Práctica 01: Análisis de Casos

CRISTIAN GARCÍA NIEVES

Septiembre 2023

1 Objetivo de la Práctica

El objetivo principal de esta práctica es que los estudiantes adquieran una comprensión sólida y profunda de los conceptos clave relacionados con la complejidad computacional de los algoritmos. Los tres casos principales a analizar (mejor caso, peor caso y caso promedio) permiten una evaluación completa del rendimiento de un algoritmo bajo diferentes circunstancias.

1.1 Objetivos Particulares

1.1 Comprender la Variabilidad de la Ejecución: Los algoritmos pueden comportarse de manera diferente según las características de los datos de entrada. Al analizar el mejor, peor y caso promedio, los estudiantes aprenderán a apreciar cómo un algoritmo puede variar en su rendimiento.

1.2 Aplicar Conceptos de Complejidad Computacional: La práctica ayudará a los estudiantes a aplicar conceptos fundamentales de la teoría de la complejidad computacional, como la notación O-grande (Big O), para describir y comparar el rendimiento de los algoritmos.

1.3 Desarrollar Habilidades de Análisis Crítico: A través del análisis de diferentes casos, los estudiantes mejorarán sus habilidades de análisis crítico, aprendiendo a identificar situaciones en las que un algoritmo puede ser más eficiente o ineficiente.

1.4 Preparación para Desarrollo de Algoritmos Eficientes: Al comprender los casos de mejor, peor y caso promedio, los estudiantes estarán mejor preparados para diseñar y seleccionar algoritmos eficientes en situaciones reales, lo que es fundamental en la resolución de problemas computacionales.

1.5 Promover la Resolución de Problemas: A través de la práctica, los estudiantes aprenderán a abordar y resolver problemas computacionales de manera más efectiva, seleccionando o adaptando algoritmos en función de las restricciones de tiempo y recursos.

2 Desarrollo de la Práctica

2.1 Implementación de los algoritmos

Se han implementado los siguientes métodos de ordenamiento en Python:

- **Burbuja**
- **Burbuja Optimizada**

Cada método de ordenamiento está desarrollado en una función de programación diferente, donde una vez llenado el arreglo comienza un temporizador y se finaliza en cuanto termina de ordenarse para poder calcular el tiempo que tarda en hacer el ordenamiento. A través de un menú, el usuario puede seleccionar cuál método de ordenamiento desea utilizar. Una vez seleccionado, el programa solicita al usuario ingresar el tamaño de la lista y los elementos de la misma, que serán la entrada de los algoritmos.

2.2 Análisis de Casos

2.2.1 Análisis del Algoritmo de Burbuja

Mejor Caso:

1.1 Ejemplo 1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1.2 Ejemplo 2: [5, 7, 9, 10, 12]

1.3 Ejemplo 3: [8, 10, 14, 19, 20, 23, 45, 59]

Justificación (para todos los ejemplos): Debido a que los datos ya están ordenados, el algoritmo solamente recorre el arreglo sin tener que mover los datos de lugar. Esto resulta en una ejecución eficiente en términos de tiempo.

Complejidad (para todos los ejemplos): $O(n)$ (Esto indica que el tiempo de ejecución del algoritmo crece de manera lineal con el tamaño de la entrada en el mejor caso.)

Peor Caso:

1.1 Ejemplo 1: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

1.2 Ejemplo 2: [12, 10, 9, 7, 5]

1.3 Ejemplo 3: [59, 45, 23, 20, 19, 14, 10, 8]

En el peor caso, para el algoritmo de ordenamiento de burbuja, la complejidad es $O(n^2)$. Esto se debe a que en el peor escenario, el algoritmo debe realizar $n-1$ comparaciones y $n-1$ intercambios en cada pase a través del arreglo, donde n es el número de elementos en el arreglo.

Por lo tanto, el número total de operaciones en el peor caso es aproximadamente n^2. Esto significa que la complejidad del algoritmo de burbuja en el peor

caso es cuadrática en términos de n, lo que indica un crecimiento cuadrático en el tiempo de ejecución a medida que el tamaño del conjunto de datos aumenta.

Caso Promedio:

1.1 Ejemplo 1: [1, 9, 2, 8, 3, 7, 4, 5, 6, 10]

1.2 Ejemplo 2: [9, 7, 10, 5, 12]

1.3 Ejemplo 3: [59, 8, 45, 10, 23, 14, 20, 19]

Justificación: En este caso, los datos están dispuestos de manera intercalada. Esto implica que el algoritmo de ordenamiento de burbuja tendrá un desempeño más rápido que si los datos estuvieran ordenados de mayor a menor, pero no tan rápido como si estuvieran completamente ordenados. La intercalación de los datos crea un escenario en el que el algoritmo debe realizar un número significativo de comparaciones e intercambios, pero no al mismo nivel que en el peor caso. Aunque es más eficiente que el peor caso, todavía requiere una cantidad considerable de operaciones para lograr el ordenamiento deseado.

Complejidad: $O(n^2)$

2.2.2 Análisis del Algoritmo de Burbuja Optimizada

En el algoritmo de burbuja optimizada sucede lo mismo, sin enmbargo los tiempos de ejcución reducen en la mayoría de cada caso para cada uno de los ejemplos utilizados

2.3 Comparación de Resultados

Tiempos de Ejecución en el Peor Caso:

- Ejemplo 1:
  - Burbuja: 0.00015 segundos
  - Burbuja Optimizada: 0.00006 segundos
- Ejemplo 2:
  - Burbuja: 0.000082 segundos
  - Burbuja Optimizada: 0.00007 segundos
- Ejemplo 3:
  - Burbuja: 0.00007 segundos
  - Burbuja Optimizada: 0.00008 segundos

Tiempos de Ejecución en el Mejor Caso:

- Ejemplo 1:

  - Burbuja: 0.00005 segundos
  - Burbuja Optimizada: 0.00003 segundos
- Ejemplo 2:
  - Burbuja: 0.000067 segundos
  - Burbuja Optimizada: 0.00006 segundos
- Ejemplo 3:
  - Burbuja: 0.00004 segundos
  - Burbuja Optimizada: 0.00007 segundos

Tiempos de Ejecución en el Caso Promedio:

- Ejemplo 1:
  - Burbuja: 0.000055 segundos
  - Burbuja Optimizada: 0.000051 segundos
- Ejemplo 2:
  - Burbuja: 0.000053 segundos
  - Burbuja Optimizada: 0.000067 segundos
- Ejemplo 3:
  - Burbuja: 0.000045 segundos
  - Burbuja Optimizada: 0.00002 segundos

  Como se puede observar en alguno de los casos en el método de ordenamiento de burbuja optimizado se llega a tener un menor tiempo de ejecución, sin embargo en estas pruebas que se realizaron en distintos casos, no se logra observar un cambio notorio en la majorida de los ejemplos.