Ejercicio. Método voraz

[ITIS/ITIG, Feb 2006] Un prolífico escritor tiene que escribir N obras maestras de la literatura en un tiempo conocido. Cada libro tiene un tiempo de redacción y un beneficio asociado. Se desea saber cual es el máximo beneficio alcanzable. Determinar:

- Qué es necesario suponer sobre la creación de los libros para poder aplicar un algoritmo voraz.
- Implementad el algoritmo voraz que resuelva el problema.
- En caso de no realizar esa suposición qué metodología de programación será más adecuada?. Razona muy brevemente la respuesta. (4 puntos).

Solución ejercicio. Método voraz

- Este problema es muy similar al de la mochila:
  - La capacidad de la mochila es el tiempo total disponible
  - Los objetos son los libros a escribir
  - El peso de los libros son el tiempo que tardan en escribirse
  - El valor de los libros es el beneficio alcanzable
- En el tema del método voraz vimos que la solución al problema de la mochila no se podía aplicar si no podíamos dividir los objetos
- Por tanto, la suposición que se puede realizar es que se puede dividir (una) obra en fragmentos: dividirla en tomos, o bien obtener del editor un adelanto del total sobre el porcentaje de la obra ya redactado
- El algoritmo de la mochila 0-1 es el más apropiado si no es posible fragmentar los libros

Solución ejercicio. Método voraz (cont.)

```
// t y b son el tiempo de redacción y el beneficio obtenido con cada libro
// T es el tiempo total disponible
// x es el resultado: los libros a escribir; MB es el beneficio total obtenido
proc libros(t[1..n],b[1..n],T,x[1..n],MB)
  desde i ← 1 hasta n hacer
    x[i] ← 0
  fin desde
  tiempo ← 0
  mientras tiempo < T hacer
    i ← ⟨ el mejor libro pendiente de considerar ⟩
    si tiempo + t[i]≤T entonces
      x[i] ← 1 ; tiempo ← tiempo + t[i]
    si no
      x[i] ← (T - tiempo) / t[i] ; tiempo ← T
    fin si
  fin mientras
  MB ← 0
  desde i ← 1 hasta n hacer MB ← MB + x[i]*b[i]
fin proc
```

Solución ejercicio. Método voraz (cont.)

- En la línea que aparece en verde en el algoritmo anterior no se especifica exactamente cómo obtener *"el mejor libro pendiente de considerar"*
- Se puede recorrer cada vez la lista de libros, calcular la relación beneficio/tiempo de redacción y seleccionar el máximo (entre los libros no considerados en iteraciones anteriores)
- Sin embargo, este enfoque no es muy eficiente
- Una solución más eficiente consiste en ordenar (en orden decreciente) los libros por su beneficio por unidad de tiempo
- Podemos utilizar cualquier algoritmo de ordenación: *mergesort* o *quicksort*
- En lugar de devolver los elementos ordenados, tenemos que devolver **los índices de los elementos**, para devolver x en el mismo orden que los *arrays* t y v

Solución ejercicio. Método voraz (cont.)

```
proc libros(t[1..n],b[1..n],T,x[1..n],MB)
  crear  r[1..n], ro[1..n]
  desde i ← 1 hasta n hacer
    x[i] ← 0
    r[i] ← b[i] / t[i]
    ro[i] ← i //contiene índices
  fin desde
  mergesort(r,ro)
  j ← 1
  tc ← 0 //tiempo consumido
  mientras tc < T ∧ j ≤ n hacer
    i ← ro[j] ; j ← j+1
    si tc + t[i]≤T entonces
      x[i] ← 1 ; tc ← tc + t[i]
    si no
      x[i] ← (T - tc) / t[i] ; tc ← T
    fin si
  fin mientras
  MB ← 0
  desde i ← 1 hasta n hacer
    MB ← MB + x[i]*b[i]
  fin desde
fin proc
```

```
proc mergesort(r[1..n],S[1..n])
  h ← ⌊ n/2 ⌋ ; m ← n-h
  crear U[1..h], V[1..m]
  si n>1 entonces
    U[1..h] ← S[1..h]
    V[1..m] ← S[h+1..n]
    mergesort(r,U)
    mergesort(r,V)
    combinar(r,U,V,S)
  fin si
fin proc

proc combinar(r[1..n],U[1..h],V[1..m],S[1..h+m])
  i ← 1 ; j ← 1 ; k ← 1
  mientras i ≤ h Y j ≤ m hacer
    si r[U[i]] > r[V[j]] entonces
      S[k] ← U[i] ; i ← i+1
    si no S[k] ← V[j] ; j ← j+1
    k ← k+1
  fin mientras
  si i>h entonces S[k..h+m] ← V[j..m]
  si no S[k..h+m] ← U[i..h]
fin proc
```

Ejercicio. Divide y Vencerás

[ITIS/ITIG, Feb 2006] Dado un vector de elementos ordenado en orden no decreciente y un rango numérico, [A..B], diseñad un algoritmo recursivo mediante la técnica de divide y vencerás que calcule la suma del número de veces que aparecen los elementos de rango en el vector. La complejidad debe ser lineal en el peor caso y logarítmica en el mejor. (2 puntos).

Solución ejercicio. Divide y Vencerás

- La entrada a este algoritmo es el vector de elementos X[1..n] y los límites del rango A y B
- El resultado es un número que indica cuántos elementos de X están en el rango [A..B]
- Como el vector está ordenado, hay que hacer lo siguiente:
  - buscar la posición del primer elemento del rango en X
  - buscar la posición del último elemento del rango en X
  - calcular el número de elementos entre ambas posiciones
- Hay que tener en cuenta que puede no haber ningún elemento del rango, o bien un solo elemento
- Una forma de resolverlo es utilizando un algoritmo parecido al de búsqueda binaria para encontrar la posición **del primer elemento** de un vector mayor o igual al buscado

Solución ejercicio. Divide y Vencerás (cont.)

```
fun suma_rango(X[1..N],A,B)
   ini ← busca_1(X,A,1,N)
   fin ← busca_1(X,B,ini,N)
   mientras fin ≤ N ∧ X[fin] = B
   hacer {1}
      fin ← fin+1
   fin mientras
   devolver fin-ini
fin fun
```

```
fun busca_1(S[1..n], x, inf, sup)
   si inf > sup entonces
      devolver inf
   si no
      mitad ← ⌊ (inf+sup) / 2 ⌋
      // aunque haya encontrado x,
      // sigue buscando a su izquierda
      si  x ≤ S[mitad] entonces
         devolver busca_1(S, x, inf, mitad-1)
      si no
         devolver busca_1(S, x, mitad+1, sup)
      fin si
   fin si
fin fun
```

- busca_1() devuelve la primera aparición del elemento buscado, o bien la posición donde debería estar si el elemento no está en el vector
- En el caso mejor, las llamadas a busca_1() son de orden logaritmico (en cualquier caso), y el bucle {1} no se ejecuta (el elemento B no está en X)
- En el caso peor, el bucle {1} tiene coste lineal

Ejercicio. Programación Dinámica

[ITIS/ITIG, Sep 2008]Consideremos un mapa formado por N países. Queremos viajar entre países. Cada vez que atravesemos una frontera tenemos que pagar una tasa. Todas las tasas son conocidas. Diseña un algoritmo **dinámico** que determine el coste del viaje más barato entre dos países concretos dados. Detalla lo siguiente:

- (0,5 puntos) La relación recursiva.
- (2 puntos) El procedimiento o función que implemente el algoritmo.

Solución ejercicio. Programación Dinámica

```
proc turismo(mapa[1..n,1..n],inicio,final,coste)
   desde i ← 1 hasta n hacer
      desde j ← 1 hasta n hacer
         D[i,j] ← mapa[i,j]
      fin desde
   fin desde
   desde k ← 1 hasta n hacer
      desde i ← 1 hasta n hacer
         desde j ← 1 hasta n hacer
            D[i,j] ← minimo(D[i,j],D[i,k]+D[k,j])
         fin desde
      fin desde
   fin desde
   coste ← D[inicio,final]
fin proc
```