str;

    }

    /\*\* Returns the index of the first CharData object in this list

     \*  that has the same chr value as the given char,

     \*  or -1 if there is no such object in this list. \*/

    public int indexOf(char chr) {

        int counter = 0;

        ListIterator iterator = listIterator(0);

        if (size > 0) {

        while (iterator.hasNext()) {

            if (iterator.current.cp.chr == chr) {

                return counter;

            }

            iterator.next();

            counter++;

        }

    }

      return -1;

    }

    /\*\* If the given character exists in one of the CharData objects in this list,

     \*  increments its counter. Otherwise, adds a new CharData object with the

     \*  given chr to the beginning of this list. \*/

    public void update(char chr) {

        int i = indexOf(chr);

        if ( i > -1 ) {

            ListIterator iterator = listIterator(i);

            Node temp = iterator.current;

            temp.cp.count++;

        } else {

            addFirst(chr);

        }

    }

    /\*\* GIVE If the given character exists in one of the CharData objects

     \*  in this list, removes this CharData object from the list and returns

     \*  true. Otherwise, returns false. \*/

    public boolean remove(char chr) {

        if (indexOf(chr) == -1) {

            return false;

        }else{

            ListIterator iterator = listIterator(indexOf(chr) - 1);

            Node temp = iterator.current;

            if (temp == first) {

                first = temp.next;

            }else {

                iterator.next();

                if (iterator.hasNext()) {

                    temp.next = iterator.current.next;

                }

                else {

                    temp.next = null;

                }

            }

        }

            size--;

            return true;

    }

    /\*\* Returns the CharData object at the specified index in this list.

     \*  If the index is negative or is greater than the size of this list,

     \*  throws an IndexOutOfBoundsException. \*/

    public CharData get(int index) {

        if (index >= size || index < 0){

        throw new IndexOutOfBoundsException();

        } else {

            int counter = 0;

            Node current = first;

            while(counter < index){

                current = current.next;

                counter++;

            }

            return current.cp;

        }

    }

    /\*\* Returns an array of CharData objects, containing all the CharData objects in this list. \*/

    public CharData[] toArray() {

        CharData[] arr = new CharData[size];

        Node current = first;

        int i = 0;

        while (current != null) {

            arr[i++]  = current.cp;

            current = current.next;

        }

        return arr;

    }

    /\*\* Returns an iterator over the elements in this list, starting at the given index. \*/

    public ListIterator listIterator(int index) {

        // If the list is empty, there is nothing to iterate

        if (size == 0) return null;

        // Gets the element in position index of this list

        Node current = first;

        int i = 0;

        while (i < index) {

            current = current.next;

            i++;

        }

        // Returns an iterator that starts in that element

        return new ListIterator(current);

    }

    public void main (String args[]) {

            String test = "committee ";

            for (int i = test.length() - 1; i >= 0; i--) {

                update(test.charAt(i));

            }

            System.out.println(this);

            System.out.println(indexOf('c'));

            remove('e');

            // System.out.println(this);

            // System.out.println(get(2));

    }

}

import java.util.HashMap;

import java.util.Random;

public class LanguageModel {

    // The map of this model.

    // Maps windows to lists of charachter data objects.

    HashMap<String, List> CharDataMap;

    // The window length used in this model.

    int windowLength;

    // The random number generator used by this model.

    private Random randomGenerator;

    /\*\* Constructs a language model with the given window length and a given

     \*  seed value. Generating texts from this model multiple times with the

     \*  same seed value will produce the same random texts. Good for debugging. \*/

    public LanguageModel(int windowLength, int seed) {

        this.windowLength = windowLength;

        randomGenerator = new Random(seed);

        CharDataMap = new HashMap<String, List>();

    }

    /\*\* Constructs a language model with the given window length.

     \* Generating texts from this model multiple times will produce

     \* different random texts. Good for production. \*/

    public LanguageModel(int windowLength) {

        this.windowLength = windowLength;

        randomGenerator = new Random();

        CharDataMap = new HashMap<String, List>();

    }

    public static void main(String[] args) {

        int windowLength = Integer.parseInt(args[0]);

        String initialText = args[1];

        int generatedTextLength = Integer.parseInt(args[2]);

        Boolean randomGeneration = args[3].equals("random");

        String fileName = args[4];

        // Create the LanguageModel object

        LanguageModel lm;

        if (randomGeneration)

        lm = new LanguageModel(windowLength);

        else

        lm = new LanguageModel(windowLength, 20);

        // Trains the model, creating the map.

        lm.train(fileName);

        // Generates text, and prints it.

        System.out.println(lm);

        System.out.println(lm.generate(initialText, generatedTextLength));

        }

    /\*\* Builds a language model from the text in the given file (the corpus). \*/

        public void train(String fileName) {

            String window = "";

            char c;

            In in = new In(fileName);

            // Reads just enough characters to form the first window

            for (int i = 0; i < windowLength; i++ ){

                window = window + in.readChar();

            }

            // Processes the entire text, one character at a time

            while (!in.isEmpty()) {

                // Gets the next character

                    c = in.readChar();

                // Checks if the window is already in the map

                List probs = CharDataMap.get(window);

                // If the window was not found in the map

                if (probs == null){

                    // Creates a new empty list, and adds (window,list) to the map

                    probs =  new List();

                    CharDataMap.put(window, probs);

                }

                // Calculates the counts of the current character.

                probs.update(c);

                // Advances the window: adds c to the window’s end, and deletes the

                // window's first character.

                window = window.substring(1) + c;

            }

             // The entire file has been processed, and all the characters have been counted.

             // Proceeds to compute and set the p and cp fields of all the CharData objects

             // in each linked list in the map.

             for (List i : CharDataMap.values()){

                  calculateProbabilities(i);

             }

            }

    // Computes and sets the probabilities (p and cp fields) of all the

    // characters in the given list. \*/

    public void calculateProbabilities(List probs) {

            ListIterator iterator = probs.listIterator(0);

            int counter = 0;

            while (iterator.hasNext()) {

                counter = counter + iterator.current.cp.count;

                iterator.next();

            }

            iterator = probs.listIterator(0);

            double dubcounter = 0;

            while (iterator.hasNext()) {

                iterator.current.cp.p =  (double)iterator.current.cp.count / counter;

                dubcounter = dubcounter + iterator.current.cp.p;

                iterator.current.cp.cp = dubcounter;

                iterator.next();

            }

    }

    // Returns a random character from the given probabilities list.

    public char getRandomChar(List probs) {

        double randomal = randomGenerator.nextDouble();

        ListIterator iterator = probs.listIterator(0);

        while ((randomal > iterator.current.cp.cp)) {

              iterator.next();

        }

        return iterator.current.cp.chr;

    }

    /\*\*

     \* Generates a random text, based on the probabilities that were learned during training.

     \* @param initialText - text to start with. If initialText's last substring of size numberOfLetters

     \* doesn't appear as a key in Map, we generate no text and return only the initial text.

     \* @param numberOfLetters - the size of text to generate

     \* @return the generated text

     \*/

    public String generate(String initialText, int textLength) {

        String str = initialText;

        if (str.length() < windowLength) {

            return str;

        }

            while(str.length() < (initialText.length() + textLength)){

                String windowtexstring = str.substring(str.length() - windowLength);

                List num =  CharDataMap.get(windowtexstring);

                if (num == null) {

                    return str;

                }

                str = str + getRandomChar(num);

            }

        return str;

    }

    /\*\* Returns a string representing the map of this language model. \*/

    public String toString() {

        StringBuilder str = new StringBuilder();

        for (String key : CharDataMap.keySet()) {

            List keyProbs = CharDataMap.get(key);

            str.append(key + " : " + keyProbs + "\n");

        }

        return str.toString();

    }

}