Comenzado el	martes, 6 de noviembre de 2018, 15:07
Estado	Finalizado
Finalizado en	miércoles, 7 de noviembre de 2018, 22:44
Tiempo empleado	1 día 7 horas
Puntos	15/15
Calificación	**10** de 10 (**100**%)

Pregunta 1

Correcta

Puntúa 2 sobre 2

El cálculo del valor a^n (para simplificar, asumimos a > 0 y n >= 0 y sabiendo que si *n = 0* entonces $a^0 = 1$) es igual a multiplicar *n* veces el número *a* por sí mismo. Por caso, $5^3 = 5 * 5 * 5 = 125$. Sabiendo esto, ¿cuál de las siguientes sería una *definición recursiva* matemáticamente correcta de la operación *potencia(a, n)*?

Seleccione una:

a.

potencia(a, n)
(a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a, n) si n > 0

b.

potencia(a, n)
(a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a-1, n) si n > 0

c.

potencia(a, n)
(a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a + potencia(a, n-1) si n > 0

d.

potencia(a, n)
(a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a, n-1) si n > 0

¡Correcto!

¡Correcto!

La respuesta correcta es:

potencia(a, n)
(a y n enteros, a > 0 y n >= 0)

= 1 si n == 0

= a * potencia(a, n-1) si n > 0

Pregunta 2

Correcta

Puntúa 1 sobre 1

Suponga que se quiere plantear una ***definición recursiva*** del concepto de ***bosque***. ¿Cuál de las siguientes propuestas generales es correcta y constituye la mejor definición?

Seleccione una:

a.
Un bosque es un conjunto de árboles que puede estar vacío, o puede contener uno o más árboles agrupados con otro bosque. ✓

¡Correcto!

b.
Un bosque es un conjunto de árboles que puede estar vacío, o puede contener n árboles (con n > 0).

c.
Un bosque es un bosque.

d.
Un bosque es un conjunto que puede contener uno o más árboles agrupados con otro bosque.

¡Correcto!

La respuesta correcta es:
Un bosque es un conjunto de árboles que puede estar vacío, o puede contener uno o más árboles agrupados con otro bosque.

Pregunta 3

Correcta

Puntúa 1 sobre 1

¿Cuáles de los siguientes son *factores esenciales* a tener en cuenta cuando se realiza el *análisis de la eficiencia de un algoritmo* (por ejemplo, para efectuar una comparación de rendimiento entre dos algoritmos que resuelven el mismo problema)?

Seleccione una o más de una:

- [x] a. El consumo de memoria. ✓

  ¡Correcto!

- [x] b. La complejidad aparente del código fuente. ✓

  ¡Correcto!

- [ ] c. El diseño de la interfaz de usuario.

- [x] d. El tiempo de ejecución. ✓

  ¡Correcto!

¡Correcto!

Las respuestas correctas son:
El tiempo de ejecución.,

El consumo de memoria.,

La complejidad aparente del código fuente.

Pregunta 4

Correcta

Puntúa 1 sobre 1

Suponga que la versión recursiva de la función para el cálculo del factorial que se presentó en clases fuese redefinida en la forma siguiente:

```
def factorial(n):
    f = n * factorial(n - 1)
    if n == 0:
        return 1
    return f
```

¿Cuá es el problema (si lo hubiese) con esta versión de la función?

Seleccione una:

a.
Para calcular el factorial necesita un ciclo for que genere los números a multiplicar. Así planteada, la función hace un único primer cálculo incompleto y lo retorna.

b.
No hay ningún problema. Funciona correctamente.

c.
Contiene todos los elementos que debería tener una función recursiva bien planteada, pero en el orden incorrecto: la condición para chequear si n es cero debería estar antes que la llamada recursiva para evitar la recursión infinita. ✓

¡Correcto!

d.
La función siempre retorna un 1, sin importar cual sea el valor de n.

¡Correcto!

La respuesta correcta es:
Contiene todos los elementos que debería tener una función recursiva bien planteada, pero en el orden incorrecto: la condición para chequear si n es cero debería estar antes que la llamada recursiva para evitar la recursión infinita.

Pregunta 5

Correcta

Puntúa 2 sobre 2

A lo largo del curso hemos visto la forma de plantear programas controlados por menú de opciones, y sabemos que esos programas incluyen un ciclo para el control del proceso. Pero también podríamos hacer un planteo basado en recursión, sin usar el ciclo, en forma similar a la que se muestra aquí:

```
__author__ = 'Cátedra de AED'


def menu():
    print('1. Opcion 1')
    print('2. Opcion 2')
    print('3. Salir')
    op = int(input('Ingrese opcion: '))
    if op == 1:
        print('Eligio la opcion 1...')
    elif op == 2:
        print('Eligio la opcion 2...')
    elif op == 3:
        return
    menu()


# script principal...
menu()
```

¿Hay algún problema o contraindicación respecto del uso y aplicación de esta versión recursiva para la gestión de un menú? Seleccione la respuesta que mejor describa este punto.

Seleccione una:

a.
Esta versión recursiva posiblemente sea más simple y compacta que la versión basada en un ciclo, pero la versión recursiva pide memoria de stack cada vez que se invoca, por lo que es más conveniente la iterativa.

¡Correcto! De todos modos, si la idea es usar el menú sólo para un par de operaciones y terminar de inmediato, no habrá problema alguno. Lo malo sería que el programa funcione en forma permanente, las 24 horas del día y todos los días... pues en ese caso, se corre el serio riesgo de provocar un overflow de stack.

b.
Esta versión recursiva es completamente equivalente a la versión iterativa. No hay ningún motivo para preferir la una sobre la otra. Da lo mismo si el programador usa la recursiva o la iterativa.

c.
La versión recursiva que se mostró no funciona. Sólo permite cargar una vez la opción elegida, e inmediatamente se detiene.

d.

No hay ningún problema ni contraindicación. Y no sólo eso: la versión recursiva es más simple y compacta que la versión basada en un ciclo, por lo cual es incluso preferible usarla.

¡Correcto!

La respuesta correcta es:
Esta versión recursiva posiblemente sea más simple y compacta que la versión basada en un ciclo, pero la versión recursiva pide memoria de stack cada vez que se invoca, por lo que es más conveniente la iterativa.

Pregunta 6

Correcta

Puntúa 1 sobre 1

¿Cuál de los siguientes era el nombre verdadero del matemático conocido como *Fibonacci*, que fue quien definió la famosa *Sucesión de Fibonacci*?

Seleccione una:

a.
Carl Gauss

b.
Leonardo Pisano ✓

¡Correcto!

c.
Giuseppe Peano

d.
Leonhard Euler

¡Correcto!

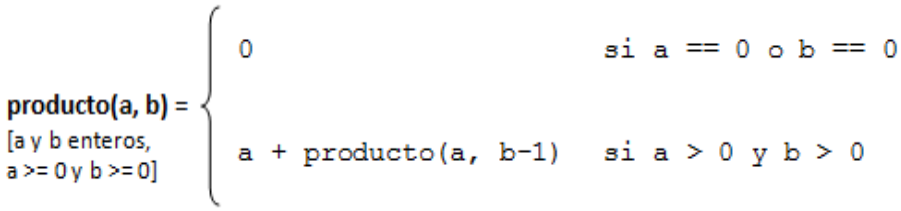
La respuesta correcta es:
Leonardo Pisano

Pregunta 7

Correcta

Puntúa 2 sobre 2

El producto de dos números *a* y *b* mayores o iguales cero, es en última instancia *una suma*: se puede ver como sumar *b* veces el número *a*, o como sumar *a* veces el número *b*. Por ejemplo: 5 * 3 = 5 + 5 + 5 o bien 5 * 3 = 3 + 3 + 3 + 3 + 3. Sabiendo esto, se puede intentar hacer una definición recursiva de la operación producto(a, b) [a >= 0, b >= 0], que podría ser la que sigue:



¿Es correcto el siguiente planteo de la función *producto(a, b)*?

```
def producto(a, b):
    if a == 0 or b == 0:
        return 0
    return a + producto(a, b-1)
```

Seleccione una:

a.
No. La propia definición previa del concepto de producto es incorrecta: un producto es una multiplicación, y no una suma...

b.
Sí. Es correcto. ✓

¡Correcto!

c.
No. La última línea debería decir *return a + producto(a-1, b-1)* en lugar de *return a + producto(a, b-1)*.

d.
No. La función sugerida siempre retornará 0, sean cuales fuesen los valores de *a* y *b*.

¡Correcto!

La respuesta correcta es:
Sí. Es correcto.

Pregunta 8

Correcta

Puntúa 1 sobre 1

En la tabla siguiente se muestra un resumen comparativo entre las versiones iterativa (basada en ciclos) y recursiva del algoritmo de cálculo del término n-ésimo de la Sucesión de Fibonacci. Note que se han dejado sin completar los casilleros que corresponden al tiempo de ejecución para ambos casos.

Cálculo del término *n-ésimo* de Fibonacci [F(n)]	**Complejidad código fuente**	**Consumo de memoria**	**Tiempo de ejecución**
Versión iterativa	Aceptable	Constante (no depende de *n*)	????
Versión recursiva	Optima	Proporcional a *n* (lineal)	????

¿Cuál es la estimación para el tiempo de ejecución de ambos algoritmos?

Seleccione una:

a.
Versión iterativa: tiempo proporcional a *n* (lineal) - Versión recursiva: tiempo proporcional a *n* (lineal)

b.
Versión iterativa: tiempo constante (no depende de *n*) - Versión recursiva: tiempo proporcional a b^n (exponencial, para algún $b > 1$)

c.
Versión iterativa: tiempo proporcional a *n* (lineal) - Versión recursiva: tiempo proporcional a b^n (exponencial, para algún $b > 1$) ✓

¡Correcto! Efectivamente... la cantidad de instancias recursivas que llegan a crearse a lo largo de todo el proceso recursivo está en el orden de 1.8^n (lo cual es muy mala noticia...)

d.
Versión iterativa: tiempo proporcional a b^n (exponencial, para algún $b > 1$) - Versión recursiva: tiempo proporcional a *n* (lineal)

¡Correcto!

La respuesta correcta es:
Versión iterativa: tiempo proporcional a *n* (lineal) - Versión recursiva: tiempo proporcional a b^n (exponencial, para algún $b > 1$)

Pregunta 9

Correcta

Puntúa 1 sobre 1

¿Cuáles de las siguientes propuestas generales son ***ciertas*** en relación al segmento de memoria conocido como *Stack Segment*?

Seleccione una o más de una:

- [x] a.
  El Stack Segment funciona como una pila (o apilamiento) de datos (modalidad *LIFO*: Last In - First Out): el último dato en llegar, se almacena en la cima del stack, y será por eso el primero en ser retirado. ✓

  ¡Correcto!

- [ ] b.
  El Stack Segment se utiliza como soporte interno *solamente* en el proceso de invocación a funciones recursivas: se va llenando a medida que la cascada recursiva se va desarrollando, y se vacía a medida que se produce el proceso de regreso o vuelta atrás.

- [ ] c.
  El Stack Segment funciona como una cola (o fila) de datos (modalidad *FIFO*: First In - First Out): el primer dato en llegar, se almacena al frente del stack, y será por eso el primero en ser retirado.

- [x] d.
  El Stack Segment se utiliza como soporte interno en el proceso de invocación a funciones *(con o sin recursividad)*: se va llenando a medida que se desarrolla la cascada de invocaciones, y se vacía a medida que se produce el proceso de regreso o vuelta atrás. ✓

  ¡Correcto!

¡Correcto!

Las respuestas correctas son:
El Stack Segment funciona como una pila (o apilamiento) de datos (modalidad *LIFO*: Last In - First Out): el último dato en llegar, se almacena en la cima del stack, y será por eso el primero en ser retirado.,

El Stack Segment se utiliza como soporte interno en el proceso de invocación a funciones *(con o sin recursividad)*: se va llenando a medida que se desarrolla la cascada de invocaciones, y se vacía a medida que se produce el proceso de regreso o vuelta atrás.

Pregunta 10

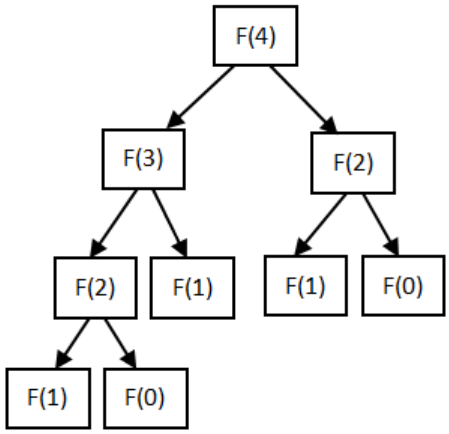
Correcta

Puntúa 2 sobre 2

Sabemos que la recursión es una de las técnicas o estrategias básicas para el planteo de algoritmos y que una de sus ventajas es que permite el diseño de algoritmos compactos y muy claros, aunque al costo de usar algo de memoria extra en el stack segment y por consiguiente también un algo de tiempo extra por la gestión del stack. Algunos problemas pueden plantearse en forma directa procesando recursivamente diversos subproblemas menores. Tal es el caso de la *sucesión de Fibonacci*, en la cual cada término se calcula como la suma de los dos inmediatamente anteriores [esto se expresa con la siguiente relación de recurrencia: *F(n) = F(n-1) + F(n-2)* con *F(0) = 0* y *F(1) = 1*]. La función sencilla que mostramos a continuación que calcula el término n-ésimo de la sucesión en forma recursiva:

```
def fibo(n):
    if n <= 1:
        return 1
    return fibo(n-1) + fibo(n-2)
```

Sin embargo, un inconveniente adicional es que la *aplicación directa* de *dos o más invocaciones recursivas* en el planteo de un algoritmo podría hacer que un mismo subproblema se resuelva más de una vez, incrementando el tiempo total de ejecución. Por ejemplo, para calcular fibo(4) el árbol de llamadas recursivas es el siguiente:



y puede verse que en este caso, se calcula *2 veces* el valor de *fibo(2)*, *3 veces* el valor de *fibo(1)* y *2 veces* el valor de *fibo(0)*... con lo que la cantidad de llamadas a funciones para hacer *más de una vez el mismo trabajo* es de ***7*** (= 2 + 3 + 2).

Suponga que se quiere calcular el valor del sexto término de la sucesión. Analice el árbol de llamadas recursivas que se genera al hacer la invocación ***t = fibo(6)*** **¿Cuántas veces en total la función *fibo()* se invoca para hacer más de una vez el mismo trabajo, SIN incluir en ese conteo a las invocaciones para obtener *fibo(0)* y *fibo(1)*?**

Seleccione una:

- a. 10 ✓ ¡Correcto! Efectivamente: *fibo(4)* se invoca 2 veces, *fibo(3)* se invoca 3 veces y *fibo(2)* se invoca 5 veces...
- b. 11
- c. 0
- d. 12

¡Correcto!

La respuesta correcta es: 10

Pregunta 11

Correcta

Puntúa 1 sobre 1

¿En cuáles de las siguientes situaciones el uso de *recursividad* está efectivamente recomendado? (Más de una respuesta puede ser válida. Marque todas las que considere correctas).

Seleccione una o más de una:

- [x] a. Recorrido y procesamiento de estructuras de datos no lineales como árboles y grafos. ✓

  ¡Correcto!

- [ ] b. Siempre que se pueda escribir una definición recursiva del problema.

- [ ] c. Nunca.

- [x] d. Generación y procesamiento de imágenes y gráficos fractales (figuras compuestas por versiones más simples de la misma figura original). ✓

  ¡Correcto!

¡Correcto!

Las respuestas correctas son:
Recorrido y procesamiento de estructuras de datos no lineales como árboles y grafos.,

Generación y procesamiento de imágenes y gráficos fractales (figuras compuestas por versiones más simples de la misma figura original).