

Doble Grado en Ingeniería Informática y Matemáticas

Algorítmica

**Práctica 5 - Programación dinámica**

David Kessler Martínez

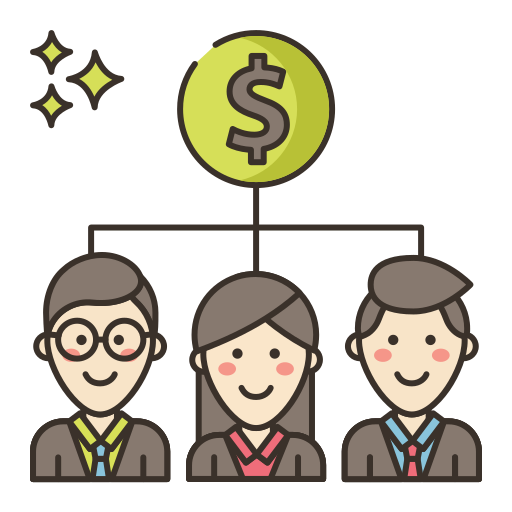
[dkesslerm03@correo.ugr.es](mailto:dkesslerm03@correo.ugr.es)

Santiago López Cerro

[santiloce@correo.ugr.es](mailto:santiloce@correo.ugr.es)

Antonio Javier Rodríguez Romero

[antoniojrr@correo.ugr.es](mailto:antoniojrr@correo.ugr.es)



Granada, curso 2022/2023

índice

**0. Introducción** 3

**0.1. Objetivos** 3

**0.2. Autores** 3

**1. P3: Problema de asignación de recursos** 4

**1.1. Definición del problema** 4

**1.2. Algoritmo basado en P.D** 4

[**1.2.1. Descripción del algoritmo**](#_td802qfouubu) **4**

[**1.2.2. Comprobación del Principio de Optimalidad de Bellman**](#_dkb8f6qqpqdz) **5**

**1.3. Conclusiones** 6

0. introducción

0.1. Objetivos

Con esta práctica lo que se pretende es resaltar la utilidad de las técnicas de resolución de problemas “Programación Dinámica”. Para ello trataremos de diseñar un algoritmo que implemente dicha técnica y viendo una vez que cumple el Principio de Optimalidad de Bellman podremos determinar qué se ha obtenido la mejor solución posible al problema planteado.

0.2. Autores

David Kessler Martínez - 33%

Antonio Javier Rodríguez Romero - 33%

Santiago López Cerro - 33%

Esta práctica ha sido realizada de forma conjunta, siendo por ello mismo por lo que hemos asignado de forma equitativa los porcentajes de la carga de trabajo. Para la búsqueda del algoritmo todos hemos ido poniendo ideas en común y respecto a la documentación del trabajo, cada uno ha ido desarrollando diversas partes hasta finalmente llegar al resultado final.

1. P3: Problema de asignación de recursos
   1. Definición del problema

Nuestro problema de asignación de recursos nos presenta una situación en la que tenemos **r unidades de un recurso** (p.ej. monetarias, personal, maquinaria) que deben asignarse a ***n* proyectos**. Las unidades de un recurso que se asignan a cada proyecto, **j**, deben estar entre 0 y r . Cada proyecto nos proporcionará un beneficio en función de las unidades de recursos que se le asignen. La solución a este problema es encontrar la asignación de recursos a proyectos que **maximice** el beneficio total.

Esta asignación será almacenada en un vector, X, donde la posición i-ésima del vector será la fracción de r que se le asignará al proyecto i-ésimo. Las componentes de este vector serán decididas una vez tengamos el beneficio máximo de la asignación.

Este beneficio total, (donde j es la fracción de recursos asignada al trabajo *i*) será almacenado en una matriz de orden n\*r. Supondremos que el beneficiado asociado a asignar 0 unidades de recursos a un proyecto será 0.

* 1. Algoritmo basado en Programación Dinámica

# 1.2.1. Descripción del algoritmo

Para la resolución del ejercicio, una de las mejores opciones que podemos tomar a la hora de diseñar un algoritmo es utilizar la técnica de la Programación Dinámica, consiguiendo una alta eficiencia en ejecución, aunque hagamos un mayor uso de memoria.

Esta se basará en la descomposición del problema en subcasos más pequeños para ir componiendo poco a poco la solución final.

La idea será representar nuestro problema como una dupla ***(n,r)***, donde **n** serán el número de proyectos a realizar y ***r*** los recursos a repartir. De esta forma, los subcasos que encontramos serán tomando dupas ***(i,j)***, con **0<=i<=n** y **0<=j<=r**, consiguiendo la opción óptima en cada subcaso a partir del anterior tratado.

Además, otro de los puntos de la Programación Dinámica es el ahorro de recursos del ordenador al guardar las soluciones de cada caso, de manera que no se tengan que calcular una y otra vez.

Consideramos () con 1<=i<=n y 1<=j<=r, de manera que cada uno de los elementos de esta matriz representará el beneficio de emplear **j unidades de recurso** en el **trabajo i**.

Para la resolución del problema también requerimos una matriz donde guardaremos en cada elemento **M(i,j)** el máximo beneficio posible que se puede obtener con **los primeros *i* trabajos y *j* unidades de recursos.**

Así la fila 0, como no toma ningún trabajo, estará rellena de 0. Por otro lado, la columna 0, al no tener ningún recurso que añadir, también estará llena de 0.

Por otro lado, en el caso de la fila 1, como solo contamos con un trabajo, tendrá los beneficios de asignar el máximo de recursos del que se disponen a este, suponiendo que el número de recursos que se le asignen a cada proyecto será directamente proporcional con el beneficio que este nos dará, lo cual tiene sentido deducir.

A partir de este punto, cada elemento **M(i,j)** se decidirá por comparación con elementos ya descritos en la matriz o datos de la matriz original. Podrá ocurrir que:

* M(i,j) = M(i-1,j): implicaría que al proyecto *i* no se le asignan recursos por ahora, y el valor máximo se mantiene en la distribución que se había realizado únicamente con los trabajos previos.
* M(i,j) = B(i,j): en este caso, el beneficio máximo se obtendrá de dedicarle todos los recursos disponibles al proyecto i.
* M(i,j) = M(i,j-k) + B(i,k), con 1<=k<j: lo cual implicaría tomar una de las soluciones calculadas con menos recursos pero con el mismo número de proyectos y utilizar los recursos restantes en realizar el proyecto i.

Se tomará el mayor valor de los descritos previamente.

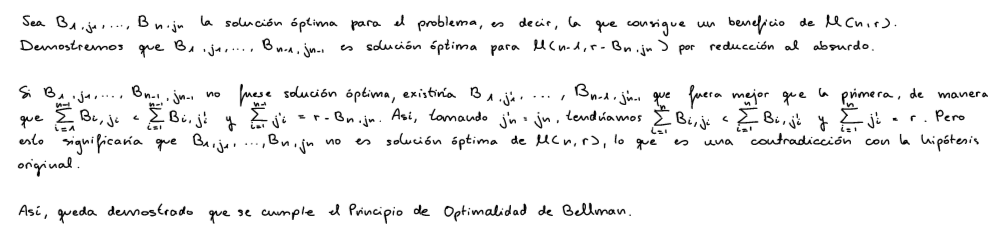
Sin embargo, en el último caso, aunque el problema no lo especifique explícitamente, se podría dar una incongruencia ya que podría ocurrir que se dedicaran recursos en dos ocasiones distintas al mismo proyecto.

Para solucionar esto, cada elemento de la matriz además de guardar el beneficio máximo, guardará un vector con todos los proyectos a los que ya se le han dedicado recursos y a los que no. De esta forma, antes de probar a añadir los beneficios que añadiría el proyecto i, se comprobará si este ya se ha realizado en cada caso, de forma que si es así, no se tomará éste como válido.

Una vez rellena la matriz, podremos obtener la secuencia de beneficios que hemos ido obteniendo de cada trabajo, es decir, cuántos recursos le hemos dedicado a cada uno, partiendo del último elemento de la tabla y realizando el proceso inverso al de construcción de la tabla. Iremos probando cada uno de los casos posibles descritos previamente viendo cuál se cumple, desplazándonos así dentro de la matriz hasta llegar a uno en el que se hayan invertido todos los recursos restantes, finalizando el proceso.

Observamos que con este algoritmo, nos aseguramos de que todas las subsecuencias de la solución que obtendremos, que será **M(n,r)**, serán las óptimas para ese número de trabajos y de recursos, cumpliendose así el Principio de Optimalidad de Bellman.

# 1.2.2. Comprobación del Principio de Optimalidad de Bellman



* 1. Conclusiones

Como vemos la resolución de este problema conlleva un alto coste computacional, para ello mediante subproblemas optimales conseguimos hallar la solución óptima en función de subproblemas de menor tamaño . Para ello tendremos que ir **guardando en memoria los subcasos** que vayamos resolviendo lo que nos permitirá evitar repetir cálculos y **mejorar la eficiencia**. Es por tanto que en nuestro caso mediante la implementación de programación dinámica, a cambio de consumir un espacio equivalente al número de recursos de los que dispongamos por la matriz asociada a los trabajos y sus beneficios, obtendremos un tiempo menor de ejecución gracias a la eficiencia proporcionada por este método.