Ingeniería en Informática de Sistemas. Curso 2002/03

Estructuras de Datos y de la Información (Grupo B)

Segundo Parcial. Examen Final Junio 2003

Ejercicio 1. (3 puntos) El árbol de Fibonacci es un árbol binario de números enteros que se define de la siguiente manera: Si n = 0 ó n = 1, el árbol consta de un único nodo. Si n > 1, el árbol consta de un nodo raíz con el árbol de Fibonacci de orden n-1 como hijo izquierdo y el árbol de Fibonacci de orden n-2 como hijo derecho.

Se pide:

a.) Implementar en C++ el árbol de Fibonacci.

b.) Calcular cuantas hojas hay en el árbol de Fibonacci de orden n.

c.) Calcular cual es la profundidad del árbol de Fibonacci de orden n.

Ejercicio 2. (2 puntos) Dos árboles binarios son similares como espejos si ambos están vacíos o si ambos no están vacíos y el subárbol izquierdo de cada uno de ellos es un reflejo igual al del subárbol derecho del otro. Implementar un método en C++ para determinar si dos árboles son similares como espejos.

Ejercicio 3. (5 puntos) Implementar en C++, utilizando otros TADs conocidos, el TAD MERCADO que modela el comportamiento de un *mercado de trabajo* simplificado, donde las personas pueden ser contratadas y despedidas por las empresas. La especificación de MERCADO usará (entre otros) los TADs PERSONA (con tipo principal *Persona*) y EMPRESA (con tipo principal *Empresa*), que se suponen ya implementados. Ambos tipos están equipados con las operaciones de igualdad y orden. Se desea que MERCADO ofrezca un tipo principal *Mercado* junto con las operaciones que se describen a continuación:

- *Nuevo* : Genera un mercado vacío, sin ninguna información.
- *Contrata* : Altera un mercado, efectuando la contratación de cierta persona como empleado de cierta empresa.
- *despide* : Altera un mercado, efectuando el despido de cierta persona que era antes empleado de cierta empresa.
- *empleados* : Consulta los empleados de una empresa, devolviendo el resultado como secuencia ordenada de personas.
- *esEmpleado* : Averigua si es cierto o no que una persona dada es empleado de una empresa dada.
- *esPluriempleado* : Averigua si es cierto o no que una persona es empleado de más de una empresa.

Soluciónes:

Ejercicio 1.

a.)

```
TArbinDinamico<int> arbolFibonacci (int n){
   TArbinDinamico<int> res;

   if ((n==0) || (n==1))
      res = TArbinDinamico(0, n, 0);
   else
      res = TArbinDinamico(arbolFibonacci(n-1), n, arbolFibonacci(n-2));
   return res;
}
```

b.)

Número de hojas = $\begin{cases} 1 & n=0 \\ fib(n)+fib(n-1) & n>0 \end{cases}$

Donde fib(n) es la función de Fibonacci

c.) Profundidad = n

Ejercicio 2.

```
template <class TElem>
bool TArbinDinamico<TElem>::similares(const TArbinDinamico<TElem> & ar){
  if (esVacio() && ar.esVacio())
    return true;
  else
    return  raiz()==ar.raiz() && hijoIz().similares(ar.hijoDr()) &&
            hijoDr().similares(ar.hijoIz());
}
```

Ejercicio 3.

```
class TMercado{
public:
  TMercado ();
  // ~TMercado ();
  void contrata (const TEmpresa&, const TEmpleado&) throw EAccesoIndebido();
  void despide (const TEmpresa&, const TEmpleado&) throw EAccesoIndebido();
  TSecuenciaDinamica<TEmpleado> empleados (const TEmpresa&);
  bool esEmpleado (const TEmpresa&, const TEmpleado&);
  bool esPluiempleado (const TEmpleado&);

private:
  TArbus<TEmpresa,TSecuenciaDinamica<TEmpleado>> _mercado;
  TArbus<TEmpleado,TsecuenciaDinamica<TEmpresa>> _empleados;
}
```

```
TMercado::TMercado (){
  _mercado = new TArbus<TEmpresa,TSecuenciaDinamica<TEmpleado>> ();
  _empleados = new TArbus<TEmpleado,TsecuenciaDinamica<TEmpresa>> ();
}

void TMercado::contrata (const TEmpresa& e, const TEmpleado& p)
  throw EAccesoIndebido(){

  TSecuenciaDinamica<TEmpleado> xemp;
  TSecuenciaDinamica<TEmpresa> xco;

  xemp = _mercado.consulta(e);
  xemp.reinicia();
  while ((!xemp.esfin()) && (xemp.actual() < p))
    xemp.avanza();
  if ((!xemp.esfin()) && (xemp.actual() == p))
    throw EAccesoIndebido();
  else
    xemp.inserta(p);
  _mercado.inserta(e, xemp);
  xco = _empleados.consulta(p);
  xco.reinicia();
  xco.inserta(e);
  _empleados.inserta(p, xco);
}

void TMercado::despide (const TEmpresa&, const TEmpleado&)
  throw EAccesoIndebido(){

  TSecuenciaDinamica<TEmpleado> xemp;
  TSecuenciaDinamica<TEmpresa> xco;

  xemp = _mercado.consulta(e);
  xemp.reinicia();
  while ((!xemp.esfin()) && (!(xemp.actual() == p)))
    xemp.avanza();
  if (xemp.esfin())
    throw EAccesoIndebido();
  else
    xemp.borra();
  _mercado.inserta(e, xemp);
  xco = _empleados.consulta(p);
  xco.reinicia();
  while ((!xco.esfin()) && (!(xco.actual() == e)))
    xco.avanza();
  xco.borra();
  _empleados.inserta(p, xco);
}

TSecuenciaDinamica<TEmpleado> TMercado::empleados (const TEmpresa&){
  return _mercado.consulta(e);
}
```



```
bool TMercado::esEmpleado (const TEmpresa&, const TEmpleado&){
  TSecuenciaDinamica<TEmpleado> xemp;

  xemp = _mercado.consulta(e);
  xemp.reinicia();
  while ((!xemp.esfin()) && (xemp.actual() < p))
    xemp.avanza();
  return (!xemp.esfin() && (xemp.actual() == p));
}

bool TMercado::esPluiempleado (const TEmpleado&){
  TSecuenciaDinamica<TEmpresa> xco;
  int cont = 0;

  xco = _empleados.consulta(p);
  xco.reinicia();
  while (!xco.esfin()){
    cont++;
    xco.avanza();
  }
  return (cont > 1);
}
```