Buscando canciones

2009 - Pablo A. Haya

Etiquetas Aprendizaje basado en proyectos, Programación, C, Herramientas de desarrollo

Características Se precisa experiencia básica en el lenguaje de programación que se emplee (ej. C), y desenvolverse con cierta soltura en el manejo de herramientas de desarrollo (ej. gcc y make). Preparada para equipos de 3 integrantes, 2 semanas de duración y con una dedicación de 6h/semana/integrante. Se puede combinar fácilmente con metodologías de desarrollo ágil. La actividad está preparada para ser implementada en C pero su adaptación a otro lenguaje no requiere grandes cambios menores. La preparación de la misma por parte del equipo se puede realizar mediante un puzle.

Contacto con el autor http://pablohaya.com/contact
Fuente de la obra http://pablohaya.com/tag/buscandocanciones
Preguntas acerca de la distribución http://pablohaya.com/tag/licencia

Última modificación 2011-11-09 16:01

Historial de asignaturas

2009-2010. Proyecto de Programación. Ingeniería Informática. Universidad Autónoma de Madrid

Enunciado de la actividad: Buscando canciones

Resumen

Esta actividad consiste en realizar la implementación del un sistema informático que permite encontrar similitudes en nombres de ficheros. Se aplicará como ejemplo descrito en este documento. Los objetivos docentes que se persiguen son practicar técnicas de trabajo en grupo y herramientas de desarrollo un proyecto software.

Se dan por supuesto manejo básico de programación en C y de las herramientas de desarrollo *gcc* y *make*.

Requisitos funcionales

Se quiere desarrollar una aplicación informática que permita encontrar similitudes en nombres de ficheros (*canciones_iguales*). En particular, se piensa trabajar con dos o más listados de nombres de ficheros de canciones. La herramienta debe ser capaz de identificar qué nombres de ficheros corresponden con la misma canción. Únicamente se va a considerar el nombre del archivo para hacer la comparación, teniendo en cuenta que este nombre puede incluir más información que el propio título, y que se han podido cometer errores tipográficos al crearlo.

Así, por ejemplo, la canción "*La Casa Por El Tejado*" del grupo *Fito Y Fitipaldis*, podría corresponder a los siguientes nombres de fichero:

- Fito Y Fitipaldis - [Lo Mas Lejos A Tu Lado] 01 La Casa Por El Tejado.mp3
- Fito y Fitipaldis la Casa por el Tejado.mp3
- Fito&Fitipaldis La Casa Por El Tejado.mp3
- Fitoy Fitipaldis la casa por tejado.mp3

A la hora de realizar el programa se van a hacer las siguientes suposiciones:

1. El nombre del grupo siempre estará incluido en el título y siempre se encontrará antes del título de la canción.
2. No se contemplarán títulos de canciones que sólo contengan dígitos.
3. Cualquier información adicional sobre la canción (ej. título del disco, nombre del propietario…) tendrán que aparecer entre corchetes o entre paréntesis.

Los listados con los nombres de los ficheros de canciones estarán contenidos en ficheros. La aplicación recibirá un fichero origen y N ficheros destino que contendrán dichos listados. La aplicación debe mostrar por la salida estándar todas aquellas canciones pertenecientes a los ficheros destino cuyo título sea similar a alguno contenido en el fichero origen.

La similitud entre dos canciones se definirá como un valor entre 0 y 1 configurable por el usuario siendo 1 cuando los títulos son idénticos. El usuario especificará a partir de qué valor dos canciones se consideran similares.

Un ejemplo de invocación de aplicación podría ser:

$ ./canciones_iguales 0.2 fichorigen fichdest1 fichdest2 … fichdestN

Formato de los ficheros

Cada fichero, independientemente que sea origen o destino, contendrá un conjunto indefinido de líneas. Cada línea contendrá el nombre de un fichero de música y finalizará con un salto de línea (habrá que tener en cuenta las diferentes codificaciones en Windows, Linux y Mac Os).

Los títulos sólo contendrán caracteres que se encuentren codificados como caracteres imprimibles de la tabla ASCII.

Formato de salida de la aplicación

Por cada título del fichero destino que se encuentre alguna semejanza con algún título del fichero origen, se extraerá una línea que contenga el título y la correspondencia que se haya encontrado. Para distinguir en qué fichero destino se ha encontrado, los títulos irán precedidos del nombre del fichero y el carácter ':'.

Requisitos no funcionales

La aplicación tendrá que ser capaz de:

- Manejar títulos de ficheros de al menos 256 caracteres.
- Soportar tamaño de ficheros de al menos 1000 canciones.
- Poder configurarse por línea de comandos el parámetro *semejanza*.
- No deberá permitirse que haya memoria sin liberar.

Diseño de la aplicación

En la Figura 1 se puede apreciar el diseño propuesto para la implementación del proyecto. Se pueden distinguir tres módulos: *Principal*, *Utilidades de Cadenas de Caracteres* y *Distancia de Edición*. El módulo *Principal* será el encargado de leer los nombres de las canciones que se encuentren en el fichero origen, y compararlos con los nombres que se encuentran en el resto de los ficheros.

El módulo *Distancia de Edición* permitirá calcular una medida de similitud entre dos cadenas de caracteres de manera que sea capaz de responder si dos títulos son lo suficientemente parecidos como para ser considerados la misma canción.

El módulo *Utilidades de Cadenas de Caracteres* se encargará de convertir los nombres de las canciones que reciba en un nombre canónico que identifique unívocamente cada canción. Se eliminarán la extensión del archivo, caracteres que no sean alfabéticos y aquella información adicional que no pertenece al título.

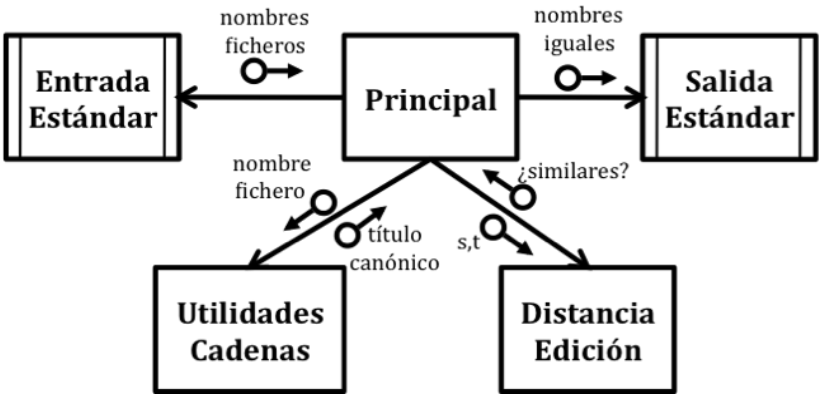

Figura 1. Diseño de la aplicación Títulos Iguales

El Algoritmo 1 muestra el pseudocódigo de lo que podría ser el módulo principal de la aplicación.

Algoritmo 1: Módulo principal

```
CANCIONES_IGUALES(UMBRAL, F_ORIGEN, F_DESTS[1..n])
Entrada: Umbral, fichero origen y lista de ficheros destino
Salida: títulos coincidentes

1  i ← 0
2  mientras no FINFICHERO(F_ORIGEN) :
3        i ← i + 1
4        T[i] ← CANONICO(LEELINEA(F_ORIGEN))
5  num_titulos_origen ← i
6  desde i ← 1 hasta n
7        mientras no FINFICHERO(F_DESTS[i]):
8                t ← CANONICO(LEELINEA(F_DESTS[i]))
9                desde j ← 1 hasta num_titulos_origen
10                    si SIMILITUD(t, T[j]) > UMBRAL:
11                        IMPRIME(i, t, T[j])
```

La totalidad de la práctica se va a realizar en dos semanas (véase guía de actividades de la semana 1 y la semana 2). En consecuencia, se ha dividido la implementación en dos partes, una para cada semana.

Parte I: Semana 1

En esta primera parte se van a implementar los siguientes módulos:

Módulo Distancia de edición (distediccion)

La **distancia de edición** mide cuán diferente es una cadena de caracteres de otra. Dadas dos cadenas de caracteres *s[1..n]* y *t[1..m]*, la distancia de edición *d(s,t)* se define como el mínimo número de transformaciones (véase más abajo) que permiten cambiar la cadena *s* en la cadena *t*. Existen múltiples formas de

calcular esta distancia. En esta práctica implementaremos la distancia de edición de Levenshtein.

Las transformaciones permitidas son:

- Reemplazar (R) : un carácter por otro.
- Insertar (I): un nuevo carácter.
- Eliminar (E): un carácter.

Así, para transformar el "Mal" en el "Bien" se precisan cuatro cambios, esto es, la distancia sería cuatro. A continuación se muestran dos posibles caminos de transformación:

Mal		**Mal**	
Bal	M⇒B (R)	Bal	M ⇒ B (R)
Bial	Inserta i (I)	Bil	a ⇒ i (R)
Biel	a ⇒ e (R)	Bie	l ⇒ e (R)
Bien	l ⇒ n (R)	**Bien**	Inserta n (I)

En el problema que nos atañe, no es necesario calcular los pasos intermedios, sino que nos vale con hallar el número pasos. Esta distancia se calcula mediante una relación de recurrencia que nos da el valor de la distancia para cada par de fragmentos de ambas palabras. Así, d(2,3) para el ejemplo anterior, correspondería a la distancia entre el fragmento "Ma" y el fragmento "Bie". Siguiendo con el ejemplo anterior, en nuestro caso, nos interesaría calcular d(3,4), y en general, d(n,m). Como se puede observar en la Figura 2, el cálculo de la distancia para cada i y j, depende de las distancias calculadas para distintas combinaciones de i-1 y j-1.

$$d(i,j) = \min \begin{cases} d(i-1,j) + \text{COSTO\_INSERTAR} \\ d(i-1,j-1) + \text{COSTO\_REEMPLAZAR} \\ d(i,j-1) + \text{COSTO\_ELIMINAR} \end{cases}$$

$$d(0,0) = 0$$
$$d(i,0) = d(i-1,0) + \text{COSTO\_INSERTAR}$$
$$d(0,j) = d(0,j-1) + \text{COSTO\_ELIMINAR}$$

Figura 2. Relación de recurrencia para el cálculo de la distancia de edición

Nótese, que en la relación de la Figura 2 se puede asignar un coste distinto para cada operación. En el ejemplo anterior, el coste de cada operación es fijo e igual a uno. Ahora bien, dependiendo del problema puede ser interesante asignar un

coste distinto a cada operación. Por ejemplo, la operación de Reemplazar tiene sentido asignarle coste 2 ya que equivale a una eliminación e inserción.

Para realizar este cálculo nos vamos a apoyar en un tabla auxiliar en la que se irán almacenando los costes intermedios obtenidos para cada par de sufijos.

		j	0	1	2	3	4
	i			B	i	e	n
0			0	1	2	3	4
1		M	1	1	2	3	4
2		a	2	2	2	**3**	4
3		I	3	3	3	3	**4**

Figura 3. Tabla auxiliar con las distancias intermedias para cada par de fragmentos de las dos cadenas originales

La tabla de la Figura 3 nos permite averiguar la distancia para cada par de sufijos. Por ejemplo, la distancia anterior entre "*Ma*" y "*Bie*" corresponde a i = 2 y j=3 que consultando la tabla obtenemos d(2,3) = 3. En particular, la distancia que nos interesa para nuestro problema es aquella en la que los índices coinciden con la longitud de cada cadena. En este ejemplo sería d(3,4) = 4.

El cálculo de esta tabla se obtiene a partir del siguiente algoritmo de programación dinámica:

Algoritmo 2: Distancia edición

```
EDITDIST(s[1..n],t[1..m])
Entrada: s, t cadenas de caracteres
Salida: distancia de edición entre s y t

1   d[0,0] ← 0
2   desde i ← 1 hasta n:
3       d[i,0] ← d[i-1,0] + COSTO_INSERTAR
4   desde j ←1 hasta m:
5       d[0,j] ← [0,j-1] + COSTO_BORRAR
6   desde i ← 1 hasta n:
7       desde j←1 hasta m:
8           si s[n] = t[j]
9               d[i,j] ← d[i-1,j-1]
10          si no
11              d[i,j] ← min(  d[i-1,j] + COSTO_INSERTAR,
                               d[i-1,j-1] + COSTO_REEMPLAZAR(s[i], t[j]),
                               d[i,j-1] + COSTO_BORRAR)
12  devuelve d[n,m]
```

Este módulo deberá incluir la siguiente función:

- **FE1:** Esta función recibe dos cadenas de caracteres y devuelve un número entero que se corresponde con la distancia de edición de ambas cadenas. En caso de que hubiera algún error se devolvería -1 (valor absurdo para una distancia).

Se implementará una aplicación cliente (*clidistediccion*) que pruebe la función anterior. Esta aplicación, importando el módulo distancia de edición, calculará la distancia entre dos cadenas de caracteres que se pasen como parámetros. Un ejemplo de cómo se invocaría esta aplicación sería:

```
$ ./clidistediccion Mal Bien
editdist(Mal,Bien)=4
```

Módulo Utilidades para cadenas de caracteres (utilcadena)

Este módulo va a incluir aquellas operaciones con cadenas de caracteres necesarias para el proyecto y que podrán ser reutilizadas en otros proyectos.

En esta primera parte se implementarán dos funciones cuyo prototipo tiene que respetarse **rigurosamente**. Lo único que se puede cambiar es el nombre de las mismas:

- **FS1**: Esta función copia una cadena de caracteres en otra eliminando la extensión del archivo, que queda delimitada por todos los caracteres que se encuentren después del último punto. Si no existiera ningún punto, la cadena se copia tal cual. Esta función copia como máximo *max* caracteres desde origen al destino. Si los caracteres son menos que el máximo definido se rellenará con '\0' el resto de la cadena destino, en caso contrario se copiará hasta el máximo permitido. La función recibe dos cadenas de caracteres (origen y destino), y el máximo número de caracteres a copiar. Devuelve la cadena destino, o NULL si ha habido algún error. A continuación se muestran un par de ejemplos de uso:

  ```
  char dst[30];

  FS1(dst, "01-metallica_enter sadman.mp3\n", sizeof (dst));

  /* dst = "01-metallica_enter sadman\0\0\0\0\0" */
  ```

  en cambio, si el tamaño de *dst* fuera 20,

  ```
  char dst[20];

  FS1(dst, "01-metallica_enter sadman.mp3\n", sizeof (dst));

  /* dst = ''01-metallica_enter \0'' */
  ```

- **FS2**: Esta función copia una cadena de caracteres en otra eliminando aquellos caracteres especificados en una tercera cadena. Esta función copia como máximo max caracteres desde origen al destino. Si los caracteres son menos que el máximo definido se rellenará con '\0' el resto de la cadena destino, en caso contrario se copiará hasta el máximo

permitido. La función recibe tres cadenas de caracteres (cadena destino, cadena origen y delimitadores a eliminar) y un entero que indica el número máximo de caracteres a copiar. Devuelve la cadena destino, o NULL si ha habido algún error. Vease el ejemplo:

```
char dst[30];

FS2(dst, "01-metallica_enter sadman”, "-,._ ", sizeof (dst));

/* dst = ‘‘01metallicaentersadman\0\0\0\0\0\0\0’’; */
```

Cada equipo será libre de implementar todas aquellas funciones auxiliares que precisen incluir en este módulo, aunque **se penalizará negativamente si se incluyen funciones ya definidas en la biblioteca estándar de C**.

Se implementará una aplicación cliente (*clistringutil*) que pruebe las funciones anteriores. Esta aplicación, importando el módulo utilidades de cadenas, extraerá la extensión y eliminará aquellos caracteres determinados en la cadena de referencia. Un ejemplo de cómo se invocaría esta aplicación y un posible resultado sería:

```
$ ./clistringutil "01-metallica_enter sadman.mp3" "?.*- "
01metallicaentersadman
```

Módulo Comparación de ficheros: Líneas iguales

Este módulo constituye una primera aproximación o prototipo del Módulo Principal de la aplicación que se desarrollará en la segunda semana. Si nos fijamos en los requisitos, los datos de entrada se obtienen de dos o más ficheros. El primero de ellos contiene la lista de títulos con la que se va a comparar el resto. Este módulo deberá comparar las cadenas de títulos contenidas en el primer fichero (fichero origen) con todas las cadenas contenidas en el resto de ficheros (ficheros destino) y mostrar aquellas cadenas que sean exactamente iguales. Se va a emplear una estrategia similar a la expuesta en el pseudocódigo del Algoritmo 1 para realizar esta comparación. Esto es:

1. Se lee completamente el primer fichero y se almacena en memoria.
2. Para cada uno de los ficheros restantes se lee línea a línea y se compara el título leído con todos los títulos del fichero origen almacenados. Si las cadenas son iguales se almacena ese título.
3. Una vez que se han leído todos los ficheros, se muestran en la salida estándar todos los títulos idénticos encontrados.

La ventaja de hacer esta implementación antes del Módulo Principal de la aplicación es que ésta no requiere ninguna integración con el resto del sistema y nos permite probar la validez del algoritmo de forma independiente.

El programa que se tiene que desarrollar deberá llamarse *líneas_iguales.c*. Para la comparación de las líneas se deberá usar la función *strcmp* de la biblioteca estándar. Si las líneas coinciden se imprimirá la línea indicando el nombre del fichero destino donde se ha encontrado otra igual, el número de línea en ese fichero y el número de línea en el fichero origen.

Suponiendo que se dispone de los siguientes ficheros:

fich1.txt	fich2.txt	fich3.txt
Bob Dylan - Like a Rolling Stone Radio Futura - Escuela de Calor John Lennon – Imagine The Beatles - Hey Jude Duncan Dhu - En algun lugar Joan Manuel Serrat – Mediterraneo The Who - My Generation Paco Lucia - Entre dos aguas Camaron - La leyenda del tiempo Metallica - Enter Sandman	Metallica - Enter Sandman Nivarna - Smells Like Teen Spirit The Who - My Generation The Beatles - Hey Jude John Lennon – Imagine Bob Dylan - Like a Rolling Stone	Paco Lucia - Entre dos aguas Paco Lucia - Entre dos aguas Duncan Dhu - En algun lugar Radio Futura - Escuela de Calor El Ultimo De La Fila – Insurreccion Joan Manuel Serrat – Mediterraneo Camaron - La leyenda del tiempo Los Piratas - Promesas que no valen nada

El programa se ejecuta con la siguiente instrucción:

```
$ ./lineas_iguales fich1.txt fich2.txt fich3.txt
```

El resultado del programa debería ser:

```
fich2.txt,1 L10: Metallica - Enter Sandman
fich2.txt,3 L7: The Who - My Generation
fich2.txt,4 L4: The Beatles - Hey Jude
fich2.txt,5 L3: John Lennon – Imagine
fich2.txt,6 L1: Bob Dylan - Like a Rolling Stone
fich3.txt,1 L8: Paco Lucia - Entre dos aguas
fich3.txt,2 L8: Paco Lucia - Entre dos aguas
fich3.txt,3 L5: Duncan Dhu - En algun lugar
fich3.txt,4 L2: Radio Futura - Escuela de Calor
fich3.txt,6 L6: Joan Manuel Serrat – Mediterraneo
fich3.txt,7 L9: Camaron - La leyenda del tiempo
```

Parte II: Semana 2

En la segunda semana se deberá terminar la implementación de los módulos del proyecto, y se realizará la integración de los mismos.

Módulo Distancia de edición (distediccion)

Este módulo se ampliará con la siguiente función:

- **FE2:** Esta función recibe dos cadenas de caracteres y un umbral de similitud, y devuelve verdadero (1) si las dos cadenas son similares o falso (0) en caso contrario. La similitud se medirá empleando la siguiente distancia normalizada

$$dnorm(s,t) = \frac{2*d(s,t)}{\alpha*\ (long(s) + long(t))\ + d(s,t)}$$

Donde d(s,t) es la distancia de edición, α es un parámetro que se inicializa

a 1, y long(s) es la longitud de la cadena s. Por tanto, si la distancia de edición normalizada (*dnorm*) entre las dos cadenas es igual o está por encima del umbral se considerarán similares, y si está por debajo serán disimilares (pueden ser distintas siendo similares).

Se modificará la aplicación cliente (*clidistediccion*) que prueba el módulo distancia de edición para que se pase como parámetro un número real entre 0 y 1 que indique el umbral de similitud. La aplicación devolverá verdadero o falso según sean similares o no. Un ejemplo de cómo se invocaría esta aplicación y un posible resultado sería:

```
$ ./clidistediccion 0.8 Mal Bien
falso
```

Módulo Utilidades para cadenas de caracteres (utilcadena)

En esta segunda parte se implementarán dos nuevas funciones cuya interfaz tiene, de nuevo, que respetarse. Lo único que habrá que cambiar es el nombre de las mismas:

- **FS3**: Esta función copia una cadena de caracteres en otra eliminando el contenido que se encuentra entre paréntesis '(' ')' o entre corchetes '[' ']´, y los propios separadores. La función copia como máximo *max* caracteres desde origen al destino. Si los caracteres son menos que el máximo definido se rellenará con '\0' el resto de la cadena destino, en caso contrario se copiará hasta el máximo permitido. Si no existieran los caracteres anteriores, la cadena se copia tal cual. Sólo será necesario contemplar un nivel de anidación de paréntesis o corchetes. Así, una vez detectado un separador de apertura, se eliminarán todos los caracteres que se encuentren hasta que aparezca el primer separador de cierre (independientemente de si entre estos caracteres se encuentran nuevos separadores de apertura).

  La función recibe dos cadenas de caracteres (origen y destino), y el máximo número de caracteres. Devuelve la cadena destino, o NULL si ha habido algún error. A continuación se muestran un par de ejemplos de uso:

  ```
  char dst[20];

  FS3(dst, "enter sadman [metallica].mp3\n", sizeof (dst));

  /* dst = ''enter sadman .mp3\n\0\0'' */
  ```

  En cambio, si hubiera corchetes anidados

  ```
  char dst[25];

  FS3(dst, "enter sadman [metal[li]ca].mp3\n", sizeof (dst));

  /* dst = ''enter sadman ca].mp3\n\0\0\0\0'' */
  ```

- **FS4**: Esta función recibe una cadena de caracteres y la convierte en

mayúsculas. Nótese que se cambia el contenido de la cadena original, y que no se realiza copia de la misma. Devuelve la cadena, o NULL si ha habido algún error. Un ejemplo de aplicación sería:

```
char dst[] = "01metallicaentersadman";

FS4(dst);

/* dst = ''01METALLICAENTERSADMAN\0'' */
```

Se modificará la aplicación cliente (*cliutilcadena*), que prueba el módulo de utilidades de cadena de caracteres, para añadir las dos funciones anteriores. Un ejemplo de cómo se invocaría esta aplicación y un posible resultado sería:

```
$ ./cliutilcadena "01-metallica_enter sadman [METALLICA].mp3" "?.*- "
01METALLICAENTERSADMAN
```

Módulo Principal: Canciones iguales

Este módulo incluirá la función principal de la aplicación. Su implementación se basará en el Módulo Comparación de ficheros (líneas iguales) realizada en la Parte I. Realizará un algoritmo similar pero se empleará en la comparación de los títulos de las canciones la función FE2 definida en el módulo Distancia de Edición, en vez de la función *strcmp*. Asimismo se compararán los títulos canónicos devueltos por la función que se describe a continuación en vez de las líneas exactas contenidas en los ficheros.

Se añadirá una función a este módulo que permita transformar los títulos leídos de los archivos en títulos canónicos.

- **FM1:** Recibe el nombre de una canción y genera un nombre canónico. Este nombre se genera a partir de las funciones definidas en el *Módulo de Utilidades de Cadenas* como el nombre del fichero en mayúsculas eliminando la extensión, caracteres de no alfabéticos (ej. ?.*-) y dígitos, así como la información que aparezca entre corchetes y paréntesis. Recibe dos cadenas de caracteres (origen y destino) y el máximo número de caracteres. Devuelve la cadena destino o NULL si ha habido algún error. A continuación se muestra un ejemplo de uso:

```
char dst[25];

FM1(dst, "01-metallica_enter sadman [METALLICA].mp3\n", sizeof
(dst));

/* dst = ''METALLICAENTERSADMAN\0\0\0\0\0'' */
```

Mejoras al proyecto

1. **Aplicación Títulos Iguales**, Eliminar las restricciones impuestas en el apartado de Requisitos, de manera que la aplicación funcione con un mayor número de nombres de ficheros utilizando memoria dinámica.

2. **Módulo Edición Distancia**. Modificar el algoritmo de Edición de Distancia para que sólo se almacene la fila que se está calculando y la anterior, en vez de toda la matriz.

3. **Módulo Edición Distancia.** Incluir una función que devuelva una secuencia de operaciones que permita reconstruir el camino para transformar la cadena *s* en la cadena *t*.

4. **Módulo Edición Distancia.** Encontrar los valores de COSTO_INSERTAR, COSTO_BORRAR, COSTO_REEMPLAZAR que dan mejores resultados. Razonar la respuesta según los nombres de fichero que se prueben

5. **Módulo Utilidades de Cadena** (Tanto en la función FS1 como FS2). permitir emplear la misma cadena como origen y destino.

6. **Módulo Utilidades de Cadena**. Contemplar en la función FS4 que se puedan incluir separadores anidados.

Entregables y criterios de corrección

El resultado de la actividad se plasma en dos entregas.

Semana 1:

La entrega de esta parte consistirá en un archivo comprimido que contendrá los siguientes ficheros:

- *distediccion.c* version 1: Archivo fuente que incluye las funciones correspondientes a la semana 1 del módulo de Distancias de Edición.
- *clidistediccion.c* versión 1: Aplicación de prueba del módulo Distancia de Edición.
- *utilcadena.c* version 1: Archivo fuente que incluye las funciones correspondientes a la semana 1 del módulo de Utilidades de Cadenas.
- *cliutilcadena.c* version 1: Aplicación de prueba del módulo Utilidades de Cadenas.
- *lineas_iguales.c*: Aplicación que permite comparar líneas de fichero.
- *Makefile* version 1: Archivo *Makefile* que genera las tres siguientes aplicaciones: *clieditdist*, *clistringutil* y *líneas_iguales*.

Criterio de corrección

En particular, tendrá que incluir un archivo *Makefile* que deberá incluir un objetivo que compile los tres ejecutables del entregable (*clidistediccion*, *cliutilcadena*, *cmplineas*).

El código entregado se evaluará según el siguiente criterio:

ACEPTABLE –

1. Las aplicaciones *clidistediccion*, *cliutilcadena* y *cmplineas* deben superar casos de prueba similares a los que se especifican en el enunciado.

2. Han sido entregado antes de la fecha límite indicada

NO ACEPTABLE – El documento no cumple los criterios anteriores.

Semana 2:

La entrega de esta parte consistirá en un archivo comprimido que contendrá los siguientes ficheros:

- *distediccion.c* version 2: Archivo fuente que incluye todas las funciones del módulo de Distancias de Edición.
- *clidistediccion* versión 2: Aplicación de prueba del módulo Distancia de Edición.
- *utilcadena.c* version 2: Archivo fuente que incluye todas las funciones del módulo de Utilidades de cadenas.
- *cliutilcadena.c* version 2: Aplicación de prueba del módulo Utilidades de cadenas.
- *canciones_iguales*.c: Módulo principal de la aplicación.
- *Makefile* version 2: Archivo *Makefile* que genera las tres siguientes aplicaciones: *clidistediccion*, *clistringutil* y *canciones_iguales*.

Criterio de corrección

La nota final de la parte II se regirá por los criterios de la tabla adjunta.

Se calcula según la calificación obtenida en cada apartado.

NO ACEPTABLE: Se obtiene un No Aceptable en alguno de los apartados.

ACEPTABLE: Se obtiene Aceptable en todos los apartados.

NOTABLE: Se obtiene, al menos, dos notables en uno de los apartados. Excepcionalmente con uno solo.

EXCELENTE: Se obtiene, al menos, dos excelentes en uno de los apartados. Excepcionalmente con uno solo.

Continúa en la siguiente cara con la rúbrica