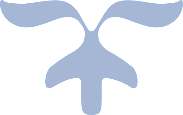


Práctica 2: algoritmos APROXIMADOS: selección de actividades PONDERADA

Algoritmos Avanzados. Grado en Ingeniería Informática.



10 de NOVIEMBRE de 2017

Jorge Aranda García

Jose vicente bañuls garcía

# **INTRODUCCIÓN**

El objetivo de esta práctica es permitir a los alumnos una profundización en los algoritmos aproximados, el porqué son necesarios, así como su comparación con los algoritmos voraces. El problema propuesto en esta ocasión es similar al propuesto en la práctica anterior, pero en este caso se ha incluido un array en el que se indica un beneficio que se obtendría si se llevase a cabo la actividad correspondiente con ese índice. Precisamente al incluir ese array de beneficios, la función de maximización consiste en realizar un número de actividades compatibles que proporcionen el mayor beneficio posible. Para ello se han propuesto dos algoritmos voraces, que serán comparados con tres de aproximación. Los cinco serán detallados a continuación.

# **ENTORNO DE DESARROLLO**

De nuevo, como sucedió en la anterior entrega, nos hemos decantado por Eclipse Java Neon y la versión utilizada de JDK ha sido JDK 1.8.0\_60. La alta familiarización que tenemos con esta herramienta, además del buen resultado que obtuvimos en la práctica 1, nos han hecho seguir utilizándola, a pesar de la necesidad de tener que crear una clase Main para ejecutar el código, algo que no es necesario en la herramienta BlueJ. Además, debido a la necesidad de trabajar en paralelo, bien en la edición, bien en la edición del código hemos decidido utilizar una herramienta de control de versiones. En este caso hemos utilizado GitHub para el repositorio y SourceTree como herramienta de visualización de la misma debido a su facilidad de uso, así como de la integración que tiene con Eclipse.

# **ALGORITMOS**

A continuación, pasarán a comentarse de forma detallada cada uno de los algoritmos, así como a justificar algunos conceptos de implementación.

## **Algoritmo Voraz 1. Función de selección: Coger el máximo beneficio**

El primer algoritmo propuesto tiene una función de selección que consiste en lo siguiente. Se coge la actividad que tenga mayor beneficio, siempre va a formar parte de la solución. A partir de aquí se intenta coger la siguiente actividad de máximo beneficio, si es compatible con las seleccionadas anteriormente será parte de la solución final. El beneficio de la actividad seleccionada se suma al beneficio parcial. Para ello, se ordena el array de beneficios mediante un array de índices. El método de ordenación escogido ha sido ShellShort (la justificación se encontrará en el anexo). Usamos el método de ordenación para poder ir directamente a por la actividad de mayor beneficio, sin tener que recorrer el array de una forma tal que nos obligase a comprobar si la actividad candidata era compatible y además de máximo beneficio. Cuando se han explorado todas las actividades se muestra el beneficio total. La justificación del uso de esta función de selección es que ha sido utilizada en la asignatura para la resolución de problemas similares, además de que al hablar de maximización lo primero que se viene a la cabeza es coger las actividades con mayor beneficio. También cabe destacar que, a diferencia de la práctica en la que se iban escogiendo las actividades de forma que solo había que comprobar si la actividad candidata se solapaba justamente con la anterior seleccionada, en este caso al no haber ningún tipo de orden de tiempo de las actividades, es necesario comprobar que no se solape con ninguna de las escogidas anteriormente. Para ello se implementa una función auxiliar que se encarga de ello. La cabecera de este método auxiliar es:

**private** **boolean** esCompatible(**int** [] c, **int** [] f, **boolean**[] sol, **int** act){…}

La cabecera del método que se corresponde con este algoritmo es:

***public******int*** *seleccionActividadesPonderadoVoraz (****int*** *[] c,* ***int*** *[] f,* ***int*** *[] b){…}*

Donde c, es el array de tiempos de comienzos, f, el array de tiempos de fin y b el array de beneficios de cada actividad.

Cabe destacar que la complejidad de este método depende directamente de los dos bucles anidados que hay. El primero es el que selecciona la actividad de mayor beneficio, el segundo, dentro de la función esCompatible(…), que comprueba que la actividad candidata sea compatible con todas las seleccionadas. Por lo tanto la complejidad es de O(n2).

## **Algoritmo Voraz 2. Función de selección: Coger la máxima tasa de beneficio/duración**

Para esta actividad la función de selección consiste en escoger la actividad que tenga mayor tasa de beneficio/duración. Estas tasas se almacenan en un array, en la que cada posición i del array almacena lo siguiente: b[i]/f[i]-c[i].

Este array se ordena como se ha hecho en el apartado anterior para no tener que comprobar si es el que tiene la mayor tasa, simplemente, se ordena, y se accede a la actividad deseada. Si ésta es compatible con todas las actividades seleccionadas anteriormente, o si es la primera, se selecciona y se almacena en el array de solución final. La justificación de esta función de selección consiste en ofrecer una visión distinta que no dependa solo del beneficio, sino que también se tenga en cuenta de alguna forma la duración.