Laboratorio 3

Pilas y la transformación recursivo a iterativo

Estructura de Dades

2n Curs GEI
Grup 2

24/11/2020

Ton Llucià Senserrich
47125160T

Eduard Sales Jové
49539818A

Exercici 1

public void push(E elem)

Declarem un nou element, i li ficarem el valor de "E elem" que és el que ens passen per paràmetre, i l'afegirem a la capçalera de la llista modificant la referència de l'objecte nou per tal que apunti al següent objecte de la pila.

public E top()

Comprova que la llista no esta buida, i en cas que no ho estigui retorna el valor que hi ha en la posició top.

public void pop()

En primer lloc comprovarem que la llista no esta buida
en cas que no ho estigui executem la següent línia "top=top.link" la qual fica el següent objecte a la posició top, eliminant així la referència a l'objecte top de l'stack.

public boolean isEmpty()

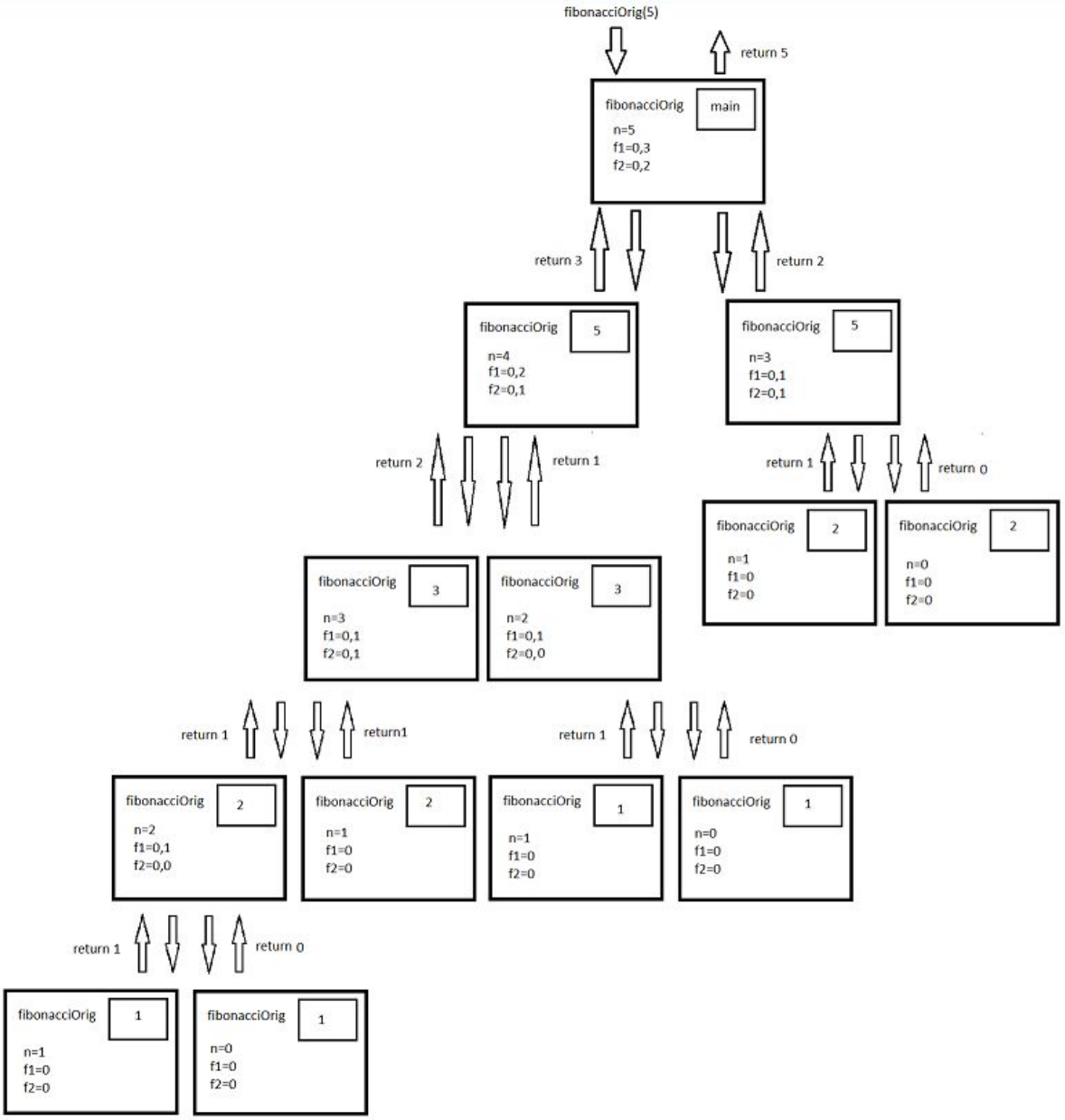
La funció retornarà true en cas que top sigui null.

Exercici 2

Per tal de transformar el codi en iteratiu afegim unes variables locals per tal de treballar amb més facilitat.

```
1. public static int fibonacciOrig(int n) {
2.      int f1 = 0,f2 = 0;
3.      if (n <= 1)
4.              return n;
5.      else {
6.              f1 = fibonacciOrig(n - 1);
7.              f2 = fibonacciOrig(n - 2);
7.              return f1+f2;
8.      }
9. }
```

I això es podria representar en un gràfic de crides en forma d'arbre, per exemple representem la crida a la funció fibonacciOrig(5).



Bàsicament la transformació a iteratiu consisteix en la implementació d'una pila i l'ús d'una classe per tal d'encapsular les crides que recursivament es fan a la funció.

Aquesta classe d'encapsulació consta de tres tipus de crida:

-call: conte la crida al cas simple, és a dir si el valor de n és més petit o igual a 1 retornarà 1 i eliminarà l'element actual de la pila, i si pel contrari no és el cas simple el manté a la cua però li canvia el tipus de crida a resume2 i afegeix un altre element a la cua amb n=n-1 i amb el tipus de crida call.

-resume1: conte la part de l'algorisme en què es fica a la cua l'element amb valor n=n-2 i es modifica l'element actual assignant f1=return_.
-resume2: és la funció de sortida de l'algorisme, on se sumen les variables f1 i f2 i es posa el resultat a return_ per tal de fer-lo servir després i elimina un element de la llista.

Exercici 3

partition amb while(partition2)

Mentre que el inf sigui més petit que sup farem algunes comprovacions:
-En cas que el valor en l'array v de posició inf sigui més petit o igual que el valor de pivot, augmentarem inf per tal de mirar que els valors de l'esquerra siguin més petits que pivot.

-En cas que el valor en l'array v de posició sup-1 sigui més gran que pivot, disminuirem sup, per comprovar que els valors de la dreta siguin més grans que pivot.

-Repetirem això fins que no trobi cap valor que compleixi el requisit, en aquest moment farem un swap entre inf i sup-1, i augmentarem i disminuirem respectivament inf i sup, perquè voldrà dir que hem trobat un valor més gran que pivot a l'esquerra i un de més petit a la dreta, cosa que voldrà dir que eols podem intercanviar.

Un cop acabem el bucle retornarem inf que serà la posició per la qual dividirem l'array en la següent iteració.

funció quick sort iteratiu(quickSortIter)

Aquesta funció l'hem implementat de manera que al case RESUME el que fem és eliminar l'element que ja hem tractat abans al case CALL i afegim a la cua les parts de la dreta i de l'esquerra per tal de tractar-les després a call, al case call el que fem és trobar el pivot i el seu valor, apartem el pivot i cridem a la funció partition per tal d'ordenar aquella part de l'array i trobem el position que és la posició per on haurem de partir seguidament en l'altre case.

Hem tingut problemes a l'hora de pensar la implementació, ja que era diferent dels exemples fets anteriorment, al principi va costar, però després ens vam adonar que la qüestió era cridar a partition i seguidament eliminar l'element i crear els dos resultants de la partició de l'array.