**Trabajo Practico N°1:**

**Problema 2**

Alumno: Moyano Andrés Martín

Patrón: 110017

Profesor: Echevarría Pablo

Jefe de Trabajos Prácticos: Marín Germán

**INDICE:**

1. Supuestos Pág. 2
2. Diseño Pág. 2
3. Análisis Pág. 4
4. Seguimiento Pág. 7
5. Tiempos de ejecución Pág. 7
6. Informe de resultados Pág. 10
7. Conclusión Pág. 12
8. **Supuestos:**
   1. El Algoritmo solamente admite una lista de tuplas con el formato (identificador, kilometro). El kilometraje admitido son de números mayores o iguales a 0.
   2. No se considera la posibilidad de números negativos o valores no numéricos.
   3. La carrera se ordenará por distancia. Se ordena antes de cualquier análisis de las postas y se realiza incluso si la carrera estuviera ordenada de ese modo antes de recibida como entrada. Esto se realiza para garantizar el correcto funcionamiento del algoritmo.
   4. La distancia entre cualquier par de postas consecutivas es como máximo 7 km.
   5. El algoritmo asume que **la última posta cubre el final de la carrera y que el primer tramo se asume que Juan puede recorrer hasta 7 km sin reponer agua**. No contempla un destino final fuera de la última posta.
   6. Supuesto que no sé si es necesario: Se asume que todas las carreras son realizables por Juan Curuchet (postas cada 7 o menos kilómetros)
9. **Diseño:**
   1. Se creó un algoritmo greedy que resuelve el problema de Juan Curuchet utilizando la siguiente regla:

**- Si entre la primera posta agregada y la primera posta hay más de 7 kilómetros se agrega esta última a la lista garantizando que Juan pueda recorrer en los primeros kilómetros de la carrera.**  
**- Si la posta n está a más de 7 kilómetros de la última posta de agua visitada se debe visitar la posta n - 1**.

**- Si la última posta visitada está a más de 7 kilómetros de la última posta, se agregará también la mencionada a la lista y así poder finalizar la carrera**

* 1. Para el retorno de la función se utiliza una lista que almacena las tuplas originales de las postas que se deben visitar las cuales siguen conservando el formato: (identificador, kilometro) almacenado en la lista original, la cual fue ordenada para garantizar el orden por kilometraje.
  2. **PseudoCodigo:**

def postas\_hitratacion(mapa\_carrera):

    mapa\_carrera.sort(key=lambda posta: posta[1])

    postas\_hitratacion = []

    ultima\_posta = 0

    cantidad\_postas = len(mapa\_carrera)

    for i in range(1, cantidad\_postas):

        distancia\_actual = mapa\_carrera[i][1] - mapa\_carrera[ultima\_posta][1]

        if distancia\_actual > 7:

            if ultima\_posta == 0 and mapa\_carrera[i - 1][1] > 7:

                postas\_hitratacion.append(mapa\_carrera[ultima\_posta])

            postas\_hitratacion.append(mapa\_carrera[i - 1])

            ultima\_posta = i - 1

    if mapa\_carrera[cantidad\_postas - 1][1] - mapa\_carrera[ultima\_posta][1] > 7:

        postas\_hitratacion.append(mapa\_carrera[-1])

    return postas\_hitratacion

* 1. **¿Es Óptimo?**

Si es óptimo, dado a que:

En cada paso, se recarga **lo más lejos posible de la última recarga de agua pero** sin correr el riesgo de quedarse sin agua entre postas. Eso asegura que Juan **maximiza el tramo recorrido entre recargas**, y por lo tanto necesita **menos paradas**.

En este contexto, algoritmo greedy puede obtener la solución óptima global porque:

- La carrera es lineal, sin retrocesos o caminos alternativos.

- Las características del problema sigue una regla simple abarcable por un algoritmo greedy; la única limitación es la distancia entre postas.

**- Una vez que se toma una decisión local, no se perjudican decisiones futuras.**

* 1. **Demostración de optimalidad:**

Se demostrará por absurdo que el algoritmo entregado encuentra la solución óptima global:  
**La Solución obtenida contiene postas innecesarias:**

Sea la carrera con postas y la solución proporcionada con y la solución con .

Dado que el algoritmo **solo agrega una posta a la solución cuando es estrictamente necesario**, es decir, justo antes de que la distancia supere los 7 km máximos sin agua implicaría que en la solución S' hay un tramo de 7 km sin cubrir, lo que provocaría que Juan no pueda completar la carrera. Si existiese una solución con menos postas inevitablemente sería una solución donde Juan no puede completar la carrera violando la restricción fundamental del problema. Por lo tanto, la solución S' no es válida, concluyendo que no puede haber una mejor que la encontrada por el algoritmo dado a que llevaría al absurdo.

1. **Análisis:**

Se asume que el ordenamiento de las postas en la carrera tiene complejidad de dado a que Python utiliza el algoritmo de ordenamiento TimSort que en promedio cuenta con esa complejidad.

A cada elemento de la lista de postas se lo consulta en promedio 1 vez con una complejidad de por cada una de estas operaciones. Dado a que se realizan n de estas consultas la complejidad dentro del for es de .

La complejidad temporal resultante es una suma de estos procesos, ninguno es despreciable:

1. **Seguimiento, una carrera con 10 postas:**

Se generó una carrera aleatoria con el generador de carreras proporcionado en la entrega de este TP1, la misma es una versión reducida de 10 postas para una mejor comprensión sin explicaciones redundantes:

Carrera = [('Posta 1', 5), ('Posta 2', 7), ('Posta 3', 11), ('Posta 4', 14), ('Posta 5', 17), ('Posta 6', 24), ('Posta 7', 26), ('Posta 8', 28), ('Posta 9', 30), ('Posta 10', 35)]

* Se analiza la distancia entre Posta 2 y la Posta 1.

No se agregó ninguna posta a la solución. 2 < 7

* Se analiza la distancia entre Posta 3 y la Posta 2.

No se agregó ninguna posta a la solución. 6 < 7

* Se analiza la distancia entre Posta 4 y la Posta 3.

La distancia entre la primera Posta y la primera agregada a la solución es mayor a 7 por lo que se la agregara para garantizar que Juan pueda terminar la carrera.

Se agrega la Posta 3 que es la última comprobada que esta una distancia de 7 o menos de la última agregada Posta 1.

* Se analiza la distancia entre Posta 5 y la Posta 4.

No se agregó ninguna posta a la solución. 6 < 7

* Se analiza la distancia entre Posta 6 y la Posta 5.

Se agrega la Posta 5 que es la última comprobada que esta una distancia de 7 o menos de la última agregada: Posta 3.

* Se analiza la distancia entre Posta 7 y la Posta 6.

Se agrega la Posta 6 que es la última comprobada que esta una distancia de 7 o menos de la última agregada: Posta 5.

* Se analiza la distancia entre Posta 8 y la Posta 7.

No se agregó ninguna Posta a la solución. 4 < 7

* Se analiza la distancia entre Posta 9 y la Posta 8.

No se agregó ninguna Posta a la solución. 6 < 7

* Se analiza la distancia entre Posta 10 y la Posta 9.

Se agrega la Posta 9 que es la última comprobada que esta una distancia de 7 o menos de la última agregada 6.

* No se agrega la última Posta porque con la última agregada es suficiente para terminar la carrera.

Resultado: [('Posta 1', 5), ('Posta 3', 11), ('Posta 5', 17), ('Posta 6', 24), ('Posta 9', 30)].

1. **Tiempos de ejecución:**
   1. **Seguimiento de tiempos:**

Método de seguimiento

Se realizaron **5 series de 5 ejecuciones** del algoritmo para valores de que muestren el comportamiento para distintas longitudes de carreras (100, 200, 300, 400, 500).  
En cada serie, se ejecutó el algoritmo **5 veces por cada valor de** , y posteriormente se calculó el promedio de los tiempos obtenidos en cada caso. Los resultados fueron redondeados a 5 cifras significativas (promedios redondeados posterior al cálculo para no perder calidad en la medición).

Entre cada serie existe un intervalo de 1 a 2 minutos.

Se entrega un promedio final para cada .

Las ejecuciones se realizaron en computador con un

Intel Core i5-7200U CPU 2.50GHz 2 Núcleos, Procesadores lógicos 4 y 8 GB de RAM

* + 1. Mediciones con n = 10:

|  |  |  |
| --- | --- | --- |
| Serie | Valores (s) | Promedio (s) |
| 1 | 0.0010319, 0.00099707, 0.00099802, 0.0019946, 0.00099754 | 0.0012038 |
| 2 | 0.00099754, 0.00099921, 0.0010173, 0.0019949, 0.00099754 | 0.0012013 |
| 3 | 0.00039697, 0.00099826, 0.0, 0.00099564, 0.0010765 | 0.00069346 |
| 4 | 0.001029491, 0.00099826, 0.0010560, 0.00099778, 0.0021944 | 0.0012552 |
| 5 | 0.0010188, 0.00099850, 0.00099730, 0.00099707, 0.0020947 | 0.0012213 |
| Final | - | 0.0011150 |

* + 1. Mediciones con n = 20:

|  |  |  |
| --- | --- | --- |
| Serie | Valores (s) | Promedio (s) |
| 1 | 0.0046291, 0.0029812, 0.0070539, 0.0032499, 0.0066805 | 0.0049189 |
| 2 | 0.0033746, 0.0086150, 0.0016527, 0.0050554, 0.0048153 | 0.0047026 |
| 3 | 0.0031967, 0.0021510, 0.0037014, 0.0035973, 0.0072021 | 0.0039697 |
| 4 | 0.0045125, 0.0073783, 0.0019944, 0.0038679, 0.0070126 | 0.0087356 |
| 5 | 0.0018582, 0.0019758, 0.0055623, 0.0023048, 0.0033798 | 0.0030162 |
| Final | - | 0.0050686 |

* + 1. Mediciones con n = 30:

|  |  |  |
| --- | --- | --- |
| Serie | Valores (s) | Promedio (s) |
| 1 | 0.0036464, 0.0047038, 0.0068932, 0.0034564, 0.0088539 | 0.0055107 |
| 2 | 0.0034463, 0.0026920, 0.0064046, 0.0048625, 0.0026751 | 0.0040161 |
| 3 | 0.0031176, 0.0096705, 0.0037587, 0.0029581, 0.0048871 | 0.0048784 |
| 4 | 0.0037551, 0.0024540, 0.0083053, 0.0035586, 0.0030792 | 0.0042305 |
| 5 | 0.0020511, 0.0056889, 0.0082655, 0.0043428, 0.0079732 | 0.0056643 |
| Final | - | 0.0048600 |

* + 1. Mediciones con n = 40:

|  |  |  |
| --- | --- | --- |
| Serie | Valores (s) | Promedio (s) |
| 1 | 0.0055568, 0.017852, 0.0083337, 0.0087528, 0.0084298 | 0.0097900 |
| 2 | 0.0092516, 0.0069830, 0.011478, 0.0064156, 0.0065401 | 0.0081336 |
| 3 | 0.0046232, 0.0085335, 0.0049775, 0.0090234, 0.018895 | 0.0092104 |
| 4 | 0.0099382, 0.0050633, 0.011515, 0.0064349, 0.010726 | 0.0087356 |
| 5 | 0.0069132, 0.010814, 0.0062304, 0.0079052, 0.0053067 | 0.0074338 |
| Final | - | 0.0086607 |

* + 1. Mediciones con n = 50:

|  |  |  |
| --- | --- | --- |
| Serie | Valores (s) | Promedio (s) |
| 1 | 0.0097153, 0.010168, 0.010743, 0.0091324, 0.011659 | 0.010284 |
| 2 | 0.0053484, 0.013424, 0.011081, 0.0078700, 0.0065496 | 0.0088547 |
| 3 | 0.020819, 0.018085, 0.012109, 0.0093474, 0.0056440 | 0.013201 |
| 4 | 0.012436, 0.0076582, 0.011702, 0.013659, 0.0085180 | 0.0107954 |
| 5 | 0.011531, 0.010084, 0.0080488, 0.0077024, 0.0081491 | 0.0091031 |
| Final | - | 0,010448 |

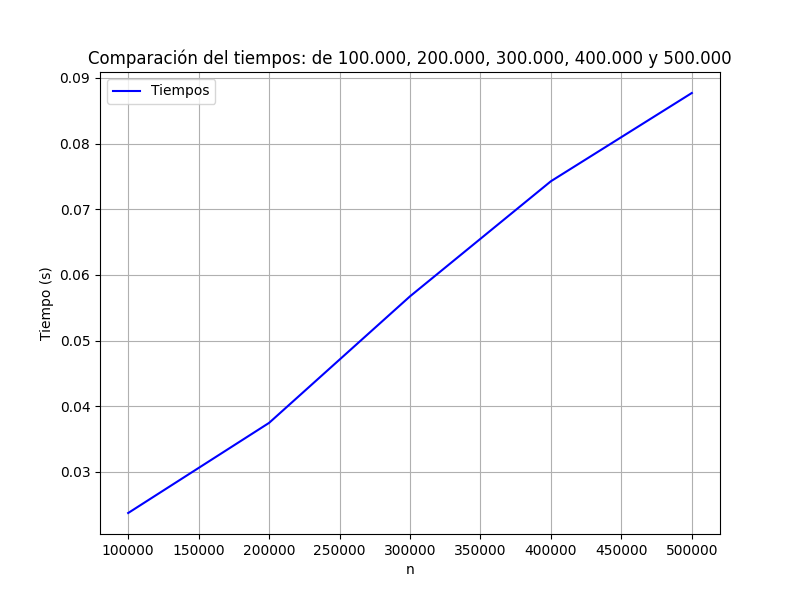


Fig. 1: Evolución del crecimiento de los resultados obtenidos con los n propuestos

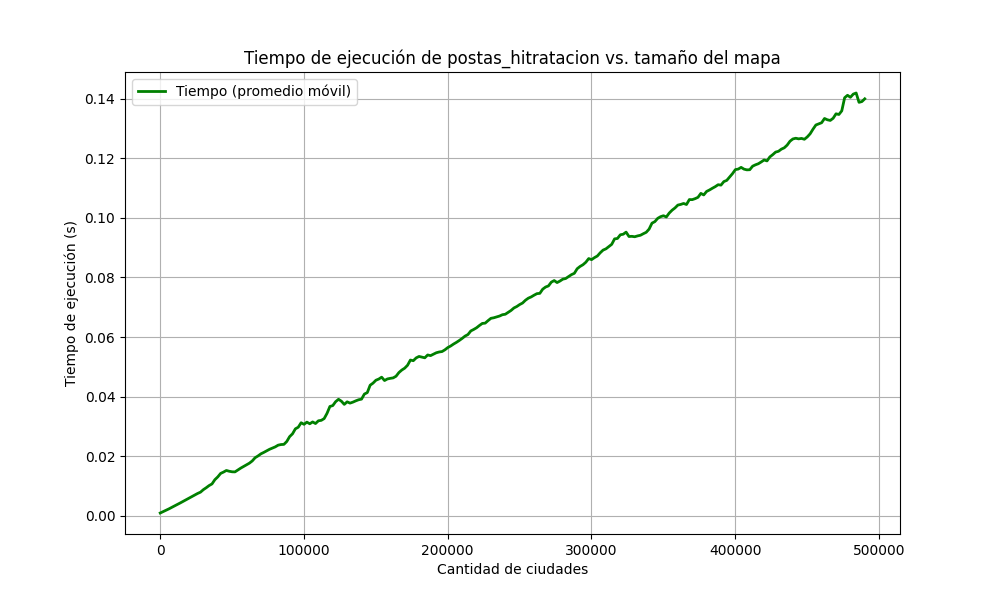


Fig. 2: Tiempos de ejecución promediados para n entre 100 y 500.000

1. **Informe de resultados:**

La función postas\_hitratacion presenta un comportamiento de tiempos de ejecución acorde a su complejidad esperada de Esta complejidad se deriva del uso de un algoritmo de ordenamiento (típicamente Timsort en Python, que opera en seguido de un recorrido lineal para seleccionar las postas de hidratación.

Los resultados reflejan un crecimiento general en el tiempo de ejecución con respecto a :

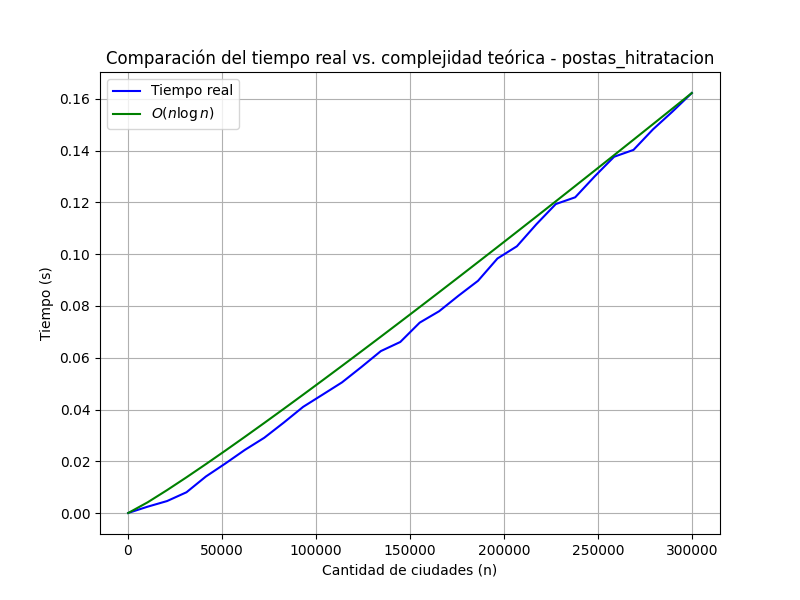


Fig. 3 Tiempos teóricos vs. Tiempos obtenidos

Se probaron 30 n valores, desde 100 hasta 500000, para cada uno se ejecutó el algoritmo y se guardó el tiempo que le tomo procesar los n números. Posteriormente se superpone la curva obtenida con la evolución de los tiempos reales y la complejidad temporal esperada. Los resultados varían casi imperceptiblemente a la curva esperada comprobando que el algoritmo se ajusta a la complejidad determinada inicialmente en el **análisis**.

Se utilizó un computador con un AMD Ryzen 5 5600H con gráficos Radeon y 8 GB de RAM para la generación así evitando ruido en los resultados, no habiendo una desviación significativa en algunos valores de n.

La causa del ruido puede deberse al uso de caché, ejecución en paralelo de otros procesos en el sistema, etc. A continuación se mostrará un gráfico de una métrica de error que posee mucho ruido y se aleja significativamente de la curva esperada. La grafica fue hecha en el computador utilizado para las métricas de **Tiempos de ejecución**, esta cuenta con menos núcleos, así como una menor cantidad de caché y de menor velocidad, lo que afecta al resultado:

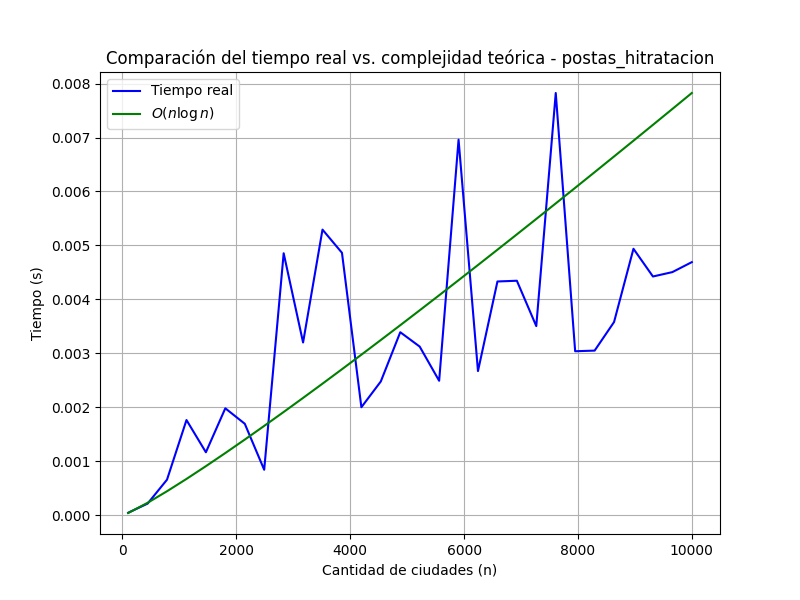


Fig. 4 Tiempos teóricos vs Tiempos obtenidos en Intel Core i5 gen 7

1. **Conclusión:**

La lógica de los algoritmos greedy de ajusta perfectamente al problema de la carrera de Juan Curuchet lo que provoca que se obtenga la solución óptima. Esto demuestra que los algoritmos greedy son una herramienta confiable cuando sus limitaciones de ajustan al propio problema, dando una solución válida en poco tiempo aun con carreras de más de 500000 postas. La única limitación es saber identificar si la solución es efectivamente la óptima o solo una aproximación de la solución, lo que implica un mínimo de conocimiento de teoría de algoritmos.