

Diseño y Análisis de Algoritmos Práctica 4 (P4)

Programación dinámica

|  |  |  |
| --- | --- | --- |
| Uxío Merino | Iván Castro | Mario Chan |
| [uxio.merino@udc.es](mailto:uxio.merino@udc.es) | [ivan.castro1@udc.es](mailto:ivan.castro1@udc.es) | [mario.chan@udc.es](mailto:mario.chan@udc.es) |

# Introducción

En esta práctica se han implementado dos algoritmos de programación dinámica en Python, cuya función es, dados dos strings ( o listas de enteros) A y B, comprobar si un tercer string C es una mezcla válida de ambos. Se considera una mezcla válida aquella que cumpla que: el string resultante C contiene todos los caracteres que están en A y B con el mismo número de repeticiones de cada carácter, y el orden de los caracteres de A y B se preserva en C. Ambas implementaciones de los algoritmos han sido validadas con una función test, comprobando que son correctas, pudiendo así continuar con la medición de tiempos de ejecución y calculo empírico de sus complejidades.

# Contexto

## ¿Qué se está midiendo?

Las medidas se corresponden con los tiempos que necesitan los algoritmos para ejecutarse, a través de la resta del tiempo antes y después de la ejecución. Los algoritmos reciben tres strings y realizan la comprobación explicada en la introducción. Hay 3 escenarios: un vocabulario con solo 2 caracteres, un alfabeto que contenga todos los caracteres del código ASCII y un alfabeto de igual tamaño que los strings iniciales A y B.

## ¿Dónde se está midiendo?

Dichas mediciones se han llevado a cabo sobre un ordenador con las siguientes características:

* + - * Sistema operativo: Windows 10 20H2 x64
      * Entorno: Spyder (Phyton 3.8)
      * Procesador: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
      * Memoria RAM: 16.00 *GB*

## Unidades de tiempo

Los tiempos se han medido en *μs* usando la función de Python *perfcounter*, que devuelve el tiempo en segundos, multiplicada por 10e6.

# Medición de tiempos y análisis de la complejidad

A continuación, se presentan los resultados obtenidos para los algoritmos a estudiar (tiempos de ejecución, cota ajustada, ligeramente subestimada y sobrestimada). La medición de tiempos pequeños está automatizada en el código, basándose en un umbral de confianza de 1000 *μs* (por defecto).

# - Algoritmo ‘isMixtureDP’:

Se han repetido varias veces las mediciones, siendo estas las mejores series. Se acompañan de una cota ligeramente subestimada (f(n) = n), una cota ajustada (g(n) = n2) y una cota ligeramente sobrestimada (h(n) = n2.5). Las cotas son las mismas para los 3 escenarios.

La cota ligeramente subestimada f(n) crece asintóticamente más despacio que el tiempo de ejecución t(n), resultando en una tendencia de t(n) / f(n) a ∞ (diverge). La cota ajustada g(n) representa la tasa de crecimiento de t(n), por lo que tiende a una constante, y la cota sobrestimada crece asintóticamente más rápido, llevando a t(n)/ h(n) a converger a 0.

# Escenario 1: vocabulario de dos elementos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **t(n)** | **t(n) / f(n)** | **t(n) / g(n)** | **t(n) / h(n)** |
| 20 | 1651.90000 | 82.59500 | 4.12975 | 0.92344 |
| 40 | 7010.80000 | 175.27000 | 4.38175 | 0.69282 |
| 80 | 21976.30000 | 274.70375 | 3.43380 | 0.38391 |
| 160 | 82543.70000 | 515.89813 | 3.22436 | 0.25491 |
| 320 | 395398.2000 | 1235.61938 | 3.86131 | 0.21585 |
| 640 | 1619354.50000 | 2530.24141 | 3.95350 | 0.15628 |
| 1280 | 6247466.70000 | 4880.83336 | 3.81315 | 0.10658 |
| 2560 | 24473377.90000 | 9559.91324 | 3.73434 | 0.07381 |
|  | | **Cota subestimada** | **Cota ajustada**  Cte ∈ [03.22, 3.95] | **Cota sobre-estimada** |

* + 1. **Escenario 2: todos los caracteres del código ASCII**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **t(n)** | **t(n) / f(n)** | **t(n) / g(n)** | **t(n) / h(n)** |
| 20 | 1141.00000 | 57.05000 | 2. 85250 | 0.63784 |
| 40 | 4528.20000 | 113.20500 | 2. 83013 | 0.44748 |
| 80 | 14814.20000 | 185.17750 | 2. 31472 | 0.25879 |
| 160 | 67927.20000 | 424.54500 | 2. 65341 | 0.20977 |
| 320 | 250358.90000 | 782.37156 | 2. 44491 | 0.13667 |
| 640 | 948453.10000 | 1481.95797 | 2. 31556 | 0.09153 |
| 1280 | 3571821.40000 | 2790.48547 | 2. 18007 | 0.06093 |
| 2560 | 14283539.00000 | 5579.50742 | 2.17950 | 0.04308 |
|  | | **Cota subestimada** | **Cota ajustada**  Cte ∈ [2.18, 2.85] | **Cota sobre-estimada** |

* + 1. **Escenario 3: mismo tamaño que los strings iniciales**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **t(n)** | **t(n) / f(n)** | **t(n) / g(n)** | **t(n) / h(n)** |
| 20 | 1016.20000 | 50.81000 | 2.54050 | 0.56807 |
| 40 | 3055.60000 | 76.39000 | 1.90975 | 0.30196 |
| 80 | 13759.60000 | 171.99500 | 2.14994 | 0.24037 |
| 160 | 58886.20000 | 368.03875 | 2.30024 | 0.18185 |
| 320 | 224968.60000 | 703.02688 | 2.19696 | 0.12281 |
| 640 | 916995.80000 | 1432.80594 | 2.23876 | 0.08849 |
| 1280 | 3579685.20000 | 2796.62906 | 2.18487 | 0.06107 |
| 2560 | 14531128.50000 | 5676.22207 | 2.21727 | 0.04382 |
|  | | **Cota subestimada** | **Cota ajustada**  Cte ∈ [1.91, 2.54] | **Cota sobre-estimada** |

# Algoritmo ‘isMixtureCX’:

Se han repetido varias veces las mediciones, siendo estas las mejores series. Se acompañan de una cota ligeramente subestimada (f(n) = log(n)), una cota ajustada (g(n) = n) y una cota ligeramente sobrestimada (h(n) = n1.5). Las cotas son las mismas para los 3 escenarios, sin embargo, no ajusta igual de bien todos los escenarios ya que al algoritmo le lleva un tiempo distinto ejecutarse para cada escenario. Non obstante, como la complejidad no varía, las cotas permanecen iguales para todos los escenarios.

La cota ligeramente subestimada f(n) crece asintóticamente más despacio que el tiempo de ejecución t(n), resultando en una tendencia de t(n) / f(n) a ∞ (diverge). La cota ajustada g(n) representa la tasa de crecimiento de t(n), por lo que tiende a una constante, y la cota sobrestimada crece asintóticamente más rápido, llevando a t(n)/ h(n) a converger a 0.

# Escenario 1: vocabulario de dos palabras

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **t(n)** | **t(n) / f(n)** | **t(n) / g(n)** | **t(n) / h(n)** |
| 20 | 153.86390 | 51.36103 | 7.69319 | 1.72025 |
| 40 | 289.76650 | 78.55136 | 7.24416 | 1.14540 |
| 80 | 644.95490 | 147.18188 | 8.06194 | 0.90135 |
| 160 | 1262.43970 | 248.74807 | 7.89025 | 0.62378 |
| 320 | 2500.30000 | 433.45369 | 7.81344 | 0.30032 |
| 640 | 4862.50000 | 752.53795 | 7.59766 | 0.30032 |
| 1280 | 12510.50000 | 1748.59156 | 9.77383 | 0.27319 |
| 2560 | 17789.90000 | 2266.87542 | 6.94918 | 0.13735 |
|  | | **Cota subestimada** | **Cota ajustada**  Cte ∈ [6.95,8.06] | **Cota sobre-estimada** |

# Escenario 2: todos los caracteres del código ASCII

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **t(n)** | **t(n) / f(n)** | **t(n) / g(n)** | **t(n) / h(n)** |
| 20 | 76.72820 | 25.61250 | 3.83641 | 0.85785 |
| 40 | 170.31990 | 46.17118 | 4.25800 | 0.67325 |
| 80 | 312.85020 | 71.39395 | 3.91063 | 0.43722 |
| 160 | 750.27950 | 147.83326 | 4.68925 | 0.37072 |
| 320 | 1063.70000 | 184.40375 | 3.32406 | 0.18582 |
| 640 | 1823.60000 | 282.22688 | 2.84937 | 0.11263 |
| 1280 | 8130.90000 | 1136.45522 | 6.35227 | 0.17755 |
| 2560 | 10567.20000 | 1346.52392 | 4.12781 | 0.08158 |
|  | | **Cota subestimada** | **Cota ajustada**  Cte ∈ [3.84, 4.68] | **Cota sobre-estimada** |

# Escenario 3: mismo tamaño que los strings iniciales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **t(n)** | **t(n) / f(n)** | **t(n) / g(n)** | **t(n) / h(n)** |
| 20 | 76.72820 | 25.61250 | 8.88961 | 1.98778 |
| 40 | 170.31990 | 46.17118 | 3.40095 | 0.53774 |
| 80 | 312.85020 | 71.39395 | 6.69970 | 0.74905 |
| 160 | 750.27950 | 147.83326 | 3.97447 | 0.31421 |
| 320 | 1063.70000 | 184.40375 | 6.26250 | 0.35008 |
| 640 | 1823.60000 | 282.22688 | 5.09891 | 0.20155 |
| 1280 | 8130.90000 | 1136.45522 | 8.45375 | 0.23629 |
| 2560 | 10567.20000 | 2829.64729 | 8.67438 | 0.17144 |
|  | | **Cota subestimada** | **Cota ajustada**  Cte ∈ [3.4, 8.88] | **Cota sobre-estimada** |

# Conclusiones

Una vez obtenidos los resultados y medida la complejidad a partir de los tiempos de ejecución, podemos concluir lo siguiente:

El algoritmo 1 está mejor ajustado por su cota que el segundo, ya que la constante a las que tiende está acotada en un intervalo menor, independientemente del escenario aplicado. En el segundo algoritmo, debido a que los tiempos de ejecución son muy diferentes entre los distintos escenarios pero la complejidad no varía, las constantes a las que tiende el algoritmo en cada escenario oscilan entre unos valores mayores.

Comparando ambas cotas, vemos que la primera de ellas es el cuadrado de la segunda (n2 en el primer caso por n en el segundo), por lo que también se puede afirmar que la complejidad del segundo, pese a estar peor ajustado, es menor.