Estructuras de Datos (Sistemas), curso 2004/05

Segunda Práctica – Comparador de programas

Introducción

Un profesor de programación se siente frustrado debido a que sus alumnos copian las prácticas de manera masiva. Tras meditar profundamente una solución a su problema, decide crear un programa para comparar los códigos fuente de las prácticas de sus alumnos, de modo que le ayude a decidir si un programa ha sido copiado. Dicho programa debería ser capaz de emparejar dos ficheros de texto y mostrar por pantalla uno frente al otro.

Por ejemplo, dados los siguientes ficheros de texto (se numeran las líneas):

```
1 int maximo(int v[],int n) {
2   int i,maxPos;
3   /* recorremos el vector  */
4   /* para buscar el maximo */
5   maxPos = 0;
6   for (i=1;i<n;i++) {
7     if (v[i] > v[maxPos]) {
8       maxPos = i; } }
9 return maxPos}
```

Fichero 1

```
1 int maximo(int v[],int n) {
2   /* recorremos el vector  */
3   int i,maxPos;
4   maxPos = 0;
5   for (i=1;i<n;i++) {
6     if (v[maxPos] < v[i]) {
7       maxPos = i; } }
8 return maxPos}
```

Fichero 2

Un emparejamiento trivial (línea a línea) sería el siguiente:

```
int maximo(int v[],int n) {      1 = 1   int maximo(int v[],int n) {
  int i,maxPos;                  2 # 2     /* recorremos el vector  */
  /* recorremos el vector  */    3 # 3     int i,maxPos;
  /* para buscar el maximo */    4 # 4     maxPos = 0;
  maxPos = 0;                    5 # 5     for (i=1;i<n;i++) {
  for (i=1;i<n;i++) {            6 # 6       if (v[maxPos] < v[i]) {
    if (v[i] > v[maxPos]) {      7 # 7         maxPos = i; } }
      maxPos = i; } }            8 # 8   return maxPos}
return maxPos}                   9 # *
```

En la columna central se muestran los números de las líneas de ambos ficheros que se han emparejado, separados por el símbolo (=) cuando las líneas son iguales y por el símbolo (#) cuando las líneas son distintas. Se ha tenido que insertar una línea en blanco (recuadro rallado, asterisco en el número de línea) al final del segundo fichero para poder emparejar todas las líneas. Con este emparejamiento trivial el resultado sería que los ficheros coinciden en una línea y difieren en 8 líneas.

Es posible lograr un emparejamiento mejor si insertamos líneas en blanco no sólo al final del fichero menor sino en *puntos estratégicos* de ambos ficheros:

int maximo(int v[],int n) {	1 = 1	int maximo(int v[],int n) {
	* # 2	/* recorremos el vector */
int i,maxPos;	2 = 3	int i,maxPos;
/* recorremos el vector */	3 # *	
/* para buscar el maximo */	4 # *	
maxPos = 0;	5 = 4	maxPos = 0;
for (i=1;i<n;i++) {	6 = 5	for (i=1;i<n;i++) {
if (v[i] > v[maxPos]) {	7 # 6	if (v[maxPos] < v[i]) {
maxPos = i; } }	8 = 7	maxPos = i; } }
return maxPos}	9 = 8	return maxPos}

Con este emparejamiento “inteligente” conseguimos emparejar 6 líneas iguales, a costa de insertar 3 líneas en blanco en ambos ficheros.

Existen muchos emparejamientos posibles, no siempre va a ser posible emparejar todas las líneas iguales, y no está permitido emparejar líneas de forma que se "crucen" (En el ejemplo anterior no sería posible emparejar simultáneamente la línea 2 del primer fichero con la 3 del segundo y la línea 3 del primer fichero con la línea 2 del segundo). Es decir, la única operación permitida es insertar líneas en blanco en los ficheros, y tras ello los emparejamientos son correlativos. También debe quedar claro que las comparaciones están basadas en líneas completas, no en caracteres (o dos líneas son iguales o son distintas).

Descripción formal

Se dispone de dos documentos formados por una secuencia de líneas. El primer documento tiene $\boldsymbol{n_1}$ líneas, las cuales se almacenan en el vector $\mathbf{D_1[1..}\boldsymbol{n_1}\mathbf{]}$. El segundo documento tiene $\boldsymbol{n_2}$ líneas, las cuales se almacenan en el vector $\mathbf{D_2[1..}\boldsymbol{n_2}\mathbf{]}$.

Una solución válida del problema del emparejamiento se puede expresar mediante dos vectores, $\mathbf{S_1[1..}\boldsymbol{n_t}\mathbf{]}$ y $\mathbf{S_2[1..}\boldsymbol{n_t}\mathbf{]}$, que almacenan índices (números de línea) a los vectores $\mathbf{D_1}$ y $\mathbf{D_2}$ respectivamente, y representan una manera particular de emparejar los dos documentos: La línea $\mathbf{S_1[}\boldsymbol{i}\mathbf{]}$ del primer documento está emparejada con la línea $\mathbf{S_2[}\boldsymbol{i}\mathbf{]}$ del segundo documento. Si $\mathbf{S_1[}\boldsymbol{i}\mathbf{]}$ vale cero significa que se ha insertado una línea en blanco en el primer documento, y si $\mathbf{S_2[}\boldsymbol{i}\mathbf{]}$ vale cero significa que se ha insertado una línea en blanco en el segundo documento. $\boldsymbol{n_t}$ es un valor en el rango $[\max(\boldsymbol{n_1},\boldsymbol{n_2})..\boldsymbol{n_1}+\boldsymbol{n_2}]$.

De entre todas las soluciones válidas (emparejamientos posibles), estaremos interesados en obtener aquella que sea **optima**, en el sentido de minimizar una determinada función de coste que se describirá posteriormente.

El emparejamiento “inteligente” del ejemplo anterior estaría representado por los vectores:

$\mathbf{S_1}$	1	0	2	3	4	5	6	7	8	9
$\mathbf{S_2}$	1	2	3	0	0	4	5	6	7	8

Por lo tanto, se deben cumplir las siguientes reglas respecto a los vectores solución:

- Si $\mathbf{S_1[}\boldsymbol{i}\mathbf{]} > 0$ y $\mathbf{S_2[}\boldsymbol{i}\mathbf{]} > 0$ significa que las líneas $\mathbf{D_1[S_1[}\boldsymbol{i}\mathbf{]]}$ y $\mathbf{D_2[S_2[}\boldsymbol{i}\mathbf{]]}$ están emparejadas y al mostrar la solución se imprimen en *i*-ésima posición.
- Si $\mathbf{S_1[}\boldsymbol{i}\mathbf{]} = 0$ y $\mathbf{S_2[}\boldsymbol{i}\mathbf{]} > 0$ significa que se ha insertado una línea en blanco en el primer documento la cual se empareja con la línea $\mathbf{D_2[S_2[}\boldsymbol{i}\mathbf{]]}$ del segundo.

- Si $\mathbf{S_1[}i\mathbf{]} > 0$ y $\mathbf{S_2[}i\mathbf{]} = 0$ significa que se ha insertado una línea en blanco en el segundo fichero la cual se empareja con la línea $\mathbf{D_1[S_1[}i\mathbf{]]}$ del primero.
- No se puede dar que $\mathbf{S_1[}i\mathbf{]} = 0$ y $\mathbf{S_2[}i\mathbf{]} = 0$, ya que eso significa insertar líneas en blanco en ambos ficheros y emparejarlas, lo que no tiene sentido (incrementa el coste de la solución)
- Ya que se debe respetar la secuencia de líneas, los vectores solución deben cumplir que, si eliminamos los valores 0, formen una secuencia ascendente que comience en 1 y termine en n_1 (para $\mathbf{S_1}$) o n_2 (para $\mathbf{S_2}$). Las reglas anteriores fijan el valor de n_t.

La función de coste consiste en valorar los emparejamientos de acuerdo al siguiente baremo:

- +3 si se empareja con una línea en blanco insertada. ($\mathbf{S_1[}i\mathbf{]} = 0$ ó $\mathbf{S_2[}i\mathbf{]} = 0$)
- +2 si se emparejan líneas distintas ($\mathbf{D_1[S_1[}i\mathbf{]]} \neq \mathbf{D_2[S_2[}i\mathbf{]]}$)
- -4 si se emparejan líneas iguales ($\mathbf{D_1[S_1[}i\mathbf{]]} = \mathbf{D_2[S_2[}i\mathbf{]]}$)

El coste de una solución consiste en la suma de los costes de todos los emparejamientos. El objetivo es encontrar la solución cuyo coste es mínimo (si existieran varias soluciones con coste igual al mínimo, cualquiera de ellas se podría presentar como la solución óptima).

En los ejemplos, el emparejamiento trivial tendría un coste de 13 y el emparejamiento inteligente de -13.

A continuación se muestra un trozo de código que generaría al azar una solución válida (por supuesto, no tiene porque ser la óptima) del problema. El propósito de éste ejemplo es ilustrar la manera en que se representa una solución.

```
I1 := 1; I2 := 1; I := 1;
while (I1 <= N1) and (I2 <= N2) do
begin
  { Se tienen tres opciones: }
  case random(3) of
    0 : begin { Emparejar D1[I1] con D2[I2] }
          S1[I] := I1; S2[I] := I2; I1 := I1+1; I2 := I2+1
        end;
    1 : begin { Insertar línea en blanco en D1 }
          S1[I] := 0;  S2[I] := I2; I2 := I2+1
        end;
    2 : begin { Insertar línea en blanco en D2 }
          S1[I] := I1; S2[I] := 0;  I1 := I1+1
        end;
  end; { case }
  I := I+1
end; { while }
{ Insertar líneas en blanco en fichero agotado. Solo se itera uno
  (o ninguno) de los dos bucles siguientes }
while I1 <= N1 do
begin
  S1[I] := I1; S2[I] := 0;  I1 := I1+1; I := I+1
end;
while I2 <= N2 do
begin
  S1[I] := 0;  S2[I] := I2; I2 := I2+1; I := I+1
end;
{ Tamaño de la solución }
Nt := I-1;
```

Requisitos de la Práctica

La práctica consistirá en crear un programa que pida el nombre de dos ficheros de texto y muestre por pantalla el resultado de su emparejamiento óptimo.

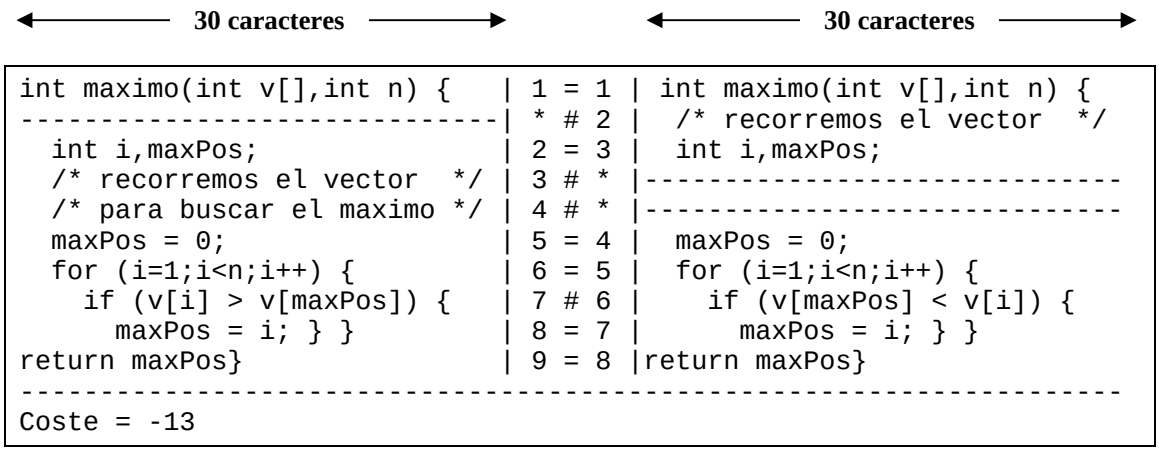
Tras pedir los nombres de los ficheros, el programa leerá su contenido almacenándolo en dos vectores de strings, $\mathbf{D_1}$ y $\mathbf{D_2}$. (una línea en cada elemento de vector).

Utilizando alguna de las estrategias de diseño contempladas en la asignatura (Divide y vencerás, fuerza bruta, backtracking, programación dinámica o algoritmo voraz) el programa calculará el emparejamiento óptimo de ambos ficheros representando la solución de la manera indicada anteriormente (vectores $\mathbf{S_1}$ y $\mathbf{S_2}$).

Por último, el programa presentará esa solución, de la siguiente manera: Se muestran una serie de líneas, cada una representando un emparejamiento, conteniendo:

- Los primeros 30 caracteres de la línea correspondiente al primer fichero, o 30 guiones si es una línea insertada.
- El número de línea a que corresponde en el primer fichero, o un asterisco si es una línea insertada.
- El símbolo (=) si ambas líneas coinciden o (#) si no coinciden.
- El número de línea a que corresponde en el segundo fichero, o un asterisco si es una línea insertada.
- Los primeros 30 caracteres de la línea correspondiente al segundo fichero, o 30 guiones si es una línea insertada.

Al final del todo se imprimirá una línea indicando el coste de esa solución. La salida correspondiente al ejemplo del emparejamiento “inteligente” sería algo parecido a lo siguiente (se recompensará el que se realicen esfuerzos para que la salida tenga un aspecto agradable):



Se puede suponer que ninguno de los ficheros superará las 1000 líneas de texto. En la evaluación se tendrá en cuenta la eficiencia de la estrategia utilizada para resolver el problema.

Documentación que se debe entregar

Se deberá entregar el código fuente del programa que resuelve la práctica (se recomienda el uso del lenguaje Eiffel, aunque alternativamente se permite realizarlo en Java, C, C++ o Delphi/Kylix) y un documento (en formato Word, Latex (DVI) o Acrobat PDF) donde se indique lo siguiente:

- El tipo de estrategia(s) de diseño (divide y vencerás, fuerza bruta, backtracking, programación dinámica o algoritmo voraz) utilizada para resolver el problema planteado. Si se utiliza un algoritmo voraz debe incluirse una demostración de que con él se obtiene la solución óptima.
- Una breve descripción (1 o 2 páginas) del modo en que se ha aplicado la estrategia al problema.
- Una evaluación en notación asintótica de la eficiencia del programa (no es imprescindible que sea exacta: se pueden utilizar cotas inferiores y superiores).

Los ficheros de las prácticas se deben depositar en el directorio **$HOME/ed04_05/p2** de la cuenta del usuario en **jair** y se recogerán automáticamente el 17 de junio a las 10:00.

Comprobación de los programas

En la página de la asignatura se publicarán casos de prueba para que podáis evaluar vuestros programas.