TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

Materia: PROGRAMACIÓN 1

Tema: Análisis de algoritmos eficiencia y optimización en Python

Estudiantes:

- Grassi Pablo J. - pjgrassi@gmail.com
- Holsbach Andrés Maximiliano - andresholsbach@gmail.com

Contenido

Introducción

El análisis de algoritmos es la ciencia aplicada al estudio de la eficiencia de estos mismos. Su objetivo principal es evaluar el rendimiento en función del tiempo de ejecución y utilización de recursos.

Marco Teórico

Conozcamos un algoritmo:

Según el libro: *Introducción to Algorithms* de Thomas H. Cormen, junto con Charles E. Leiserson y Ronald L. Rivest, un algoritmo en un conjunto bien definido de pasos para resolver un problema, en este caso computacional.

¿Cómo se compone un algoritmo?

- Debe ser finito.
- Debe tener aceptar entradas.
- Debe tener pasos bien definidos.
- Debe tener al menos una salida.

Interés compuesto: los intereses generados se van sumando al capital inicial, y se van generando nuevos intereses, generando esto un incrementos constante del capital resultante.

Objetivo del análisis

Conocer el comportamiento del algoritmo bajo distintas circunstancias es crucial para diseñar soluciones eficientes, y optimizar recursos.

Para ello contamos con los siguientes conceptos.

- Eficiencia temporal: evalúa el tiempo de ejecución según la entrada.
- Eficiencia espacial: determina cuanta memoria precisa el algoritmo para ejecutarse.
- Escalabilidad: analiza el comportamiento del algoritmo sometido a incrementos de datos de entrada

Enfoques de análisis

Existen dos formas de abordar el análisis de algoritmos:

- Análisis asintótico: evalúa el comportamiento del algoritmo a medida que el tamaño de la entrada de datos incrementa. Para ello utilizamos la notación Big-O, para describir la complejidad temporal y espacial, proporcionándonos una estimación teórica de su eficiencia sin necesidad de implementarlo.
- Análisis empírico: consta de ejecutar el algoritmo sometido a distintos valores de entradas, para medir su rendimiento en tiempo de ejecución y uso de recursos. Esto entrega datos experimentales realiza sobre el comportamiento del algoritmo.

Ambos enfoques son esenciales, nos permiten obtener la estimación teórica, prescindiendo del tipo de lenguaje o hardware, y luego comprobar su desempeño para validar el análisis teórico y obtener resultados reales.

Desarrollo de algoritmos

Se analizarán dos algoritmos para el cálculo de interés compuesto.

- En este primer ejemplo observamos el calculo del interés compuesto con un script iterativo.

```
from interesCompuesto import valorFuturoIterativo

"""

Ejemplo de uso:

Para simular el análisis de el algoritmo, generamos una lista con valores aleatorios.

Los datos de la inversion se asemejaran a una inversión de el plazo establecido.

El resultado serán el valor final de la inversión y el tiempo que tardó en calcularlo, para cada iteración.

"""

valoresIniciales = [642226.93, 760322.26, 104757.69, 399684.60, 395746.39, 740615.86, 959015.46, 854345.91, 725954.37, 165218.05]  # Valores iniciales de ejemplo


for i in (valoresIniciales):

    principal = i  # Inversión inicial de la lista

    tasaInteres = 0.75   # Tasa de interés fija del 75%

    periodos = 30 # Periodos fijos de 30 años


    # Calcular el valor futuro

    valorFinal, tiempo = valorFuturoIterativo(principal, tasaInteres, periodos)


    # Mostrar resultados

    print(f"{tasaInteres * 100};{periodos};{principal:.2f};{valorFinal:.2f};{tiempo}")
```

- Captura de pantalla con resultados

```
$ python InteresConBucle.py
75.0;30;642226.93;12555383225376.46;0.0026226043701171875

75.0;30;760322.26;14864118745509.97;0.002384185791015625

75.0;30;104757.69;2047987840925.93;0.001430511474609375

75.0;30;399684.60;7813738552323.40;0.00286102294921875

75.0;30;395746.39;7736747486607.72;0.002384185791015625

75.0;30;740615.86;14478863328094.57;0.0019073486328125

75.0;30;959015.46;18748523390884.10;0.0026226043701171875

75.0;30;854345.91;16702258665925.12;0.002384185791015625

75.0;30;725954.37;14192234697300.43;0.0021457672119140625

75.0;30;165218.05;3229973451127.95;0.002384185791015625
```

- En este segundo ejemplo observaremos la optimización del algoritmo utilizando una fórmula cerrada para calcular el interés compuesto

```
from interesCompuesto import valorFuturoFormula

"""

Ejemplo de uso:

Para simular el análisis de el algoritmo, generamos una lista con valores aleatorios.

Los datos de la inversion se asemejaran a una inversión de el plazo establecido.

El resultado serán el valor final de la inversión y el tiempo que tardó en calcularlo, para cada iteración.

"""

valoresIniciales = [642226.93, 760322.26, 104757.69, 399684.60, 395746.39, 740615.86, 959015.46, 854345.91, 725954.37, 165218.05]  # Valores iniciales de ejemplo


for i in (valoresIniciales):

    principal = i  # Inversión inicial de la lista

    tasaInteres = 0.75   # Tasa de interés fija del 75%

    periodos = 30 # Periodos fijos de 30 años


    # Calcular el valor futuro

    valorFinal, tiempo = valorFuturoFormula(principal, tasaInteres, periodos)


    # Mostrar resultados

    print(f"{tasaInteres * 100};{periodos};{principal:.2f};{valorFinal:.2f};{tiempo}")
```

- Captura de pantalla con resultados.

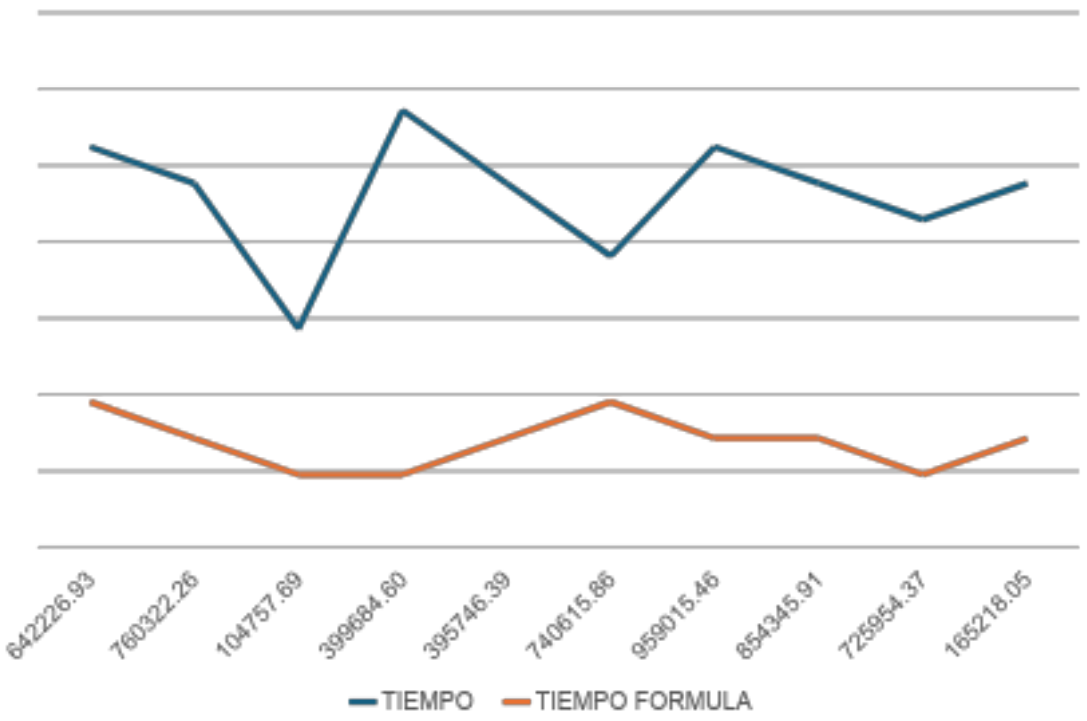
```
$ python InteresConFormula.py
75.0;30;642226.93;12555383225376.46;0.0054836273193359375
75.0;30;760322.26;14864118745509.97;0.0007152557373046875
75.0;30;104757.69;2047987840925.93;0.0002384185791015625
75.0;30;399684.60;7813738552323.40;0.0002384185791015625
75.0;30;395746.39;7736747486607.72;0.000476837158203125
75.0;30;740615.86;14478863328094.57;0.000476837158203125
75.0;30;959015.46;18748523390884.11;0.0007152557373046875
75.0;30;854345.91;16702258665925.12;0.002384185791015625
75.0;30;725954.37;14192234697300.43;0.0002384185791015625
75.0;30;165218.05;3229973451127.95;0.0007152557373046875
```

Resultados obtenidos

- Ambos algoritmos devuelven exactamente el mismo valor final, lo podemos observar en la tabla de análisis

BUCLE FOR		FORMULA DIRECTA	
INICIAL	FINAL	INICIAL	FINAL
642226,93	12555383225376,40	642226,93	12555383225376,40
760322,26	14864118745509,90	760322,26	14864118745509,90
104757,69	2047987840925,93	104757,69	2047987840925,93
399684,60	7813738552323,40	399684,60	7813738552323,40
395746,39	7736747486607,72	395746,39	7736747486607,72
740615,86	14478863328094,50	740615,86	14478863328094,50
959015,46	18748523390884,10	959015,46	18748523390884,10
854345,91	16702258665925,10	854345,91	16702258665925,10
725954,37	14192234697300,40	725954,37	14192234697300,40
165218,05	3229973451127,95	165218,05	3229973451127,95

- Utilizar la fórmula constante mejora significativamente la eficiencia, tanto en tiempo de ejecución como en recursos computacionales.
- Compartimos un gráfico comparativo de ambos algoritmos



- Para el ejemplo iterativo O(m x n), la complejidad es O(n), varía según los datos ingresados donde
  - m: es el tamaño de los valores iniciales
  - n: es el número de periodos
- Para el ejemplo de fórmula directa O(m) la complejidad es O1, independiente de datos ingresados.

Conclusión

El enfoque basado en la fórmula es significativamente más eficiente y estable que el modo iterativo, más aún cuando los datos incrementan su tamaño.

Bibliografía:

- "Introduction to Algorithms" de Thomas H. Cormen, Charles E. Leiserson y Ronald L. Rivest, una referencia clásica en el estudio de algoritmos.
- "Fundamentos de Algoritmia" de Gilles Brassard y Paul Bratley, que aborda el diseño y análisis de algoritmos.