PRÁCTICA Nº 7: 1 sesión

(del 31 de Mayo al 4 de Junio de 2004)

Montículos. Simulación del funcionamiento de un ascensor

0. OBJETIVO

Esta práctica tiene dos objetivos:

- La manipulación de árboles binarios representados mediante vectores.
- La utilización de montículos (*heaps*) como colas de prioridad.

1. INTRODUCCIÓN TEÓRICA

Un montículo de mínimos, *min heap*, es un árbol binario completo tal que, el valor de la clave de cada nodo es menor o igual que las claves de sus nodos hijos (si los tiene). Además, sus hijos cumplen la propiedad de montículo de mínimos.

De la definición se deduce que la clave contenida en el nodo raíz de un *min heap* es la menor clave del árbol. Pero esto no quiere decir que sea la única con ese valor.

La estructura montículo tiene interesantes aplicaciones, por ejemplo, la ordenación de arrays (algoritmo *heapsort*) o el mantenimiento de las llamadas colas de prioridad. Las colas de prioridad son un tipo especial de colas donde todo elemento que se inserta en la cola lleva asociado una prioridad, de forma que el elemento que se borra de la cola es el primero entre los que tengan la mínima (o máxima) prioridad.

Los montículos, como árboles binarios completos que son, se pueden representar de forma eficaz mediante una estructura secuencial (un array). La interfaz de la clase será la siguiente:

```
class Monticulo //de mínimos
{
    public:
        typedef int Valor;
        Monticulo (void);
        bool Insertar (Valor);
        bool EliminarMinimo (void);
        bool Minimo (Valor &);
        bool MonticuloVacio (void);
    private:
        typedef Valor Vector [MAX];
        Vector info;
        int fin;
};
```

2. PLANTEAMIENTO DE LA PRÁCTICA

Como ejemplo de utilización de la clase *Monticulo*, se propone plantear la simulación del funcionamiento de un ascensor:

Una empresa instaladora de ascensores, nos encarga diseñar el sistema de control para sus ascensores. De forma que, podamos instalarlos en cualquier edificio con tal de indicar, al principio, el número de plantas de las que consta dicho edificio (incluyendo la planta baja) y con un máximo de 64 plantas.

La filosofía de funcionamiento del ascensor es la siguiente:

(a) Todas las peticiones realizadas por los usuarios son tratadas de igual forma, no importa que se trate de llamadas internas o externas (al ascensor).

(b) Para que el movimiento del ascensor se realice de manera eficiente, las llamadas se priorizarán y serán atendidas en función de la prioridad asociada. Este valor dependerá fundamentalmente de la posición actual del ascensor y del sentido de la marcha. En concreto, el algoritmo para asignar valores de prioridad a las llamadas es el siguiente:

Sean,

sentido igual a +1 si el ascensor sube y -1 si el ascensor está bajando

piso el valor numérico del piso desde donde se llama al ascensor

actual el valor numérico del piso donde se encuentra actualmente el ascensor

Entonces,

Si (se puede llegar a ***piso*** desde ***actual*** siguiendo el sentido de la marcha)

prioridad ← piso *sentido

sino //es decir, es preciso cambiar el sentido de la marcha

prioridad ← piso * (-sentido)

Nota 1: La condición (*se puede llegar a piso desde actual siguiendo el sentido de la marcha*) se puede evaluar como: (*piso * sentido > actual * sentido*)

Nota 2: Se pueden generar valores negativos, pero esto no supone ningún problema.

Como la medida de prioridad utilizada está relacionada con lo cerca que está el piso de la posición actual, se interpreta como mejor prioridad el valor de prioridad mínimo.

3. REALIZACIÓN DE LA PRÁCTICA

Con este planteamiento, hay que escribir un programa que simule el funcionamiento del ascensor de forma que las peticiones se realicen mediante pulsaciones del teclado y después se mueva el ascensor a los pisos que se han solicitando previamente.

Para gestionar las llamadas solicitadas y la prioridad asignada a cada una de ellas, es preciso mantener 2 colas de prioridad (montículos de mínimos). Una de las colas almacenará las peticiones que se pueden atender siguiendo el sentido actual de la marcha y la otra cola guardará las peticiones que requieren que se cambie el sentido de la marcha del ascensor para ser atendidas.

Hay que tener en cuenta que, antes de visualizar el movimiento del ascensor, recogeremos todas las peticiones de pisos en los que queremos parar. Una vez terminado todo el recorrido, daremos la opción de volver a introducir nuevas peticiones. Para ello se propone que, mientras no se introduzca el carácter '$' se supone que son peticiones realizadas en el mismo momento y que se visualizarán en conjunto. Utilizaremos el carácter '-' cuando deseemos salir del programa.

Las llamadas se simularán con pulsaciones de teclas de la siguiente manera: Teclas '0'..'9' indican pisos del 0 al 9, teclas 'A'..'Z' indican pisos del 10 al 36 (incluir 'Ñ' que debe corresponder con el piso 24) y teclas 'a'..'z' indican pisos del 37 al 63 (incluir 'ñ' que corresponde con el piso 51). Por tanto, habrá que realizar una función que convierta estos caracteres a los números de piso correspondientes para poder utilizar la función de visualización del movimiento del ascensor.

La visualización del proceso se realizará utilizando **la clase proporcionada por el profesor**, llamada `Ascensor.cpp` y que contiene dos métodos básicos (se recomienda consultar el resto de los métodos en el propio archivo fuente):

> **`void DibujarEdificio(int plantas, int ventanas, int ascensor)`** muestra por pantalla el dibujo del edificio, indicándole el número de plantas del edificio y el número de ventanas en cada planta, junto con la posición inicial en la que se encuentra el ascensor.
>
> **`int MoverAscensor(int inicial, int final)`** muestra por pantalla el movimiento del ascensor indicándole el estado inicial y final del movimiento.

El esquema algorítmico básico de la simulación será el siguiente:

> 1) Leer el número de plantas y de ventanas que tendrá el edificio y dibujarlo.
>
> Mientras no sea el final (indicado por el carácter '-') hacer:
>
> > 2) Leer y almacenar en la cola correspondiente todas las peticiones que se introduzcan por teclado (el final de datos se indicará mediante el carácter '$'), calculando la prioridad que tendrá cada una en función de la posición del ascensor y del sentido de la marcha.
> >
> > 3) Mover el ascensor para atender todas las peticiones almacenadas. Primero se atenderán todas las peticiones en el sentido de la marcha del ascensor y después, si es necesario, se cambiará el sentido de la marcha del ascensor para atender al resto de las peticiones. En cualquier caso, el ascensor siempre se debe desplazar al piso más próximo (menor prioridad).
>
> Fin_mientras

Hay que tener en cuenta que los montículos almacenan básicamente el valor del piso desde donde se realiza la llamada pero con signo positivo o negativo, dependiendo del sentido de la marcha. Esto quiere decir que a la hora de extraer valores de los montículos para mover el ascensor, habrá que considerar el valor absoluto de las prioridades, puesto que no hay pisos negativos.

Nota: Se debe incluir en el proyecto, además de la clase `ascensor`, el fichero "`conio.c`" proporcionados por el profesor.

4. ENTREGA DEL PROGRAMA

La entrega de esta práctica debe incluir **el código** con el programa principal y las funciones del programa que implemente el funcionamiento del ascensor (**`##movimiento.cpp`**) que contendrá básicamente el esquema algorítmico antes mencionado, y la implementación de la clase 'Monticulo' (**`##monticulo.cpp`** y **`##monticulo.h`**).

ENTREGA DE PROGRAMAS:
Antes de (11 de Junio de 2004)