TP1 - Algo III - Algoritmos Greedies

Martín Acuña - Manuel Horn

April 2023

1 Problema de selección de actividades

El problema de selección de actividades consiste en, dado un conjunto de actividades con un rango horario (tanto inicio como fin están acotados) para cada una, encontrar un subconjunto maximal de actividades que no se superpongan en el tiempo.

Se puede demostrar que este problema se puede resolver aplicando una estrategia que involucre un algoritmo "Greedy" con complejidad lineal.

El algoritmo es iterativo. La elección greedy en cada iteración va a ser buscar la actividad que termine lo mas antes posible y al mismo tiempo no se superponga con la última de las actividades ya seleccionadas (no necesariamente se va a encontrar siempre).

Se propone el siguiente algoritmo para resolver el problema con su explicación a continuación:

Algorithm 1 maximizar(actividades)

1:	bucketSort(actividades)		O(n)
2:	ultima = 0		O(1)
3:	res = []		O(1)
4:	**for** act in actividades **do**	n iteraciones de O(1):	O(n)
5:	**if** $act.princpio \geq ultima$ **then**		O(1)
6:	$ultima = act.final$		O(1)
7:	$res.agregar(act.indice)$		O(1)
8:	**end if**		
9:	**end for**		
10:	**return** (res, len(res))		O(1)

Dado un conjunto de intervalos de números naturales (actividades), se le asigna un índice a cada uno y se lo ordena crecientemente por horario de fin. Dado que los horarios de fin son enteros acotados, se ordenan los intervalos

usando "Bucket Sort" en O(n), siendo n la longitud del conjunto de intervalos (tamaño de entrada).

El algoritmo recibe una lista de actividades con sus respectivos horarios de inicio y fin. Primero se ordenan de acuerdo al criterio mencionado, esto facilita la elección Greedy que se hace en cada iteración, ya que solamente con un barrido lineal se puede encontrar dicha actividad.
La variable 'ultima' va actualizando el último horario de fin agregado al conjunto 'res', que inicialmente es una lista vacía. Se recorre el conjunto de inicio a fin, agregando una actividad cuando dicha es una elección Greedy, y eventualmente llega a la solución óptima al llegar al final del conjunto. Por último, el algoritmo devuelve un subconjunto óptimo con su tamaño.

2 Demostración de correctitud

Para demostrar que dicho algoritmo resuelve el problema encontrando una solución óptima, hay ver que la solución parcial obtenida en cada iteración es un subconjunto de una solución óptima del problema general.

Teorema:
Este algoritmo encuentra una solución óptima para el problema de selección de actividades.

Demostración:
Sea S_i la solución parcial en la iteración i ($0 \leq$ i $\leq$ n, con n la longitud del conjunto), hay que probar que para todo i en ese rango existe una solución óptima S_* que contiene a S_i. Esto se prueba por inducción.

Caso Base:
i=0 (antes de entrar al ciclo). $S_0 = \emptyset$, y como $\emptyset$ está incluido en todo conjunto, se cumple.

Hipótesis Inductiva: $\exists S_* tq. S_i \subseteq S_*$

Paso Inductivo: QVQ $\exists S' tq. S_{i+1} \subseteq S'$

Si la actividad considerada en la iteración i+1 no se agrega, es porque interseca con algún intervalo de S_i, $S_i = S_{i+1}$, y por H.I., $S_{i+1} \subseteq S_*$
En el caso contrario, sea a_k la actividad que el algoritmo eligió en la iteración i+1, a_k puede pertenecer o no a S_*:

Si $a_k \in S_*$:
$S_{i+1} = S_i \cup a_k \Longrightarrow$S$_{i+1} \subseteq S*$, entonces directamente se toma $S' = S_*$

Si $a_k \notin S_*$:

Hay que encontrar otra solución óptima S' tq. $(S_i \cup a_k) \subseteq S'$. Se puede hacer esto buscando una actividad $a_j \in S_*$ que puede ser sustituida por a_k. Para esto, se considera a a_j tq. $a_j \in (S_* - S_i)$ que termine antes que todas en ese conjunto y sea $S' = (S_* - a_j) \cup a_k$, hay que demostrar que S' es una solución óptima válida.

Como el algoritmo eligió a_k en vez de a_j, esto implica que el fin de a_k es menor o igual que el de a_j y el principio de a_k es mayor o igual que el fin de la solución anterior.

Como $S_* = (S_i \cup a_j \cup R)$ (donde R es el conjunto de actividades de S_* cuyos inicios son todos mayores que el fin de a_j) y S_* es una solución válida, entonces $fin(S_i) \leq inicio(a_j)$ y $fin(a_j) \leq inicio(R)$, así se define $S' = (S_i \cup a_k \cup R)$. Por lo expuesto, S' es válida.

Por último, se puede ver que $|S'| = |S_*|$, y por ser S_* una solución óptima, S' también lo es (porque ambas son máximas).

■

3 Complejidad temporal

A simple vista en el pseudocódigo, se puede ver que la complejidad temporal es O(n), con n la cantidad de actividades.

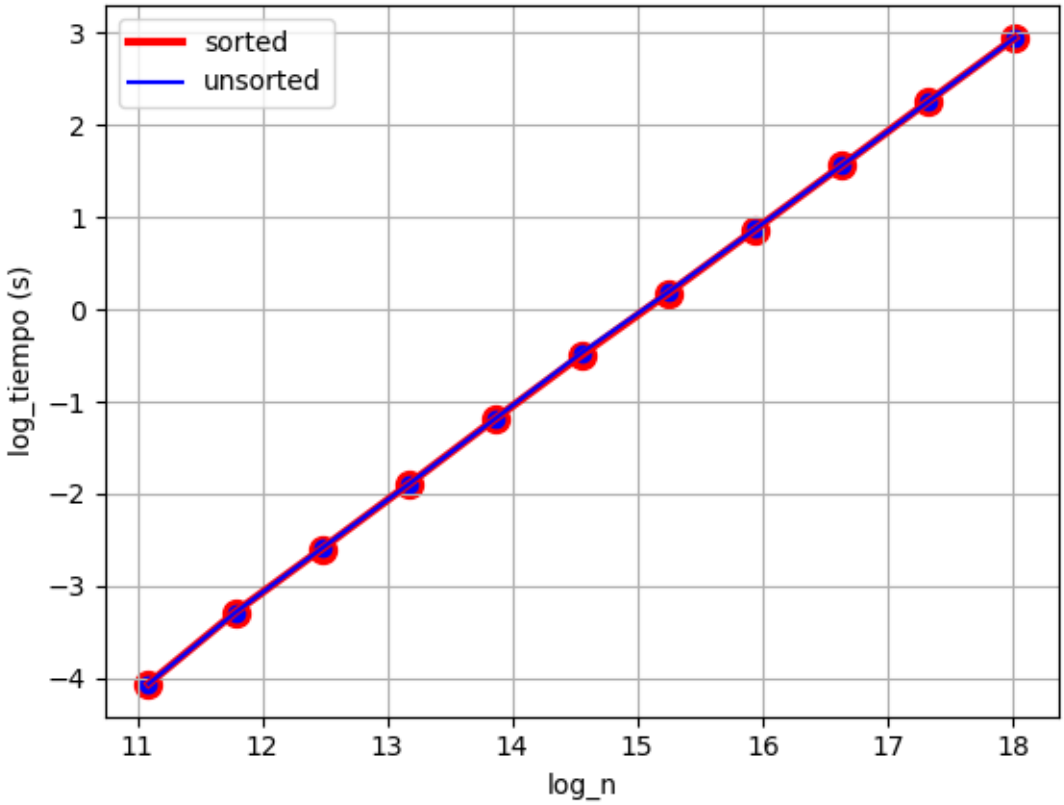
A continuación se justifica la complejidad temporal empíricamente exponiendo los resultados obtenidos de pasar el algoritmo por una serie de tests aleatorios de tamaños varios.

Para verificar el comportamiento asintótico del algoritmo, se generaron dos conjuntos de instancias. Uno de ellos con las actividades ordenadas según su horario de fin y el otro sin ordenar (siempre respetando la precondición de instancia válida).

Luego, se midieron los tiempos que tardó el algoritmo con cada instancia para ambos conjuntos.

Para cada conjunto se calculó el coeficiente de Pearson entre el tamaño de las instancias y sus respectivos tiempos y estos dieron muy cercanos a 1 (0.999941044634488 para las ordenadas y 0.9999230845546057 para las no ordenadas). Esto dice que la correlación entre las dos variables es prácticamente lineal.

Como las entradas generadas tienen una relación exponencial entre sus tamaños y la relación entre el tamaño y el tiempo es lineal, se ajustaron los datos a una escala logarítmica (para visualizar mejor).



Si en la escala lineal se cumple $time_{unsorted} = time_{sorted} * c$, entonces en la escala logarítmica se cumple $\log(time_{unsorted}) = \log(time_{sorted} * c) = \log(time_{sorted}) + \log(c)$, así que $\log(c)$ es la diferencia entre los tiempos. En el gráfico ambos tiempos parecen no diferir, para ver eso se calculó el valor de esa constante $\log(c)$. Podemos estimar este valor calculando la diferencia entre ambas rectas. El resulado de c fue c=1.003556895214288, que es aproximadamente 1, y por lo tanto $\log(c) \sim 0$. Así que la diferencia entre ambos tiempos es prácticamente nula.

4 Conclusiones

Dado que el algoritmo usa BucketSort, el ordenamiento no se basa en comparaciones entre actividades, sino en hacer un conteo para ellas. Entonces no discrimina entre las instancias ordenadas y las desordenadas. Por lo tanto, se concluye que no existe un conjunto de instancias en que mejore el tiempo de ejecución del algoritmo.