Práctica sobre Estrategias Algorítmicas

Objetivos:

- Practicar la estrategia algorítmica voraz para un problema real.

Programa a realizar:

Se deberá desarrollar un programa que devuelva una determinada cantidad de dinero tratando de minimizar el número de monedas o billetes que se devuelven. El programa permitirá dos modos de operación: a) monedas/billetes infinitos (donde se supone que no tenemos limitación en el número de monedas/billetes a devolver de cualquier tipo) y b) monedas/billetes limitados (donde gestionaremos una cantidad de monedas o billetes de cada tipo).

Pasos a seguir:

El programa preguntará inicialmente el modo de operación y, seguidamente, de manera iterativa irá preguntando cantidades a devolver (el usuario introducirá un 0 cuando quiera salir del programa) e informando del número de monedas/billetes de cada tipo que se deben devolver.

Para el caso de **monedas/billetes infinitos** la implementación de la función de devolver cambio ya se ha visto en clase de teoría para arrays:

```
int cambio(int x, int valor[n],int solucion[n])
{
for (int i=0;i<n;i++) solucion[i]=0;
int i=0, suma=0;
while (suma<x && i<n)
        if (suma+valor[i]<=x)
                {
                solucion[i]++;
                suma+=valor[i];
                }
        else
                i++;
if (suma==x) return 1;
else
        {
        for (int i=0;i<n;i++) solucion[i]=0;
        return 0;
        }
}
```

Práctica sobre Estrategias Algorítmicas

Cambios a realizar sobre esta implementación:

- **gestión de valores de monedas/billetes a través de TAD vectordinámico**. Recordad que esta estructura debe estar ordenada de mayor a menor valor facial antes de pasársela a la función indicada arriba. El programa permitirá al menos gestionar los tipos de monedas existentes para el euro y para el dólar. Por tanto, al principio del programa, además del modo de operación se preguntará el tipo de moneda (euro/dólar) y se creará y rellenará (por valor descendente) un TAD vectordinámico con los valores de las monedas existentes de euro o dólar, según corresponda. Por ejemplo, para euro se crearía un vector con los siguientes valores: 50000, 20000, 10000, 5000, 2000, 1000, 500, 200, 100, 50, 20, 10, 5, 2, 1. Donde los siete primeros valores son el valor en céntimos de los billetes y los últimos valores son el valor en céntimos de las monedas.

- **gestión de solución a través de TAD vectordinámico**. En lugar de almacenar la solución en un array estático, la solución se almacenará en un vectordinámico que se generará antes de invocar a la función de cambio y que la función de cambio rellenará indicando que número de monedas/billetes se devuelven de cada tipo.

Para el caso de **monedas/billetes limitados** hay que gestionar también el stock de monedas existentes de cada tipo. Para ello:

- Al principio del programa, tras haber seleccionado el modo de operación de monedas limitadas, el programa leerá de un fichero el número de monedas y billetes disponibles de cada tipo. La gestión del stock se llevará también mediante el TAD vectordinámico. En concreto, se creará y rellenará (inicialmente con los valores leídos del fichero externo) un vectordinámico (*stock*) que ha de ser pasado como parámetro adicional a la función cambio. Será necesario generalizar la función de modo que: 1) de sólo cambio de la correspondiente moneda o billete si hay stock, y 2) una vez que llegue a una solución actualice el stock de monedas para la siguiente operación de devolución de cambio. Además, el programa no sólo informará del cambio que se le da a un usuario sino también del stock de monedas y billetes que quedan tras realizar esa devolución. Todas las operaciones de impresión se deben realizar desde el programa principal (nunca desde la función de devolver cambio). Al finalizar el programa, el fichero de stocks se actualizará para que almacene las monedas existentes de cada tipo para la próxima ejecución del programa.
- Dentro del lazo donde el programa pide repetidamente cantidades a devolver a los usuarios también se permitirá aumentar el stock. Esto es, para el caso de monedas limitadas, el programa presentará dos opciones: a) devolver cambio, y b) aumentar el stock. En esta última situación, supondréis que cuando se aumenta el stock porque se fue a buscar cambio al banco siempre se aumenta en 10 unidades el stock de cada tipo de moneda o billete.

Recordad que la estrategia voraz, aunque exista solución, no necesariamente la encuentra. Por ejemplo:

Devolver 0,31€ cuando disponemos sólo de: 1 moneda de 0,20€, 1 moneda de 0,10€, 1 moneda de 0,05€ y 3 monedas de 0,02€.

Para este caso, la aproximación voraz no tiene solución, mientras que una no voraz podría encontrar una solución de 5 monedas: 1 moneda de 0,20€, 1 moneda de 0,05€ y 3 monedas de 0,02€.

Probad este ejemplo para comprobar que efectivamente con esta configuración de monedas no se encuentra solución.

Generalización para cualquier tipo de moneda:

Por último, generalizar el programa de modo que pueda gestionar cualquier tipo de moneda (no sólo euros/dólares). Estrategia sugerida:

- Almacenar en un fichero externo los valores de las monedas existentes y el nombre de la moneda. Por ejemplo:

  euro
  50000, 20000, 10000, 5000, 2000, 1000, 500, 200, 100, 50, 20, 10, 5, 2, 1

Al principio del programa simplemente se leerá ese fichero (y el fichero de stock, si estamos en la opción de monedas limitadas) y se operará de acuerdo a los valores faciales de esa moneda.

Probad ejecuciones con distintas monedas para ver que en función de los valores faciales existentes encontramos o no la solución óptima. Por ejemplo, con un sistema monetario de valores faciales 6, 4, y 1 para devolver 8 unidades la aproximación voraz daría 1 moneda de 6 y 2 de 1 (total: 3 monedas) mientras que la solución óptima es 2 de 4 (2 monedas).