Algorítmica

2º Grado en Ingeniería Informática

Guión de prácticas

Algoritmos de Programación Dinámica

Algoritmos de Programación Dinámica

1. Objetivo

El objetivo de la práctica consiste en que el alumno sea capaz de analizar un problema y resolverlo mediante la técnica de Programación Dinámica, siendo capaz de justificar su eficacia en términos de optimalidad. Se expondrá un problema que será resuelto en clase por el profesor. Posteriormente, se expone un conjunto de problemas que deberán ser resueltos por el estudiante.

2. Diseño de algoritmos de Programación Dinámica

2.1. Descripción del problema de ejemplo

Supongamos que compramos un refresco en una máquina y, tras pagar, se nos tiene que devolver un cambio de N céntimos con el mínimo número de monedas. Se asume que hay n **tipos de monedas** {1...n} cuyo valor es $\{d_1, d_2, \ldots, d_n\}$. Adicionalmente, se pone la restricción de que exista un **número finito de monedas** de cada tipo $\{m_1, m_2, \ldots, m_n\}$. También se asume que $d_1 = 1$. ¿Qué algoritmo debería seguir la máquina para alcanzar este objetivo?

2.2. Solución: Diseño de componentes y del algoritmo

- **Resolución del problema por etapas:** El problema se puede resolver por etapas. En cada etapa se seleccionaría echar o no echar una moneda de un tipo dado. Supondremos que las monedas están ordenadas de menor a mayor valor d_i para asegurar una solución óptima factible desde las etapas más tempranas.
- **Ecuación recurrente**: La solución depende de las etapas (tipos de monedas a considerar devolver, que notaremos como *i*) y del cambio que quede por devolver, que notaremos como *j*. Llamaremos al valor T(i,j) al mínimo número de monedas a devolver para un cambio *j*, supuesto que consideramos echar o no echar monedas de tipo 1, echar o no echar monedas de tipo 2, echar o no echar monedas de tipo 3, … hasta echar o no echar monedas de tipo i.

  En la etapa i consideramos echar monedas de tipo *i*. Caben dos posibles decisiones:

  - Devolver una moneda de tipo *i*: En tal caso, el mínimo número de monedas a devolver sería $1+T(i, j-d_i)$; es decir, 1 (por la moneda que estamos echando, y el mínimo número de monedas a devolver para un cambio que resta el valor de la moneda que estamos devolviendo, supuesto que podemos seguir echando más monedas de tipo i.
  - No devolver una moneda de tipo i: En tal caso, el mínimo número de monedas a devolver será el mismo que cuando ni siquiera habíamos considerado echar una moneda de tipo i.

Con estas dos decisiones posibles, y dado que el problema es de minimización, tendríamos que podríamos expresar la ecuación recurrente como:

$$T(i,j)= \min\{T(i-1, j), 1+T(i, j-d_i)\}$$

Los casos base para esta ecuación serían:

- Sin cambio a devolver, j=0: T(i,0)= 0 . Da igual el tipo de moneda a devolver que estemos considerando; si no hay que devolver cambio, el mínimo número de monedas a devolver es de 0.
- Considerando echar monedas de sólo $d_1 = 1$ unidad. En tal caso, T(1, j)= j ; es decir, para devolver un cambio j con monedas de 1 céntimo necesitamos j monedas.

- **Valor objetivo:** Se desea conocer el valor T(n,N), el mínimo número de monedas a devolver para un cambio N, suponiendo que consideramos echar cualquier tipo de monedas desde el {1...n}.

- **Verificación del cumplimiento del P.O.B:** Cualquier solución óptima, forzosamente, debe estar formada por subsoluciones óptimas. En este caso se cumple, dado que T(i,j), que asumimos óptimo, podrá tener valores T(i-1, j) o 1+T(i, j-d_i). Estos valores también son óptimos, dado que la ecuación recurrente siempre va escogiendo el mínimo entre ambos valores. No es posible que exista un valor menor que el mínimo entre ambos valores para T(i,j).

- **Diseño de la memoria:**

  - Para resolver el problema, T(i,j) se representará como una matriz.
  - Esta matriz tendrá n filas. Cada fila i estará asociado a un tipo de moneda i.
  - Esta matriz tendrá N+1 columnas. Cada columna estará asociada a cada posible cambio a devolver entre {0..N}.
  - Cada celda de la matriz T(i,j) contendrá el mínimo número de monedas a devolver para un cambio j, suponiendo que estamos considerando devolver monedas entre todos los tipos desde el 1 hasta el i.
  - La memoria se rellenará de la siguiente forma:
    - En primer lugar, se rellenan las celdas correspondientes a los casos base.
    - En segundo lugar, se rellenarán las filas {2, 3, …., n}, en orden secuencial.
    - Cada fila se rellenará en orden creciente de columnas {1...N}.

Con este diseño, construimos el algoritmo de Programación Dinámica como sigue:

ALGORITMO T, V= Monedas(N, n, {d_1, d_2, …, d_n})

```
T← matriz de n filas indexadas {1...n} y N columnas indexadas {0..N}
Para cada fila i en {1...n}, hacer:
        T(i,0)= 0
Para cada columna j en {1...n}, hacer:
        T(1, j)= j

Para cada fila i en {2..n}, hacer:
        Para cada columna j en {1..N}, hacer:
                T(i,j)= min{T(i-1, j), 1+T(i, j-di)}

V← T(n, N)
Devolver T, V
```

- **Recuperación de la solución:** La solución se recuperará desde el valor objetivo de la matriz calculada previamente, recorriendo hacia atrás la recurrencia. El siguiente algoritmo devuelve la solución S (el conjunto de monedas concreto a devolver), suponiendo un número de monedas de cada tipo infinito:

```
ALGORITMO S= RecuperaMonedas( T(1..n, 0..N) : Tabla resultante del algoritmo
Monedas )

    S← ∅
    i← n, j← N
    Mientras j <> 0, hacer:
        Si i>1 y T(i,j)= T(i-1, j), hacer:
            i← i-1
        En otro caso, hacer:
            j← j-di
            Añadir moneda de tipo i a S
    Devolver S
```

- **Recuperación de la solución en el caso de que haya un número finito $\{m_1, m_2, \ldots, m_n\}$ de cada tipo de moneda**

Para resolver este problema de forma óptima completamente, habría que reformular la ecuación inicial levemente e incluir restricciones. El enunciado pide que modifiquemos sólo la función de recuperación de la solución, por lo que podría darse el caso de que, en algún sistema monetario, el algoritmo completo (construcción de la solución y recuperación de la misma) no fuese óptimo.

La idea general del algoritmo de recuperación podría ser la siguiente:

- Deberíamos llevar un contador de cada tipo de monedas que se ha devuelto $\{c_1, c_2, \ldots, c_n\}$. Cada valor c_i estaría inicialmente inicializado a 0 (porque no se ha decidido echar ninguna moneda).
- Cuando estamos evaluando T(i,j), se podría considerar echar una moneda de tipo i sólo si se cumple que $c_i < m_i$. En caso contrario, sólo se debería considerar la decisión T(i-1, j) para recorrer hacia atrás la solución hasta un caso base.
- Si se decide devolver una moneda de tipo i, se debe actualizar el contador $c_i \leftarrow c_i + 1$
- Si se consigue llegar al caso base T(i,0), se devuelven las monedas seleccionadas.
- Si se llega al caso base T(1, j), con j>0, y no quedan monedas de tipo 1, entonces se decide que no hay solución (en este caso el algoritmo no es óptimo porque podría haber alguna decisión en algún T(i,j), distinta a la tomada, que sí devolviese algún conjunto de monedas).

El algoritmo de recuperación de la solución modificado es como sigue:

```
ALGORITMO S= RecuperaMonedas2( T(1..n, 0..N) : Tabla resultante del algoritmo
Monedas )

    C← {c1, c2, …, cn}
    Para cada i en {1..n}, hacer:
        ci ← 0
    S← ∅
    i← n, j← N
```

Mientras j <> 0, hacer:

Si T(i,j) <> T(i, j-d_i) ó c_i >=m_i , hacer:
i← i-1
En otro caso, hacer:
j← j-d_i
Añadir moneda de tipo i a S
Si i < 1, hacer:
Devolver "No hay solución"
Devolver S

3. Ejercicios propuestos

Para cada uno de los ejercicios siguientes, se debe:

- Diseñar e implementar un algoritmo de Programación Dinámica que resuelva el problema, así como el algoritmo de recuperación de la solución.
- Analizar la optimalidad (mediante demostración o contraejemplo) del algoritmo diseñado.

3.1. Ejercicio 1

A lo largo de un río hay n aldeas. En cada aldea, se puede alquilar una canoa para viajar a otras aldeas que estén a favor de la corriente (resulta casi imposible remar a contra corriente). Para todo posible punto de partida i y para todo posible punto de llegada j más abajo en el río $(i < j)$, se conoce el coste de alquilar una canoa para ir desde i hasta j, $c(i, j)$ (si ese trayecto concreto no existe, entonces $c(i, j) = \infty$. Sin embargo, puede ocurrir que el coste del alquiler desde i hasta j sea mayor que el coste total de una serie de alquileres más breves. En tal caso, se puede devolver la primera canoa en alguna aldea k situada entre i y j y seguir el camino alquilando en k una nueva canoa (no hay costes adicionales por cambiar de canoa de esta manera).

Diseñe un algoritmo basado en programación dinámica para determinar el coste mínimo del viaje en canoa desde todos los puntos posibles de partida i a todos los posibles puntos de llegada j $(i < j)$. Aplique dicho algoritmo en la resolución del caso cuya matriz de costos es la siguiente:

	1	2	3	4	5
1	0	3	3	∞	∞
2	-	0	4	7	∞
3	-	-	0	2	3
4	-	-	-	0	2
5	-	-	-	-	0

3.2. Ejercicio 2

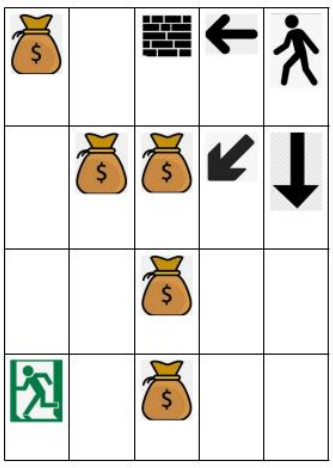
Una empresa realiza planificaciones de viajes aéreos entre n ciudades. Para ir de una ciudad i a una ciudad j puede ser necesario coger varios vuelos distintos. El tiempo de un vuelo directo de i a j será (si existe) $T(i,j)$ (que puede ser distinto de $T(j,i)$). Hay que tener en cuenta que si cogemos un vuelo de i a k y después otro de k a j, será necesario esperar un tiempo de *escala* adicional $E(k)$ en el aeropuerto de k, con lo que el tiempo de ese viaje de i a j será de $T(i,k)+T(k,j)+E(k)$.

Se desea conocer la forma de volar desde cualquier ciudad i hasta cualquier otra j en el menor tiempo posible. Diseñad e implementad un algoritmo de Programación dinámica que resuelva este problema. Aplicadlo para resolver el problema, para $n=4$, cuya matriz de tiempos es la siguiente, suponiendo que $E(k)=1\ \forall k$.

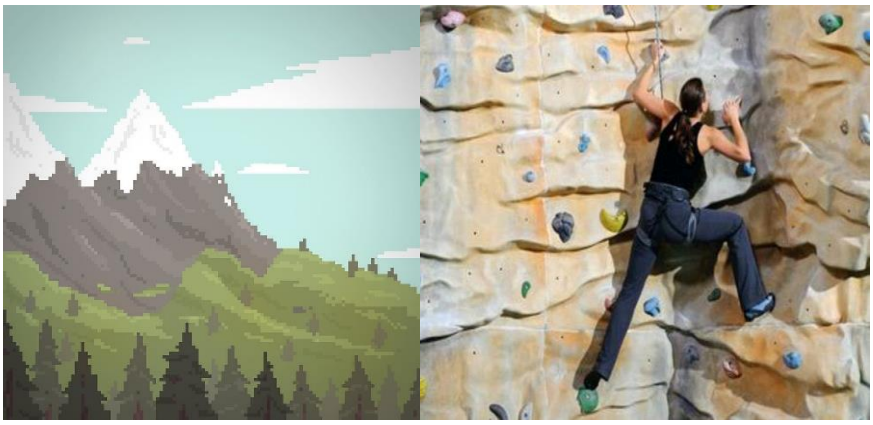
$T[i,j]$	1	2	3	4
1	0	2	1	3
2	7	0	9	2
3	2	2	0	1
4	3	4	8	0

3.3. Ejercicio 3

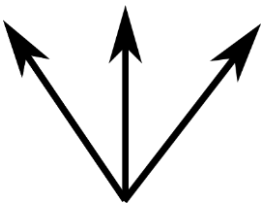
Un videojuego se juega por turnos y se representa en un mapa cuadriculado bidimensional de f filas y c columnas. El jugador siempre entra al mapa por la esquina superior derecha, y sale por la esquina inferior izquierda. En cada turno, los posibles movimientos del jugador son: ir 1 casilla a la izquierda, ir 1 casilla abajo, o moverse 1 posición a la casilla inferior izquierda. Cada casilla del mapa puede estar vacía, contener un muro, o contener una bolsa de oro. Todas las casillas son transitables salvo las que tienen muros. El objetivo consiste en llegar a la salida pudiendo recoger tanto oro como sea posible (pasar por tantas casillas que contengan una bolsa como se pueda). En el ejemplo siguiente, el jugador puede conseguir un máximo de 3 bolsas de oro con los movimientos permitidos.



3.4. Ejercicio 4



Tenemos que ponernos en forma, y hemos decidido empezar subiendo la "Pixel Mountain". Para ello, tenemos que ascender desde una posición baja hasta la cumbre. Evidentemente, una parte muy importante de toda ascensión es decidir por dónde deberíamos subir. En nuestra ascensión siempre subimos (sin desfallecer) pero podemos movernos directamente hacia arriba, o desplazarnos a la izquierda o derecha en diagonal:



En cada posible posición de la montaña tenemos un coste asociado de la dificultad que tiene llegar a esa posición. El objetivo es calcular el recorrido con menor dificultad:

Un ejemplo:

2	8	9	**5**	8
4	4	6	**2**	3
5	7	5	6	**1**
3	2	5	**4**	8

De esta montaña 5x4 se puede observar que el mejor camino (el más fácil) es el marcado en negrita, es decir, el que implica la dificultad de **4+1+2+5 = 12.** El plan (secuencia de pasos) serían las posiciones 3-4-3-3 (si numeramos desde 0).

4. Evaluación de la práctica

Se deben resolver todos los ejercicios propuestos en el apartado 3 de este guión. Cada ejercicio se valorará sobre 10 de la siguiente forma:

1. **(2 puntos)** Diseño de resolución por etapas y ecuación recurrente
2. **(2 puntos)** Diseño de la memoria
3. **(1 punto)** Verificación del P.O.B.
4. **(2 puntos) Diseño** del algoritmo de cálculo de coste óptimo.
5. **(2 puntos) Diseño** del algoritmo de recuperación de la solución.
6. **(1 punto) Implementación** de los algoritmos de cálculo de coste óptimo y recuperación de la solución.

La valoración de la práctica se dará como una calificación numérica entre 0 y 10. **Todos los problemas propuestos tendrán la misma calificación máxima de 10/3**.

5. Entrega y presentación de la práctica

Se deberá entregar un documento en PDF (memoria de prácticas) realizado en equipos de 3 personas, conteniendo los siguientes apartados:

1. Solución detallada a los problemas propuestos, conteniendo los apartados descritos en el apartado 4 de este guión, para cada problema propuesto.

La práctica deberá ser entregada por PRADO, en la fecha y hora límite explicada en clase por el profesor. No se aceptarán, bajo ningún concepto, prácticas entregadas con posterioridad a la fecha límite indicada. La entrega de PRADO permanecerá abierta con, al menos, una semana de antelación antes de la fecha límite, por lo que todo alumno tendrá tiempo suficiente para entregarla.

El profesor, en clase de prácticas, realizará controles de las prácticas a discreción, que consistirán en presentaciones de los estudiantes de cada equipo (powerpoint) y/o entrevistas individuales con el fin de asegurar de que los estudiantes alcanzan las competencias deseadas. Estas entrevistas y/o presentaciones se realizarán en las sesiones de evaluación de prácticas, previamente anunciadas en clase por el profesor.

La **no asistencia** a una sesión de evaluación de prácticas por un estudiante supondrá la **calificación de 0 (no presentado) a la práctica que deba presentar, independientemente de la calificación obtenida en la memoria de prácticas**.

IMPORTANTE: Antes de las sesiones de evaluación, cada estudiante deberá prepararse para:

- Realizar una presentación powerpoint de la práctica de **máximo 5 minutos**, si el profesor lo exige.
- Conocer a fondo todos los algoritmos resueltos, así como los pasos para diseñar e implementar los algoritmos.
- Conocer las respuestas a las preguntas requeridas en el apartado 4 para cada problema.

El desconocimiento o la ausencia del material indicado podrá suponer la calificación de 0 en la práctica.