Filogenia y búsqueda de patrones en la música de los trovadores mediante la distancia Levenshtein[1]

Angel Manuel Olmos

Con el fin de poder comprender el significado de un conjunto de datos dado, el investigador necesariamente ha de encontrar un patrón, un comportamiento coherente, o una explicación que permita especular con teorías que se puedan acercar a las reglas que generaron los datos observados.

La dificultad del análisis y ordenación crece exponencialmente con el número de elementos significativos para nuestra investigación. Encontrar similitudes y diferencias macroscópicas entre dos vasos producidos en una misma fábrica es bastante fácil si tomamos solamente como parámetros observables el perímetro del borde y el volumen que pueda alojar en su interior. No obstante, a medida que vamos incrementando el número de parámetros, como pueda ser la densidad del vidrio, el color, aumenta la dificultad de extraer conclusiones sobre el origen real de tales diferencias, y encontrar una explicación y prever posteriores resultados.

Una de las eiencias que más problemas tiene en este aspecto es la genética. Cada gen de un ser vivo está compuesto de millares, si no millones, de proteínas enlazadas de una forma en concreto. Encontrar diferencias y similitudes entre secciones de estas cadenas es una prioridad a la hora de poder establecer dependencias y filogenias de especies.

[1] Gran parte del contenido de este trabajo se presentó en Medieval and Renaissance Conference (Bristol, 2002), y fue financiado por la Université Paris IV-Sorbonne.

La música monódica comparte, aunque pueda despertar alguna sorpresa a primera vista, el mismo problema que la genética. Toda melodía se compone de un número finito de elementos, que se combinan y se repiten de la forma que el compositor o intérprete lo desea. El conjunto de estos elementos, que llamaremos alfabeto, son las alturas de las notas musicales. También exite otro alfabeto de duraciones, y podría añadirse otros tantos tales como intensidades o timbres. Desafortunadamente, para la música escrita, solamente los dos primeros alfabetos tienen la posibilidad de ser estudiados, dado que el resto no son definibles de forma objetiva, porque están sujetos a la interpretación.

Muchos musicólogos han dedicado sus energías y esfuerzos a encontrar patrones melódicos, préstamos de otras obras, o comportamientos del desarrollo de una obra musical, ya sea monódica o polifónica, por el método de la búsqueda aleatoria. Los hallazgos de patrones mediante este sistema están muy condicionados y limitados. Si desconocemos lo que buscamos, no podremos hallar nada.

La matemática nos enseña que es posible diseñar algoritmos mediante los cuales se pueden encontrar patrones en cadenas de datos, como las melodías musicales, sin que sea necesario que se le diga qué ha de buscar. De esta forma son los datos mismos los que generan patrones, y no el investigador el que intenta ajustar un patrón preestablecido a un conjunto de datos.

El objetivo de este artículo es el de mostrar con un ejemplo práctico cómo estas búsquedas se pueden aplicar al reconocimiento de patrones, y búsqueda de dependencias melódicas entre una colección de melodías.

Para ello hemos de definir brevemente las herramientas que utilizaremos en esta tarea. El lector interesado en un desarrollo teórico más completo de estos conceptos de base puede consultar la variada bibliografía existente al respecto[2]. En primer lugar introducimos la llamada Distancia Levenshtein (LD). LD mide la similitud entre dos cadenas de datos, a las que nos referiremos como cadena origen (s) y cadena final (t). LD es el número mínimo de inserciones, eliminaciones o substituciones de elementos de la cadena (s) para convertirla en (t).

Si la cadena s="Casa" y t="Cama", entonces LD(s,t)=1, puesto que solamente hace falta una substitución de caracteres. También podríamos decir que es 2, porque se puede eliminar primero la "s" de "Casa" y luego introducir la "m" de cama, pero LD se define como el mínimo número de operaciones necesarias, por lo que su valor es 1.

El ejemplo anterior es muy elemental, pero por ejemplo, medir la distancia entre los dos párrafos anteriores, que no son sino cadenas de letras, es más trabajoso. El algoritmo definido por Vladimir Levenshtein en 1966[3] nos previene de introducir operaciones innecesarias o redundantes, y nos asegura que obtiene el mínimo número de operaciones de las anteriormente descritas.

Con el fin de evitar excesivo aparataje matemático, presento una versión en Python del algoritmo que calcula la LD entre dos cadenas a y b. Esta formulación no es única, y tiene muchas variantes dependiendo del problema al que se aplique. La que aquí presento es aplicable de forma general. Para encontrar modificaciones y consideraciones teóricas sobre el algoritmo en sí, se puede consultar, entre otros, los trabajos de Hirschberg[4]

```
def distance(a,b):
    n, m = len(a), len(b)
    if n > m:
        # Así estamos seguros de que n <= m, para
utilizar el mínimo espacio O(min(n,m))
        a,b = b,a
        n,m = m,n

    current = range(n+1)
    for i in range(1,m+1):
        previous, current = current, [i]+[0]*m
        for j in range(1,n+1):
            add, delete = previous[j]+1,
current[j-1]+1
            change = previous[j-1]
            if a[j-1] != b[i-1]:
                change = change + 1
            current[j] = min(add, delete, change)

return current[n]
```

Veamos un ejemplo práctico de cómo trabaja el algoritmo. Supongamos que queremos saber a qué distancia están las palabras "Ismael" y "Samuel". Es decir, cuál es el camino que permite utilizar el menor número de cambios.

Aplicamos el algoritmo y obtenemos la siguiente tabla:

		-1	0	1	2	3	4	5
			S	a	m	u	e	l
-1		0	1	2	3	4	5	6
0	i	1	1	2	3	4	5	6
1	s	2	2	2	3	4	5	6
2	m	3	3	3	2	3	4	5
3	a	4	4	3	3	3	4	5
4	e	5	5	4	4	4	3	4
5	l	6	6	5	5	5	4	3

Como ambas palabras tienen 6 letras, el mejor alineamiento entre ellas es el natural, sin añadir ningún elemento más. Diferente sería el caso de dos cadenas de diferente longitud, como podemos ver en este otro ejemplo:

		-1	0	1	2	3	4	5	6	7
			P	r	o	f	e	s	o	r
-1		0	1	2	3	4	5	6	7	8
0	i	1	1	2	3	4	5	6	7	8
1	s	2	2	2	3	4	5	5	6	7
2	m	3	3	3	3	4	5	6	6	7
3	a	4	4	4	4	4	5	6	7	7
4	e	5	5	5	5	5	4	5	6	7
5	l	6	6	6	6	6	5	5	6	7

[2] Véase, por ejemplo Alberto Apostolico and Zvi Galil, eds., *Pattern Matching Algorithms* (New York, Oxford: Oxford University Press, 1997). y *Combinatorial Pattern Matching*, ed. Amihood Amir and Gad M. Landau, *12th Annual Symposium, Cpm 2001* (Jerusalem: Springer, 2001).

[3] Vladimir I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Cybernetics and Control Theory* 10 (1966).

[4] Véase, por ejemplo D. S. Hirschberg, "Serial Computations of Levenshtein Distances," in *Pattern Matching Algorithms*, ed. Alberto Apostolico and Zvi Galil (New York, Oxford: Oxford University Press, 1997).

Para alinear estas dos palabras existen estas posibilidades:

```
Ismae--l
Profesor
Ismae-l-
Profesor
Ismael--
Profesor
```

El resto de posibles alineaciones provocarían que LD sería mayor.

En el caso Ismael -> Samuel, vemos claramente que LD es 3, dado que solamente habría que cambiar la "I" por la "S", la "s" por la "a", y la "a" por la "u". En el caso Ismael -> Profesor, todas las opciones de alineación presentadas, que son las óptimas, llevan a una distancia 7.

Como el lector puede comprobar, a medida que aumenta el número de elementos, el cálculo de la distancia es más complicado. Como nota anecdótica, la distancia entre las dos frases de este párrafo es 98 (incluyendo signos de puntuación y espacios).

Las tablas que se han presentado en ambos ejemplos muestran el proceso de cálculo que sigue el algoritmo. Este prueba todas las alineaciones posibles de cada una de las letras de la primera cadena con la segunda, y se queda con la que menos operaciones de inserción/deleción necesita. Así sigue hasta el final. Cuando hay varias posibilidades, como en el caso de la "I" de Ismael -> Profesor, vemos cómo se trazan diferentes rutas alternativas de mínimas distancias. Como las diferencias son acumulativas, el número presente en la esquina inferior derecha es siempre LD.

Después de esta pequeña introducción teórica, vamos a aplicar el algoritmo general a dos cadenas musicales. Vamos a codificar las notas de la melodía "Stella splendens" del Llibre Vermell de Montserrat:

```
agafdfgbafgedfggfdc
```

E intentemos buscar el mejor alineamiento de esta otra pequeña melodía, compuesta de varias subcadenas de la anterior:

```
agafdgbafgdc
```

Aplicando el algoritmo que hemos presentado, el ordenador se da cuenta inmediatamente de la similitud del contenido de esta segunda cadena, y nos propone los siguientes alineamientos:

```
agafd-gba----f-g-dc
agafdfgbafgedfggfdc

agafd-gbaf-----g-dc
agafdfgbafgedfggfdc

agafd-gba----fg--dc
agafdfgbafgedfggfdc

agafd-gbaf----g--dc
agafdfgbafgedfggfdc

agafd-gbafg------dc
agafdfgbafgedfggfdc

agafd-gbafg-d-----c
agafdfgbafgedfggfdc
```

Según podemos ver en la tabla siguiente, el coste de conversión de una cadena en otra es 7, cifra que aparece en la esquina inferior derecha.

		-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
			a	g	a	f	d	f	g	b	a	f	g	a	d	f	g	g	f	d	c
-1		(0)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	a	1	(0)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	g	2	1	(0)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	a	3	2	1	(0)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	f	4	3	2	1	(0)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	d	5	4	3	2	1	(0)	(1)	2	3	4	5	6	7	8	9	10	11	12	13	14
5	g	6	5	4	3	2	1	1	(1)	2	3	4	5	6	7	8	9	10	11	12	13
6	b	7	6	5	4	3	2	2	2	(1)	2	3	4	5	6	7	8	9	10	11	12
7	a	8	7	6	5	4	3	3	3	2	(1)	(2)	(3)	(4)	(5)	6	7	8	9	10	11
8	f	9	8	7	6	5	4	3	4	3	2	(1)	(2)	(3)	(4)	(5)	(6)	7	8	9	10
9	g	10	9	8	7	6	5	4	3	4	3	2	(1)	(2)	(3)	(4)	(5)	(6)	(7)	8	9
10	d	11	10	9	8	7	6	5	4	4	4	3	2	2	(2)	(3)	(4)	(5)	(6)	(7)	8
11	c	12	11	10	9	8	7	6	5	5	5	4	3	3	3	3	4	5	6	7	(7)

Hemos constatado cómo este algoritmo tan simple es capaz, en pocas milésimas de segundo, de encontrar similitudes de una melodía. Sin embargo, debemos cuestionarnos si es posible diseñar un algoritmo que sea capaz de juzgar cuándo una melodía está realmente emparentada con otra, puesto que para el ordenador, todas ellas están emparentadas, aunque unas tengan una distancia mayor entre sí que otras. Uno de los problemas que se nos plantean a la hora de poder reprogramar este algoritmo para este fin es que LD no es realmente una distancia, aunque así se la denomine. Para que una aplicación sea una distancia, ha de cumplir estos tres postulados

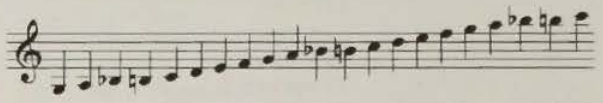
- d (a,b) ≥ 0 para todo a,b
  - Este se cumple, puesto que no hay LD negativas, dado que siempre se parte de cero, y las sustituciones siempre son sumas de números positivos.
- d (a,b) = d (b,a) para todo a,b
  - Este también se cumple. La demostración matemática es algo más larga y no merece la pena mostrarla aquí.
- d (a,b) ≤ d (a,c) + d (c,b) para todo a,b,c
  - Esta es la llamada desigualdad triangular, y no se cumple para LD, lo que invalida su carácter de distancia.

Por esta razón, no se puede acotar la LD entre dos puntos conociendo LD de esos dos puntos en relación a un tercero. Esto nos impide decir, por ejemplo, que si dos melodías tienen una LD baja respecto a una tercera, la LD entre ambas será también muy pequeña.

Por otro lado, debemos cuestionarnos si la percepción de similitud melódica se asemeja a la LD, y si no es así, si podemos modificarla para adecuarla a la realidad sonora. Varios han sido los intentos de adaptar la algorítmica a la búsqueda de similitudes musicales, con diferentes resultados[5]. La propuesta expuesta en este trabajo se dirige exclusivamente al tratamiento de la similitud melódica de la monodía, pero sus conclusiones y herramientas pueden ser fácilmente aplicables a otro tipo de música.

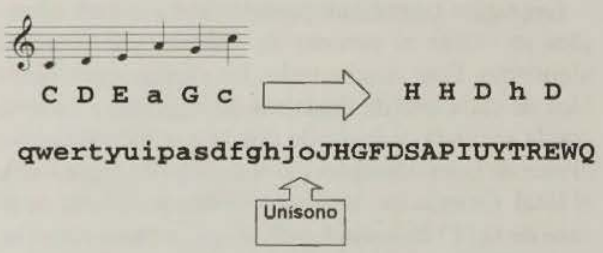
[5] Véase, por ejemplo, las aplicaciones presentadas en *Melodic Similarity. Concepts, Procedures and Applications, Computing in Musicology 11* (Cambridge MA, Boston: MIT Press, Center for Computer Assisted Research in the Humanities, 1998).

Vamos a establecer, en primer lugar, nuestro alfabeto de trabajo, que lo constituirán todas las letras que puedan formar parte de nuestras cadenas melódicas:



V A B H C D E F G a b h c d e f g a q 1 2

Solamente se han codificado las notas que aparecen en el repertorio que se va a analizar, pero es evidente que se puede extender todo lo que se quiera. El problema de trabajar con notas con alturas fijas es que si tratamos de emparejar dos melodías exactamente iguales, pero transpuestas, el algoritmo no se daría cuenta de que son la misma, porque la cadena de letras es muy diferente. Para solucionar esto, las melodías se pueden convertir a cadenas de intervalos, que tendrán, como es lógico, una letra menos de longitud, porque se necesitan dos notas para crear el primer intervalo.



Hemos creado entonces un alfabeto de intervalos que podrán representar cualquier melodía. Ahora, dos melodías idénticas o muy similares, pero transpuestas, serán transcritas en cadenas idénticas o muy similares. Se diseñó un sencillo programa llamado Torculus, que convierte cadenas melódicas en cadentas interválicas, de manera que esta conversión no entraña trabajo extra.

El repertorio que tomaremos como referencia para aplicar este algoritmo es el conjunto de piezas musicales de trobadores transcritas por Ismael Fernández de la Cuesta[6]. Es un material monódico y no se va a considerar el ritmo, dado que no es posible establecer una rítmica precisa a partir de los manuscritos que nos han llegado.

El número total de melodías contenidas en la citada obra es de 414. Como el objetivo de este

[6] Ismael Fernández de la Cuesta and Robert Lafont, *Las Cançons Dels Trobadors*, ed. Rodrigo de Zayas, *Opera Omnia* (Tolosa: Institut d'Estudis Occitans, 1979).

experimento es el de comprobar la recurrencia interna de patrones melódicos, se ha construido una serie de cadenas de intervalos que están presentes en alguna de las melodías originales. Esta simplificación del número de cadenas que se intentarán acoplar dentro del mayor número de melodías posibles permite que la velocidad de cálculo sea significativamente mayor, sin perder fiabilidad de los resultados. Además, no es objetivo de nuestro trabajo el encontrar melodías preestablecidas, aunque se podría hacer sin ningún problema. Si tomáramos todas las cadenas posibles resultantes de combinar aleatoriamente los elementos de nuestro alfabeto tendríamos el siguiente número de cada una de ellas:

Cadenas de 2 elementos:	1089
Cadenas de 5 elementos:	aprox. 40 millones
Cadenas de 10 elementos:	aprox. 1500 billones

Para efectuar la simplificación, se creó otro programa simple, llamado Porrectus, que troceó todas las melodías en todos los trocitos de 2, 5 y 10 notas consecutivas, y buscó aquellas que fueran únicas, para que no se repitieran cálculos inútiles. El número de cadenas se quedó como sigue:

Cadenas de 2 elementos:	223
Cadenas de 5 elementos:	6250
Cadenas de 10 elementos:	26751

Ya tenemos el conjunto de melodías en las que se buscarán patrones recurrentes (414), y las cadenas de intervalos que se buscarán. Ahora falta definir el método de búsqueda y los criterios que se tomarán para considerar si LD nos dice si dos cadenas son semejantes o no. Una melodía que sustituye un intervalo de segunda por uno de octava ciertamente será muy diferente a nuestros oídos si sustituye una segunda menor por una segunda mayor. Este hecho fue capital para considerar la introducción de una modificación del algoritmo clásico de cálculo de LD. Se construyó una matriz de pesos, que penaliza más los cambios entre intervalos alejados que entre los cercanos, de manera que los elementos de esta matriz quedan así

$$sustitución: \partial(i,j) = 1 - \frac{\left|\frac{1}{i-j}\right|}{2}$$

A la hora de insertar o eliminar notas, podemos encontrarnos con repercusiones de sonidos, que en realidad no constituyen una diferencia significativa. Así, la inserción o deleción de un elemento al unísono no tiene coste alguno, y se penaliza más la inserción de grandes intervalos, ya sean descendentes o ascendentes. Formulado matemáticamente, la matriz queda así:

$$inserción/\ deleción: \partial(0,i) = \partial(i,0) = 1 - \frac{\left|\frac{1}{i-17}\right|}{2}$$

Finalmente, y para que no se penalizara el hecho de que el patrón buscado no se encontrara al comienzo de la melodía analizada, se permitió que las inserciones al comienzo del patrón fueran consideradas con peso 0. El patrón se puede mover entonces a lo largo de la melodía para buscarle la mejor ubicación posible. El programa Scandicus se encargó de escanear y computar las distancias mínimas de estos patrones en todas las melodías, y elaborar una tabla de distancias. Dependiendo del número de intervalos de la cadena patrón, se fue más exigente a la hora de aceptar una identificación melódica. Cuanto más larga es la cadena patrón, más complicado es que se encuentre un referente exacto, así que se fijaron las siguientes LD para cada uno de estos tres conjuntos de cadenas patrón:

Cadenas de 2 intervalos:	LDmáxima=0
Cadenas de 5 intervalos:	LDmáxima=0
Cadenas de 10 intervalos:	LDmáxima=1

Evidentemente, cuanto más bajo sea el valor de LDmáxima, la tolerancia con la que aceptaremos que dos cadenas son semejantes será menor. Si aumentamos demasiado el valor de LDmáxima, podemos encontrar que el ordenador muestra cadenas que a nosotros nos parecen demasiado diferentes entre sí, aunque objetivamente esté más cercanas de lo que el sentido musical nos marca.

Uno de los puntos fuertes del sistema es que puede rastrear melodías migratorias. Si una parte de una melodía está a una altura, pero la segunda parte, tras un intervalo más grande, sigue el mismo diseño de la primera pero en un tono más agudo, el algoritmo detecta esta similitud, puesto que, a efectos interválicos, solamente se habrá modificado un elemento de la cadena.

Los resultados que se obtuvieron al comparar todas las cadenas de n elementos (n=2, 5, 10) con las melodías originales, fueron los siguientes:

Para n=2 y d=0, las cadenas más comunes fueron

1. dcd 414 ocurrencias (99%)
2. cha 414 ocurrencias (99%)
3. edc 413 ocurrencias (98%)
4. cdc 408 ocurrencias (97%)
5. dch 401 ocurrencias (95%)

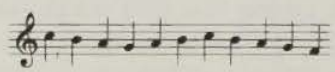
Nótese que las 5 primeras cadenas de 3 notas (n=2 intervalos), con d=0, están compuestas por grados conjuntos. Este resultado denota la excasa incidencia de los saltos en este repertorio

Para n=5 y d=0, las cadenas más comunes fueron

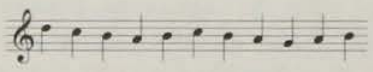
1. ahchaG 227 ocurrencias (54%)
2. cdefed 195 ocurrencias (46%)
3. chaGah 171 ocurrencias (40%)
4. dedcha 165 ocurrencias (39%)
5. hcdcha 161 ocurrencias (38%)

Al igual que ocurría con n=2, con n=5 (cadenas de 6 notas), las 5 primeras del ranking solamente contienen grados conjuntos. Para encontrar la primera cadena que contuviera un salto melódico deberíamos descender al puesto 56, con 60 ocurrencias (14%). Este resultado era previsible con n=2, pero no deducible de él.

Para n=10 y d=1, las cadenas más comunes fueron, con diferencia, estas dos:



44 ocurrencias (10, 5%)



37 ocurrencias (8, 9%)

Para cadenas tan largas, es sorprendente que en 1 de cada 10 melodías podamos encontrar alguna de estas dos cadenas de notas, o su transposición a otra altura.

La conclusión de este trabajo no es otra sino la de ofrecer esta nueva herramienta de análisis y búsqueda de melodías escondidas. Aplicaciones directas de este algoritmo incluyen:

- Búsqueda de melodías predeterminadas
  - Melodías tipo L'homme armé
  - Entonaciones modales
  - etc.
- Búsqueda de melodías invertidas o retrogradadas
  - Una simple modificación del algoritmo presentado que no lleva más de 5 minutos de programar daría la vuelta a las cadenas y podría encontrar cualquier transformación de las habituales
- Filiación de melodías
  - Si admitimos que las modificaciones a las melodías son mayores a medida que nos alejemos en el tiempo o en el espacio de la fuente original, un análisis n-dimensional de un conjunto de fuentes que ofrece la misma melodía arrojaría inmediatamente su árbol de dependencias
- Búsqueda de incipits exactos o aproximados en bases de datos tipo RISM

En resumen, todas aquellas búsquedas que los musicólogos han estado practicando a mano durante muchos años. La extensión del algoritmo a la polifonía es inmediata, ya que la polifonía la entiende el ordenador como líneas melódicas independientes. Si quisiéramos hacer búsquedas que implicaran a dos voces simultáneamente, ya habría que introducir mayores cambios al algoritmo, lo que dejamos para posteriores investigaciones.

APÉNDICE I

Debido a la capacidad del ordenador en el que se diseñaron los programas en su día, presento su listado en el obsoleto QBasic, ya que era el compilador que mi ordenador portátil podía ejecutar sin problemas de memoria en aquel momento. Se trata de programas hechos a medida del experimento, y algunas instrucciones pueden ser algo crípticas. No obstante, ofrezco estos listados por si al lector le pueden ser útiles para reproducir los pasos aquí dados.

Formato del archivo de entrada de melodías primario:

Datos		
Título	**Fuente**	**Melodía**
139a. Celh que non vòl ausir chançons	G 68v	aaaaGabaaGFGaaaGFEFGFEDDaaaaGabaaGFGaaaGFEFGFEDDDEDDCFG aacacdchaGFhaGaGFEFGFEDCCDEFGECDEFEDaaaaGaba

```
'Torculus
'Programa conversor de secuencias melódicas a interválicas
'Angel Manuel Olmos, 2002
'Université, Paris IV-Sorbonne
'-------------------------------------------------------
'

DECLARE FUNCTION convierte$ (melodia$)
DECLARE FUNCTION cogelinea$ (datos$)
CLEAR
CLS
version$ = "0.9b"

TYPE cancion
   titulo AS STRING * 60
   fuente AS STRING * 20
   melodia AS STRING * 300
END TYPE

DIM filebuffer AS cancion

DIM t$(3)
DIM puntoycoma(2)

'Inicio

PRINT "Torculus.  Versiónn"; version$
PRINT "Programa conversor de secuencias melódicas a interválicas"
PRINT "Angel Manuel Olmos, 2002"
PRINT "Universit, Paris IV-Sorbonne"
PRINT "-------------------------------------------------------"
PRINT
INPUT "Introduce fichero origen:", datos$
INPUT "Introduce fichero destino:", destino$

  OPEN datos$ FOR BINARY ACCESS READ AS #1
  OPEN destino$ FOR RANDOM ACCESS WRITE AS #2 LEN = LEN(filebuffer)
```

```
'Proceso

WHILE flag = 0
  linea$ = cogelinea$(datos$)
  IF linea$ = "seacabo" THEN flag = 1: GOTO salefuera

  'Separa los campos (El fichero fuente tiene titulo y melodia)
  i = 1
   FOR n = 1 TO LEN(linea$)
   IF RIGHT$(LEFT$(linea$, n), 1) = ";" THEN puntoycoma(i) = n: i = i + 1: GOTO adios
adios:
    NEXT n

    t$(1) = LEFT$(linea$, puntoycoma(1) - 1)
    t$(2) = MID$(linea$, puntoycoma(1) + 1, (puntoycoma(2) - 1) - puntoycoma(1))
    t$(3) = MID$(linea$, puntoycoma(2) + 1)

filebuffer.titulo = t$(1)
filebuffer.fuente = t$(2)
filebuffer.melodia = convierte$(t$(3))

PUT #2, , filebuffer

salefuera:
WEND

CLOSE #1
CLOSE #2

PRINT "-Hecho!"


FUNCTION cogelinea$ (datos$) STATIC

bit$ = " ": a$ = "": flag = 0
WHILE flag = 0
GET #1, , bit$
IF bit$ = CHR$(13) OR bit$ = CHR$(10) THEN flag = 1: GET #1, , bit$: GOTO sale
IF EOF(1) THEN a$ = "seacabo": GOTO sale
a$ = a$ + bit$
IF a$ = "seacabo" THEN flag = 1

sale:
WEND

cogelinea$ = a$

END FUNCTION

FUNCTION convierte$ (melodia$)

distancia$ = "V ABHC D EF G abhc d ef g q12"
convertida$ = ""
intervalos$ = "qwertyuipasdfghjoJHGFDSAPIUYTREWQ"
```

```
FOR n = 1 TO LEN(melodia$) - 1
  y$ = MID$(melodia$, n, 1)
  FOR p = 1 TO LEN(distancia$)
    IF y$ = MID$(distancia$, p, 1) THEN matchy = p
  NEXT p
  IF matchy > LEN(distancia$) THEN PRINT "ERROR!!"
  z$ = MID$(melodia$, n + 1, 1)
  FOR k = 1 TO LEN(distancia$)
    IF z$ = MID$(distancia$, k, 1) THEN matchz = k
 NEXT k

 'Comprueba los si's

'  IF MID$(distancia$, matchy, 1) = "B" OR MID$(distancia$, matchy, 1) = "b"
'  THEN sibbajo = 1 ELSE sibbajo = 0
'  IF MID$(distancia$, matchz, 1) = "B" OR MID$(distancia$, matchz, 1) = "b"
'  THEN sibalto = 1 ELSE sibalto = 0
'  IF MID$(distancia$, matchy, 1) = "H" OR MID$(distancia$, matchy, 1) = "h"
'  THEN sihbajo = 1 ELSE sihbajo = 0
'  IF MID$(distancia$, matchz, 1) = "H" OR MID$(distancia$, matchz, 1) = "h"
'  THEN sihalto = 1 ELSE sihalto = 0

'Convierte

  semitonos = matchz - matchy + 16
  convertida$ = convertida$ + MID$(intervalos$, semitonos + 1, 1)
  NEXT n
  convierte$ = convertida$

END FUNCTION
```

```
'Porrectus
'Programa creador de sub-secuencias de tamaño definido
'Angel Manuel Olmos, 2002
'Université, Paris IV-Sorbonne
'-------------------------------------------------------
'$DYNAMIC
CLEAR
CLS
version$ = "0.9b"
TYPE cancion
titulo AS STRING * 60
fuente AS STRING * 20
melodia AS STRING * 300
END TYPE

DIM filebuffer AS cancion
DIM buffer2 AS cancion


'Comienzo
PRINT "Porrectus.  Versión "; version$
PRINT "Programa creador de sub-secuencias de tamaño definido"
PRINT "Angel Manuel Olmos, 2002"
PRINT "Université, Paris IV-Sorbonne"
PRINT "-------------------------------------------------------"
PRINT

INPUT "Introduce nombre de archivo de datos:", datos$
INPUT "Introduce archivo de salida:", salida$
PRINT
INPUT "Introduce longitud de la cadena:", long$

longitud = VAL(long$)

OPEN datos$ FOR RANDOM ACCESS READ AS #1 LEN = LEN(filebuffer)
OPEN "temporal.bin" FOR RANDOM ACCESS WRITE AS #2 LEN = LEN(buffer2)

WHILE NOT EOF(1)
GET #1, , filebuffer
t$ = filebuffer.melodia
FOR n = 1 TO (LEN(RTRIM$(t$)) - longitud)
buffer2 = filebuffer
r$ = MID$(RTRIM$(t$), n, longitud)
buffer2.melodia = r$
PUT #2, , buffer2
NEXT n

WEND
PRINT : PRINT "-Hecho!"

CLOSE #1: CLOSE #2

'Descarta cadenas repetidas y escribe archivo final

DIM buffer3 AS cancion
```

```
  'Halla el fin del archivo temporal
  OPEN "temporal.bin" FOR RANDOM ACCESS READ AS #1 LEN = LEN(buffer2)
  inmax = 0
  WHILE NOT EOF(1)
  inmax = inmax + 1
  GET #1, , buffer2
  WEND
  PRINT "Número de cadenas procesadas: "; inmax
  CLOSE #1

infin = 0

OPEN "temporal.bin" FOR RANDOM ACCESS READ AS #1 LEN = LEN(buffer2)
OPEN salida$ FOR RANDOM ACCESS WRITE AS #2 LEN = LEN(buffer3)

FOR n = 1 TO inmax
  LOCATE 20, 1: PRINT "Hecho: "; (INT(n * 10000 / inmax)) / 100
  GET #1, n, buffer2
  rep = 0
    FOR i = n + 1 TO inmax
      GET #1, i, buffer3
      IF buffer3.melodia = buffer2.melodia THEN rep = 1: GOTO salee
    NEXT i
salee:
  IF rep = 0 THEN infin = infin + 1: PUT #2, infin, buffer2
NEXT n
PRINT : PRINT "Cadenas sin repeticiones: "; infin
CLOSE #1: CLOSE #2
KILL "temporal.bin"
```

```
'Scandicus
'Programa de búsqueda de patrones en melodias monódicas
'Angel Manuel Olmos, 2002
'Universit, Paris IV-Sorbonne
'-------------------------------------------------------

CLS
CLEAR
    intervalos$ = "qwertyuipasdfghjoJHGFDSAPIUYTREWQ"
DIM cadena
DIM melodia
DIM d(500, 10)
TYPE cancion
   titulo AS STRING * 60
   fuente AS STRING * 20
   melodia AS STRING * 300
END TYPE

DIM buffer AS cancion

DIM buffer2 AS cancion



version$ = "0.9a"

PRINT "Scandicus. Versión "; version$
PRINT "Programa de búsqueda de patrones en melodias monódicas"
PRINT "Angel Manuel Olmos, 2002"
PRINT "Universit, Paris IV-Sorbonne"
PRINT "-------------------------------------------------------"
PRINT : PRINT

INPUT "Introduce archivo de cadenas:", cadenas$
INPUT "Introduce archivo de melodias interválicas:", melodias$
INPUT "Introduce archivo de salida:", salida$
INPUT "Introduce la cota K:", k
INPUT "Introduce longitud de cadena:", ll
INPUT "Introduce cadena de comienzo:", start
INPUT "Introduce cadena de final:", final
PRINT



'Crea matriz de pesos
  DIM peso(33, 33)
  FOR i = 1 TO 33
  IF i = 17 THEN peso(0, i) = 0 ELSE peso(0, i) = 1 - (1 / ABS(i - 17)) / 2
  peso(i, 0) = peso(0, i)
  FOR j = 1 TO 33
  IF i - j = 0 THEN peso(i, j) = 0 ELSE peso(i, j) = 1 - (1 / ABS(i - j)) / 2
  NEXT j
  NEXT i
```

```
'Comienza la comparación
CLS
OPEN cadenas$ FOR RANDOM ACCESS READ AS #1 LEN = LEN(buffer2)
OPEN melodias$ FOR RANDOM ACCESS READ AS #2 LEN = LEN(buffer)
OPEN salida$ FOR BINARY ACCESS WRITE AS #3

   'Lee cuántas cadenas y melodías hay
   pcadenas = 0: pmelodias = 0
   WHILE NOT EOF(1)
   pcadenas = pcadenas + 1
   GET #1, , buffer2
   WEND
   CLOSE #1
   OPEN cadenas$ FOR RANDOM ACCESS READ AS #1 LEN = LEN(buffer2)
   WHILE NOT EOF(2)
   pmelodias = pmelodias + 1
   GET #2, , buffer
   WEND
   CLOSE #2
   OPEN melodias$ FOR RANDOM ACCESS READ AS #2 LEN = LEN(buffer)
   PRINT "N£mero total de melod¡as: "; pmelodias
   PRINT "N£mero total de cadenas:  "; pcadenas

'Comienza el cálculo

FOR pointcad = start TO final
r = 0
flag = 0
GET #1, pointcad, buffer2
cad$ = buffer2.melodia
c$ = LTRIM$(RTRIM$(cad$))

        REDIM cadena(LEN(c$))
        FOR i = 1 TO LEN(c$)
        FOR j = 1 TO 33
        IF MID$(c$, i, 1) = MID$(intervalos$, j, 1) THEN cadena(i) = j
        NEXT j
        NEXT i

FOR pointmel = 1 TO pmelodias - 1

LOCATE 22, 1: PRINT "Analizando cadena: "; pointcad; c$
LOCATE 22, 30: PRINT "melodía "; pointmel

GET #2, pointmel, buffer2
mel$ = buffer2.melodia
m$ = LTRIM$(RTRIM$(mel$))
        REDIM melodia(LEN(m$))
        FOR i = 1 TO LEN(m$)
        FOR j = 1 TO 33
        IF MID$(m$, i, 1) = MID$(intervalos$, j, 1) THEN melodia(i) = j
        NEXT j
        NEXT i
```

```
'Trocea la melodia

  FOR principio = 1 TO LEN(m$) - ll + 1
  trozo = principio + ll - 2
  WHILE trozo < LEN(m$)
  trozo = trozo + 1


minimo = 999999
flag = 0

            'Calcula la distancia entre la cadena y el cacho

             d(0, 0) = 0: r = 0

             FOR i = principio TO trozo

             r = r + 1
               d(r, 0) = d(r - 1, 0) + peso(ABS(melodia(i) - cadena(0)), 0)
             NEXT i

             FOR j = 1 TO ll
               d(0, j) = d(0, j - 1) + peso(0, ABS(cadena(j) - melodia(0)))
             NEXT j
             r = 0
             FOR i = principio TO trozo
             r = r + 1
             FOR j = 1 TO ll

               IF melodia(i) <> cadena(j) THEN
                 a = d(r, j - 1) + peso(0, ABS(cadena(j) - melodia(i)))
                 b = d(r - 1, j) + peso(ABS(cadena(j) - melodia(i)), 0)
                 c = d(r - 1, j - 1) + peso(melodia(i), cadena(j))
                 IF a <= b THEN min = a ELSE min = b
                 IF c < min THEN min = c
                 d(r, j) = min: GOTO rrr
              END IF
              IF melodia(i) = cadena(j) THEN d(r, j) = d(r - 1, j - 1)

rrr:

             IF d(r, j) <= k THEN flag = 1
             'PRINT d(r, j)
             NEXT j
             'PRINT "-------"; "Principio="; principio; "   Final="; trozo
             IF flag = 1 THEN flag = 0 ELSE flag = 0: trozo = 9999: GOTO sigue

             NEXT i
             IF minimo > d(r, ll) THEN minimo = d(r, ll)
             IF minimo <= k THEN PRINT "MATCH! "; MID$(m$, principio, trozo - principio +
1); " -> "; c$; "
Minimo:"; minimo: incr = incr + 1: minimo = 9999: GOTO siguiente

sigue:
  WEND

  NEXT principio
```

```
siguiente:
NEXT pointmel

pone$ = c$ + "," + STR$(incr) + CHR$(13) + CHR$(10)
PUT #3, , pone$: incr = 0
NEXT pointcad

CLOSE #1: CLOSE #2: CLOSE #3
END
```