class Nodo {

int dato;

Nodo siguiente;

Nodo(int dato) {

this.dato = dato;

this.siguiente = null;

}

}

class Lista {

Nodo cabeza;

// a) Iterativo: pertenece

public boolean perteneceIter(int x) {

Nodo actual = cabeza;

while (actual != null) {

if (actual.dato == x) {

return true;

}

actual = actual.siguiente;

}

return false;

}

// a) Recursivo: pertenece

public boolean perteneceRec(Nodo nodo, int x) {

if (nodo == null) {

return false;

}

if (nodo.dato == x) {

return true;

}

return perteneceRec(nodo.siguiente, x);

}

// b) Iterativo: borrar

public void borrarIter(int x) {

if (cabeza == null) return;

if (cabeza.dato == x) {

cabeza = cabeza.siguiente;

return;

}

Nodo actual = cabeza;

while (actual.siguiente != null && actual.siguiente.dato != x) {

actual = actual.siguiente;

}

if (actual.siguiente != null) {

actual.siguiente = actual.siguiente.siguiente;

}

}

// b) Recursivo: borrar

public Nodo borrarRec(Nodo nodo, int x) {

if (nodo == null) return null;

if (nodo.dato == x) return nodo.siguiente;

nodo.siguiente = borrarRec(nodo.siguiente, x);

return nodo;

}

// c) Iterativo: largo

public int largoIter() {

int count = 0;

Nodo actual = cabeza;

while (actual != null) {

count++;

actual = actual.siguiente;

}

return count;

}

// c) Recursivo: largo

public int largoRec(Nodo nodo) {

if (nodo == null) {

return 0;

}

return 1 + largoRec(nodo.siguiente);

}

// d) Iterativo: snoc

public void snocIter(int x) {

Nodo nuevo = new Nodo(x);

if (cabeza == null) {

cabeza = nuevo;

return;

}

Nodo actual = cabeza;

while (actual.siguiente != null) {

actual = actual.siguiente;

}

actual.siguiente = nuevo;

}

// d) Recursivo: snoc

public Nodo snocRec(Nodo nodo, int x) {

if (nodo == null) {

return new Nodo(x);

}

nodo.siguiente = snocRec(nodo.siguiente, x);

return nodo;

}

// Métodos de apoyo para imprimir la lista

public void imprimir() {

Nodo actual = cabeza;

while (actual != null) {

System.out.print(actual.dato + " -> ");

actual = actual.siguiente;

}

System.out.println("null");

}

// Método main para probar las implementaciones

public static void main(String[] args) {

Lista lista = new Lista();

lista.snocIter(1);

lista.snocIter(2);

lista.snocIter(3);

lista.imprimir(); // 1 -> 2 -> 3 -> null

System.out.println("Largo (iterativo): " + lista.largoIter()); // 3

System.out.println("Pertenece 2 (iterativo): " + lista.perteneceIter(2)); // true

lista.borrarIter(2);

lista.imprimir(); // 1 -> 3 -> null

// Pruebas recursivas

lista.cabeza = lista.snocRec(lista.cabeza, 4);

lista.imprimir(); // 1 -> 3 -> 4 -> null

System.out.println("Largo (recursivo): " + lista.largoRec(lista.cabeza)); // 3

System.out.println("Pertenece 3 (recursivo): " + lista.perteneceRec(lista.cabeza, 3)); // true

lista.cabeza = lista.borrarRec(lista.cabeza, 3);

lista.imprimir(); // 1 -> 4 -> null

}

}

Ejercicio 2: Lista simplemente encadenada

class Lista {

Nodo cabeza;

// a) Invertir (iterativo)

public Lista invertirIter() {

Lista invertida = new Lista();

Nodo actual = cabeza;

while (actual != null) {

Nodo nuevo = new Nodo(actual.dato);

nuevo.siguiente = invertida.cabeza;

invertida.cabeza = nuevo;

actual = actual.siguiente;

}

return invertida;

}

// a) Invertir (recursivo)

public Nodo invertirRec(Nodo nodo, Nodo invertida) {

if (nodo == null) return invertida;

Nodo nuevo = new Nodo(nodo.dato);

nuevo.siguiente = invertida;

return invertirRec(nodo.siguiente, nuevo);

}

public Lista invertir() {

return new Lista(invertirRec(cabeza, null));

}

// b) Iterativo: estaOrdenada

public boolean estaOrdenadaIter() {

if (cabeza == null) return true;

Nodo actual = cabeza;

while (actual.siguiente != null) {

if (actual.dato > actual.siguiente.dato) {

return false;

}

actual = actual.siguiente;

}

return true;

}

// b) Recursivo: estaOrdenada

public boolean estaOrdenadaRec(Nodo nodo) {

if (nodo == null || nodo.siguiente == null) return true;

if (nodo.dato > nodo.siguiente.dato) return false;

return estaOrdenadaRec(nodo.siguiente);

}

// c) Iterativo: insOrd

public void insOrdIter(int elem) {

Nodo nuevo = new Nodo(elem);

if (cabeza == null || cabeza.dato >= elem) {

nuevo.siguiente = cabeza;

cabeza = nuevo;

return;

}

Nodo actual = cabeza;

while (actual.siguiente != null && actual.siguiente.dato < elem) {

actual = actual.siguiente;

}

nuevo.siguiente = actual.siguiente;

actual.siguiente = nuevo;

}

// c) Recursivo: insOrd

public Nodo insOrdRec(Nodo nodo, int elem) {

if (nodo == null || nodo.dato >= elem) {

Nodo nuevo = new Nodo(elem);

nuevo.siguiente = nodo;

return nuevo;

}

nodo.siguiente = insOrdRec(nodo.siguiente, elem);

return nodo;

}

public void insOrd(int elem) {

cabeza = insOrdRec(cabeza, elem);

}

// d) Iterativo: cuenta

public int cuentaIter(int elem) {

int count = 0;

Nodo actual = cabeza;

while (actual != null) {

if (actual.dato == elem) {

count++;

}

actual = actual.siguiente;

}

return count;

}

// d) Recursivo: cuenta

public int cuentaRec(Nodo nodo, int elem) {

if (nodo == null) return 0;

return (nodo.dato == elem ? 1 : 0) + cuentaRec(nodo.siguiente, elem);

}

// e) Iterativo: maximo

public int maximoIter() {

if (cabeza == null) throw new RuntimeException("La lista está vacía.");

int max = cabeza.dato;

Nodo actual = cabeza.siguiente;

while (actual != null) {

if (actual.dato > max) {

max = actual.dato;

}

actual = actual.siguiente;

}

return max;

}

// e) Recursivo: maximo

public int maximoRec(Nodo nodo) {

if (nodo == null) throw new RuntimeException("La lista está vacía.");

if (nodo.siguiente == null) return nodo.dato;

int max = maximoRec(nodo.siguiente);

return Math.max(nodo.dato, max);

}

// f) Iterativo: promedio

public double promedioIter() {

if (cabeza == null) throw new RuntimeException("La lista está vacía.");

int sum = 0, count = 0;

Nodo actual = cabeza;

while (actual != null) {

sum += actual.dato;

count++;

actual = actual.siguiente;

}

return (double) sum / count;

}

// f) Recursivo: promedio

public double promedioRec(Nodo nodo, int[] sumCount) {

if (nodo == null) return 0;

sumCount[0] += nodo.dato;

sumCount[1]++;

promedioRec(nodo.siguiente, sumCount);

return (double) sumCount[0] / sumCount[1];

}

public double promedio() {

int[] sumCount = {0, 0};

return promedioRec(cabeza, sumCount);

}

// g) Iterativo: tomar\_n

public int tomarNIter(int n) {

if (cabeza == null || n < 1) throw new RuntimeException("La lista no es lo suficientemente larga.");

Nodo actual = cabeza;

for (int i = 1; i < n; i++) {

if (actual.siguiente == null) throw new RuntimeException("La lista no es lo suficientemente larga.");

actual = actual.siguiente;

}

return actual.dato;

}

// g) Recursivo: tomar\_n

public int tomarNRec(Nodo nodo, int n) {

if (nodo == null || n < 1) throw new RuntimeException("La lista no es lo suficientemente larga.");

if (n == 1) return nodo.dato;

return tomarNRec(nodo.siguiente, n - 1);

}

// h) Iterativo: cambiar

public Lista cambiarIter(int n, int m) {

Lista nuevaLista = new Lista();

Nodo actual = cabeza;

while (actual != null) {

int valor = (actual.dato == n) ? m : actual.dato;

nuevaLista.snocIter(valor);

actual = actual.siguiente;

}

return nuevaLista;

}

// h) Recursivo: cambiar

public Nodo cambiarRec(Nodo nodo, int n, int m) {

if (nodo == null) return null;

int valor = (nodo.dato == n) ? m : nodo.dato;

Nodo nuevo = new Nodo(valor);

nuevo.siguiente = cambiarRec(nodo.siguiente, n, m);

return nuevo;

}

public Lista cambiar(int n, int m) {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = cambiarRec(cabeza, n, m);

return nuevaLista;

}

// Métodos de apoyo para imprimir la lista

public void imprimir() {

Nodo actual = cabeza;

while (actual != null) {

System.out.print(actual.dato + " -> ");

actual = actual.siguiente;

}

System.out.println("null");

}

// Método main para probar las implementaciones

public static void main(String[] args) {

Lista lista = new Lista();

lista.snocIter(1);

lista.snocIter(2);

lista.snocIter(3);

lista.imprimir(); // 1 -> 2 -> 3 -> null

Lista invertida = lista.invertirIter();

invertida.imprimir(); // 3 -> 2 -> 1 -> null

System.out.println("Esta ordenada (iterativo): " + lista.estaOrdenadaIter()); // true

lista.insOrdIter(0);

lista.imprimir(); // 0 -> 1 -> 2 -> 3 -> null

System.out.println("Cuenta 2 (iterativo): " + lista.cuentaIter(2)); // 1

System.out.println("Maximo (iterativo): " + lista.maximoIter()); // 3

System.out.println("Promedio (iterativo): " + lista.promedioIter()); // 1.5

System.out.println("Elemento en posición 2 (iterativo): " + lista.tomarNIter(2)); // 1

Lista cambiada = lista.cambiarIter(1, 5);

cambiada.imprimir(); // 0 -> 5 -> 2 -> 3 -> null

}

}

Ejercicio 3: Lista simplemente encadenada con suma acumulativa

class Lista {

Nodo cabeza;

// i) Iterativo: sumaAcum

public Lista sumaAcumIter() {

Lista nuevaLista = new Lista();

if (cabeza == null) return nuevaLista;

int suma = 0;

Nodo actual = cabeza;

while (actual != null) {

suma += actual.dato;

nuevaLista.snocIter(suma);

actual = actual.siguiente;

}

return nuevaLista;

}

// i) Recursivo: sumaAcum

public Nodo sumaAcumRec(Nodo nodo, int suma) {

if (nodo == null) return null;

suma += nodo.dato;

Nodo nuevo = new Nodo(suma);

nuevo.siguiente = sumaAcumRec(nodo.siguiente, suma);

return nuevo;

}

public Lista sumaAcum() {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = sumaAcumRec(cabeza, 0);

return nuevaLista;

}

// j) Iterativo: concat

public Lista concatIter(Lista otraLista) {

Lista nuevaLista = new Lista();

Nodo actual = cabeza;

while (actual != null) {

nuevaLista.snocIter(actual.dato);

actual = actual.siguiente;

}

actual = otraLista.cabeza;

while (actual != null) {

nuevaLista.snocIter(actual.dato);

actual = actual.siguiente;

}

return nuevaLista;

}

// j) Recursivo: concat

public Nodo concatRec(Nodo nodo1, Nodo nodo2) {

if (nodo1 == null) return nodo2;

nodo1.siguiente = concatRec(nodo1.siguiente, nodo2);

return nodo1;

}

public Lista concat(Lista otraLista) {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = concatRec(cabeza, otraLista.cabeza);

return nuevaLista;

}

// k) Iterativo: prefijo

public Lista prefijoIter(int n) {

Lista nuevaLista = new Lista();

Nodo actual = cabeza;

for (int i = 0; i < n && actual != null; i++) {

nuevaLista.snocIter(actual.dato);

actual = actual.siguiente;

}

return nuevaLista;

}

// k) Recursivo: prefijo

public Nodo prefijoRec(Nodo nodo, int n) {

if (nodo == null || n <= 0) return null;

Nodo nuevo = new Nodo(nodo.dato);

nuevo.siguiente = prefijoRec(nodo.siguiente, n - 1);

return nuevo;

}

public Lista prefijo(int n) {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = prefijoRec(cabeza, n);

return nuevaLista;

}

// Métodos de apoyo para imprimir la lista

public void imprimir() {

Nodo actual = cabeza;

while (actual != null) {

System.out.print(actual.dato + " -> ");

actual = actual.siguiente;

}

System.out.println("null");

}

// Método main para probar las implementaciones

public static void main(String[] args) {

Lista lista = new Lista();

lista.snocIter(1);

lista.snocIter(2);

lista.snocIter(3);

lista.imprimir(); // 1 -> 2 -> 3 -> null

Lista sumaAcum = lista.sumaAcumIter();

sumaAcum.imprimir(); // 1 -> 3 -> 6 -> null

// Probar suma acumulativa recursiva

Lista sumaAcumRec = lista.sumaAcum();

sumaAcumRec.imprimir(); // 1 -> 3 -> 6 -> null

// Concatenación de listas

Lista otraLista = new Lista();

otraLista.snocIter(4);

otraLista.snocIter(5);

otraLista.snocIter(6);

Lista concatenadaIter = lista.concatIter(otraLista);

concatenadaIter.imprimir(); // 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null

Lista concatenadaRec = lista.concat(otraLista);

concatenadaRec.imprimir(); // 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null

// Prefijo de la lista

Lista prefijoIter = lista.prefijoIter(2);

prefijoIter.imprimir(); // 1 -> 2 -> null

Lista prefijoRec = lista.prefijo(2);

prefijoRec.imprimir(); // 1 -> 2 -> null

}

}

class Nodo {

int dato;

Nodo siguiente;

public Nodo(int dato) {

this.dato = dato;

this.siguiente = null;

}

}

class Lista {

Nodo cabeza;

// a) Iterativo: snoc

public void snocIter(int dato) {

Nodo nuevo = new Nodo(dato);

if (cabeza == null) {

cabeza = nuevo;

} else {

Nodo actual = cabeza;

while (actual.siguiente != null) {

actual = actual.siguiente;

}

actual.siguiente = nuevo;

}

}

// a) Recursivo: snoc

public Nodo snocRec(Nodo nodo, int dato) {

if (nodo == null) return new Nodo(dato);

nodo.siguiente = snocRec(nodo.siguiente, dato);

return nodo;

}

public void snoc(int dato) {

cabeza = snocRec(cabeza, dato);

}

// b) Iterativo: invertir

public Lista invertirIter() {

Lista invertida = new Lista();

Nodo actual = cabeza;

while (actual != null) {

Nodo nuevo = new Nodo(actual.dato);

nuevo.siguiente = invertida.cabeza;

invertida.cabeza = nuevo;

actual = actual.siguiente;

}

return invertida;

}

// b) Recursivo: invertir

public Nodo invertirRec(Nodo nodo, Nodo previo) {

if (nodo == null) return previo;

Nodo siguiente = nodo.siguiente;

nodo.siguiente = previo;

return invertirRec(siguiente, nodo);

}

public Lista invertir() {

Lista invertida = new Lista();

invertida.cabeza = invertirRec(cabeza, null);

return invertida;

}

// c) Iterativo: insOrd

public void insOrdIter(int dato) {

Nodo nuevo = new Nodo(dato);

if (cabeza == null || cabeza.dato >= dato) {

nuevo.siguiente = cabeza;

cabeza = nuevo;

} else {

Nodo actual = cabeza;

while (actual.siguiente != null && actual.siguiente.dato < dato) {

actual = actual.siguiente;

}

nuevo.siguiente = actual.siguiente;

actual.siguiente = nuevo;

}

}

// c) Recursivo: insOrd

public Nodo insOrdRec(Nodo nodo, int dato) {

if (nodo == null || nodo.dato >= dato) {

Nodo nuevo = new Nodo(dato);

nuevo.siguiente = nodo;

return nuevo;

}

nodo.siguiente = insOrdRec(nodo.siguiente, dato);

return nodo;

}

public void insOrd(int dato) {

cabeza = insOrdRec(cabeza, dato);

}

// d) Iterativo: cuenta

public int cuentaIter(int dato) {

int contador = 0;

Nodo actual = cabeza;

while (actual != null) {

if (actual.dato == dato) contador++;

actual = actual.siguiente;

}

return contador;

}

// d) Recursivo: cuenta

public int cuentaRec(Nodo nodo, int dato) {

if (nodo == null) return 0;

return (nodo.dato == dato ? 1 : 0) + cuentaRec(nodo.siguiente, dato);

}

public int cuenta(int dato) {

return cuentaRec(cabeza, dato);

}

// e) Iterativo: maximo

public int maximoIter() {

if (cabeza == null) throw new RuntimeException("La lista está vacía.");

int maximo = cabeza.dato;

Nodo actual = cabeza.siguiente;

while (actual != null) {

if (actual.dato > maximo) maximo = actual.dato;

actual = actual.siguiente;

}

return maximo;

}

// e) Recursivo: maximo

public int maximoRec(Nodo nodo) {

if (nodo == null) return Integer.MIN\_VALUE;

return Math.max(nodo.dato, maximoRec(nodo.siguiente));

}

public int maximo() {

return maximoRec(cabeza);

}

// f) Iterativo: promedio

public double promedioIter() {

if (cabeza == null) throw new RuntimeException("La lista está vacía.");

int suma = 0, cuenta = 0;

Nodo actual = cabeza;

while (actual != null) {

suma += actual.dato;

cuenta++;

actual = actual.siguiente;

}

return (double) suma / cuenta;

}

// f) Recursivo: promedio

public double promedioRec(Nodo nodo, int[] sumCount) {

if (nodo == null) return sumCount[0];

sumCount[0] += nodo.dato;

sumCount[1]++;

return promedioRec(nodo.siguiente, sumCount) / sumCount[1];

}

public double promedio() {

int[] sumCount = {0, 0};

return promedioRec(cabeza, sumCount);

}

// g) Iterativo: tomar\_n

public int tomarNIter(int n) {

if (cabeza == null || n < 1) throw new RuntimeException("La lista no es lo suficientemente larga.");

Nodo actual = cabeza;

for (int i = 1; i < n; i++) {

if (actual.siguiente == null) throw new RuntimeException("La lista no es lo suficientemente larga.");

actual = actual.siguiente;

}

return actual.dato;

}

// g) Recursivo: tomar\_n

public int tomarNRec(Nodo nodo, int n) {

if (nodo == null || n < 1) throw new RuntimeException("La lista no es lo suficientemente larga.");

if (n == 1) return nodo.dato;

return tomarNRec(nodo.siguiente, n - 1);

}

// h) Iterativo: cambiar

public Lista cambiarIter(int n, int m) {

Lista nuevaLista = new Lista();

Nodo actual = cabeza;

while (actual != null) {

int valor = (actual.dato == n) ? m : actual.dato;

nuevaLista.snocIter(valor);

actual = actual.siguiente;

}

return nuevaLista;

}

// h) Recursivo: cambiar

public Nodo cambiarRec(Nodo nodo, int n, int m) {

if (nodo == null) return null;

int valor = (nodo.dato == n) ? m : nodo.dato;

Nodo nuevo = new Nodo(valor);

nuevo.siguiente = cambiarRec(nodo.siguiente, n, m);

return nuevo;

}

public Lista cambiar(int n, int m) {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = cambiarRec(cabeza, n, m);

return nuevaLista;

}

// i) Iterativo: sumaAcum

public Lista sumaAcumIter() {

Lista nuevaLista = new Lista();

if (cabeza == null) return nuevaLista;

int suma = 0;

Nodo actual = cabeza;

while (actual != null) {

suma += actual.dato;

nuevaLista.snocIter(suma);

actual = actual.siguiente;

}

return nuevaLista;

}

// i) Recursivo: sumaAcum

public Nodo sumaAcumRec(Nodo nodo, int suma) {

if (nodo == null) return null;

suma += nodo.dato;

Nodo nuevo = new Nodo(suma);

nuevo.siguiente = sumaAcumRec(nodo.siguiente, suma);

return nuevo;

}

public Lista sumaAcum() {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = sumaAcumRec(cabeza, 0);

return nuevaLista;

}

// j) Iterativo: concat

public Lista concatIter(Lista otraLista) {

Lista nuevaLista = new Lista();

Nodo actual = cabeza;

while (actual != null) {

nuevaLista.snocIter(actual.dato);

actual = actual.siguiente;

}

actual = otraLista.cabeza;

while (actual != null) {

nuevaLista.snocIter(actual.dato);

actual = actual.siguiente;

}

return nuevaLista;

}

// j) Recursivo: concat

public Nodo concatRec(Nodo nodo1, Nodo nodo2) {

if (nodo1 == null) return nodo2;

nodo1.siguiente = concatRec(nodo1.siguiente, nodo2);

return nodo1;

}

public Lista concat(Lista otraLista) {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = concatRec(cabeza, otraLista.cabeza);

return nuevaLista;

}

// k) Iterativo: prefijo

public Lista prefijoIter(int n) {

Lista nuevaLista = new Lista();

Nodo actual = cabeza;

for (int i = 0; i < n && actual != null; i++) {

nuevaLista.snocIter(actual.dato);

actual = actual.siguiente;

}

return nuevaLista;

}

// k) Recursivo: prefijo

public Nodo prefijoRec(Nodo nodo, int n) {

if (nodo == null || n <= 0) return null;

Nodo nuevo = new Nodo(nodo.dato);

nuevo.siguiente = prefijoRec(nodo.siguiente, n - 1);

return nuevo;

}

public Lista prefijo(int n) {

Lista nuevaLista = new Lista();

nuevaLista.cabeza = prefijoRec(cabeza, n);

return nuevaLista;

}

// Métodos de apoyo para imprimir la lista

public void imprimir() {

Nodo actual = cabeza;

while (actual != null) {

System.out.print(actual.dato + " -> ");

actual = actual.siguiente;

}

System.out.println("null");

}

// Método main para probar las implementaciones

public static void main(String[] args) {

Lista lista = new Lista();

lista.snocIter(1);

lista.snocIter(2);

lista.snocIter(3);

lista.imprimir(); // 1 -> 2 -> 3 -> null

Lista sumaAcum = lista.sumaAcumIter();

sumaAcum.imprimir(); // 1 -> 3 -> 6 -> null

Lista sumaAcumRec = lista.sumaAcum();

sumaAcumRec.imprimir(); // 1 -> 3 -> 6 -> null

Lista otraLista = new Lista();

otraLista.snocIter(4);

otraLista.snocIter(5);

otraLista.snocIter(6);

Lista concatenadaIter = lista.concatIter(otraLista);

concatenadaIter.imprimir(); // 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null

Lista concatenadaRec = lista.concat(otraLista);

concatenadaRec.imprimir(); // 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null

Lista prefijoIter = lista.prefijoIter(2);

prefijoIter.imprimir(); // 1 -> 2 -> null

Lista prefijoRec = lista.prefijo(2);

prefijoRec.imprimir(); // 1 -> 2 -> null

}

}