Práctica dirigida de Algoritmia

Tema: Representación algorítmica

1. La secuencia de Fibonacci es una sucesión de números enteros en la que cada número es la suma de los dos números anteriores, comenzando con 0 y 1. Formalmente, la secuencia se define como:

   $$F(1) = 0$$
   $$F(2) = 1$$
   $$F(n) = F(n-1) + F(n-2) \quad \text{para } n > 2$$

   Utilizando esta definición, se te pide que implementes una función en R llamada `fibonacci`, que tome como argumento un número entero positivo n y retorne un vector con los primeros n términos de la secuencia de Fibonacci.

2. La secuencia de Fibonacci es una sucesión de números enteros donde cada número es la suma de los dos números anteriores, comenzando con 0 y 1. Esta secuencia puede calcularse utilizando diferentes métodos, uno de los cuales es un enfoque iterativo.

   Se te pide que implementes una función en R llamada `fibonacci_iterative`, que tome como argumento un número entero positivo n y retorne el n-ésimo término de la secuencia de Fibonacci usando un enfoque iterativo.

   Detalles de la Implementación

   - Si $n = 1$, la función debe retornar 0 (el primer término de la secuencia).
   - Si $n = 2$, la función debe retornar 1 (el segundo término de la secuencia).
   - Para valores de n mayores que 2, la función debe calcular el n-ésimo término de la secuencia utilizando un bucle iterativo que actualice dos variables para almacenar los dos términos previos de la secuencia.

3. Implementa una función en R que calcule el n-ésimo término de la secuencia de Fibonacci utilizando un enfoque recursivo. Después de

implementar la función, intenta calcular el término 100 de la secuencia. ¿Qué observas al intentar calcular el término 100? ¿Cómo afecta la recursión a la eficiencia del cálculo?; ¿Ocurre lo mismo que con el enfoque iterativo? ¿Cómo comparas el rendimiento de ambos enfoques para calcular grandes valores de n?

4. Implementa una función en R llamada `suma` que tome como argumento un número entero positivo n y calcule la suma de los primeros n números consecutivos (es decir, la suma de 1 a n). La función debe utilizar un bucle `for` para realizar la suma e imprimir el resultado en el formato:

   "La suma de los primeros n números consecutivos es = `suma`".

   Prueba la función con diferentes valores de n y verifica que los resultados sean correctos.

5. Implementa una función en R llamada `suma` que tome como argumento un número entero n. La función debe realizar lo siguiente:

   a) **Validación:** Si el valor de n es mayor que 0, la función debe calcular la suma de los primeros n números consecutivos (es decir, la suma de 1 a n) utilizando un bucle `for`. Después, debe imprimir el resultado en el formato:

      "La suma de los n números consecutivos es: `suma`".

   b) **Manejo de errores:** Si el valor de n es 0 o un número negativo, la función debe imprimir un mensaje indicando que "El número insertado es inválido".

   Prueba la función con diferentes valores de n, incluyendo casos positivos y negativos, para verificar que el comportamiento sea el esperado.

6. Implementa una función en R llamada `an_pa` que tome como argumentos tres valores:

   - **a1:** El primer término de una progresión aritmética.
   - **n:** El número de términos.
   - **r:** La razón o diferencia común entre los términos consecutivos.

La función debe calcular y mostrar el n-ésimo término (a_n) de la progresión aritmética utilizando la fórmula:

$$a_n = a_1 + (n-1) \times r$$

7. Implementa una función en R llamada `PA_nter` que tome como argumentos tres valores:

   - **a1:** El primer término de una progresión aritmética.
   - **n:** El número de términos que se desean generar.
   - **r:** La razón o diferencia común entre los términos consecutivos.

   La función debe generar y retornar un vector que contenga los primeros n términos de la progresión aritmética, utilizando la fórmula:

   $$a_n = a_1 + (n-1) \times r$$

8. Implementa una función en R llamada `cal_sum_ter` que tome como argumentos tres valores:

   - **a1:** El primer término de una progresión aritmética.
   - **n:** El número de términos que se desean calcular.
   - **r:** La razón o diferencia común entre los términos consecutivos.

   La función debe realizar lo siguiente:

   a) **Generar los términos:** Calcular y almacenar los primeros n términos de la progresión aritmética utilizando la fórmula:

   $$a_i = a_1 + (i-1) \times r$$

   donde i es el índice del término (de 1 a n).

   b) **Calcular la suma:** Calcular la suma de estos n términos.

   c) **Retorno:** La función debe retornar una lista que contenga:

   - Un vector con los primeros n términos de la progresión aritmética.

- La suma de estos términos.

9. Implementa una función en R llamada `suma_n_cuadrados` que tome como argumento un número entero positivo n. La función debe realizar lo siguiente:

   a) **Generar los términos:** Calcular y almacenar en un vector los cuadrados de los primeros n números enteros. Es decir, el vector debe contener los valores $1^2, 2^2, 3^2, \ldots, n^2$.

   b) **Calcular la suma:** Calcular la suma de estos cuadrados.

   c) **Retorno:** La función debe retornar una lista que contenga:

   - Un vector con los cuadrados de los primeros n números enteros.
   - La suma de estos cuadrados.

   Ejemplo de uso: Si `n = 5`, la función debe retornar:

   - Un vector con los términos `c(1, 4, 9, 16, 25)`, que representa los cuadrados de los números del 1 al 5.
   - La suma de estos términos, que sería 55.

   Prueba la función con diferentes valores de `n` para verificar que los resultados sean correctos.

10. Implementa una función en R llamada `poligono` que tome como argumento un número entero n, que representa el número de lados de un polígono. La función debe realizar lo siguiente:

    a) **Identificación del Polígono:** La función debe identificar el tipo de polígono según el número de lados n. Para valores de n entre 3 y 12, la función debe imprimir el nombre del polígono correspondiente (por ejemplo, "triángulo" para 3 lados, "cuadrilátero" para 4 lados, etc.). Si n es mayor que 12, la función debe imprimir "El polígono tiene n lados".

    b) **Cálculo del Ángulo Interior:** Si n es mayor o igual a 3, la función debe calcular y mostrar el valor del ángulo interior de cada vértice del polígono, utilizando la fórmula:

$$\text{Ángulo interior} = \frac{(n-2) \times 180}{n} \text{ grados}$$

c) **Manejo de Errores:** Si n es menor que 3, la función debe imprimir un mensaje indicando que "Con el número de lados proporcionado no se puede formar un polígono".

Ejemplo de uso: Si `n = 5`, la función debe identificar el polígono como un pentágono y calcular su ángulo interior, mostrando ambos resultados.

Prueba la función con diferentes valores de `n` para verificar que los resultados sean correctos.

11. Implementar una función en R que muestre en letras el número de la cara de un dado obtenido al azar.

12. Implementa una función en R que sume todos los números pares menores o iguales a N (Número entero positivo cualquiera). La función debe devolver el resultado de la suma solo si N es positivo.

Ordenamiento por Inserción

Dado el arreglo inicial $\langle 5, 2, 4, 6, 1, 3 \rangle$, aplicamos el algoritmo de inserción paso a paso.

Paso 1: Inicialización

- Array original: $\langle 5, 2, 4, 6, 1, 3 \rangle$
- Comienza con $i = 2$, donde key $= A[i] = 2$.

Paso 2: i = 2, insertar 2

1. key $= A[2] = 2$
2. $j = i - 1 = 1$, entonces $j = 1$.
3. Comparación: $A[j] = 5 >$ key $= 2$, entonces se cumple la condición del `while`.
4. Desplazamiento: $A[j + 1] = A[2] = 5$, el array es ahora $\langle 5, 5, 4, 6, 1, 3 \rangle$.

5. Decremento de j: $j = j - 1 = 0$.
6. El `while` termina porque $j = 0$.
7. Inserción de key: $A[j + 1] = A[1] = 2$, el array es ahora $\langle 2, 5, 4, 6, 1, 3 \rangle$.

Paso 3: i = 3, insertar 4

1. key $= A[3] = 4$
2. $j = i - 1 = 2$, entonces $j = 2$.
3. Comparación: $A[j] = 5 >$ key $= 4$, entonces se cumple la condición del `while`.
4. Desplazamiento: $A[j + 1] = A[3] = 5$, el array es ahora $\langle 2, 5, 5, 6, 1, 3 \rangle$.
5. Decremento de j: $j = j - 1 = 1$.
6. Inserción de key: $A[j + 1] = A[2] = 4$, el array es ahora $\langle 2, 4, 5, 6, 1, 3 \rangle$.

Paso 4: i = 4, insertar 6

1. key $= A[4] = 6$
2. $j = i - 1 = 3$, entonces $j = 3$.
3. Comparación: $A[j] = 5 <$ key $= 6$, no se cumple la condición del `while`.
4. Inserción directa, el array permanece $\langle 2, 4, 5, 6, 1, 3 \rangle$.

Paso 5: i = 5, insertar 1

1. key $= A[5] = 1$
2. $j = i - 1 = 4$, entonces $j = 4$.
3. Comparación: $A[j] = 6 >$ key $= 1$, se cumple la condición del `while`.
4. Desplazamiento: $A[j + 1] = A[5] = 6$, el array es ahora $\langle 2, 4, 5, 6, 6, 3 \rangle$.

5. Decremento de j: $j = j - 1 = 3$.
6. Comparación: $A[j] = 5 >$ key $= 1$, se cumple la condición del `while`.
7. Desplazamiento: $A[j + 1] = A[4] = 5$, el array es ahora $\langle 2, 4, 5, 5, 6, 3 \rangle$.
8. Decremento de j: $j = j - 1 = 2$.
9. Comparación: $A[j] = 4 >$ key $= 1$, se cumple la condición del `while`.
10. Desplazamiento: $A[j + 1] = A[3] = 4$, el array es ahora $\langle 2, 4, 4, 5, 6, 3 \rangle$.
11. Decremento de j: $j = j - 1 = 1$.
12. Comparación: $A[j] = 2 >$ key $= 1$, se cumple la condición del `while`.
13. Desplazamiento: $A[j + 1] = A[2] = 2$, el array es ahora $\langle 2, 2, 4, 5, 6, 3 \rangle$.
14. Decremento de j: $j = j - 1 = 0$.
15. Inserción de key: $A[j + 1] = A[1] = 1$, el array es ahora $\langle 1, 2, 4, 5, 6, 3 \rangle$.

Paso 6: i = 6, insertar 3

1. key $= A[6] = 3$
2. $j = i - 1 = 5$, entonces $j = 5$.
3. Comparación: $A[j] = 6 >$ key $= 3$, se cumple la condición del `while`.
4. Desplazamiento: $A[j + 1] = A[6] = 6$, el array es ahora $\langle 1, 2, 4, 5, 6, 6 \rangle$.
5. Decremento de j: $j = j - 1 = 4$.
6. Comparación: $A[j] = 5 >$ key $= 3$, se cumple la condición del `while`.
7. Desplazamiento: $A[j + 1] = A[5] = 5$, el array es ahora $\langle 1, 2, 4, 5, 5, 6 \rangle$.
8. Decremento de j: $j = j - 1 = 3$.
9. Comparación: $A[j] = 4 >$ key $= 3$, se cumple la condición del `while`.
10. Desplazamiento: $A[j + 1] = A[4] = 4$, el array es ahora $\langle 1, 2, 4, 4, 5, 6 \rangle$.

11. Decremento de j: $j = j - 1 = 2$.
12. Inserción de key: $A[j + 1] = A[3] = 3$, el array es ahora $\langle 1, 2, 3, 4, 5, 6 \rangle$.

El arreglo final ordenado es $\langle 1, 2, 3, 4, 5, 6 \rangle$.