

▷ 1. **Procedimientos y funciones con listas**

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

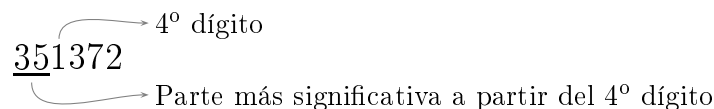
- Borrar el elemento de la posición i de una lista.
- Saturar una lista a uno elemento dado x : si tenemos la lista l poner en cada posición i el máximo entre x y $l[i]$.
- Borrar todas las apariciones del elemento x de una lista.
- Filtar los elementos primos de la lista.
- Insertar un elemento en la 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.

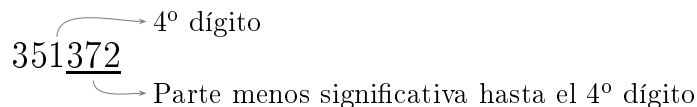
Parte más significativa Escribe una función que devuelva la parte más significativa desde el n -ésimo dígito de un número.

Ejemplo



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

Ejemplo



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

Solución

Puesto que el programa funciona igual para base 10 que para cualquier otra base. Haremos el programa que valga para cualquier base.

Una primera solución usándo la el operador potencia, predeterminado en Python.

```
def most_significant(number, position, base):  
    """  
    This function returns the most significant part  
    of a number in the given base from position  
    Example  
    number = 351372  
    pos = 4  
    base = 10  
    retruns 35  
  
    @type number: int  
    @type position: int  
    @param position: position>0  
    @type base: int  
    @rtype: int  
    """  
  
    return number / (base**position)  
  
def least_significant(number, position, base):  
    """  
    This function returns the least significant part  
    of a number in the given base from position.  
  
    Example  
    number = 351372  
    pos = 4  
    base = 10  
    retruns 372  
  
    @type number: int  
    @type position: int  
    @param position: position>0  
    @type base: int  
    @rtype: int  
    """  
  
    return number%(base**(position-1))  
  
def digit_in_pos(number, position, base):  
    """  
    This function returns the digit  
    of a number in the given base in the given position.  
  
    Example  
    number = 351372  
    pos = 4  
    base = 10  
    retruns 1  
  
    @type number: int  
    @type position: int  
    @param position: position>0  
    @type base: int  
    @rtype: int  
    """  
  
    return most_significant(number, position-1, base)%base  
  
def main():  
    print most_significant(351372, 4, 10)  
    print digit_in_pos(351372, 4, 10)  
    print least_significant(351372, 4, 10)
```

```
main()
```

Si el queremos practicar bucles podemos sustituir el programa anterior con una llamada a calcular la potencia de un número

```
def power(number, exp):
    res = 1
    i = 0
    while i < exp:
        res *= number
        exp += 1
    return res

def most_significant(number, position, base):
    """
    This function returns the most significant part
    of a number in the given base from position
    Example
    number = 351372
    pos = 4
    base = 10
    retruns 35

    @type number: int
    @type position: int
    @param position: position > 0
    @type base: int
    @rtype: int
    """

    return number / power(base, position)

def least_significant(number, position, base):
    """
    This function returns the least significant part
    of a number in the given base from position.

    Example
    number = 351372
    pos = 4
    base = 10
    retruns 372

    @type number: int
    @type position: int
    @param position: position > 0
    @type base: int
    @rtype: int
    """

    return number % power(base, position - 1)

def digit_in_pos(number, position, base):
    """
    This function returns the digit
    of a number in the given base in the given position.

    Example
    number = 351372
    pos = 4
    base = 10
    retruns 1

    @type number: int
    @type position: int
    @param position: position > 0
    @type base: int
    @rtype: int
    """

    return most_significant(number, position - 1, base) % base
```

```
def main():
    print most_significant(351372, 4, 10)
    print digit_in_pos(351372, 4, 10)
    print least_significant(351372, 4, 10)
```

```
main()
```

Pero en lugar de calcular la potencia podemos hacer una versión que haga los cálculos directamente

```
def most_significant(number, position, base):
    """
    This function returns the most significant part
    of a number in the given base from position
    Example
    number = 351372
    pos = 4
    base = 10
    retruns 35

    @type number: int
    @type position: int
    @param position: position>0
    @type base: int
    @rtype: int
    """
    i = 0
    while i<position:
        number = number/base
        i += 1
    return number

def least_significant(number, position, base):
    """
    This function returns the least significant part
    of a number in the given base from position.

    Example
    number = 351372
    pos = 4
    base = 10
    retruns 372

    @type number: int
    @type position: int
    @param position: position>0
    @type base: int
    @rtype: int
    """
    i = 0
    part = 0
    pot = 1
    while i<position-1:
        digit = number%base
        part = digit*pot + part
        number = number/base
        pot *= base
        i += 1
    return part

def digit_in_pos(number, position, base):
    """
    This function returns the digit
    of a number in the given base in the given position.

    Example
    number = 351372
    pos = 4
    base = 10
    retruns 1
```

```

@ttype number: int
@ttype position: int
@param position: position>0
@ttype base: int
@rttype: int
"""
i = 0
while i<position:
    digit = number%base
    number = number/base
    i += 1
return digit

def main():
    print most_significant(351372, 4, 10)
    print digit_in_pos(351372, 4, 10)
    print least_significant(351372, 4, 10)

main()

```

▷ 3. Generación de cadenas

En cada línea del fichero estándar de entrada (input) se leen cuatro caracteres: un signo (que puede ser +, -), un espacio en blanco, una letra L1, una letra L2; colocadas en el orden descrito. Se supone que las letras son mayúsculas. Codifíquese un programa para grabar, por cada línea leída, una línea con el siguiente contenido:

- Si el signo es “+”, una lista en orden alfabético ascendente de las letras que hay entre L1 y L2 (ambas inclusive), separadas entre sí por una coma, y finalizada la lista con un punto.

Puede ocurrir que la lista resulte vacía (cuando la letra L1 es posterior a la letra L2); por ejemplo, esta es una posible ejecución:

```

+ EH
E,F,G,H.
+ HE
.

```

- Si el signo es “-”, lo mismo, pero en orden descendente: por ejemplo:

```

- HE
H,G,F,E.
- EH
.

```

Solución

```

def ascend(s):
    res=""
    i=ord(s[0])
    end=ord(s[1])
    while i<=end:
        res+=chr(i)
        i+=1
    return res
def descend(s):

```

```

    res=""
    i=ord(s[0])
    end=ord(s[1])
    while i>=end:
        res+=chr(i)
        i-=1
    return res

def analyze(s):
    if s[0]=="+":
        return ascend(s[2:4])
    else:
        return descend(s[2:4])

```

▷ 4. Conversión de caracteres a números

Seguramente, en muchos de los programas que ya has realizado, te ha ocurrido que cuando el programa tiene que leer una variable de tipo numérico e introduces un carácter que no sea un dígito, por ejemplo un letra *a*, el programa da un error de ejecución y no continúa. Nos proponemos hacer unos ejercicios para evitar este problema.

Conversión de caracteres a números en base 10 Escribe un programa que lea una secuencia de dígitos y nos indique si dicha secuencia puede o no ser un número, en caso de que sí pueda ser un número, entonces tenemos que *tener* realmente el valor de dicho número.

Conversión de caracteres a números en una base arbitraria Escribe un programa para trabajar con números en base arbitraria. El programa tiene que leer una secuencia e indicar si dicha secuencia puede o no ser un número en la base que estamos considerando. Si efectivamente la secuencia puede ser un número en dicha base, entonces tenemos que *tener* realmente el valor de dicho número.

(Para más información consúltase la pista 1.)

Solución

El problema de transformar una cadena de caracteres en un entero es independiente de la base, por tanto abordaremos directamente este problema.

Para bases $2 \leq b \leq 10$ no hay demasiado problema, puesto que cada dígito tiene su valor. Para bases $b > 10$ hay que escoger caracteres para dígitos cuyo valor sean mayor o igual que 10. Para ello emplearemos letras mayúsculas

A	↪	10
B	↪	11
C	↪	12
D	↪	13
E	↪	14
F	↪	15
...		

```

"""
    Ascii representation of numbers
"""

def value(char):

```

```

"""
This function returns the numerical value of
a single char:
"0" -> 0
"1" -> 1
....
"9" -> 9
"a" -> 10
"b" -> 11
@type char: string
@param char: len(char)=1 it must be a numeric digit or a
lowercase ascii standard letter
@rtype: int
"""

if "0" <= char <= "9":
    return ord(char)-ord("0")
elif "a" <= char <= "z":
    return 10 + ord(char) - ord("a")
else:
    return -1

def digit2ascii(digit):
    """
    This function returns the ascii letter corresponding to a digit
    @type digit: int
    @rtype: string
    """
    if 0 <= digit <= 9:
        return chr( digit + ord("0") )
    else:
        return chr( digit - 10 + ord("a") )

def str2num(s_num, base):
    """
    This function converts the string s_num to an integer. s_num is
    a natural number represented in base.
    @type s_num: string
    @type base: int
    @param base: base>=2
    @param s_num: the digits of s_num must be lower than base.
    For it we consider the funtion value: value(s_num[i])<base
    @rtype: int
    """
    num = 0
    i = 0
    while i < len(s_num):
        num = num * base + value(s_num[i])
        i += 1
    return num

def num2str(num, base):
    """
    This function returns the representation of num in base. The digits
    above 10 are represented with letters a, b, c,
    @type num: int
    @param num: num>0
    @type base: int
    @param base: base>=2
    @rtype: string
    """
    s_num = ""
    while num > 0:
        s_num = digit2ascii(num % base) + s_num
        num = num / base
    return s_num

def main():
    """
    The main function

```

```

"""
num_ori = 160
s_num = num2str(num_ori, 16)
print s_num
num = str2num(s_num, 16)
print num, num == num_ori

s_num = num2str(num_ori, 2)
print s_num
num = str2num(s_num, 2)
print num, num_ori == num

main()

```

▷ 5. Traza de algoritmos de ordenación

Se desea observar cómo funcionan diversos métodos de ordenación, al trabajar con un *array*. Por ello, se propone incorporar en cada uno de los métodos que consideremos una traza adecuada que muestre con claridad el *modus operandi* de los diferentes algoritmos.

Ordenación por selección El algoritmo de selección consiste en seleccionar en cada etapa la componente mínima (o máxima) e intercambiarla con la que está en la posición que debería ocupar dicha componente.

Una manera de seguir el algoritmo de selección, consiste en ver en cada etapa la componente seleccionada y la componente con la que va a intercambiar posiciones. (Para más información consúltase la pista 2.)

Ejemplo

Vector inicial:	M	U	R	C	I	E	L	A	G	O
Etapla 1:	A							M		
Etapla 2:		C		U						
Etapla 3:			E			R				
Etapla 4:				G					U	
Etapla 5:					I					
Etapla 6:						L	R			
Etapla 7:							M	R		
Etapla 8:								O	R	
Etapla 9:									R	U
Vector final:	A	C	E	G	I	L	M	O	R	U

Ordenación por inserción El algoritmo de ordenación por inserción consiste en ordenar en la etapa n las n -primeras componentes del array hasta que tenemos todas las componentes ordenadas. Para ello, en la etapa n podemos suponer que tenemos las $n - 1$ primeras componentes ordenadas y, por comparaciones sucesivas, colocamos la componente de posición n en el lugar adecuado con respecto a las anteriores.

Para seguir el método de inserción, en la etapa n podemos ver como las primeras n componentes están ordenadas. (Para más información consúltase la pista 3.)

Ejemplo

Vector inicial: M U R C I E L A G O

Etapa 1:	M
Etapa 2:	M U
Etapa 3:	M R U
Etapa 4:	C M R U
Etapa 5:	C I M R U
Etapa 6:	C E I M R U
Etapa 7:	C E I L M R U
Etapa 8:	A C E I L M R U
Etapa 9:	A C E G I L M R U
Vector final:	A C E G I L M O R U

Solución

Ordenación por selección La ordenación por selección es muy sencilla: se selecciona el mínimo elemento del vector, se coloca en la primera posición y se repite el proceso para el resto del vector.

La implementación de esta idea consiste en un doble bucle, el índice *i* que recorre el bucle externo indica la posición a partir de la cual buscaremos el mínimo elemento del vector. El bucle interno, recorre el vector desde la posición *i*-ésima hasta el final buscando el menor de los elementos, el índice de dicho elemento se almacena en la variable *min*.

Una vez que tenemos seleccionado el mínimo elemento, *pmin*, a partir de la posición *i*-ésima, basta con intercambiar el contenido de dichas posiciones.

```
def seleccion(lst):
    """
    Ordena la lista lst de menor a mayor. Para ello en cada vuelta óselecciona
    el menor elemento y lo coloca en la óposicin adecuada.
    @param lst: lista con los valores a ordenar
    @type lst: list

    """
    for i in xrange(len(lst)):
        pmin = i
        for j in xrange(i, len(lst)):
            if lst[j] < lst[pmin]:
                pmin = j
        lst[i], lst[pmin] = lst[pmin], lst[i]
```

Para trazar el algoritmo insertamos, creamos una lista vacía salvo por las posisicones que se intercambias e imprimimos dicha lista:

```
éóé
print " : {0}".format(" ".join(map(str, lst)))

def seleccion(lst):
    """
    Ordena la lista lst de menor a mayor. Para ello en cada vuelta óselecciona
    el menor elemento y lo coloca en la óposicin adecuada.
    @param lst: lista con los valores a ordenar
    @type lst: list

    """
    print " : {0}".format(" ".join(map(str, range(len(lst)))))
    print " : {0}".format(" ".join(map(str, lst)))
    for i in xrange(len(lst)):
```

```

pmin = i
for j in xrange(i, len(lst)):
    if lst[j] < lst[pmin]:
        pmin = j
lst[i], lst[pmin] = lst[pmin], lst[i]
trz = [" "] * len(lst)
trz[i] = lst[i]
trz[pmin] = lst[pmin]

```

Ordenación por inserción Si suponemos que las $i - 1$ primeras coordenadas de un vector están ordenadas, ordenar las i primeras es una tarea sencilla: basta con recolocar el elemento de la posición i en su lugar adecuado. Si suponemos que el vector está ordenado de forma creciente, debemos buscar el primer elemento que sea menor o igual que el queremos recolocar y desplazar las casillas una posición. Para ordenar completamente un vector utilizamos un doble bucle.

```

def insercion(l):
    """
    Ordena la lista lst de menor a mayor. Utiliza el método de la óinserccin
    ordenada, en cada vuelta se toma el elemento iê-simo y se inserta de forma
    ordenada en los primeros elementos ya ordenados.
    @param l: lista con los valores a ordenar
    @type l: list
    """
    for i in xrange(1, len(lst)):
        j = i - 1
        while (j >= 0) and (lst[j+1] < lst[j]):
            lst[j], l[stj+1] = l[j+1], lst[j]
            j = j - 1

```

Para trazar el algoritmo insertamos, imprimimos la parte inicial de la lista

```

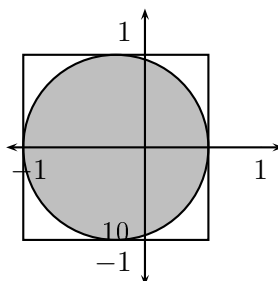
def insercion(lst):
    """
    Ordena la lista l de menor a mayor. Utiliza el método de la óinserccin
    ordenada, en cada vuelta se toma el elemento iê-simo y se inserta de forma
    ordenada en los primeros elementos ya ordenados.
    @param l: lista con los valores a ordenar
    @type l: list
    """

    print " : {0}".format(" ".join(map(str, range(len(lst)))))
    print " : {0}".format(" ".join(map(str, lst)))
    for i in xrange(1, len(lst)):
        j = i - 1
        while (j >= 0) and (lst[j+1] < lst[j]):
            lst[j], lst[j+1] = lst[j+1], lst[j]
            j = j - 1
        print "{0} : {1}".format(i, " ".join(map(str, lst[: (i+1)])))

```

▷ 6. 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 tante los aciertos y deduzca de ahí una aproximación de π . (Para más información consúltese la pista 4.)

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

Solución

Para simular el lanzamiento de dardos necesitamos una función aleatoria uniformemente distribuida que nos devuelva números en el intervalo real $(-1, 1)$. Para ello usaremos la función `uniform` del módulo `random` (<http://docs.python.org/library/random.html#random.uniform>) que devuelve un número aleatorio uniformemente distribuido en un intervalo arbitrario (a, b) .

Una vez que se cuenta con esta función el programa es bastante sencillo, basta simular una determinada cantidad de veces el experimento de lanzar un dardo. Es decir, calcular de forma aleatoria un valor para las coordenadas (x, y) que indican la posición en la que ha caído el dardo, y comprobar si este dardo ha caído dentro o fuera del círculo. Tras la realización de estos experimentos tenemos dos datos, el número de dardos lanzados y el número de dardos que han caído dentro del círculo. A partir de estos datos calculamos la aproximación a π como indica el enunciado.

```
import random

def esta_dentro(x,y):
    return (x*x + y*y) <= 1
def aproximarPI(dardos):
    dentro=0
    i=0
    while i<dardos:
        x=random.uniform(-1,1)
        y=random.uniform(-1,1)
        if esta_dentro(x,y):
            dentro+=1
        i+=1
    return 4*(float(dentro)/dardos)

def main():
    random.seed()
```

```

    pi=aproximarPI(100000)
    print pi

main()

```

Este método de aproximación, aunque divertido, es muy ineficiente, tendrás que pedir a tu programa que lance un gran número de dardos para obtener un valor aproximado sólo en unas pocas cifras decimales.

▷ 7. 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,

$$(1, \text{numPar} - 1), (2, \text{numPar} - 2), \dots$$

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

$$\dots, (\text{numPar} - 2, 2), (\text{numPar} - 1, 1)$$

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 16.)

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.

Solución

```
'''
Conjetura de Goldbach
Todo par es suma de dos primos
'''

def esprimo(n):
    i = 2;
    primo = True;
    while (i<n) and primo:
        primo = (n%i != 0)
        i+=1
    return primo

def descomponer(numPar):
    conseguido = False
    sumando1 = 1
    while not conseguido:
        sumando1 = sumando1 + 1
        sumando2 = numPar - sumando1
        conseguido = esprimo(sumando1) and esprimo(sumando2)
    return conseguido, sumando1, sumando2

def main(tope):
    k = 2
    seCumpleHastaK = True
    while seCumpleHastaK and k<tope:
        k = k + 2
        posible, sum1, sum2 = descomponer(k)
        if posible:
            print "{0} = {1} + {2}".format(k,sum1,sum2)
        else:
            seCumpleHastaK = False
    if not seCumpleHastaK:
        print 'La conjetura es falsa'

main(500)
```

▷ 8. 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.

Solución

```
#!/usr/bin/python

"""
Program that stores primes in a file
"""

def is_prime(prime_list, num):
    """
    This function check if num is prime.
    This functions uses the list of primes less than num.
    @type num: int
    @param num: num>2
    @type prime_list: list of int
    @param prime_list: contains the primes less than num
    """
    i = 0
    while prime_list[i]**2 < num and num%prime_list[i] != 0:
        i+=1
    return prime_list[i]**2 > num

def next_prime(prime_list, num):
    """
    This function computes the next prime number after num.
    This functions uses the list of primes less than num.
    Whenever a prime is found the prime_list is updated
    @type num: int
    @param num: num>2
    @type prime_list: list of int
    @param prime_list: contains the primes less than num
    """
    num = num+1
    while not is_prime(prime_list, num):
        num += 1
    return num

def write_primes(filename, prime_list, n_primes):
    """
    This function appends the following from prime n_primes in filename
    @type filename: string
    @type prime: int
    @param prime: prime is prime number
    @type n_primes: int
    @param n_primes: n_primes>=0
    @type prime_list: list of int
    @param prime_list: list of primes less than prime
    """
    flp = open(filename,"a")
    i = 0
    while i<n_primes:
        prime = next_prime(prime_list, prime_list[-1])
        prime_list.append(prime)
        flp.write("{0}\n".format(prime))
        i += 1
    flp.close()

def read_primes(filename):
    """
    This function reads the first primes from a file.
    Each line contains a prime number
    @type filename: string
    @rtype: list of int
    @returns: the list of first primes
    """
```

```

"""
flp = open(filename,"r")
line = flp.readline()
prime_list = []
while line != "":
    prime_list.append(int(line))
    line = flp.readline()
flp.close()
return prime_list

def main(n_primes):
    """
    Main program
    @type n_primes: int
    @param n_primes: n_primes >= 0
    """
    filename = "primes.txt"
    prime_list = read_primes(filename)
    write_primes(filename, prime_list, n_primes)

main(10)

```

▷ 9. Triángulo de Pascal

Consideremos el triángulo de Pascal,

$$\begin{array}{ccccccc}
 & & & & 1 & & \\
 & & & 1 & & 1 & \\
 & & 1 & & 2 & & 1 \\
 & 1 & & 3 & & 3 & & 1 \\
 1 & & 4 & & 6 & & 4 & & 1 \\
 & & & \dots & & & & &
 \end{array}$$

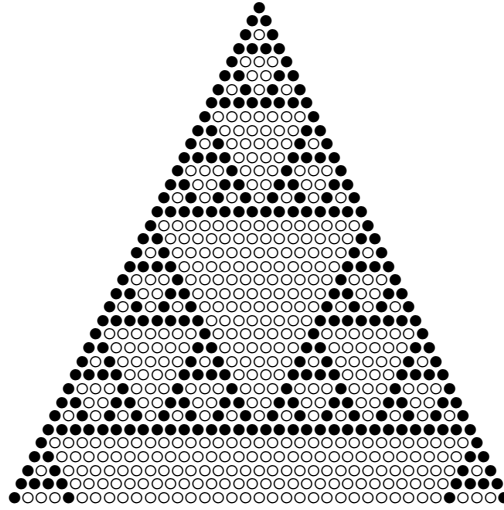
cuyos elementos pueden generarse bien directamente mediante números combinatorios, siendo $\binom{i}{j}$ el elemento j -ésimo de la fila i -ésima (para $0 \leq j \leq i$),

$$\begin{array}{ccccccccccc}
 & & & & \binom{0}{0} & & & & \\
 & & & \binom{1}{0} & & \binom{1}{1} & & & \\
 & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & \\
 & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & \\
 \binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4} \\
 & & & \dots & & & & &
 \end{array}$$

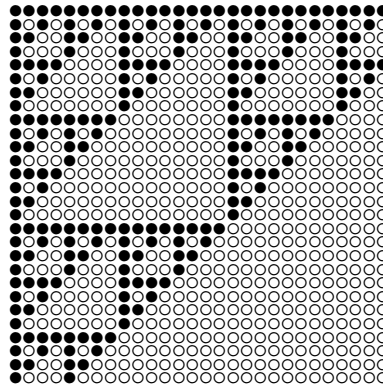
o bien sumando los dos situados sobre él:

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

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,



y sustituyendo cada “●” por un asterisco (‘*’), y cada “○” por un blanco (‘ ’) para usar los caracteres disponibles.

Solución

Vamos a resolver este ejercicio de distintas maneras, empezando con una muy sencilla, hasta llegar a un programa eficiente.

S.9.1 Solución directa

Al rotar el triángulo de Pascal como pide el enunciado,

$$\begin{array}{ccccccc}
 \binom{0}{0} & \binom{1}{1} & \binom{2}{2} & \binom{3}{3} & \cdots & \binom{j}{j} & \cdots \\
 \binom{1}{0} & \binom{2}{1} & \binom{3}{2} & \binom{4}{3} & \cdots & \binom{1+j}{j} & \cdots \\
 \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
 \binom{i}{0} & \binom{i+1}{1} & \binom{i+2}{2} & \binom{i+3}{3} & \cdots & \binom{i+j}{j} & \cdots \\
 \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots
 \end{array}$$

se observa que el elemento situado en la fila i -ésima y en la columna j -ésima es $\binom{i+j}{j}$, para $i, j \geq 0$. Como consecuencia, se tiene un primer algoritmo que consiste en escribir los elementos de la matriz correspondiente, de `rows` filas y `cols` columnas, que reflejan el tamaño de la pantalla,


```
def pascal(rows, cols):
    for f in range(rows):
        print_row(f, cols)
```

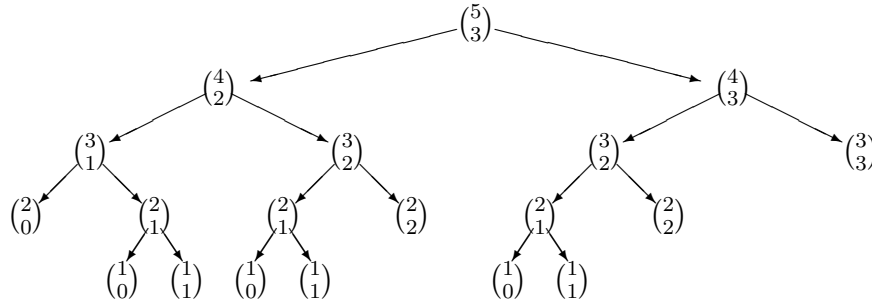
donde `print_row` consiste en lo siguiente:

```
def print_row(f, cols):
    for c in range(cols):
        if comb(f+c, c) % 2 == 1:
            print '*',
        else:
            print '.',
    print
```

donde $\text{comb}(m, n) = \binom{m}{n} = \frac{m!}{n!(m-n)!}$ para $0 \leq n \leq m$, y puede hallarse iterativa o recursivamente.

S.9.2 Solución tabulando los números combinatorios

Puede objetarse un inconveniente al algoritmo propuesto: el cálculo independiente de cada elemento $\binom{m}{n}$ presenta redundancias, tanto si se calcula recursivamente,



como en su versión iterativa:

$$\dots, \frac{(m-2) \dots (m-n)}{(n-1)!}, \frac{(m-1) \dots (m-n)}{n!}, \dots$$

$$\dots, \frac{(m-1) \dots (m-n+1)}{(n-1)!}, \frac{m \dots (m-n+1)}{n!}, \dots$$

Una alternativa es tabular los números combinatorios $\binom{i+j}{j}$, para $0 \leq i \leq \text{rows}, 0 \leq j \leq \text{cols}$. Llamando *Comb* a la tabla mencionada, tenemos lo siguiente,

$$\text{Comb}[i+j, j] \in \mathbb{N}, \text{ para } (i, j) \in \{0, \dots, \text{rows}\} \times \{0, \dots, \text{cols}\}$$

Ahora esta tabla se rellena fácilmente así:

```
def fill_row(t, row):
    j=1
    n=len(t[row])
    while j<n:
        t[row][j]=t[row-1][j]+t[row][j-1]
        j=j+1

def build_table(rows, cols):
    t = [None]*rows
    t[0]=[1]*cols
    i=1
    while i<rows:
        t[i]=[1]*cols
        fill_row(t, i)
        i=i+1

    return t
```

Una vez construida la tabla, reproducir la figura resulta casi igual que en la versión anterior: basta con cambiar en ella cada llamada `comb(m, n)`, que calcula un número combinatorio, por la consulta de la tabla para encontrar su valor: `t[m][n]`.

S.9.3 Solución tabulada y usando la paridad de los números combinatorios

Un inconveniente que tiene ahora la solución anterior es el del desbordamiento de los números que surgen en las expresiones, debido al rápido crecimiento de los números combinatorios. Sin embargo, para resolver este problema no es necesario recurrir a dichos números: puesto que finalmente sólo estamos interesados en la paridad de los números $\binom{m}{n}$, basta con calcular sólo este dato y registrarlo en una tabla de paridades,

$$Paridad[i + j, j] \in \{\text{True}, \text{False}\} \text{ para } (i, j) \in \{0, \dots, \text{rows}\} \times \{0, \dots, \text{cols}\}$$

donde `True` indica que número es par mientras que `False` indica que es impar.

```
def fill_row(t, row):
    j=1
    n=len(t[row])
    while j<n:
        t[row][j]= not (t[row-1][j]==t[row][j-1])
        j=j+1

def build_table(n,m):
    t = [None]*n
    t[0]=[True]*m
    i=1
    while i<n:
        t[i]=[True]*m
        fill_row(t,i)
        i=i+1

    return t
```

Una vez rellena la tabla con ceros y unos, reproducir la figura es aún más sencillo que antes, ya que los elementos son ahora sólo booleanos:

```
def print_table(t):
    for l in t:
        for b in l:
            if b:
                print ".",
            else:
                print "*",
        print
```

S.9.4 Solución avanzando por filas

Finalmente, observamos que, para construir cada fila de la tabla, basta con mantener la fila anterior, pudiendo evitarse así el gasto de memoria de la tabla completa, que se reduce al de una única fila de paridades:

$$Paridad[j] \in \{\text{True}, \text{False}\} \text{ para } j \in \{0, \dots, \text{rows}\}$$

La primera fila se rellena con una hilera inicial de unos, y se muestra en la pantalla al mismo tiempo la línea de asteriscos correspondiente.

```
def print_row(t):
    for a in t:
        if a:
```

```

        print "*",
    else:
        print ".",
    print

def new_row(t):
    i=1
    n=len(t)
    while i<n:
        t[i]= not (t[i]==t[i-1])
        i=i+1

def pascal(rows,cols):
    t = [True]*cols
    i=0
    while i<rows:
        print_row(t)
        new_row(t)
        i=i+1

```

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 importante 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.

▷ 10. *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 $[5, 5, 2, 2, 1, 4, 1, 1, 5, 6]$.

Solución

En esta solución vamos a considerar un array de tamaño variable

```
int const N=20;
```

```
typedef struct Vector {
    int tam;
    int datos[N];
} Vector;
```

Este es uno de los programas clásicos de búsqueda, buscamos el primer elemento que *se repita exactamente k veces*. Por tanto procedimiento que busca tal elemento se corresponde a tal esquema:

```
void kRep(Vector const& v, int const k,
          bool & existe, int & valor) {
    /* Entrada: vector v y entero k
       Salida: boolean existe y entero valor.
       Busca en el array v un elemento que
       se repita exactamente k veces.
       Si existe tal elemento existe es true
       y se devuelve en el parámetro de salida valor.
       Si no existe el parámetro existe es false */
    int i = 0;
    while (i < v.tam && rep(v,i)!=k) {
        i++;
    }
    existe = i<v.tam;
    if (existe)
        valor = v.datos[i];
}
```

Este procedimiento usa la función `rep` que calcula el número de repeticiones de un elemento dentro del vector

```
int rep(Vector const& v, int const pos) {
    /* Función que devuelve el número de repeticiones
       del valor en la posición pos. 0<=pos<v.tam */
    int cont=0;
    for (int i=0; i<v.tam; i++)
        if (v.datos[i]==v.datos[pos])
            cont++;
    return cont;
}
```

Por último usamos un pequeño programa de prueba

```
void genera (Vector & v) {
    v.tam = rand()%N;
    for (int i=0; i<v.tam; i++) {
        v.datos[i]=rand()%(v.tam/2);
    }
}

ostream& operator << (ostream& out, Vector const& v) {
    for (int i=0; i<v.tam; i++) {
```

```

        if (i>0) out << ", ";
        out << v.datos[i];
    }
    return out;
}
main() {
    srand(time(NULL));
    Vector v;
    genera(v);
    cout << v << endl;
    cout << "Número de repeticiones: " << endl;
    int k;
    cin >> k;
    bool existe;
    int valor;
    kRep(v,k,existe,valor);
    if (existe)
        cout << "El valor " << valor
            << " se repite " << k << " veces"
            << endl;
    else
        cout << "Ningún elemento se repite exactamente "
            << k << " veces"
            << endl;
}

```

▷ 11. 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 consúltase la pista 5.)

Solución

```
"""
spiral run of a matrix
"""

import sys

def build(size):
    """
    Function that build a n*n matrix
    @type n: integer
    """
    mat = []
    elem = 0
    i = 0
    while i < size:
        row = []
        j = 0
        while j < size:
            row.append(elem)
            j += 1
            elem += 1
        mat.append(row)
        i += 1
    return mat

def print_mat(mat):
    """
    Function that print a matrix
    @type mat: matrix
    """
    for row in mat:
        for cell in row:
            print "{0:2}".format(cell),
        print

def print_lst(lst):
    """
    Function that print a matrix
    @type mat: list
    """
    print "list with {0} elements: ".format(len(lst))
    for cell in lst:
        print "{0}".format(cell),
    print

def run_corner(mat, row, col, dim, direction):
    """
    This function add the elements of a "corner" of mat.
    The origin is the position (row, col). The dimension of
    the corner is dim.
    direction indicates the order which the elements are run:
    1 indicates left-down
    -1 indicates up-right.

    The function returns the position of the next element to be run.

    @type mat: matrix
    @type col: int
    @type row: int
    @type dim: int
    @type direction: int
    @param col: 0<=col<len(matrix), 0<=col+direction*(dim-1)<len(matrix)
    @param row: 0<=row<len(matrix), 0<=row+direction*(dim-1)<len(matrix)
    @param dim: dim<=len(matrix)
    @param direction: direction == 1 or direction == -1
    @rtype: list

    Examples:
```

```

mat = [[ 0, 1, 2, 3, 4],
        [ 5, 6, 7, 8, 9],
        [10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24]]

row = 1
col = 1
dim = 3
direction = 1
returns [6, 7, 8, 13, 18]

row = 3
col = 2
dim = 2
direction = -1
returns [17, 18, 23]

"""
i = 0
elements = []
while i < dim:
    elements.append(mat[row][col])
    col += direction
    i += 1
col -= direction # the previous loop finished when i==dim,
# col is one position ahead.
row += direction
i = 0
while i < dim-1:
    elements.append(mat[row][col])
    row += direction
    i += 1
row -= direction # the previous loop finished when i==dim,
# col is one position ahead.
col -= direction
return elements

def run_left_down(mat, row, col, dim):
    """
    Function that computes the left-down corner of the matrix
    of dimension dim starting at (row, col).
    @type mat: matrix
    @type col: int
    @type row: int
    @type dim: int
    @param col: 0<=col<len(matrix), 0<=col+(dim-1)<len(matrix)
    @param row: 0<=row<len(matrix), 0<=row+(dim-1)<len(matrix)
    @param dim: dim<=len(matrix)
    @rtype: a tuple of 3 elements, 2 integers and a list

    Examples:
    mat = [[ 0, 1, 2, 3, 4],
            [ 5, 6, 7, 8, 9],
            [10, 11, 12, 13, 14],
            [15, 16, 17, 18, 19],
            [20, 21, 22, 23, 24]]
    row = 1
    col = 1
    dim = 3
    returns (3, 2, [6, 7, 8, 13, 18])
    """
    return run_corner(mat, row, col, dim, 1)
def run_right_up(mat, row, col, dim):
    """
    Function that computes the right-up corner of the matrix
    of dimension dim starting at (row, col).
    @type mat: matrix
    @type col: int
    @type row: int
    @type dim: int

```

```

@param col: 0<=col<len(matrix), 0<=col+(dim-1)<len(matrix)
@param row: 0<=row<len(matrix), 0<=row+(dim-1)<len(matrix)
@param dim: dim<=len(matrix)
@rtype: a tuple of 3 elements, 2 integers and a list

Examples:
mat = [[ 0, 1, 2, 3, 4],
        [ 5, 6, 7, 8, 9],
        [10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24]]

row = 3
col = 2
dim = 2
returns [17, 16, 11]

"""
return run_corner(mat, row, col, dim, -1)

def test_run_right_up():
    mat = [[ 0, 1, 2, 3, 4],
            [ 5, 6, 7, 8, 9],
            [10, 11, 12, 13, 14],
            [15, 16, 17, 18, 19],
            [20, 21, 22, 23, 24]]
    print run_right_up(mat, 3, 2, 2)

    mat = build(6)
    print run_right_up(mat, 5, 4, 5)

def spiral(mat):
    """
    Function that print the elements of the matrix
    in "spiral".
    @type mat: matrix

    Examples:
    mat = [[0, 1, 2, 3, 4],
            [5, 6, 7, 8, 9],
            [10, 11, 12, 13, 14],
            [15, 16, 17, 18, 19],
            [20, 21, 22, 23, 24]]
    returns [0, 1, 2, 3, 4, 9, 14, 19, 24, 23, 22, 21, 20, 15, 10, 5, 6, 7, 8, 13, 18, 17, 16, 11, 12]

    mat = [[0, 1, 2, 3, 4, 5],
            [6, 7, 8, 9, 10, 11],
            [12, 13, 14, 15, 16, 17],
            [18, 19, 20, 21, 22, 23],
            [24, 25, 26, 27, 28, 29],
            [30, 31, 32, 33, 34, 35]]
    returns [0, 1, 2, 3, 4, 5, 11, 17, 23, 29, 35,
            34, 33, 32, 31, 30, 24, 18, 12,
            6, 7, 8, 9, 10, 16, 22, 28,
            27, 26, 25, 19, 13,
            14, 15, 21, 20]

    """
    i = 0
    j = 0
    dim = len(mat)
    elements = []
    while dim>0:
        lst = run_left_down(mat, i, i, dim)
        elements.extend(lst)
        lst = run_right_up(mat, len(mat)-i-1, len(mat)-i-2, dim-1)
        elements.extend(lst)
        dim -= 2
        i += 1
    return elements

```



```

def main(size):
    """
    Main Program
    """
    mat = build(size)
    elements = spiral(mat)
    print_mat(mat)
    print_lst(elements)

if __name__ == "__main__":
    if len(sys.argv) >= 2:
        size = int(sys.argv[1])
    else:
        size = 6
    main(size)

```

▷ 12. 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 consúltase la pista 6.)

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

Solución

Reverso de un número

```

int reverso(int const num) {
    /*
    Dev: el reverso del número n expresado en BASE.
    Se ignoran los posibles errores debidos a
    que los números se salgan del rango permitido
    al tipo int.
    Eje: el reverso de 1245 en base 10 es 5421
    */
    int m = 0;
    int n = num;
    while (n > 0) {
        int d = n % BASE;

```

```

        m = m * BASE + d;
        n = n / BASE;
    }
    return m;
}

```

Número palíndromo

```

bool esPalindromo (int const num) {
    return num == reverso(num);
}

```

▷ 13. 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 12).

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

$$\begin{array}{ccccccc}
 & & 59 & & 154 & & 605 \\
 & & + 95 & & + 451 & & + 506 \\
 59 & \rightarrow & \frac{}{154} & \rightarrow & \frac{}{605} & \rightarrow & \frac{}{1111} \rightarrow 1111
 \end{array}$$

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 consúltase la pista 7.)

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

7 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–).

Solución

```
def inverse(n,base):
    """
    Computes the inverse of "n" when represented in base "base"
    @type n: integer
    @param n: n>0
    @type base: integer
    @param base: base>1
    @rtype: integer
    """
    inv=0
    while n>0:
        d = n%base
        n = n/base
        inv = inv*base + d
    return inv

def conjecture(n):
    """
    This procedure prints all the numbers starting in "n" to test the palindrome conjecture.
    @type n: integer
    @param n: n>0
    """
    inv = inverse(n,10)
    while n!=inv:
        print "%d is not a palindrome" % n
        n=n+inv
        inv=inverse(n,10)
    print "found: %d" % n
```

▷ 14. Suma Acumulada

Supongamos que tenemos una lista v lleno de 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
---	---	----	----

Solución

```
#include <iostream>
using namespace std;
int const N = 5;
typedef int Vector[N];
ostream& operator << (ostream & out, Vector const& v) {
    out << '[';
    for (int i=0; i<N; i++) {
        if (i>0) out << ',';
        out << v[i];
    }
    out << ']';
    return out;
}
void suma(Vector & v, int const pos) {
    for (int i=N-1; i>=pos; i-) {
        v[i] += v[i-1];
    }
}
void suma(Vector & v) {
    for (int i=1; i<N; i++) {
        suma(v,i);
    }
}
void genera(Vector & v) {
    for (int i=0; i<N; i++)
        v[i] = rand()%(2*N);
}
main() {
    Vector v;
    genera(v);
```

```

    cout << "Original: " << v << endl;
    suma(v);
    cout << "Final: " << v << endl;
}

```

▷ 15. Entropía de un vector

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

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

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:

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

Escribe una función que calcule la entropía total de un vector. (Para más información consúltase la pista 8.)

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 entropía es 0;
- y para la última 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.

Solución

```

import random
def entropia_pos(v, pos):
    """This function computes the entropy of position pos in v
    @type v: list of numbers
    @type pos: integer
    @param pos: pos < len(v)
    @rtype: integer
    """
    en = 0
    j = pos + 1
    n = len(v)
    while j < n:
        if v[pos] > v[j]:
            en += 1
        j += 1

```

```

        j += 1
    return en

def entropy(v):
    """This function computes the entropy of vector v
    @type v: list of numbers
    @rtype: integer
    """
    en = 0
    i = 0
    n = len(v)
    while i < n:
        en += entropy_pos(v, i)
        i += 1
    return en

n = 10
v = [0] * n
i = 0
while i < len(v):
    v[i] = random.randint(0, n)
    i += 1
print v, entropy(v)
i = 0
while i < len(v):
    print entropy_pos(v, i)
    i += 1

```

▷ 16. 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.

Solución

Solución “normal”

```

def even_filter(l):
    out = []
    i = 0
    n = len(l)

```

```

    while i<n:
        if l[i]%2!=0:
            out.append(l[i])
            i+=1
    return out

def mod24_filter(l):
    out = []
    i=0
    n=len(l)
    while i<n:
        if l[i]==3 or (l[i]*l[i])%24==1:
            out.append(l[i])
            i+=1
    return out

def is_prime(n):
    i=2
    while i*i < n and n%i!=0:
        i+=1
    return i*i>n

def prime_filter(l):
    out = []
    i=0
    n=len(l)
    while i<n:
        if is_prime(l[i]):
            out.append(l[i])
            i+=1
    return out

print prime_filter(mod24_filter(even_filter(range(2,1000))))

```

Y esta solución usando orden superior (las funciones se pueden pasar como parámetros).

```

def filter(l,f):
    out = []
    i=0
    n=len(l)
    while i<n:
        if f(l[i]):
            out.append(l[i])
            i+=1
    return out

def is_odd(n):
    return n%2==1
def even_filter(l):
    return filter(l,is_odd)

def is_mod24(n):
    return n==3 or n*n%24==1
def mod24_filter(l):
    return filter(l,is_mod24)

def is_prime(n):
    i=2
    while i*i < n and n%i!=0:
        i+=1
    return i*i>n
def prime_filter(l):
    return filter(l,is_prime)

print prime_filter(mod24_filter(even_filter(range(2,1000))))

```

▷ 17. 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 como 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.

Solución

```
#include <iostream>
using namespace std;
int nivel(int const numero) {
    /* numero >= 0 */
    int factorial=1;//factorial de nivel
    int ultimoNumeroNivel=0;//Último número del nivel
    int nivel=0;
    /* el nivel 0 es el desconocido, hay están
       las personas con número 0 */
    while (numero>ultimoNumeroNivel) {
        nivel++;
        factorial=factorial*nivel;
        ultimoNumeroNivel=ultimoNumeroNivel+factorial;
    }
    return nivel;
}
int main(int argc, char** args) {
```



```

int n1,n2;
cout << "Dame el primer número: ";
cin >> n1;
cout << "Dame el segundo número: ";
cin >> n2;
cout << "El nivel de " << n1 << " es " << nivel(n1) << endl;
cout << "El nivel de " << n2 << " es " << nivel(n2) << endl;
}

```

▷ 18. 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 consúltase la pista 9.)

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.

Solución

Los juegos propuestos se basan en el uso de monedas que tienen diversas probabilidades de obtener cara y cruz.

Podemos definir una función que, dada una probabilidad de salir cara, simule el lanzamiento de una moneda y nos informe de si se ha obtenido cara o no. Una jugada a cualquiera de los juegos básicos descritos puede verse como un procedimiento que, dado un capital, lanza la moneda correspondiente y devuelve el capital incrementado o decrementado dependiendo del resultado obtenido al lanzar la moneda.

Casi iguales pero no tanto Una jugada puede simularse con el procedimiento:

Por tres es al revés Una jugada puede simularse con el procedimiento:

Mezclando juegos Para hacer un procedimiento que realice la jugada del juego resultante de la mezcla de los dos anteriores es necesario tener información acerca del número de partidas que llevamos para saber cuál de los juegos se ha de utilizar.

Por ello, se añade una variable que nos indica el número de jugadas que llevamos. Dicho contador se va incrementando a medida que jugamos. Para que el juego funcione correctamente y siga la secuencia CCTTCCTTCCTT..., descrita en el enunciado, se hace necesario que la primera llamada al procedimiento `mezcla` se realice pasando el valor 0 al parámetro `cont`.

Un programa completo para realizar la simulación podría ser el siguiente: Dónde sólo quedan por definir los procedimientos que se ocupan de la entrada de los datos y de mostrar los resultados:

▷ 19. 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 descifrar, es la permutación del alfabeto que hayamos elegido. (Para más información consúltase la pista 10.)

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 continúe 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?

Solución

El siguiente es un programa que cifra mensajes utilizando un código monoalfabético

```

import sys
from generate_cripto import CRIPTO_FILENAME, read_criptogram

def encode_line(s, criptogram):
    """
    Encode the line s according to the given criptogram
    @type s: string
    @type criptogram: list of one letter strings
    @param s: all letters in the string appear in the criptogram
    @rtype: string
    @param: returns the codification of string "s" according to "criptogram"
    """
    enc=""
    for a in s:
        if a<>" ":
            enc+=criptogram[ord(a)-ord("A")]
    return enc

def encode(file):
    """
    Prints in the standard output the decodification of the file
    @type file: file
    """
    criptogram=read_criptogram()
    for line in file:
        print encode_line(line[:-1], criptogram)

def main():
    file_name=sys.argv[1]
    if len(sys.argv)>1:
        file = open(sys.argv[1])
    else:
        file=sys.stdin
    encode(file)

if __name__ == "__main__":
    main()

```

El siguiente es uno que decodifica usando un código monoalfabético

```

import sys
from generate_cripto import CRIPTO_FILENAME, read_criptogram

def find_pos(criptogram, letter):
    """
    This function find the position of letter 'letter' in the criptogram
    @type criptogram: list of one letter strings
    @type letter: one letter string
    @param criptogram: the letter appears in the criptogram
    @rtype: integer
    @return: pos such that criptogram[pos]==letter
    """
    i=0
    n=len(criptogram)
    while i<n and criptogram[i]!=letter:
        i+=1
    return i

def decode_line(s, criptogram):
    """
    Decode the line s according to the given criptogram
    @type s: string
    @type criptogram: list of one letter strings
    @param s: all letters in the string appear in the criptogram
    @rtype: string
    @param: returns the decodification of string "s" according to "criptogram"
    """

```

```

    dec=""
    for a in s:
        pos=find_pos(criptogram,a)
        dec+=chr(pos+ord("A"))
    return dec

def decode(file):
    """
    Prints in the standard output the decodification of the file
    @type file: file
    """
    criptogram=read_criptogram()
    for line in file:
        # line include the '\n' character so we
        # remove it by writing line[:-1]
        print decode_line(line[:-1],criptogram)

def main():
    file_name=sys.argv[1]
    if len(sys.argv)>1:
        file = open(sys.argv[1])
    else:
        file=sys.stdin
    decode(file)

if __name__ == "__main__":
    main()

```

Dichos programa supone que la clave, es decir, la permutación del alfabeto que vamos a considerar se encuentra en el archivo `CRIPTO_FILENAME` definido en el fichero `generate_cripto.py`. En dicho fichero también se encuentran los procedimientos para leer y escribir el archivo.

```

import random

CRIPTO_FILENAME="criptogram.txt"

def read_criptogram():
    """
    This function reads the criptogram from file CRIPTO_FILENAME
    @type: list of one letter strings
    """
    f=open(CRIPTO_FILENAME)
    criptogram=[]
    a=f.read(1)
    while a!="":
        criptogram.append(a)
        a=f.read(1)
    return criptogram

def suffle(alphabet):
    """
    This function suffle the give list "alphabet"
    @type aplphabet: list
    """
    i=0
    n=len(alphabet)
    while i<1000:
        pos1 = random.randint(0,n-1)
        pos2 = random.randint(0,n-1)
        aux=alphabet[pos1]
        alphabet[pos1]=alphabet[pos2]
        alphabet[pos2]=aux
        i+=1

def generate_criptogram(alphabet):
    """
    This function build the criptogram file CRIPTO_FILENAME from a given alphabet
    @type aplphabet: list of one letter strings

```


Confecciona un programa que presente las distintas opciones para cada rasgo, para que el usuario elija una, y dibuje el individuo seleccionado.

```

Estilos de pelo disponibles:
1.- Pelo denso      WWWWWWWW
2.- Pelo escaso     |||||
3.- Rapado          |""""|
4.- A raya         \\\\/////

Escoja opción: _

```

Rasgos en caras Con todo, la identificación de rasgos separados no siempre resultaba fácil. Por eso se confeccionó la versión definitiva, que presentaba cuatro individuos, como los siguientes,

```

      WWWWWWWW      \\\\/////      |""""|      |||||
      |  O O  |      |-(. .)-|      |-(o o)-|      | \ / |
@      J      @      {  "  }      [  j  ]      <  -  >
      |  ==  |      |  -  |      |  _ _  |      |  _ _  |
      \_____/      \_____/      \_____/      \_____/

-- n. 1 --      -- n. 2 --      -- n. 3 --      -- n. 4 --

```

para que el testigo describiese al sospechoso combinando los ojos de un modelo, las orejas de otro y así sucesivamente. Desarrolla esta versión final, donde ahora el menú consiste en mostrar los retratos de referencia y en pedir las correspondientes elecciones:

```

Escoja los rasgos:
Pelo:              -
Ojos:              -
Orejas/nariz:     -
Boca:              -

```

Notas bibliográficas La idea de este enunciado está tomada de [AR95].

Solución

▷ 21. Sin los elementos de las ráfagas

Una *ráfaga* 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.

Solución

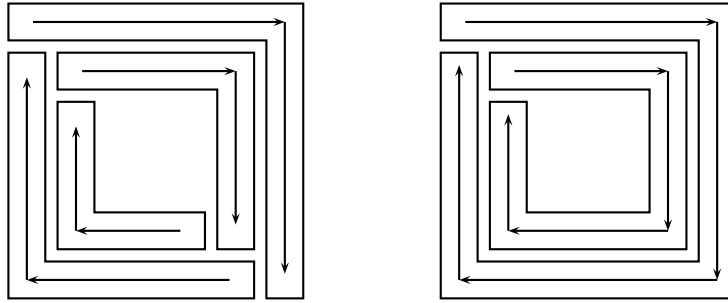
Referencias

- [AR95] Owen Astrachan and David Reed. The Applied Apprenticeship Approach to CS1. *SIGCSE Bulletin*, 27(1):1–5, 1995.
- [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 jajá!*. 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. Si leemos los datos que nos introducen carácter a carácter, podemos saber si en realidad nos han escrito un número u otra cosa. Atención a la diferencia de un dígito (los dígitos son los números de 0 al 9) si lo interpretamos como un carácter o como un número.
2. En el desarrollo de este apartado, puede ser conveniente desarrollar un subprograma auxiliar que escribe una (o dos) componente(s) del vector en su posición, dejando en blanco las demás.

3. En el desarrollo de este apartado, puede ser conveniente desarrollar un subprograma auxiliar que escriba un trozo del vector en su posición, dejando en blanco las demás.
4. 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) .
5. 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 para el anillo más exterior, otro para el siguiente y así sucesivamente.

6. Haber trabajado con el ejercicio 2 puede ser de gran ayuda para saber cómo *obtener* los dígitos que componen un número.
7. En el ejercicio 12 tienes funciones que te serán útiles.
8. Conviene definir primero una función que calcule la entropía de una posición.
9. 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.
10. 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.