DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Técnicas Algorítmicas

23 de Abril de 2023 — Algoritmos y Estructuras de datos III

Integrante	LU	Correo electrónico
Andrés Finkelsteyn	511/01	andresgfin@gmail.com
Martín Santesteban	397/20	martin.p.santesteban@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - Pabellón I
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

Descripción del problema

Dado un conjunto de actividades $\mathcal{A} = \{A_1, A_2, ..., A_m\}$, el problema de selección de actividades consiste en encontrar un subconjunto de actividades $\mathcal{S}$ de cardinalidad máxima, tal que ningún par de actividades de $\mathcal{S}$ se solapen en el tiempo. Cada actividad A_i se realiza en algún intervalo de tiempo (s_i, t_i) siendo $s_i \in \mathbb{N}$ su momento inicial y $t_i \in \mathbb{N}$ su momento final. Suponemos que $1 \leq s_i < t_i \leq 2n$ para todo $1 \leq i \leq n$.

Descripción del algoritmo

El algoritmo consta de dos partes, la primera consiste en ordenar utilizando una versión modificada de Counting Sort y la segunda en realizar la selección golosa de actividades.

El algoritmo de Counting Sort modificado ordena las actividades por su tiempo de finalización aprovechando el hecho de que ese tiempo se encuentra acotado en este problema. Para ello en primer lugar se guardan las actividades en listas dentro de un vector. Cada una de estas listas se forma con actividades que tienen el mismo tiempo de finalización y se encuentra en la componente del vector indexada por ese valor de tiempo. Luego se recorre ordenadamente el vector, copiando las actividades de las listas a un nuevo vector que es usado en la segunda parte.

El algoritmo de la selección golosa consiste en recorrer el vector de actividades ordenadas por Counting Sort y agregar aquellas que tienen un tiempo de inicio mayor que el máximo tiempo de finalización registrado hasta el momento entre las actividades ya elegidas.

Demostración de correctitud

Queremos demostrar que la solución obtenida a partir de la selección golosa de m actividades es una solución óptima. Para ello definiremos a G como el conjunto de los intervalos correspondientes a las m actividades de la solución golosa:

$$G = \{(s_{g_1}, t_{g_1}), ..., (s_{g_m}, t_{g_m})\}$$

y demostraremos primero la proposición enunciada a continuación.

Proposición: Dado $S = \{(s_{s_1}, t_{s_1}), ..., (s_{s_r}, t_{s_r})\}$ el conjunto de intervalos provenientes de una solución óptima, si se definen los conjuntos σ_i y γ_i de la siguiente forma:

$$\sigma_i = \left\{(s_{s_j}, t_{s_j}) \in S : s_{s_j} < t_{g_i}\right\}$$

$$\gamma_i = \{(s_{g_k}, t_{g_k}) \in G : s_{g_k} < t_{g_i}\}$$

entonces se cumple la desigualdad: $\#\sigma_i \leq \#\gamma_i$

Se puede notar que σ_i es el subconjunto de S compuesto por todos los intervalos correspondientes a las actividades que terminan antes del tiempo t_{g_i} y γ_i es el subconjunto de G compuesto por todos los intervalos que cumplen con esa misma misma propiedad.

Demostración: Se realizará por inducción.

Caso base: $\#\sigma_1 \leq \#\gamma_1$.
Por construcción $\#\gamma_1 = 1$.
Veamos que el cardinal $\#\sigma_1$ no puede ser mayor que 1. En caso de existir dos intervalos pertenecientes a σ_1, podemos afirmar que uno de ellos, que denotamos (s_{s_j}, t_{s_j}), debería cumplir que $t_{s_j} < t_{g_1}$ para no solaparse con el otro intervalo. Por lo tanto ese intervalo habría sido elegido en la selección golosa en lugar de (s_{g_1}, t_{g_1}).

Paso inductivo:
HI: $\#\sigma_i \leq \#\gamma_i$
Quiero ver que $\#\sigma_{i+1} \leq \#\gamma_{i+1}$

Definimos los siguientes conjuntos:

$$A_{\sigma_i} = \left\{(s_{s_j}, t_{s_j}) \in S : t_{g_i} \leq s_{s_j} < t_{g_{i+1}}\right\}$$

$$A_{\gamma_i} = \left\{(s_{g_k}, t_{g_k}) \in G : t_{g_i} \leq s_{g_k} < t_{g_{i+1}}\right\}$$

Se puede notar que A_{σ_i} es el subconjunto de S compuesto por todos los intervalos correspondientes a las actividades que terminan después del tiempo t_{g_i} y antes del tiempo $t_{g_{i+1}}$ mientras que A_{γ_i} es el subconjunto de G compuesto por todos los intervalos que cumplen con esa misma misma propiedad.

Por definición:

$$\sigma_{i+1} = \sigma_i \cup A_{\sigma_i} \wedge \sigma_i \cap A_{\sigma_i} = \emptyset \Rightarrow \#\sigma_{i+1} = \#\sigma_i + \#A_{\sigma_i}$$

$$\gamma_{i+1} = \gamma_i \cup A_{\gamma_i} \wedge \gamma_i \cap A_{\gamma_i} = \emptyset \Rightarrow \#\gamma_{i+1} = \#\gamma_i + \#A_{\gamma_i}$$

Por construcción $\#A_{\gamma_i} = 1$.
Veamos que el cardinal $\#A_{\sigma_i}$ no puede ser mayor que 1. En caso de existir dos intervalos pertenecientes a A_{σ_i}, podemos afirmar que uno de ellos, que denotamos (s_{s_j}, t_{s_j}), debería cumplir que $t_{s_j} < t_{g_{i+1}}$ para no solaparse con el otro intervalo. Por lo tanto ese intervalo habría sido elegido en la selección golosa en lugar de $(s_{g_{i+1}}, t_{g_{i+1}})$. Entonces:

$$\#A_{\sigma_i} \leq \#A_{\gamma_i}$$

Si a la desigualdad anterior le sumamos la desigualdad de la hipótesis inductiva:

$$\#A_{\sigma_i} + \#\sigma_i \leq \#A_{\gamma_i} + \#\gamma_i \Rightarrow \#\sigma_{i+1} \leq \#\gamma_{i+1}.$$

Así queda demostrado.

Como consecuencia de la proposición sabemos que:

$$\#\sigma_m \leq \#\gamma_m.$$

Vemos que $\gamma_m = G$ por definición.
Además $\sigma_m = S$ porque no existe un intervalo (s_{s_l}, t_{s_l}) perteneciente a S tal que $s_{s_l} \geq t_{g_m}$. Si existiera habría sido elegido en la solución golosa para ser parte del conjunto G a continuación del intervalo (s_{s_m}, t_{s_m}). Entonces:

$$\#S \leq \#G$$

Dado que S proviene de una solución óptima resulta que $\#S = \#G$, por lo cual queda demostrado que la solución golosa también es óptima.

Complejidad

Esta sección consiste en el cálculo de la complejidad teórica del algoritmo y en el estudio empírico de la misma. El estudio empírico incluye el análisis de conjuntos de instancias en las cuales el algoritmo requiere más tiempo de cómputo.

Complejidad teórica

La complejidad teórica del algoritmo se calcula como la suma de las complejidades de las dos partes que lo componen.

El algoritmo de Counting Sort modificado tiene la misma complejidad que la versión original porque solo difiere en una operación con la misma complejidad. Esa operación es el agregado de cada actividad a una lista mientras que en el algoritmo original es el incremento en uno del valor de un contador. Por lo tanto la primera parte del algoritmo tiene complejidad lineal.

La selección golosa de actividades consiste en iterar una única vez sobre el conjunto de actividades ya ordenadas por lo cual la segunda parte también tiene complejidad lineal. En conclusión la complejidad teórica del algoritmo es $O(n)$.

Estudio del tiempo en función de la cantidad de actividades

El objetivo de este estudio es analizar la relación funcional entre el tiempo de ejecución y el tamaño de la instancia, donde el tamaño denota la cantidad de actividades de la misma.

Hipótesis

Se espera ver una relación lineal entre el tiempo de ejecución y el tamaño de la instancia dado que la complejidad teórica es lineal.

Procedimiento

Se generaron 1000 instancias de forma aleatoria para cada tamaño de instancia, donde los tamaños son valores de la forma 2^i, para $i \in [10, 11, ..., 24]$ y se registraron sus tiempos de ejecución. A partir de los datos obtenidos se realizó una regresión lineal y se calculó el coeficiente de correlación.

Resultado

La figura 1 muestra una relación lineal entre el tiempo de ejecución y la cantidad de actividades. Se observa una dispersión apreciable de los puntos para los tamaños más chicos, pero esta disminuye a medida que los tamaños aumentan. Los tiempos de ejecución se mantienen dentro del intervalo $[0s, .., 16s]$. Los tamaños y los tiempos medidos presentan un coeficiente de correlación de 0,9973.

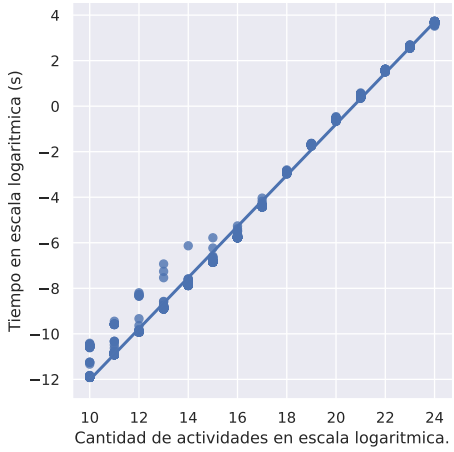


Figura 1: Regresión lineal de tiempo en función de la cantidad de actividades.

Estudio de rendimiento para distintas instancias

El objetivo de esta sección es el planteo de conjuntos de instancias que teóricamente requieren mayor tiempo de cómputo y el estudio empírico de estos conjuntos.

Hipótesis

Se propone que las instancias que incluyen un número más grande de actividades que no tienen ningún solapamiento con las demás requieren un tiempo más elevado de cómputo, asimismo aquellas instancias que presentan un número más bajo requieren un tiempo de cómputo más reducido. Para entender la causa hay que notar que las actividades que no tienen solapamientos son necesariamente incluidas en el vector que da como respuesta el algoritmo. Por lo tanto una instancia con un número alto de actividades sin solapamientos lleva a realizar más veces la operación push_back() en la implementación.

Es importante destacar que no es clara la importancia relativa de la operación push_back() dentro de la implementación y consecuentemente tampoco es clara la magnitud del efecto de un número alto de actividades sin solapamientos sobre el tiempo de cómputo. Recordando el algoritmo, esta operación aparece en la segunda parte cuando se realiza la selección golosa de actividades. Para ello se itera sobre las actividades ordenadas y se evalúa si una actividad tiene un tiempo de inicio mayor al último tiempo de finalización agregado al vector de respuesta. Si es el caso, se la agrega al vector mediante la operación mencionada y en el caso contrario no se realizan operaciones.

Procedimiento

Se diseñó un método para generar aleatoriamente instancias con un número variable de actividades que no se solapan con las demás. Para ello se fijó el número de actividades en $N = 2^{14}$ y se generaron 1000 instancias aleatorias para cada valor de $t \in \{2,3,...,20\}$, donde toda actividad A_i cumple que $|f_i - s_i| \leq t$.

La idea detrás del método desarrollado es que dentro de una instancia con N actividades que tienen un máximo tiempo de duración t bajo, la cantidad de solapamientos entre actividades debe ser baja también. Análogamente, cuanto más alto es el valor de t en una instancia, la cantidad de solapamientos también debe ser más alta. Implementando este método reiteradas veces, se vio que la cantidad de solapamientos en función de t parecería ser exponencial, y que para $t = 20$ casi todas las actividades se solapaban con alguna otra.

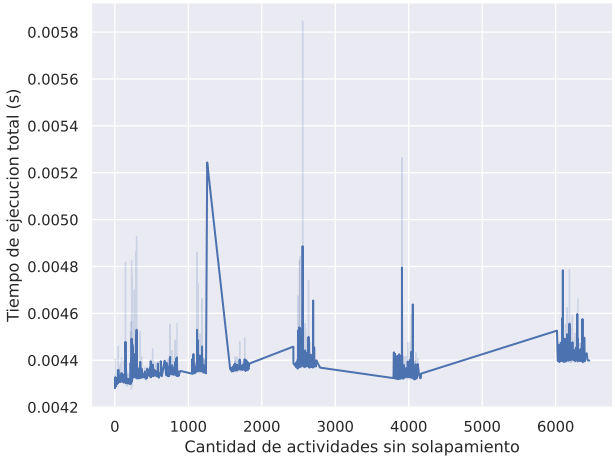
Luego de generar las instancias se ejecutó el algoritmo sobre todas ellas y se registraron sus respectivos tiempos de cómputo. Posteriormente se realizaron distintos gráficos motivados por las observaciones de los resultados.

Resultados

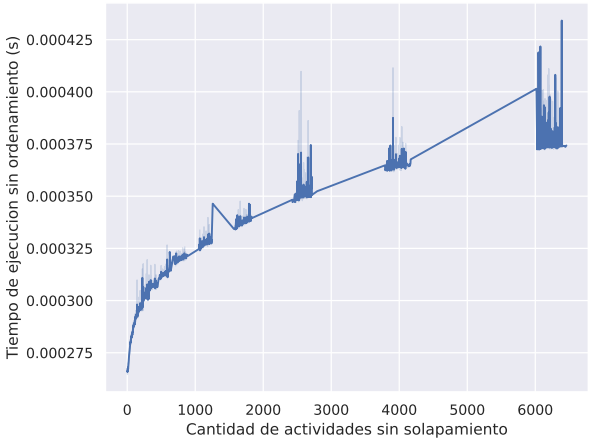
En la figura 2a se presenta el gráfico de los tiempos de cómputo en función de la cantidad de actividades sin solapamiento. Se ve que para $N = 2^{14}$ los tiempos siempre están dentro del intervalo [0.0042s, 0.0053s] y parecen

comportarse de forma más o menos constante. Estos resultados no fueron satisfactorios porque contradicen la hipótesis de que los tiempos varían en función de la cantidad de actividades sin solapamiento.

A continuación se exploró la idea de que el tiempo de ejecución de la selección golosa de actividades fuera despreciable frente al tiempo de ejecución del ordenamiento por Counting Sort modificado, haciendo que no se pueda apreciar el efecto de la variación del número de actividades sin solapamientos. Por lo tanto, se volvió a realizar el experimento pero midiendo solamente los tiempos de ejecución de la segunda parte del algoritmo. Los resultados se pueden ver en la figura 2b.



(a) Tiempo total de ejecución *vs* Cantidad de actividades no solapadas.



(b) Tiempo de ejecución *vs* Cantidad de actividades no solapadas.

La segunda experimentación muestra que los tiempos de cómputo de la segunda parte del algoritmo son siempre despreciables frente a los tiempos de cómputo de la primera parte, ya que se encuentran dentro del intervalo [0.000275s, 0.000425s]. Sin embargo permite apreciar que efectivamente los tiempos de ejecución aumentan en función de la cantidad de actividades sin solapamiento. Adicionalmente los tiempos de ejecución y la cantidad de actividades sin solapamiento parecen tener una relación logarítmica.

A partir de este resultado se consideró que la relación logarítmica podía deberse a la implementación del método push_back() de la librería estándar de $C++$. Esto motivó a estudiar los tiempos de ejecución en función de la cantidad de veces en las que se realiza la operación, cantidad que es equivalente al tamaño del vector respuesta. Los resultados, transformados por la función logaritmo, se pueden ver en la figura 3

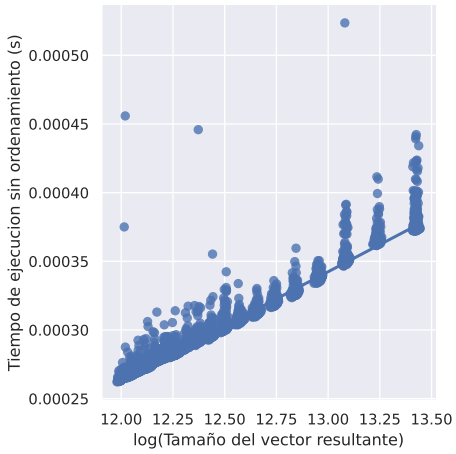


Figura 3: Regresion lineal del tiempo de ejecución en función del logaritmo del tamaño del vector resultado.

El gráfico muestra una relación lineal entre los tiempos de ejecución y la cantidad de actividades transformada por la función logaritmo, teniendo un coeficiente de correlación igual a 0,9925.

Finalmente se puede concluir que las instancias con más actividades sin solapamientos tienen un mayor tiempo de ejecución, presentando una relación logarítmica. Además se puede concluir que el ordenamiento de las actividades toma un tiempo significativamente mayor al de la selección de actividades.