UNIVERSIDAD DE GRANADA

# Práctica 4
# Programación Dinámica

**Joaquín Sergio García Ibáñez**
**Juan Navarro Maldonado**

**Índice:**

**Ejercicio 1** **2**
Diseño de resolución por etapas 2
Ecuación recurrente 2
Diseño de la memoria 2
Verificación del P.O.B. 3
Diseño del algoritmo de cálculo de coste óptimo. 3
Diseño del algoritmo de recuperación de la solución. 3
Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución. 4

**Ejercicio 2** **5**
Diseño de resolución por etapas 5
Ecuación recurrente 6
Diseño de la memoria 6
Verificación del P.O.B. 6
Diseño del algoritmo de cálculo de coste óptimo. 7
Diseño del algoritmo de recuperación de la solución. 7
Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución. 8

**Ejercicio 3** **9**
Diseño de resolución por etapas 9
Ecuación recurrente 10
Diseño de la memoria 10
Verificación del P.O.B. 10
Diseño del algoritmo de cálculo de coste óptimo. 11
Diseño del algoritmo de recuperación de la solución. 11
Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución. 11

# Ejercicio 1

*En el departamento de una empresa de traducciones se desea hacer traducciones de textos entre varios idiomas. Se dispone de algunos diccionarios. Cada diccionario permite la traducción (bidireccional) entre dos idiomas. En el caso más general, no se dispone de diccionarios para cada par de idiomas por lo que es preciso realizar varias traducciones. Dados N idiomas y M diccionarios, determine si es posible realizar la traducción entre dos idiomas dados y, en caso de ser posible, indique la cadena de traducciones de longitud mínima. Ejemplo: Traducir del latín al arameo disponiendo de los siguientes diccionarios: latín-griego, griegoetrusco, etrusco-demótico, griego-demótico, demótico-arameo (Solución: sí es posible la traducción: latín-griego-demótico-arameo, realizando tres traducciones).*

## Diseño de resolución por etapas

El problema se puede resolver por etapas. En cada etapa se seleccionaría un nodo o traducción intermedio k para cada par de nodos i,j.
La función objetivo es minimizar $D_N[i][j]$, con N el número de nodos para i,j

## Ecuación recurrente

Vamos a definir T[i][j] como la secuencia mínima de diccionarios utilizados. Lo que nosotros buscamos es T[M][N], esto es la secuencia mínima de diccionarios usados (M) para traducir el N idioma .

Los idiomas los vamos a definir como j y van a ir desde 0 hasta N
D es la matriz adyacente del grafo y D[i][j] es la distancia de traducciones entre i y j.
La ecuación recurrente será:

$$D_k[i][j] = min\{D_{k-1}[i][j],\ D_{k-1}[i][k] + D_{k-1}[k][j]\}$$

El caso base sería el siguiente:

$$D_0[i][j] = D[i][j]$$

El camino de i a j sin pasar por ningún nodo entre ellos

## Diseño de la memoria

Para resolver el problema, D[i][j] se representará como una matriz.

- Esta matriz tendrá M filas. Cada fila i estará asociado a un idioma i.
- Esta matriz tendrá N columnas. Cada columna estará asociada a otro idioma {0..N}.
- 
- Cada celda de la matriz D[i][j] contendrá el mínimo número de traducciones para un idioma j, suponiendo que estamos considerando que queremos traducir desde el i idioma.

## Verificación del P.O.B.

Para el caso base es óptimo cuando el mejor camino de traducir el idioma i al idioma j sin pasar por ningún idioma es D[i][j].
Para el caso general es óptimo debido a que habría otros nodos en el camino de i a j pasando por {1...k-1} tal que el número de traducciones intermedias sea menor que el considerado. Esto es imposible, dado que la ecuación recurrente siempre selecciona el menor número de traducciones

## Diseño del algoritmo de cálculo de coste óptimo.

```
Procedimiento Traducir(MatridAdy[0..N][0..N], P[0..N][0..N])
        Para i=0 hasta N, hacer:
                Para j=0 hasta N, hacer:
                        D[i][j]= MatrizAdy[i][j]
                        P[i][j]= 0
                Fin-Para
        Fin-Para
Para k=0 hasta N, hacer:
        Para i=0 hasta N, hacer:
                Para i=0 hasta N, hacer:
                        Si D[i][j]> D[i][k]+D[k][j], entonces:
                                D[i][j]= D[i][k]+D[k][j]
                                P[i][j]= k
                        Fin-Si
                Fin-Para
        Fin-Para
Fin-Para
Devolver D,P
```

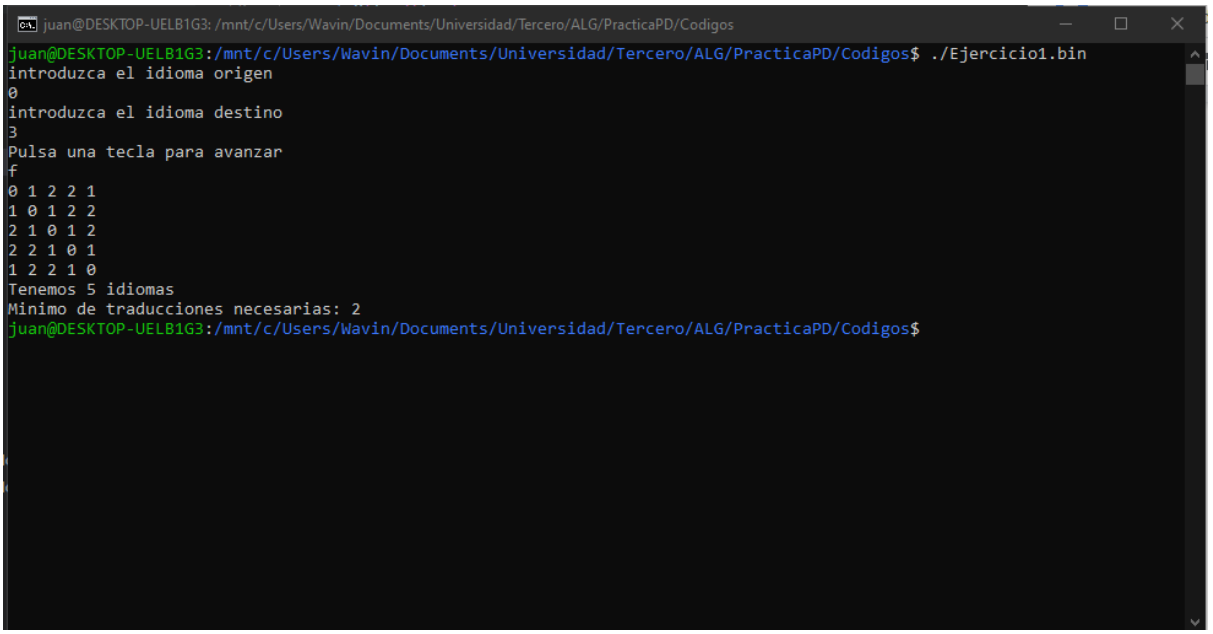
## Diseño del algoritmo de recuperación de la solución.

```
Algoritmo RecuperaSolucion(D[0..N][0..N], T[0..N][0..N])
        Mostrar D[i][j] //Distancia minima entre el idioma i y el idioma j
        Para i=0 hasta N, hacer:
                Para i=0 hasta N, hacer:
                        Mostrar T[i][j] //Secuencia de pasos para traducir del idioma i
        al idioma j

                Fin-Para
        Fin-Para
```

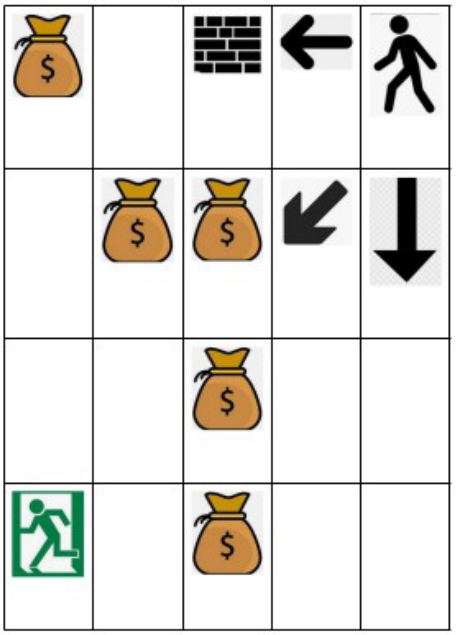
# Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución.

Nuestro algoritmo empieza con un ejemplo básico en el que tenemos 5 idiomas y 5 diccionarios que se pueden traducir entre ellos, nos hemos fijado a la hora de crear la matriz de adyacencia en el ejemplo del enunciado en el que se hace una traduccion del latin al arameo y vamos a probar a traducir desde el idioma 0 al idioma 3, nuestro algoritmo empezaría haciendo una copia a una matriz llamada D y con esa matriz comprobamos el coste de los nodos intermedios y si el coste es mejor que el actual lo metemos y con D[i][j] mostramos la distancia mínima de traducción entre i y j



# Ejercicio 2

*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.*



## Diseño de resolución por etapas

Numeramos los distintos nodos desde 1 … n, definiremos como nodo . Asumimos que D es la matriz de adyacencia del grafo, y D[i][j] es la distancia para ir directos desde el nodo i al nodo j. Debemos de tener en cuenta que la distancia entre dos nodos que no tengan arco, es decir, no son contiguos, es infinita.
Nuestra función objetivo será minimizar $D_n[i][j]$ con n el número de casillas que se recorren dentro de la matriz de nuestro mapa.

El problema es resoluble por etapas siempre que en cada etapa se considere pasar por un nodo intermedio.

# Ecuación recurrente

El objetivo es conocer $D_n[i][j]$ que es el coste mínimo para ir desde el inicio hasta la salida pudiendo coger todas las bolsas de dinero posibles hasta un maximo de 3 bolsas, considerando pasar por cualquier casilla.

El caso general supone que para ir desde el inicio hasta la salida puede pasar o no por las casillas 1 a K como intermedios, por lo que tiene dos posibilidades:

- Que el camino de i a j no pase por K
- Que el camino de i a j pase por K

Por tanto la ecuación recurrente que tendríamos es:

$$D_k[i][j] = min\{D_{k-1}[i][j],\ D_{k-1}[i][k] + D_{k-1}[k][j]\}$$

El caso base vendría dado por una matriz de 4x4, donde la salida se encuentre solamente a un paso del robot, gastando 0 de batería, por lo que:

$$D_0[i][j] = D[i][j]$$

# Diseño de la memoria

Para poder representar la solucion, necesitaremos dos matrices:

- D[i][j], que va a contener los costes de la distancia entre dos puntos
- T[i][j], que representará el mapa en el que se encuentra nuestro agente.

# Verificación del P.O.B.

Para el caso base, siempre será óptimo, ya que el mejor camino desde el inicio a la salida siempre es óptimo ya que no pasas por ninguna otra casilla al ir directamente al objetivo.

Para cualquier otro caso, también va a ser óptimo ya que debido a la ecuación recurrente siempre nos seleccionará la casilla de menor coste.

## Diseño del algoritmo de cálculo de coste óptimo.

```
Procedimiento Camino(MatrizAdy[0..N][0..N], T[0..N][0..N])
        Para i=0 hasta N, hacer:
                Para j=0 hasta N, hacer:
                        D[i][j]= MatrizAdy[i][j]
                        T[i][j]= 0
                Fin-Para
        Fin-Para
Para k=0 hasta N, hacer:
        Para i=0 hasta N, hacer:
                Para i=0 hasta N, hacer:
                        Si D[i][j]> D[i][k]+D[k][j], entonces:
                                D[i][j]= D[i][k]+D[k][j]
                                T[i][j]= k
                        Fin-Si
                Fin-Para
        Fin-Para
Fin-Para
Devolver D,T
```

## Diseño del algoritmo de recuperación de la solución.

```
Algoritmo RecuperaSolucion(T[0..N][0..N])
        Para i=0 hasta N, hacer:
                Solucion = T[i][j]+Solucion
        Fin-Para
        Devolver Solucion
Fin-Algoritmo
```

# Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución.

Para probar este algoritmo hemos decidido poner un ejemplo de un tablero 4x4 para visualizar mejor el comportamiento de nuestro algoritmo.

Cada casilla, estará evaluada por con un coste, donde las bolsas con dinero tienen el menor coste, las casillas normales, por las que puede transitar el ladrón tienen un coste algo superior y las casillas con muros tienen un coste alto, para que no sean evaluadas por el algoritmo.

Ejecutamos el código con ./Ejercicio2.bin
Para este ejercicio no es necesario pasarle como argumento nada, porque el tamaño del tablero se creará en tiempo de ejecución.

```
juan@DESKTOP-UELB1G3:/mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/PracticaPD/Codigos$ ./Ejercicio2.bin
introduzca el tamaño del mapa
4
Mostrando mapa:
_ $ X R
_ _ $ $
_ _ $ _
O _ _ $
Pulsa cualquier tecla para continuar
```

Como podemos observar el tablero que se ha generado, contiene un muro y 5 bolsas con dinero. Si aplicamos los movimientos del ladrón, los movimientos óptimos serían:

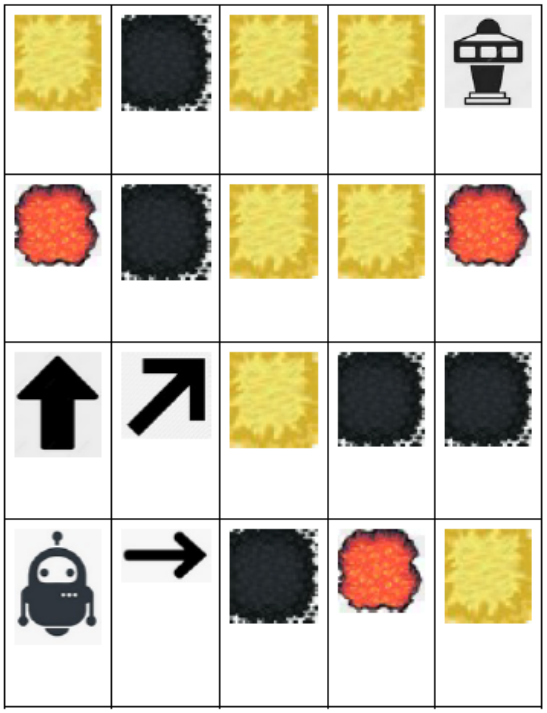
1.- Mover abajo -> Bolsa + 1
2.- Mover izquierda -> Bolsa + 1
3.- Mover diagonal -> Bolsa +1
4.- Mover abajo
5- Mover izquierda -> Objetivo

Por tanto el número de bolsas que puede coger el ladrón serán 3.

```
juan@DESKTOP-UELB1G3: /mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/PracticaPD/Codigos
_ R $ _
O _ _ $
menor 0
valor: 50
cambia pos nueva:
2 0
Pulsa cualquier tecla para continuar
c
costes :10 10000 50
Prueba mapa: 50
posF: 2
posC: 0
menor 0
valor: 10
Mostrando mapa:
_ $ X _
_ _ $ $
R _ $ _
O _ _ $
Ultima columna
menor 1
valor: 10
termina ejecucion
cambia pos nueva:
3 0
Pulsa cualquier tecla para continuar
c
Camino recorrido:
10-> 10-> 10-> 50-> 50-> 10000-> Numero de bolsas recogidas: 3
juan@DESKTOP-UELB1G3:/mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/PracticaPD/Codigos$
```

# Ejercicio 3

*Una sonda marciana representa el mapa del terreno de forma cuadricuada en 2D, mediante una tabla de f filas y c columnas. La sonda siempre se posiciona en la esquina inferior izquierda del mapa, y puede moverse de su casilla actual a su inmediatamente superior, inmediatamente derecha, o inmediatamente en la posición superior derecha. Siempre, en la esquina superior derecha encontramos su estación base, a la que debe regresar. En cada casilla del mapa puede haber un terreno que puede dificultar más o menos el avance de la sonda debido a que gaste más o menos batería. Un ejemplo puede verse en la siguiente figura:*
*La sonda tiene catalogados un total de n tipos de terreno. Pasar por una casilla del terreno de tipo i puede gastar una batería bi. El objetivo es llegar desde la posición inicial hasta la objetivo gastando la mínima batería posible.*



## Diseño de resolución por etapas

Numeramos los distintos nodos desde 1 … n, definiremos como nodo . Asumimos que D es la matriz de adyacencia del grafo, y D[i][j] es la distancia para ir directos desde el nodo i al nodo j. Debemos de tener en cuenta que la distancia entre dos nodos que no tengan arco, es decir, no son contiguos, es infinita.
Nuestra función objetivo será minimizar $D_n[i][j]$ con n el número de casillas que se recorren dentro de la matriz de nuestro mapa.

El problema es resoluble por etapas siempre que en cada etapa se considere pasar por un nodo intermedio.

## Ecuación recurrente

El objetivo es conocer $D_n[i][j]$ que es el coste mínimo para ir desde el inicio hasta la salida considerando pasar por cualquier casilla.

El caso general supone que para ir desde el inicio hasta la salida puede pasar o no por las casillas 1 a K como intermedios, por lo que tiene dos posibilidades:

- Que el camino de i a j no pase por K
- Que el camino de i a j pase por K

Por tanto la ecuación recurrente que tendríamos es:

$$D_k[i][j] = min\{D_{k-1}[i][j],\ D_{k-1}[i][k] + D_{k-1}[k][j]\}$$

El caso base vendría dado por una matriz de 2x2, donde la salida se encuentre solamente a un paso del robot, gastando 0 de batería, por lo que:

$$D_0[i][j] = D[i][j]$$

## Diseño de la memoria

Para poder representar la solucion, necesitaremos dos matrices:

- D[i][j], que va a contener los costes de la distancia entre dos puntos
- T[i][j], que representará el mapa en el que se encuentra nuestro agente.

## Verificación del P.O.B.

Para el caso base, siempre será óptimo, ya que el mejor camino desde el inicio a la salida siempre es óptimo ya que no pasas por ninguna otra casilla al ir directamente al objetivo.

Para cualquier otro caso, también va a ser óptimo ya que debido a la ecuación recurrente siempre nos seleccionará la casilla de menor coste.

## Diseño del algoritmo de cálculo de coste óptimo.

```
Procedimiento Camino(MatrizAdy[0..N][0..N], T[0..N][0..N])
        Para i=0 hasta N, hacer:
                Para j=0 hasta N, hacer:
                        D[i][j]= MatrizAdy[i][j]
                        T[i][j]= 0
                Fin-Para
        Fin-Para
Para k=0 hasta N, hacer:
        Para i=0 hasta N, hacer:
                Para i=0 hasta N, hacer:
                        Si D[i][j]> D[i][k]+D[k][j], entonces:
                                D[i][j]= D[i][k]+D[k][j]
                                T[i][j]= k
                        Fin-Si
                Fin-Para
        Fin-Para
Fin-Para
Devolver D,T
```

## Diseño del algoritmo de recuperación de la solución.

```
Algoritmo RecuperaSolucion(T[0..N][0..N])
        Para i=0 hasta N, hacer:
                Solucion = T[i][j]+Solucion
        Fin-Para
        Devolver Solucion
Fin-Algoritmo
```

## Implementación de los algoritmos de cálculo de coste óptimo y recuperación de la solución.

Vamos a poner un ejemplo para la implementación de este algoritmo, en este caso, vamos a generar un tablero 4x4. Del modo que lo hemos implementado, siempre tendremos 3 tipos de casillas, aparte de la casilla inicial y la casilla de salida, que son totalmente distintas al tipo del terreno en el que la sonda se moverá.

Los distintos tipos de terreno se representarán en la terminal con el coste de cada uno de ellos, por lo que tendremos:

1.- Coste bajo -> 10
2.- Coste medio -> 50
3.- Coste alto -> 99

La casilla objetivo, es decir la esquina superior derecha de nuestro mapa, siempre valdrá 0 para que la sonda sepa cuándo terminar la ejecución.

Procedemos a ejecutar nuestro programa con ./Ejercicio3.bin
No hace falta añadir parámetros a la ejecución, puesto que el tamaño del mapa se pedirá dentro de la ejecución del algoritmo.

```
juan@DESKTOP-UELB1G3:/mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/PracticaPD/Codigos$ ./Ejercicio3.bin
Introduzca el tamaño del mapa
4
Mostrando mapa:
50 50 10 O
99 50 50 10
10 50 99 50
R  50 99 50
Pulsa cualquier tecla para continuar
```

El camino óptimo, que debería recorrer la sonda, será:

1.- Mover arriba -> Coste 10
2.- Mover diagonal -> Coste 50
3.- Mover diagonal -> Coste 10
4.- Mover derecha -> Coste 0 (Objetivo)

Por lo tanto el coste total es de 70, el cual sería el óptimo para llegar a la casilla de destino.

```
juan@DESKTOP-UELB1G3: /mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/PracticaPD/Codigos
Mostrando mapa:
50 50 10 O
99 R  50 10
10 50 99 50
99 50 99 50
menor 1
valor: 10
cambia pos nueva:
0 2
Pulsa cualquier tecla para continuar
c
costes :-312777776 32767 0
Prueba mapa: 0
posF: 0
posC: 2
Mostrando mapa:
50 50 R  O
99 50 50 10
10 50 99 50
99 50 99 50
menor 2
valor: -312777776
Termina ejecucion
cambia pos nueva:
0 3
Pulsa cualquier tecla para continuar
c
Camino recorrido:
10-> 50-> 10-> 0-> Coste total: 70
juan@DESKTOP-UELB1G3:/mnt/c/Users/Wavin/Documents/Universidad/Tercero/ALG/PracticaPD/Codigos$
```

Podemos comprobar por tanto el recorrido que realiza la sonda paso a paso mostrado en el mapa y al final de la ejecución, una vez finalizada la ejecución del código, recuperará la solución y nos mostrará el coste total del camino.