Informática
Año 2014/2015
Facultad de CC. Matemáticas

Hoja de ejercicios 5

Subprogramas

18 de febrero de 2015

▷ 1. Procedimientos y funciones con listas

Desarrolla funciones y procedimientos (funciones sin return) para cada uno de los problemas siguientes:

- Borrar el elemento de la posición i de una lista.
- Saturar una lista a uno elemento dado x: si tenemos la lista l poner en cada posición i el máximo entre x y $l[i]$.
- Borrar todas las apariciones del elemento x de una lista.
- Filtar los elementos primos de la lista.
- Insertar un elemento en la posición i de la lista.
- Hacer una *rotación* de los elementos de la lista: el elemento en la posición $i + 1$ pase a estar en la posición i y el elemento 0 en la última posición de la lista.
- Intercambiar los elementos pares e impares: si i es una posición par intercambiar el elemento de la posición i con el de la posición $i + 1$.

▷ 2. Descomposición de un número

En este ejercicio proponemos la definición de funciones sencillas que permitan descomponer un número de múltiples formas.

Parte más significativa Escribe una función que devuelva la parte más significativa desde el n-ésimo dígito de un número.

Ejemplo

4º dígito

$\underline{35}1372$

Parte más significativa a partir del 4º dígito

Parte menos significativa Escribe una función que devuelva la parte menos significativa hasta el n-ésimo dígito de un número.

Ejemplo

4º dígito

$351\underline{372}$

Parte menos significativa hasta el 4º dígito

Dígito n-ésimo Escribe una función que nos devuelva el dígito n-ésimo de un número.

▷ 3. Conversión de caracteres a números

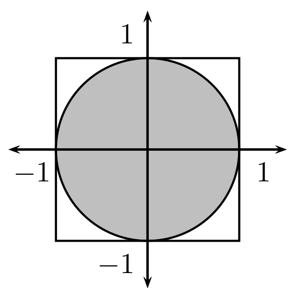
En este ejercicio meditaremos sobre el funcionamiento de la llamada a `int(s)` siendo `s` un `string`.

Conversión de caracteres a números en base 10 Escribe una función que reciba una cadena de caractéres con dígitos del 0 al 9 y devuelva el corespondiente entero.

Conversión de caracteres a números en una base arbitraria Escribe la función anterior pero para trabajar con números en una base cualquiera.

▷ 4. Aproximación hacia π con dardos

Considera el siguiente experimento: se dispone de una diana de radio unidad centrada en el origen de coordenadas y del cuadrado en el cual se inscribe dicha diana:



Si se efectúa un buen número de lanzamientos de dardos, uniformemente distribuidos en el cuadrado circunscrito $[-1,1] \times [-1,1]$, el número de los que caerán en la diana será aproximadamente proporcional a su superficie:

$$\frac{\text{número de disparos dentro}}{\text{número de disparos total}} \simeq \frac{\text{superficie de la diana}}{\text{superficie del cuadrado}}$$

y, como sabemos que

$$\frac{\text{superficie de la diana}}{\text{superficie del cuadrado}} = \frac{\pi}{4}$$

se puede estimar que

$$\pi \simeq 4\frac{\text{número de disparos dentro}}{\text{número de disparos total}}$$

aproximación que será, probablemente, tanto más precisa cuanto mayor sea el número de lanzamientos.

Se pide un programa que efectúe un buen número de lanzamientos, que tantee los aciertos y deduzca de ahí una aproximación de π. (Para más información consúltese la pista 1.)

Notas bibliográficas La idea de este enunciado y otras sobre simulación de variables aleatorias pueden ampliarse en [Ben84, Dew85], entre otras muchas referencias.

▷ 5. Conjetura de Goldbach

En una carta escrita a Leonhard Euler en 1742, Christian Goldbach (1690–1764) afirmó (sin demostrarlo) que todo número par es la suma de dos números primos. Para poner a prueba esta conjetura (hasta cierto punto, claro está), basta con avanzar a través de los primeros n números pares hasta encontrar uno que no verifica esa propiedad, o hasta llegar al último, ratificándose la conjetura hasta ese punto,

```
k = 0
seCumpleHastaK = true
while seCumpleHastaK and k <= n:
  k = [[el siguiente par]]
  [[Tantear la descomposición de |k|]]
  if [[falla el intento]]:
    [[|seCumpleHastaK| se anota como falso]]
```

donde cada tanteo se puede expresar mediante un subprograma que responda a la siguiente llamada,

```
descomponer(numPar)
```

devolviendo tres valores `conseguido, sumando1, sumando2`; esto es, un subprograma que, dado un entero `numPar` (supuestamente positivo y par), busca una descomposición del mismo en dos sumandos `sumando1` y `sumando2` primos, indicando además si lo ha `conseguido` o no. Esa descomposición de `numPar` se busca probando pares de sumandos,

$$(1, \texttt{numPar} - 1), (2, \texttt{numPar} - 2), \ldots$$

hasta que ambos sean primos o bien el primero supere al segundo, para no repetir los tanteos finales,

$$\ldots, (\texttt{numPar} - 2, 2), (\texttt{numPar} - 1, 1)$$

que son iguales a los iniciales.

Descomposición Escribe en Python el subprograma anterior, suponiendo que existe una función que verifique la primalidad de un número. (De hecho, se ha desarrollado en el ejercicio 14.)

Conjetura de Goldbach Finalmente, desarrolla el programa correspondiente al algoritmo completo, siguiendo los pasos descritos.

Notas bibliográficas Esta conjetura aparece recogida en las *Meditaciones algebraicas* de Edward Waring (1734–1793) junto con otros resultados interesantes. Te proponemos desarrollar un programa que permita comprobar los siguientes enunciados (pruébalos con los primeros millares de los números naturales):

- Conjetura: todo entero impar es un número primo o la suma de tres números primos.
- Teorema (Leonhard Euler, 1707–1783): todo entero positivo es la suma de, a lo más, cuatro cuadrados.
- Teorema (John Wilson, 1741–1793): para todo primo p, el número $(p-1)!+1$ es múltiplo de p.

En la novela [Dox00] se relata la historia de un matemático que únicamente vivió para intentar demostrar la conjetura de Goldbach.

▷ 6. Generación de primos

- Realiza una función que entero n, calcule el menor primo mayor o igual que n. Modifica esa función para que admita (y use) una lista con los primos menores o iguales a n.
- Realiza una función que genere una lista de números primos. Dicha función debe tener 2 parámetros: uno conteniendo la lista con los primeros números primos y otro con el número de números primos adicionales que se quieren generar.
- Realiza un programa que tenga como parámetro el nombre de un fichero y un entero n Ese fichero debe contener una lista con los primeros números primos. El programa debe calcular los siguientes números primos y añadirlos al fichero.

▷ 7. Triángulo de Pascal

Consideremos el triángulo de Pascal,

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 1 & & 1 & & & \\
 & & 1 & & 2 & & 1 & & \\
 & 1 & & 3 & & 3 & & 1 & \\
1 & & 4 & & 6 & & 4 & & 1 \\
 & & & & \cdots & & & &
\end{array}$$

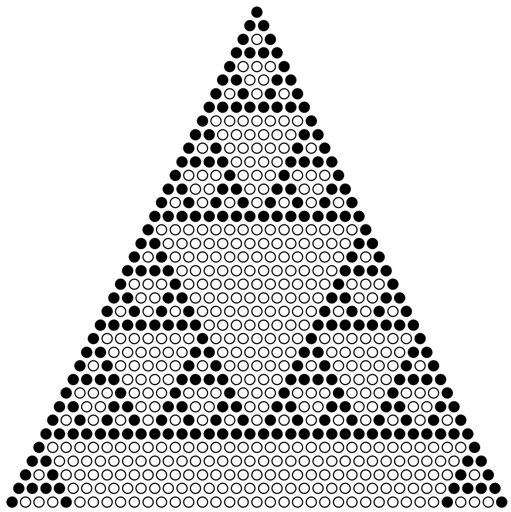
cuyos elementos pueden generarse bien directamente mediante números combinatorios, siendo $\binom{i}{j}$ el elemento j-ésimo de la fila i-ésima (para $0 \leq j \leq i$),

$$\begin{array}{ccccccccc}
 & & & & \binom{0}{0} & & & & \\
 & & & \binom{1}{0} & & \binom{1}{1} & & & \\
 & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & \\
 & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & \\
\binom{4}{0} & & \binom{4}{0} & & \binom{4}{0} & & \binom{4}{0} & & \binom{4}{0} \\
 & & & & \cdots & & & &
\end{array}$$

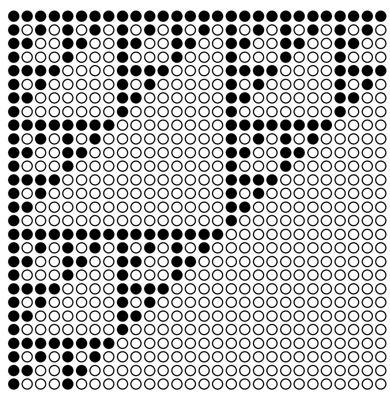
o bien sumando los dos situados sobre él:

$$\begin{array}{ccccccccccccccccccc}
 & & & & & & & & & \cdots & & & & & & & & & \\
 & 1 & & 8 & & 28 & & 56 & & 70 & & 56 & & 28 & & 8 & & 1 & \\
 & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & & \swarrow\searrow & \\
1 & & 9 & & 36 & & 84 & & 126 & & 126 & & 84 & & 36 & & 9 & & 1 \\
 & & & & & & & & & \cdots & & & & & & & & &
\end{array}$$

Sustituyendo cada número impar por un punto negro y cada número par por uno blanco, se obtiene el triángulo siguiente:



Se pide desarrollar un programa que reproduzca ese dibujo, aunque rotando la figura del siguiente modo, para aprovechar mejor la pantalla,



Notas bibliográficas En este ejercicio se han presentado variaciones sobre el tema de la construcción del triángulo, enfocando distintos problemas de eficiencia que se presentan a menudo. Sin embargo, esta sencilla construcción es abundante en propiedades: además de estar en él presentes los números naturales y, obviamente, los números combinatorios, no es difícil descubrir los triangulares (de diferentes órdenes), los de Fibonacci, etc. Consulta por ejemplo [Enz01]. Los artículos y libros sobre el tema son muy abundantes. Entre ellos, citamos el capítulo 15 de [Gar87], de donde proviene la idea de jugar con la paridad de los coeficientes binomiales. Este atrayente libro incluye, a su vez, una pequeña colección de reseñas bibliográficas sobre el mencionado triángulo.

Un poco de historia El triángulo de Pascal debe su nombre a Blaise Pascal (1623–1662), que en 1653 publicó *Traité du triangle arithmétique*, obra en la que se incluye el estudio más importate sobre el tema. Sin embargo, el triángulo de Pascal era conocido por los matemáticos árabes. Al-Samawal (1130–1180) cita un trabajo de Al-Karaji (953–1029) en el que se daba la construcción de dicho triángulo.

▷ **8.** ***k*-repeticiones en un array**

Diseña un subprograma que, a partir de una lista , v, que contiene n números enteros, y del entero $k > 0$, decida si en v existe algún valor que se repita *exactamente* k veces. Si existe alguno, el subprograma debe detenerse en cuanto encuentre el primero de ellos, devolviendo que sí y el correspondiente valor. Por el contrario, si no existe ningún valor que se repita exactamente k veces, devolverá simplemente que no.

Ejemplo Consideremos la lista $v = [8, 5, 1, 2, 1, 4, 1, 5, 1, 6]$ que tiene 10 elementos. Si pasásemos a nuestro subprograma el vector v y el entero $k = 4$, el subprograma debería encontrar, si es que existe, un valor que se repitiese exactamente 4 veces en el vector v. En este caso, sí que existe dicho valor, que es el 1, y por tanto, el subprograma debería devolver en una variable el valor cierto y en otra variable el valor que se repite exactamente k veces, el 1.

Sin embargo, si en las mismas condiciones ($k = 4$), v fuera la lista $[2, 1, 1, 2, 1, 4, 1, 5, 1, 7]$ el procedimiento debería devolver en una variable el valor falso, pues ningún valor se repite exactamente k veces. Lo mismo ocurriría con la lista $[5, 5, 2, 2, 1, 4, 1, 1, 5, 6]$.

▷ **9. Recorrido espiral**

Dada una matriz cuadrada de dimensión arbitraria escribe un subprograma que recorra la matriz en espiral.

Ejemplo Si consideramos la matriz de dimensión 4,

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

un recorrido en espiral de la misma, que comienza en la casilla (1,1) y que va hacia la derecha, mostraría los números en orden creciente 1, 2, 3, 4, ... 14, 15 y 16.

Si consideramos esta otra matriz, también de dimensión 4, su recorrido en espiral, desde la casilla (1,1) y hacia la derecha, mostraría la siguiente secuencia:

1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

1, 15, 14, 4, 9, 5, 16 2, 3, 13, 8, 12 6, 7, 11 y 10 (Para más información consúltese la pista 2.)

▷ **10. Palíndromos**

Un palíndromo es una palabra o frase que se lee igual de izquierda a derecha que de derecha a izquierda; por ejemplo, la palabra “anilina”, o las frases “anula la luz azul a la luna” y “sé verla al revés”, despreciando las diferencias entre mayúsculas y minúsculas, los espacios en blanco entre palabras y las tildes.

De igual forma, decimos que la expresión de un número en una determinada base es un palíndromo, o capicúa, si dicho número se lee igual de izquierda a derecha que de derecha a izquierda, por ejemplo 81318 en base 10, o el número nueve expresado en base 2, 1001. Obviamente, suponemos que los números siempre están expresados por su representación mínima, es decir, sin ceros redundantes. Por ejemplo, 100 en base 10 no es un palíndromo, ya que no se lee igual de derecha a izquierda de izquierda a derecha; no consideramos expresiones de 100 como 00100, que en principio también son el mismo número, ya que entonces podríamos decir que sí que es un palíndromo.

Reverso de un número Escribe una función que tenga como parámetro un número entero y devuelva el reverso de dicho número. (Para más información consúltese la pista 3.)

Número palíndromo Escribe una función que tenga como parámetro de entrada un número y nos indique si éste es palíndromo o no.

▷ 11. Conjetura para la formación de palíndromos

Consideremos el siguiente procedimiento:

> Dado un número, lo sumamos a su reverso. Si esta suma es un palíndromo, entonces paramos; y si no, repetimos el proceso con el número obtenido de dicha suma, hasta dar con un palíndromo.

Una curiosa conjetura de teoría de números afirma que, partiendo de cualquier número natural expresado en base 10, el procedimiento anterior para, y por tanto nos lleva a un palíndromo (véase el ejercicio 10).

Ejemplo Aquí vemos un ejemplo de cómo funciona la conjetura. Supongamos que partimos del número 59 lo sumamos a su reverso y obtenemos 154. Repetimos la operación con 154: la suma con su reverso es 605. Por último, al sumar 605 a su reverso obtenemos un palíndromo

$$59 \to \begin{array}{r} 59 \\ +\quad 95 \\ \hline 154 \end{array} \to \begin{array}{r} 154 \\ +\quad 451 \\ \hline 605 \end{array} \to \begin{array}{r} 605 \\ +\quad 506 \\ \hline 1111 \end{array} \to 1111$$

Conjetura Escribe un programa que lleve a cabo el procedimiento descrito por la conjetura para encontrar un palíndromo a partir de un número. (Para más información consúltese la pista 4.)

Terminación Una conjetura es una hipótesis no demostrada ni refutada. La conjetura que estamos considerando describe un método que, en caso de terminar, conduce a un palíndromo a partir de un número. Aunque el método en general termina, con algunos números no se sabe qué ocurre: por ejemplo, con el 196 se han realizado centenares de iteraciones pero no se ha conseguido llegar a un palíndromo.

Escribe un programa que lleve a cabo el procedimiento descrito por la conjetura para encontrar un palíndromo a partir de un número. El número de iteraciones tiene que limitarse para así evitar los desbordamientos y asegurar que el programa termina.

Notas bibliográficas Desde que hay personas que piensan y demuestran ha habido conjeturas. Las conjeturas son misteriosas y siempre han generado leyendas y mitos. Muchas conjeturas en matemáticas son más populares que los teoremas más importantes. En el ejercicio 5 encontrarás una de las conjeturas más conocidas. La conjetura de la formación de palíndromos aparece en [Gar95], un libro ligero, ameno y divertido del maestro de la divulgación matemática Martin Gardner (1914–).

▷ **12. Suma Acumulada**

Supongamos que tenemos una lista v llenade números. Queremos realizar una serie de sumas acumuladas sobre los valores de dicho lista de la siguiente forma:

- Al principio la lista tiene una serie de valores,

v_1	v_2	$\cdots$	v_{n-1}	v_n

- En la primera vuelta sumamos cada componente $v[i]$ a $v[i+1]$ y dejamos el valor en $v[i+1]$,

v_1	v_1+v_2	v_2+v_3	$\cdots$	$v_{n-2}+v_{n-1}$	$v_{n-1}+v_n$

- Repetimos el proceso de sumar $v[i]$ a $v[i+1]$ y dejar el valor en $v[i+1]$ comenzando por la segunda componente y después por la tercera... así hasta el final de la lista .

Escribe un programa que dada una serie de números calcule el resultado de aplicarles el proceso anterior.

Ejemplo Consideremos la lista con los valores siguientes:

4	2	5	3

En la primera iteración se realizarían las siguientes sumas en la lista

4	$4+2$	$2+5$	$5+3$

y quedaría con los valores:

4	6	7	8

En la siguiente vuelta comenzaríamos en la segunda posición y realizaríamos las siguientes sumas:

4	6	$6+7$	$7+8$

obteniendo la lista

4	6	13	15

Finalmente, en la última vuelta tendríamos que sumar las dos últimas componentes

4	6	13	$13+15$

y el resultado final sería la lista

4	6	13	28

▷ **13. Entropía de un vector**

Sea v un vector de enteros con N posiciones numeradas del 1 al N. La *entropía de una posición* $i \in \{1,\ldots,N\}$ se define como en número de posiciones mayores que i que contienen elementos menores que el de la posición $i-$ésima. Formalmente se puede definir como sigue:

$$entropiaPos(i,v) = card(\{j \in \{i+1,\ldots,N\} \mid v[i] > v[j]\})$$

donde *card* denota el cardinal o número de elementos de un conjunto. La *entropía total* del vector v se define como la suma de las entropías de sus posiciones, es decir:

$$entropia(v) = \sum_{i=1}^{N} entropiaPos(i, v)$$

Escribe una función que calcule la entropía total de un vector. (Para más información consúltese la pista 5.)

Ejemplo La entropía del vector $[7, 9, 9, 1, 5]$ se calcula así:

- para la primera posición, que contiene el elemento 7, la entropía es 2 puesto que en posiciones posteriores hay dos elementos menores, el 1 y el 5;
- para la segunda posición, primer 9, es 2 porque hay dos elementos menores a continuación, el 1 y el 5;
- para la tercera posición, segundo 9, también es 2 igual que antes;
- para la cuarta posición, el 1, la entropia es 0;
- y para la ultima posición, el 5, también es 0. De hecho, sea cual sea el valor que contenga esta última posición, la entropía es siempre 0.

Sumando la entropía de todas las posiciones, la entropía total del vector es $2+2+2+0+0 = 6$. Naturalmente, para un vector ordenado en forma creciente la entropía será siempre 0.

▷ 14. Generación de primos

Considera las siguientes propiedades relacionadas con la primalidad de un número:

- Un entero superior a 2 sólo puede ser primo si es impar.
- Un entero n superior a 3 sólo puede ser primo si verifica la propiedad $n^2 \bmod 24 = 1$.
- Un entero positivo n es primo si y sólo si no tiene divisores entre 2 y $\lfloor \sqrt{n} \rfloor$.

y, basándote en ellas, escribe las siguientes funciones:

Filtro: "mod24-1" Una función que indique si un entero verifica la propiedad siguiente:

$$n^2 \bmod 24 = 1$$

Filtro: divisores Una función que indique si un número es primo, buscando si tiene algún divisor entre 2 y $\lfloor \sqrt{n} \rfloor$.

Filtro: primos Una función que indique si un número es primo o no, descartando primero los pares, comprobando luego la propiedad "mod24-1," y finalmente buscando sus posibles divisores.

Notas bibliográficas La literatura sobre los números primos es muy abundante. La referencia [Pom83] es un interesante punto de partida para ampliar información sobre este tema.

▷ **15. El factorial en la sociedad del futuro**

(*) Ahora hay mucha policía; nos aseguran que es necesaria. Mi abuela me ha contado que antes sólo existían dos o tres policías, distintas pero iguales; ahora hay una sola, pero con tantas personas que necesitan vigilarse mutuamente. Por eso está organizada jerárquicamente, y los de un nivel vigilan a los del siguiente hasta que se llega a la ciudadanía.

Nadie conoce el primer nivel; no se sabe cuántas personas lo forman; dicen que se llama CIA y que vigila al siguiente nivel. El segundo nivel se llama BICIA; tiene una sola persona (dicen que es un rey) que vigila a las dos personas del siguiente nivel. El tercer nivel se llama TRICIA; cada uno de sus dos miembros vigila a tres personas en el siguiente nivel. Así se sigue, por varios niveles más de los que poca gente conoce el nombre; siempre se cumple que una persona del nivel n vigila a $n+1$ personas del siguiente nivel, y que a cada persona del nivel $n+1$ sólo la vigila una del nivel n; se deduce que en cada nivel hay $(n-1)!$ personas. Al último nivel lo llamamos POLICIA porque no sabemos exactamente a cuántas personas tocan. Las personas que quedamos somo ciudadanos, aunque se está extendiendo el término CIADAANO.

Mi abuela dice que antes se podía trabar conversación con cualquier persona desconocida; que antes los policías tenía un carnet y un traje especial. Ahora no es así; sólo hay una documentación, no hay una ropa especial. En busca de la prima de productividad, la policía para intempestivamente a la gente por la calle. En un principio hubo muchos altercados y tiroteos entre la misma policía; luego se impuso la costumbre de gritar el nivel para que los demás supiéramos por quién tomar partido; pero ahora, para hacerlo todo más sutil, se grita el número de la documentación. Afortunadamente la documentación está numerada consecutivamente por niveles; es fácil hacerse una idea de quién manda más, pero como los niveles son tan grandes, es difícil saber si dos números son del mismo nivel o los separan muy pocos niveles.

El otro día me pararon el agente número 47335 y la agente número 80157; me trataron de muy malos modos; no sabía que sólo estaban un nivel por encima de mí y que les podía haber interpuesto una denuncia. Ahora voy a escribir un programa en mi microcomputadora portátil para auxiliarme la próxima vez; le daré dos números de documentación y me dirá cuántos niveles los separan.

▷ **16. Juegos perdedores ganan**

(*) Queremos realizar un programa que permita simular juegos de azar y así observar su evolución a largo plazo. Para ello nos vamos a ir a nuestro casino particular en el que se encuentran los siguientes juegos:

Casi iguales pero no tanto El juego de *casi iguales pero no tanto* tiene las siguientes reglas: se lanza una moneda en la que sale cara con una probabilidad del 49.5 % y cruz con una del 50.5 %; el jugador gana un punto si sale cara y pierde un punto si sale cruz.

Escribe un programa que simule el juego a largo plazo, y que muestre los datos para que se aprecie la evolución de las ganancias o pérdidas del jugador.

Por tres es al revés El juego de *por tres es al revés* se juega con dos monedas M_1 y M_2. Al lanzar la primera de ellas, M_1, sale cara con un 9.5 % de probabilidad y cruz con un 90.5 %; al lanzar la segunda, M_2, sale cara con una probabilidad del 75.5 % y cruz con el 24.5 %. El jugador gana un punto si sale cara y pierde un punto si sale cruz. Si el capital con el que cuenta el jugador es múltiplo de tres, se lanza la moneda M_1, y si no se lanza la moneda M_2. El jugador puede comenzar con un capital arbitrario, en puntos.

Escribe un programa que simule el juego a largo plazo y que muestre los datos para que se aprecie la evolución de las ganancias o pérdidas del jugador.

Mezclando juegos Si has realizado los ejercicios anteriores, te habrás dado cuenta de que nuestro casino es un negocio rentable. En los juegos que hemos propuesto el jugador lleva la peor parte y, a largo plazo, siempre acaba perdiendo.

Supongamos que abreviamos por C al juego *casi iguales pero no tanto* y por T al juego *por tres es al revés*. Se abre una nueva mesa de juego en nuestro casino que consiste en alternar jugadas a los dos juegos anteriores de la siguiente forma CCTTCCTTCCTT..., es decir dos jugadas al juego C, dos jugadas al juego T... ¿Apostarías tus puntos a este nuevo juego?

Escribe un programa que simule el juego a largo plazo y que muestre los datos para que se aprecie la evolución de las ganancias o pérdidas del jugador. El jugador puede comenzar con un capital arbitrario, en puntos. ¿Te sorprende el resultado?

(Para más información consúltese la pista 6.)

Notas bibliográficas El resultado de mezclar juegos perdedores para encontrar un juego ganador es muy reciente y se conoce como *paradoja de Parrondo*. El propio autor, Juan Parrondo (1964–), lo explica muy claramente en el artículo [Par01]. En realidad, no se trata de una verdadera *paradoja* matemática, sino de un resultado sorprendente. El libro *Fotografiando las matemáticas* [Mar00] ofrece cuidadas fotografías que muestran cómo las matemáticas aparecen en muy diversos ámbitos de la vida. Uno de los cincuenta artículos que aglutina esta obra está dedicado al trabajo de Parrondo.

▷ 17. Código de sustitución monoalfabético

(*) Un código monoalfabético de sustitución es aquél en el que las letras del alfabeto en que se escribe el mensaje original son sustituidas por otras letras del mismo alfabeto. Es decir, dado un alfabeto, calculamos una permutación suya y, para cifrar un mensaje, simplemente cambiamos las letras según nos indique esa permutación.

Veamos cómo se cifra un mensaje con este tipo de códigos. Consideremos la siguiente permutación del alfabeto usual:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
J	W	K	U	L	Z	B	P	N	H	E	C	G	Y	V	R	M	Q	S	T	D	A	X	I	F	O	Ñ

Con este código, para cifrar la palabra "DARDO", tenemos que consultar en la primera fila cada una de las letras de la palabra y escribir la letra que se encuentra en la fila de abajo: para la D escribimos U, para la A escribimos J, y así sucesivamente hasta obtener "UJSUQ". Observa que las dos apariciones de la misma letra D en la palabra "DARDO" son codificadas por la misma letra U.

La clave necesaria en este tipo de cifrados es una permutación del alfabeto que vayamos a utilizar. Esta clave es difícil de retener de memoria y por tanto se necesita su almacenamiento y gestión. Para reducir estos inconvenientes, se puede recurrir a códigos pseudoaleatorios. Por ejemplo, a partir de una palabra que se utiliza como clave. Supongamos que queremos realizar un código monoalfabético cuya clave podamos recordar, elijamos como clave la palabra "OCULTA". Una forma posible de definir el código es:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
O	C	U	L	T	A	B	D	E	F	G	H	I	J	K	M	N	Ñ	P	Q	R	S	V	W	X	Y	Z

Hemos utilizado la palabra clave para definir el comienzo de la permutación y luego hemos seguido el orden alfabético. Si la palabra clave tiene letras repetidas, entonces basta con no escribirlas de nuevo. Por ejemplo si consideramos "CRIPTOGRAMA" como palabra clave, entonces el cifrado quedaría

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
C	R	I	P	T	O	G	A	M	B	D	E	F	H	J	K	L	N	Ñ	Q	S	U	V	W	X	Y	Z

Una permutación del alfabeto Define un subprograma que calcule una permutación aleatoria del alfabeto que vayas a utilizar para escribir los mensajes.

Una permutación de un alfabeto cualquiera Define un subprograma que dado un alfabeto cualquiera devuelva una permutación de dicho alfabeto. Ahora el alfabeto tiene que tratarse como un parámetro más del subprograma.

Almacenamiento en ficheros Uno de los grandes problemas que pueden tener este tipo de códigos es recordar la clave. En este caso la clave es la permutación del alfabeto que hemos elegido. Modifica adecuadamente el programa del apartado anterior para que la permutación del alfabeto sea guardada en un fichero.

Código monoalfabético Escribe subprogramas para cifrar y descifrar mensajes utilizando un código monoalfabético. La clave, tanto para cifrar como para descrifrar, es la permutación del alfabeto que hayamos elegido. (Para más información consúltese la pista 7.)

Elimina letras repetidas Escribe un pequeño subprograma que reciba una palabra y que construya otra *palbr* (aunque no exista en el diccionario) que sea idéntica a la primera pero que elimine las repeticiones de letras.

Ideas para la simplificación de la clave Escribe un programa que construya una permutación de un alfabeto basada en una palabra clave. Es decir, una permutación que comience con las letras no repetidas de la palabra y continue con las letras restantes en el orden alfabético.

Cifrar y descifrar con clave simplificada Escribe programa para cifrar y descifrar mensajes utilizando un código de sustitución monoalfabético basado en una palabra clave. ¿Qué ocurre con las últimas letras del alfabeto cifrado? ¿Puede esto afectar a la seguridad del código? ¿Se te ocurre alguna solución sencilla?

▷ 18. Sin los elementos de las ráfagas

Una *rafaga* en una secuencia es la aparición consecutiva de un mismo elemento. Un elemento *e tiene ráfagas* en una secuencia si hay alguna ráfaga formada por *e*.

- Escribe un subprograma que elimine de una secuencia todos los elementos que tienen ráfagas.

  Ejemplo El subprograma debería convertir la secuencia

  $$1, 2, 2, 2, 4, 5, 7, 5, 4, 2, 1, 1$$

  en

  $$4, 5, 7, 5, 4.$$

- Haz otro subprograma que sólo elimine los elementos que tengan ráfagas más largas que un cierto n.

  Ejemplo Si consideramos la secuencia

  $$1, 2, 2, 2, 4, 5, 7, 5, 4, 2, 1, 1$$

  al quitar los elementos con ráfagas de más de 2 elementos, quedaría

  $$1, 4, 5, 7, 5, 4, 1, 1.$$

▷ **19. Matrices**

En este ejercicio vamos a desarrollar una estructura de datos y los subprogramas necesarios para realizar algunas operaciones sobre matrices.

Definición del tipo Describe un tipo de datos adecuado para guardar matrices $n \times m$, donde n y m pueden variar durante la ejecución del programa.

Llenar y mostrar Queremos ahora dos procedimientos que permitan utilizar fácilmente las matrices definidas con el tipo anterior. Uno de los procedimientos se encargará de llenar con datos aleatorios una matriz. El segundo de los procedimientos se encargará de escribir los datos de una matriz por pantalla.

Leer de un fichero Escribe una función para leer una matriz de un fichero de texto. Debes describir el formato del fichero en el que está guardada la matriz.

Cálculo de la matriz traspuesta Implementa una función o procedimiento que calcule la matriz traspuesta a una dada.

Matrices adjuntas Dada una matriz M y una posición (i, j), dentro de ella, la matriz adjunta C se define como la matriz formada por los elementos de M que no se encuentran en la fila i ni en la columna j. Implementa un subprograma que calcule matrices adjuntas.

Cálculo del determinante Implementa una función que nos permita calcular el determinante de una matriz considerando la estructura de datos definida en el apartado anterior. Puede ser útil el subprograma para el cálculo de la matriz adjunta desarrollado en otro apartado.

Multiplicación Escribe el código adecuado para multiplicar matrices.

Composicion Definimos la operación *composición* de matrices, que denotamos por $[-|-]$, como sigue: Si A es una matriz $n \times m$ y B es una matriz $n \times k$, entonces la composición de A y B, $C = [A|B]$, es una matriz de $n \times (m + k)$ donde las primeras m columnas de C son las de la matriz A y las últimas k columnas de C son las de la matriz B.

▷ **20. Buscar nombres de funciones.**

Escribe una función que, dado un programa Python correcto, devuelva una lista con los nombres de todas las funciones que aparecen en dicho programa.

▷ **21. Palabras de un texto.**

Escribe una función que, dado un string, devuelva una lista con las palabras de dicho string. Consideraremos que son letras todos los caracteres que no esten en los dos strings `string.whitespace` y `string.punctuation` del módulo string.

Referencias

[Ben84] John Bentley. Programming Pearls. *Communications of the ACM*, January 1984.

[Dew85] A. K. Dewdney. Cinco piezas sencillas para bucle y generador de números aleatorios. *Investigación y Ciencia*, 105:94–99, June 1985.

[Dox00] Apostolos Doxiadis. *El tío Petros y la conjetura de Goldbach.* Ediciones B, 2000.

[Enz01] Hans Magnus Enzensbergen. *El diablo de los números: un libro para todos aquéllos que temen a las matemáticas.* Siruela, 2001.

[Gar87] Martin Gardner. *Carnaval matemático.* Alianza Editorial, S. A, 1987.

[Gar95] Martin Gardner. *¡Ajá! Inspiración ¡ajá!.* Labor, 1995.

[Mar00] Luisa Marqués, editor. *Fotografiando las matemáticas.* Carroggio, 2000.

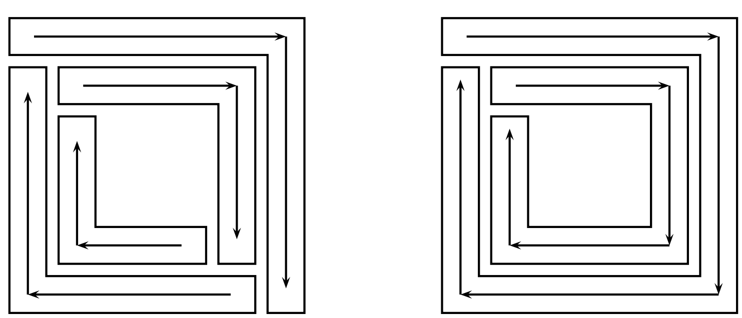
[Par01] Juan Parrondo. Perder + perder = ganar. Juegos de azar paradójicos. *Investigación y Ciencia*, 298, July 2001.

[Pom83] Carl Pomerance. A la búsqueda de los números primos. *Investigación y Ciencia*, 77:80–99, February 1983.

Pistas

1. Para desarrollar una solución adecuada del programa te será útil escribir una función que devuelva un número aleatorio uniformemente distribuido en un intervalo arbitrario (a, b).

2. Una buena forma de resolver el problema es pensar en descomposiciones de recorridos que, iterados, permitan pasar por los datos de la matriz en espiral. Observa, por ejemplo, los siguientes dibujos:



En la figura de la izquierda, hay dos recorridos. Uno de los recorridos es ir hacia la derecha y luego bajar y el otro recorrido consiste en ir hacia la izquierda y luego subir. Estos recorridos tienen que estar parametrizados por la longitud del lado que se quiere dibujar.

En la figura de la derecha, hay un recorrido pora el anillo más exterior, otro para el siguiente y así sucesivamente.

3. Haber trabajado con el ejercicio 2 puede ser de gran ayuda para saber cómo *obtener* los dígitos que componen un número.

4. En el ejercicio 10 tienes funciones que te serán útiles.

5. Conviene definir primero una función que calcule la entropía de una posición.

6. Una parte muy importante de la solución al problema consiste en encontrar la definición de las funciones que simulen los procesos aleatorios que intervienen en los diversos juegos.

7. La utilización del alfabeto de cifrado es alta: necesitamos consultarlo una vez para cada letra del mensaje original; por tanto, conviene tener dicho alfabeto en un soporte que permita un rápido acceso.