EXAMEN 2016-2017 Julio

Ejercicio 1: ¿Qué TDA elegirías?

a) Implementar un conjunto ordenado de enteros (con funciones básicas de insertar, borrar, y buscar) de entre: un ABB (árbol binario de búsqueda no balanceado), un APO(árbol parcialmente ordenado implementado mediante un heap) y un vector ordenado.

Vector ordenado:

- Buscar $O(log_2 n)$
- Insertar $O(n)$
- Borrar $O(n)$

ABB:

- Buscar $O(n)$
- Insertar $O(n)$
- Borrar $O(n)$

Claramente en un APO, como lo tratamos como un heap, es menos eficiente (tripleee).

Ejercicio 2:

Una lista L se dice que está ordenada de forma ascendente por índices (orden indexado) si la secuencia {L(indices(0)),L(indices(1)),...L(indices(indices.size()-1))}. (siendo L(i) el elemento en la posición i-ésima), es una secuencia ordenada de forma ascendente. Necesariamente indices debe ser una lista de enteros que indica una secuencia de índices en L.

Implementa una función booleana que reciba L (L puede ser cualquier tipo de dato) y un índice I y devuelva true si L está ordenada de forma ascendente con respecto al índice.

```
template <class T>
T& obtener_elemento(list<T> &L, int pos){
    list<T>::iterator it =L.begin();
    for (int i =0; i < pos; i++)
        it++:
    return *it;
}

template<class T>
bool ordenada(list<T> &L, list<int> &indices){

    list<int>::iterator it;
    T anterior;
    bool ordenada=true;
```

```
    for (it=indices.begin(); it!= indices.end() && ordenada; it++){
        if (obtener_elemento(L, *it) < anterior)
            ordenada=false;
        anterior=obtener_elemento(L, *it);
    }
    return ordenada;
}
```

Ejercicio 3: true si un árbol binario de enteros ab2 coincide con un subárbol de otro ab1 (deben coincidir también los valores de las etiquetas).

```
bool es_subarbol(const bintree<int> &ab1, const bintree<int> &ab2){
    bintree<int>::nodo n =ab1.raiz();
    bintree<int> der, izq;
    if (ab1==ab2)
        return true;
    else{
        podar_derecha(n, der);
        podar_izquierda(n, izq);
        return (es_subarbol(der, ab2) || es_subarbol(izq, ab2));

    }

}
```

Ejercicio 3: Implementa el TDA cola con dos pilas

```
template<class T>
class Cola{

    private:
        stack<T> pila1; //se insertan aqui
        stack<T> pila2; //se borran de aqui
    public:

        Cola();
        Cola(const Cola() c);
        ~Cola();

        bool vacia() const;
        void poner(const T& valor);
        void quitar();
        T& frente() const;
}

bool Cola::vacia()const{
    if (pila1.empty() && pila2.empty())
```

```
        return true;
    else
        return false;
}

void Cola::poner(const T& valor){
    stack<T>otro;
    pila2.push(valor);
    otro = pila2;
    pila1.clear();
    while(!otro.empty()){
        pila1.push(otro.top());
        otro.pop();
    }
}

void quitar(){
    stack<T> otro;
    pila2.pop();
    otro = pila2;
    while(!otro.empty()){
        pila1.push(otro.top());
        otro.pop();
    }
}

T& frente(){
    return pila1.top();

}
```

Ejercicio 5: TDA Índice

```
class indices{
    private:
        map<string, set<int>> datos;

    public:


        class iterator{

            private:
                map<string,set<int>>::iterator it;
                set<int>::iterator ot;
            public:
                iterator():it(0),ot(0);
                iterator(const iterator &i){it=i.it; ot=i.ot;};
                iterator& operator=;
                pair<string, set<int>>& operator *(){return *it;};
```

```
                bool operator ==(const iterator &i){return (it==i.it &&
ot== i.ot);}
                                                                    };
                bool operator !=...->


                iterator & operator ++(){
                    ot++;
                    while (*ot%2 == 0  || ot==*it.second.end()){

                        if (ot==*it.second.end()){
                            it++;
                            ot= it.begin();
                        }
                        else{
                           ot++;
                        }
                    }
                }
    };

    iterator& begin(){
        indices::iterator i;
        i.it=datos.begin();
        i.ot=*(i.it).begin();
        while (*ot%2 == 0  || ot==*it.second.end()){
            if (ot==*it.second.end()){
                it++;
                ot= it.begin();
            }
            else{
                ot++;
            }
        }
        return i;
};
```