UNIDAD 4. PROGRAMACIÓN DINÁMICA

Programación dinámica

Tabla de contenido

Introducción

La programación dinámica es un método de optimización que puede aplicarse a diferentes y numerosos problemas, algunos de los cuales ya han sido analizados en programación lineal y programación entera. Los parámetros usados en la programación dinámica pueden ser estocásticos o probabilísticos y determinísticos.

La programación dinámica tiene como finalidad encontrar una solución de un problema de optimización en forma secuencial. A diferencia de la programación lineal, la programación entera no es un algoritmo de solución única, sino más bien un método para resolver un problema grande y único solventando una secuencia de problemas más pequeños, sin importar el número de ellos. Ahora, la programación dinámica permite resolver un problema que depende del tiempo en forma de una continuidad de problemas de un sólo periodo, en donde los parámetros de cada periodo dependen del periodo que se considera; es posible que no se conozca la cuantificación de cada periodo sino hasta que éste llega.

La solución de problemas mediante la programación dinámica se basa en el llamado principio de óptimo, enunciado por Bellman (1957) y que dice: "*En una secuencia de decisiones óptima toda subsecuencia ha de ser también óptima*". Sin embargo, se debe evaluar en cada problema presentado que este principio se esté cumpliendo y analizar cómo abordar cada uno de los problemas de acuerdo a sus propias características.

Objetivos

Objetivo general

Solucionar problemas que puedan resolverse a través de la programación dinámica.

Objetivos específicos

- Identificar las generalidades de la programación dinámica y cómo aplicar la metodología para la resolución de problemas en donde se deben tomar decisiones por fases.
- Aplicar algoritmos como el problema de la mochila, el cálculo de los números de Fibonacci y el cálculo de los coeficientes binomiales, entre otros, para la resolución de problemas.
- Diferenciar los métodos que se pueden emplear en la resolución de problemas, asimilando conceptos como programación dinámica determinista, estocástica y programación lineal.

4.1 Generalidades de la programación dinámica

La programación dinámica es un método de optimización que se puede emplear para la resolución de problemas de matemática aplicada y para darle estructura a una solución óptima, definiendo el camino más adecuado para hallarla.

La programación dinámica comenzó a emplearse durante la Segunda Guerra Mundial. Algunos de los aliados (Alemania, Inglaterra, Estados Unidos y la U.R.S.S.), conformaron grupos de trabajo consagrados a la investigación, quienes posteriormente serian la base de muchos de los inventos que aparecieron durante este tiempo de guerra, inicialmente diseñados como estrategias militares para la guerra y que posteriormente contribuirían al desarrollo luego de 1945.

La primera disciplina que surgió a partir de la forma como se dio solución a muchos problemas en la guerra, fue la investigación operativa.

Después de la guerra los equipos de trabajo dedicados a la investigación, dieron inicio a nuevos temas que, como resultado final, arrojaron el planteamiento de diversos problemas que fueron abordados desde un enfoque netamente matemático y en donde la programación dinámica se comenzó a aplicar después de 1952.

Posteriormente, Richard Bellman (1920 – 1984), en el año de 1957, desarrolla el método en el área de los problemas de decisión discretos y con la ayuda de sus colaboradores se dedicó a formular modelos matemáticos capaces de solucionar diferentes problemas en los términos de la programación dinámica que, como resultado, mostraron que las ideas centrales de los métodos aplicados por ellos, en particular, las basadas en el Principio de Optimalidad, podrían ser utilizadas satisfactoriamente en la mayoría de los problemas que se estaban presentando en aquella época.

Características de un problema de programación dinámica

Para que un problema pueda ser resuelto mediante programación dinámica, debe cumplir con ciertas características, como:

- El problema puede ser dividido en etapas.
- Cada etapa tiene un número de estados asociados a ella.
- La decisión óptima de cada etapa depende sólo del estado actual y no de las decisiones anteriores.
- La decisión tomada en una etapa determina cuál será el estado de la etapa siguiente.

4.2 Resolución de un problema de programación dinámica

Para resolver un problema de programación dinámica se debe:

- **Identificar etapas, estados y variable de decisión:** Cada etapa debe tener asociada una o más decisiones (problema de optimización).
  - Cada estado debe contener toda la información relevante para la toma de decisión asociada al periodo.
  - Las variables de decisión son aquellas sobre las cuales se debe definir su valor, de modo que se pueda optimizar el beneficio acumulado y modificar el estado de la próxima etapa.
- **Describir las ecuaciones de recurrencia:** Se debe indicar cómo se acumula la función de beneficios a optimizar (función objetivo) y cómo varían las funciones de estado de una etapa a otra.
- **Solucionar:** Se debe optimizar cada sub-problema por etapas en función de los resultados de la resolución del sub-problema siguiente.

Tipos de programación dinámica

Determinista	Estocástica
Las variables que actúan sobre el sistema son predecibles	Las variables que actúan sobre el sistema son aleatorias.
	No se conoce el valor exacto de las variables, pero sí su función de distribución.
	La decisión óptima es la que minimiza el costo esperado.

Tabla 4.1. Diferencias entre programación dinámica determinista y estocástica.

4.3 Terminología

Función de Valor Óptimo: Se puede llamar a la regla que asigna valores a varios problemas dentro de un problema.

Función de Política Óptima: Es la que asocia la primera mejor decisión con cada problema**.**

Relación de recurrencia o relación recursiva: Es el producto que provoca una fórmula o grupo de fórmulas que pertenecen a varios valores de S, basado en el principio de optimalidad.

Condiciones limitantes: Asumidas como obvias desde el planteamiento del problema y desde la definición de S con cálculos necesitados como resultantes de los valores de la función de valor óptimo S para ciertos argumentos.

4.4 Etapas y estados en programación dinámica

Cuando una variable describe cuántas decisiones han sido tomadas hasta cierto momento y si el número total de decisiones es fijo, el número de etapas será igual al número de decisiones.

Las variables de estado, que son las posibles condiciones variadas en las cuales el procedimiento se encuentra en esa etapa del problema y el número de estados, pueden ser finitas o infinitas.

La decisión en cada etapa es el resultado de asignar un número de veces las variables de estado sucesivas X_n, X_{n+1} que están unidas a través de la ecuación recursiva que calcula los valores de X_{n+1} usando el valor de X_n y la decisión en el estado d_n.

Las variables de estado pertenecen al presente estado con el anterior y permiten calcular la restante cantidad de recursos escasos.

En programación dinámica existen dos procedimientos:

1. **En retroceso**: Caracterizado por tener unas condiciones terminales fijas y el cálculo de valores numéricos se realiza desde la línea terminal al punto inicial.

2. **En avance**: Caracterizado por tener unas condiciones iniciales fijas y el cálculo de valores numéricos se realiza desde la línea inicial al punto final.

4.5 Algoritmos de solución

Existe un conjunto de instrucciones o reglas bien definidas que permiten analizar problemas que se pueden resolver a través de programación dinámica. Algunos de los más importantes algoritmos de solución son:

4.5.1 El problema de la mochila

Hay un conjunto S de n objetos, en el que cada objeto i tiene un beneficio bi y un peso wi positivo. Se deben seleccionar los elementos que garanticen un beneficio máximo, pero con un peso global menor o igual que W.

Dado el conjunto S de n objetos, sea Sk el conjunto de los k primeros objetos (de 1 a k):

Se puede definir B(k,w) como la ganancia de la mejor solución obtenida a partir de los elementos de S k para una mochila de capacidad w.

La mejor selección de elementos del conjunto Sk para una mochila de tamaño w se puede definir en función de selecciones de elementos de Sk-1 para mochilas de menor capacidad.

La mejor opción para Sk coincide con la mejor selección de elementos de Sk-1 con peso máximo w (el beneficio máximo para S k coincide con el de Sk-1), o bien es el resultado de añadir el objeto k a la mejor selección de elementos de Sk-1 con peso máximo w−wk (el beneficio para Sk será el beneficio que se obtenía en Sk-1 para una mochila de capacidad w−wk más el beneficio bk asociado al objeto k).
B (k,w):

$$B\ (k,w) = \begin{Bmatrix} B\ (k-1,w) & si\ X_k = 0 \\ B\ (k-1,w-\ w_k\ ) +\ b_k & si\ X_k = 1 \end{Bmatrix}$$

Para resolver el problema se puede hallar el máximo de ambos valores:

$$B\ (k,w) = \max\{B\ (k-1,w),\quad B\ (k-1,\quad w-\ w_k\ ) +\ b_k\}$$

4.5.2 Cálculo de los números de Fibonacci

Hace referencia a la deducción de los requisitos (términos) de la sucesión de números de Fibonacci. Estos cálculos se pueden expresar de la siguiente forma:

$$Fib\ (n) = \begin{cases} 1 & si\ \ n = 0{,}1 \\ Fib\ \ (n-1) + Fib\ (n-2)) & si\ \ n\ > 1 \end{cases}$$

Para resolver este problema se puede crear un algoritmo que en tiempo lineal lo resuelva, a través de la generación de una tabla que permita ir recopilando los cálculos realizados para reutilizarlos posteriormente.

Los algoritmos obtenidos con la aplicación de esta técnica pueden llegar a ser complejos en tiempo y espacio; se debe procurar evitar una complejidad temporal de orden polinómico y simplificar su construcción.

4.5.3 Cálculo de los coeficientes binomiales

Cuando se encuentra la solución a un problema y se halla una expresión que define la solución, es complejo crear un vector o una tabla que conserve los resultados parciales. De esta forma, se observa que se requiere de una tabla bidimensional que haga referencia al cálculo de los coeficientes binomiales, definido como:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}\ si\ 0\ < k\ < n,\quad \binom{n}{0} = \binom{n}{n} = 1$$

El algoritmo recursivo que los calcula es de complejidad exponencial por la repetición de los cálculos que realiza. De esta manera, se puede plantear una notación con un tiempo de ejecución de orden O (nk) fundamentado en el triángulo de Pascal. Para su realización, es necesaria la instauración de una tabla en dos dimensiones en donde se puedan almacenar los valores intermedios que se utilizan posteriormente.

	0	**1**	**2**	**3**	**…**	**k-1**	**K**
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
…	…	…	…	…	…		
…	…	…	…	…	…	…	
n-1						C (n-1, k-1) + C (n-1, k)	
n							C (n,k)

Tabla 4.2. Almacenamiento de valores intermedios

4.5.4 La subsecuencia común máxima

En muchos problemas se quiere conocer cuál es la composición de la solución óptima, es decir, los elementos que la conforman. En este caso, se deben conservar los valores de las soluciones parciales y la forma como se llega a ellas.

En el siguiente ejemplo se crea una tabla y a partir de ella se reconstruye la solución, tratándose del cálculo de la subsecuencia común máxima.

Ejemplo 1

Dada una secuencia X = $\{x_1\ x_{2\ldots}\ x_m\}$, se dice que Z = $\{z_1\ z_{2\ldots}\ z_k\}$, es una subsecuencia de X (siendo k ≤ m) si existe una secuencia creciente $\{i_1\ i_{2\ldots}\ i_k\}$ de índices de X tales que para todo j = 1,2,…, k se tiene $X_{ij} = Z_j$

Dadas dos secuencias, X y Y, se dice que Z es una subsecuencia común de X y Y si es subsecuencia de X y subsecuencia de Y. Se debe determinar la subsecuencia de longitud máxima común a dos secuencias.

Solución:

A la longitud de la secuencia común máxima de las secuencias $X_i\ y\ Y_j$ se llamará L (i,j), siendo X_i el i – ésimo estipulo de X (es decir, $X_i = \{x_1\ x_{2\ldots}\ x_i\}$) y Y_j el j – ésimo estipulo de Y, (es decir, $Y_j = \{y_1\ y_{2\ldots}\ y_j\}$).

Se puede plantear la solución como una sucesión de decisiones, aplicando el principio de óptimo en el que en cada marcha se establece si una representación forma parte o no de la secuencia común máxima. Comenzando por los últimos caracteres de las dos secuencias X y Y, la solución viene dada por la siguiente relación en recurrencia:

$$L\ (i,j) = \begin{cases} 0 & si\ i\ = 0\ o\ j = 0 \\ L\ (i\ -1, j-\ 1) + 1 & si\ i \neq 0, \quad j \neq 0\ y\ x_i = \ y_j \\ Max\ \{L\ (i, j-1), L\ (i-1, j)\} & si\ i \neq 0, \quad j \neq 0\ y\ x_i \neq \ y_j \end{cases}$$

La salida recursiva es de orden exponencial y, por tanto, se construirá una tabla con los valores L (i,j) para evitar la repetición de cálculos ayudados de la programación dinámica. Para la construcción de la tabla suponga que X y Y son las sucesiones de valores:

X = {1 0 0 1 0 1 0 1}

Y = {0 1 0 1 1 0 1 1 0}

		0	1	2	3	4	5	6	7	8
			1	0	0	1	0	1	0	1
0		0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
2	1	0	1	1	1	2	2	2	2	2
3	0	0	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	4	4	4
5	1	0	1	2	2	3	3	4	4	5
6	0	0	1	2	3	3	4	4	5	5
7	1	0	1	2	3	4	4	5	5	6
8	1	0	1	2	3	4	4	5	5	6
9	0	0	1	2	3	4	5	5	6	6

Tabla 4.3. Cálculo de la subsecuencia común máxima.

La tabla se va construyendo por filas y rellenando de izquierda a derecha. En cada $L[i,j]$ hay dos datos: uno para la construcción de lo que viene después de la secuencia óptima y otro necesario que corresponde a la longitud de cada subsecuencia.

La salida a lo que viene después de la secuencia normal máxima de las secuencias X y Y, se encuentra en el extremo inferior derecho ($L[9,8]$) y, por tanto, su longitud es 6. Si se quiere obtener lo que viene después de la secuencia, se debe recorrer la tabla (zona sombreada) a partir de esta posición y seguir la información que indica cómo obtener las longitudes óptimas a partir de su procedencia (izquierda, diagonal o superior).

4.5.5 El problema del camino de mínimo costo

Un grafo es un par de conjuntos, V y E, de los cuales el primero contiene los vértices o nodos y el segundo es el conjunto de ramas o arcos y está formado por pares de elementos de V que se conectan entre sí. Usualmente un grafo se puede denotar como G = (V,E).

Se dice que un grafo es dirigido cuando los elementos (u,v) ϵ E son pares ordenados. De esta forma las ramas se pueden notar como u v.→

Un grafo es acíclico cuando no forma ciclos, es decir, no existe ningún camino que comience y termine en el mismo vértice.

El problema de hallar el camino de costo mínimo en un grafo dirigido, es un caso específico de lo que se llama problema de decisión en múltiples pasos. La formulación general de este problema es la siguiente:

Un juicio es vigilado, habitualmente, a tiempos t = 0,1, ... Como resultado de dicha evaluación se consigue el importe de una variable (o, eventualmente, un vector de variables) X que sirve para calificar la situación del juicio. Al par u = (t, x(t)) se le conoce como estado.

Después de cada observación de X debe aplicar una acción correctiva, tomada de un conjunto de posibles disposiciones D(u). La escogencia de una medida particular d(u) da como resultado una transformación T que da como consecuencia un nuevo estado: v = (t + 1, ˜x) = T(u, d), y que tiene incorporado un costo c(u, d). Una situación que indique para cada estado u una decisión concreta a establecer recibe el nombre de política. Se pretende encontrar una política, de tal manera que la suma de los costos de las transformaciones engendradas por las continuas decisiones sea mínima (su resultado), y recibe el nombre de política óptima.

En el problema del camino de mínimo costo, los nódulos del grafo desempeñan el papel de estados. Así, para cada u ∈ V, el acumulado D(u) de las optativas decisiones está conformado por los vértices v tales que (u, v) ∈ E, mientras que las transformaciones engendradas por una decisión son las ramas (u, v) ∈ E, y el costo de la transformación es el costo de la rama. Una política es, en este caso, una regla que indica a qué nodo pasar si se está en un vértice dado cualquiera, siendo la política óptima aquella que da, para cada u ∈ V un camino de costo mínimo que finalice en un vértice terminal.

A continuación se observa un grafo con los costos correspondientes de ir de un origen i a un destino j.

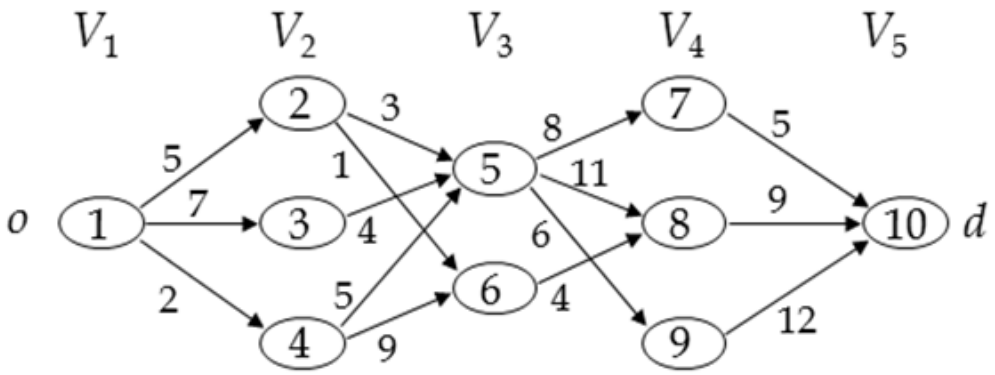


Gráfico 4.1. Grafo multietapa

Se denota por $c(u,v)$ el costo del arco (u,v).

Se debe hallar un camino de costo mínimo que vaya del nódulo *0* al nódulo *d*.

Cada V_i define una etapa del grafo por lo que todo recorrido del nódulo 0 al nódulo d tiene exactamente un vértice en cada V_i.

Ejemplo 2

Maximizar el beneficio total asignando el recurso a los r proyectos cuando se dispone de n unidades de un recurso que deben asignarse a r proyectos. Se conoce que si se dan j unidades ($0 \le j \le n$) al proyecto i resulta un beneficio $N_{i,j}$.

Primero se formulará como grafo multietapa:

- r+1 es el número de etapas.
- El proyecto i está representado por la etapa i, $1 \le i \le$ r.
- En cada etapa i, $2 \le i \le r$ hay n+1 vértices $v_{i,j}$, $0 \le j \le n$.
- Los vértices de las etapas 1 y r+1 son respectivamente, $o=v_{1,0}$ y $d=v_{r+1,n}$.
- Representa el estado en el que se dan un total de j unidades del recurso a los proyectos 1, 2, …, i-1 (vértice $v_{i,j}$, $2 \le i \le r$).
- Para todo $j \le l$ y $1 \le i < r$ los arcos son de la forma $(v_{i,j},v_{i+1,l})$.
- El costo asignado $N_{i,l-j}$ al arco $(v_{i,j},v_{i+1,l})$, $j \le l$, corresponde a asignar l-j unidades del recurso al proyecto i, $1 \le$ i $< r$.
- El costo $N_{i,l}$ es asignado a los arcos de la forma $(v_{r,j},v_{r+1,n})$.

Segundo, se formulará como grafo resultante para r=3 y n=4.

- El camino de costo máximo del nódulo o al nódulo d definirá la asignación óptima.
- Al cambiar los signos de las etiquetas, el convertir a en un problema de camino de costo mínimo cambiar los signos de las etiquetas.

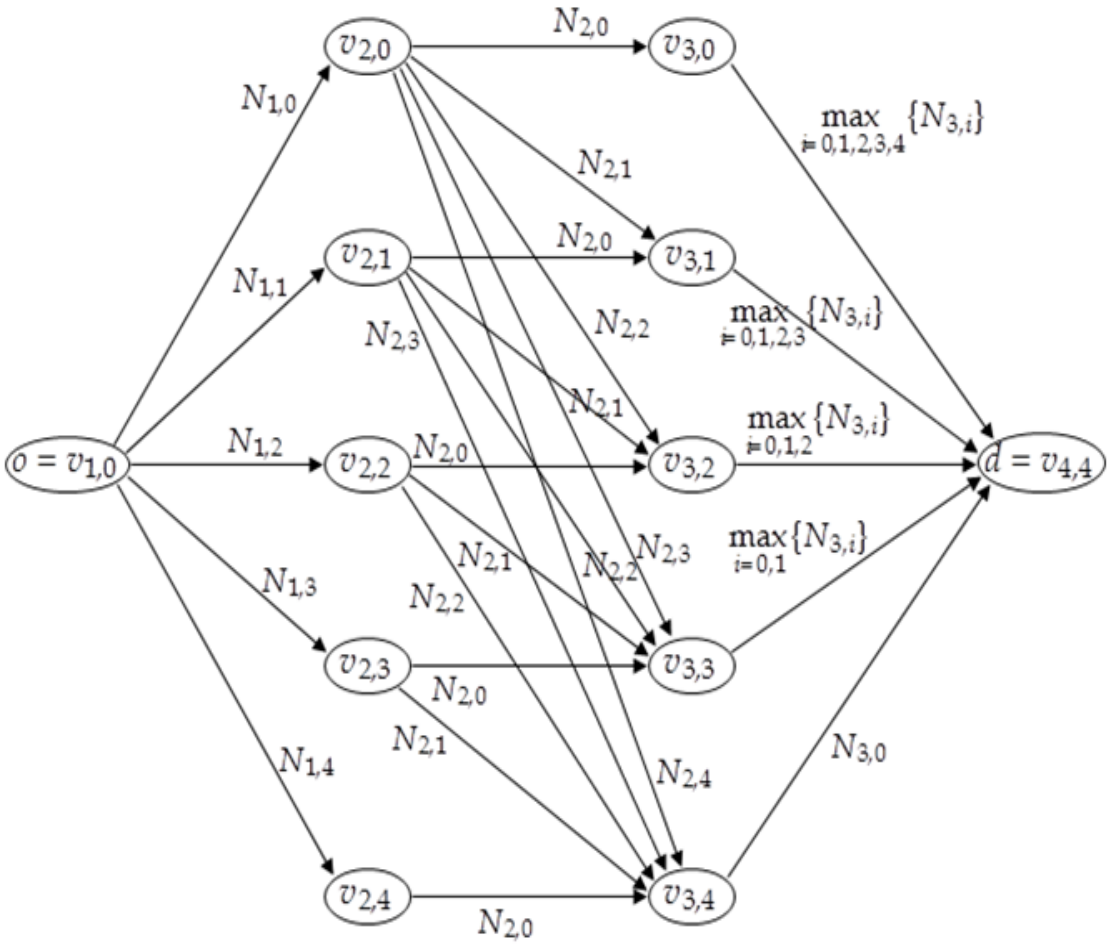


Gráfico 4.2. Formulación grafo multietapa

Solución:

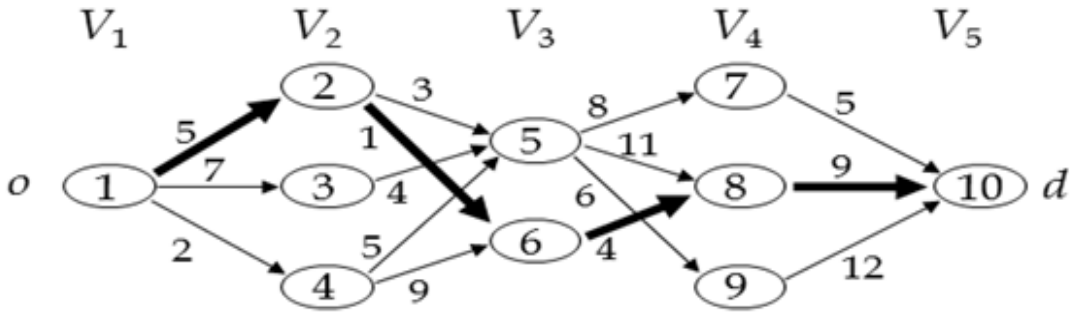


Gráfico 4.3. Solución grafo multietapa.

- Cada camino del nódulo o al nódulo d es el producto de una cadena de k-2 decisiones.

- Decisión i-ésima: determinar, a partir de un vértice v_i de V_i, un arco que tenga como origen a v_i y algún nódulo como destino a V_{i+1}.

- Principio de optimalidad: El camino de costo mínimo debe contener caminos después del camino de costo mínimo, entre otros nódulos.

Ecuación de recurrencia hacia adelante:

Sea $s(i,j)$ un camino de costo mínimo $C^*(i,j)$ desde el vértice j del conjunto V_i hasta el vértice destino d.

Entonces:

$$C(k-1,j) = \begin{Bmatrix} c\,(j,d), & si\ (j,d) \in A \\ \infty & en\ otro\ caso \end{Bmatrix}$$

$$C(i,j) = \min_{\substack{t \in V_{i+1} \\ (j,t)\in A}} \{c\,(j,1) + C(i+1,l)\}, para\ 1 \leq i \leq k-2$$

$$C(3,5) = \min\{8 + C(4,7), 11 + C(4,8), 6 + C(4,9)\} = 13$$

$$C(3,6) = 4 + C(4,8) = 13$$

$$C(2,2) = \min\{3 + C(3,5), 1 + C(3,6)\} = 14$$

$$C(2,3) = 4 + C(3,5) = 17$$

$$C(2,4) = \min\{5 + C(3,5), 9 + C(3,6)\} = 18$$

$$C(1,1) = \min\{5 + C(2,2), 7 + C(2,3), 2 + C(2,4)\} = 19$$

Falta recolectar las decisiones creadas en cada etapa que minimizan el costo:

- El valor de l que minimiza $c\,(j,l) + C*(i+1,l)$ es $D(i,j)$
- Entonces, el camino de costo mínimo es:
  v_1=1; v_2=D(1,1); v_3=D(2,D(1,1))……; etc.

D (3,5) = 7;
D (3,6) = 8
D (2,2) = 6
D (2,3) = 5
D (2,4) = 5
D (1,1) = 2
$V_1 = 1$
V_2 = D (1,1) = 2
V_3 = D (2,D(1,1)) = 6
V_4= D (3, D(2, D(1,1))) = 8

Ejemplo 3

El gerente de Tortillas Texanas ha estado trabajando para obtener un contrato con una de las compañías de comida rápida tipo mexicana más importante en Bogotá, Taco Bell. La fábrica de Tortillas Texanas es una compañía ubicada en Bogotá. En ella se fabrican tortillas de maíz y trigo. Las tortillas de maíz son utilizadas en gorditas, tacos y chalupas; las tortillas de trigo se utilizan en las quesadillas mexicanas. En días pasados se conoció que la compañía Taco Bell quería comprar 200 cajas (cada caja contiene 1.000 tortillas de trigo) a Tortillas Texanas en cada uno de los próximos 7 meses a un precio fijo.

Inmediatamente empezó a calcular las ganancias que Tortillas Texanas podría tener, llegando a la conclusión de que la compañía hará $1.000 cada mes si la empresa produce exactamente la cantidad de tortillas necesitadas en ese mes. Además, calculó la ganancia que podría percibir por fabricar más cajas necesitadas para un sólo mes. Esto se realizó suponiendo que las tortillas puedan ser producidas y congeladas hasta 4 meses (máximo). Tortillas Texanas, según estudios propios, puede fabricar más tortillas para un sólo mes si se trabaja tiempo extra y subcontratando. Esto no tendría que afectar la producción de otros productos. También ha calculado las utilidades por la producción de cantidades de tortillas (tamaños de lote), como se describe a continuación:

Tamaño del lote (cajas)	Ganancia/ miles
200	$1000
400	$2500
600	$3750
800	$4750

Tabla 4.4. Tamaños de lote

Debido a la combinación de factores económicos y de manejo de inventarios, existen economías de escala en la producción del tamaño de lote grande, pero también existen los costos (congelar las tortillas, deterioro de algunas tortillas). Pasando de un periodo de 2 meses, el deterioro por congelación se incrementa notoriamente.

Debido a que el gerente desea maximizar la ganancia total posible para la empresa en un contrato de 7 meses con Taco Bell, él desea producir el tamaño de lote que lo conducirá a esto.

Función objetivo: El problema de producción de tortillas Taco Bell requiere ser separado en una secuencia de decisiones, ya que la programación lineal no siempre considera decisiones sobre el tiempo.

Este problema requiere una secuencia de decisiones que sean realizadas con respecto al tiempo y los parámetros (demanda, ganancias, etc.) requieren ser asumidos con certeza.

La primera pregunta es saber qué tanto se debe producir cada mes que se puedan maximizar las ganancias totales para un lapso de tiempo de 7 meses y que sabiendo que las tortillas se echan a perder después de 4 meses se debe considerar la restricción de fabricar hasta 4 meses la demanda.

Sea:

X1 = demanda de un mes (200 cajas).
X2 = demanda de dos meses (400 cajas).
X3 = demanda de tres meses (600 cajas).
X4 = demanda de cuatro meses (800 cajas).

La formulación del problema será:

Maximizar:

$$1000X1+ 2500X2+ 3750X3+ 4750X4$$

Sujeto a:

$$X1+ 2X2+ 3X3+ 4X4 = 7$$
$$Xi = 0,1,2,3,... \text{ para toda i}$$

La programación dinámica enfoca este problema como una secuencia de decisiones y trabaja para tomar una serie de decisiones que conduzcan a las mayores ganancias, sujeto a cualquier restricción que exista en el problema.

Decisiones del séptimo mes

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en miles
7	200	200	$1000	0	0	$1000

Tabla 4.5. Decisión del séptimo mes

Decisiones del sexto mes:

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en miles
6	400	200	$1000	200	$1000	$2000
		400	$2500	0	0	$2500

Tabla 4.6. Decisiones de sexto mes

Decisiones del quinto mes:

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en Miles en miles
5	600	200	$1000	400	$2500	$3500
		400	$2500	200	$1000	$3500
		600	$3750	0	0	$3750

Tabla 4.7. Decisiones del quinto mes

Decisiones del cuarto mes:

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en miles
4	800	200	$1000	600	$3750	$4750
		400	$2500	400	$2500	$5000
		600	$3750	200	$1000	$4750
		800	$4750	0	0	$4750

Tabla 4.8. Decisiones del cuarto mes

Decisiones del tercer mes:

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en miles
3	1000	200	$1000	800	$5000	$6000
		400	$2500	600	$2750	$6250
		600	$3750	400	$2500	$6250
		800	$4750	200	$1000	$5750

Tabla 4.9. Decisiones de tercer mes

Decisiones del segundo mes:

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en miles
2	1200	200	$1000	1000	$6250	$7250
		400	$2500	800	$5000	$7500
		600	$3750	600	$3750	$7500
		800	$4750	400	$2500	$7250

Tabla 4.10. Decisiones del segundo mes

Decisiones del primer mes:

Mes	Demanda	Opciones de producción	Ganancia inmediata	Demanda restante	Mejor ganancia sobre la demanda restante	Ganancia total en miles
1	1400	200	$1000	1200	$7500	$8500
		400	$2500	1000	$6250	$8750
		600	$3750	800	$5000	$8750
		800	$4750	600	$3750	$8500

Tabla 4.11 Decisiones del primer mes

El programa de máximas ganancias es:

Producir en cajas mensualmente:

Mes 1	**400**	**400**	**600**
Mes 2	0	0	0
Mes 3	400	600	0
Mes 4	0	0	400
Mes 5	600	0	0
Mes 6	0	400	400
Mes 7	0	0	0

Tabla 4.12. Producir en cajas mensualmente

Problema de camino simple

Hace referencia a la búsqueda de un camino simple de longitud máxima a partir de un grafo.

Ejemplo 4

La siguiente ilustración emula una ciudad con una distribución de calles definida:

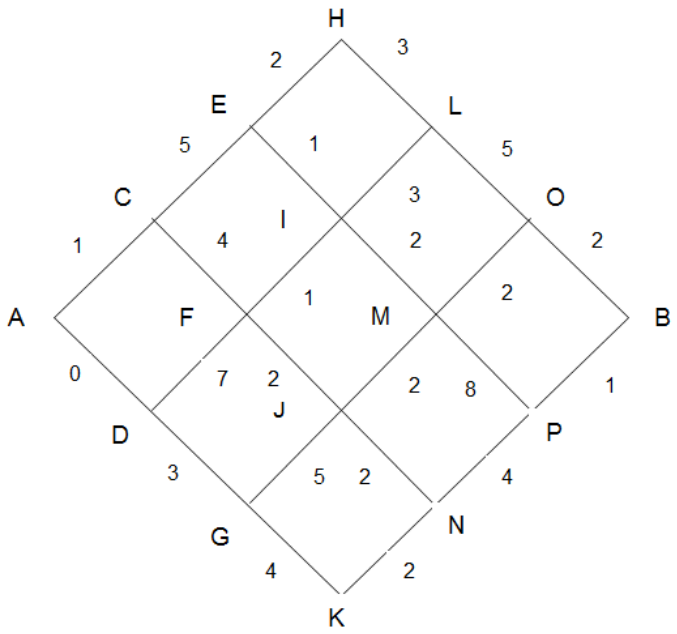


Gráfico 4.4. Problema de camino simple

Se desea encontrar el mínimo de esfuerzo total para ir de la calle A a la calle B. Este problema se puede resolver enumerando todas las alternativas posibles de caminos de la calle A a la calle B; sumando esfuerzos, bloque tras bloque de cada camino y escogiendo la suma más pequeña. Todas las calles tienen un sólo sentido y el número representado en el mapa hace ver el esfuerzo (usualmente tiempo y algunas veces costo de distancia) requerido para atravesar cada bloque individualmente de la ciudad.

Existen 20 caminos diferentes, 100 adiciones y 19 comparaciones.

$\begin{bmatrix} X \\ X/2 \end{bmatrix}$ 20 caminos diferentes

$[X-1]\begin{bmatrix} X \\ X/2 \end{bmatrix}$ 100 adiciones

$X\begin{bmatrix} X \\ X/2 \end{bmatrix}-1$ 19 comparaciones

Donde X es el número de etapas de la calle A a la calle B.

Aplicando programación dinámica, se puede tomar:

Sc como el esfuerzo mínimo de la calle C a la calle B

SD como el esfuerzo mínimo de la calle D a la calle B

No se sabe si ir diagonalmente en dirección hacia arriba o hacia abajo; adicionalmente se conocen 2 números.

- El mínimo esfuerzo para llegar de la calle C a la calle B por el mejor camino.
- El esfuerzo total requerido para llegar de la calle D a la calle B por el mejor camino.

Se añade esfuerzo mínimo de la calle C a la calle B, el esfuerzo requerido para ir de la calle A a la calle C, resultando el esfuerzo del mejor camino y empezando diagonalmente hacia arriba. Igualmente, se le hace al esfuerzo mínimo de la calle D a la calle B más el esfuerzo para ir de la calle A a la calle D, pero diagonalmente hacia abajo.

Para encontrar el esfuerzo total mínimo a la primera mejor decisión se comparan las sumas.

El esfuerzo mínimo de la calle C a la calle B y el esfuerzo mínimo de la calle D a la calle B aún son desconocidas.

Únicamente los esfuerzos a lo largo del mejor camino de la calle C a la calle B y de la calle D a la calle B son relevantes para los cálculos anteriores y los esfuerzos de los restantes caminos (9 de cada uno) no necesitan ser calculados.

Propiedad de un camino óptimo: Una vez que el primer paso es tomado, los pasos faltantes deben arrojar una solución óptima para el nuevo problema, con un reducido número de etapas. De esta manera:

$$S_A = \min_{0+\,S_D} 1 + S_c$$

La segunda idea principal indica que mientras en los dos números el esfuerzo mínimo de la calle C a la calle B y el esfuerzo mínimo de la calle D a la calle B son desconocidos inicialmente, se puede calcular el esfuerzo mínimo de la calle C a la calle B (si son desconocidos), el esfuerzo mínimo de la calle E a la calle B y el esfuerzo mínimo de la calle F a la calle B.

El esfuerzo mínimo de la calle C a la calle B:

$$\boldsymbol{S_c = \min_{4+\,S_F} 5 + S_E} \quad \textbf{y}$$

El esfuerzo mínimo de la calle D a la calle B:

$$\boldsymbol{S_D = \min_{3+\,S_G} 7 + S_F}$$

Para poder obtener el esfuerzo mínimo de la calle E a la calle B, el esfuerzo mínimo de la calle F a la calle B y el esfuerzo mínimo de la calle G a la calle B, es necesario calcular el esfuerzo mínimo de la calle I a la calle B, el esfuerzo mínimo de la calle J a la calle B, el esfuerzo mínimo de la calle H a la calle B y el esfuerzo mínimo de la calle K a la calle B y estos dependen del esfuerzo mínimo de la calle L a la calle B, el esfuerzo mínimo de la calle M a la calle B y el esfuerzo mínimo de la calle N a la calle B, los cuales, a su vez, dependen del esfuerzo mínimo de la calle O a la calle B y el esfuerzo mínimo de la calle P a la calle B.

S_O y S_P pueden ser calculados y trabajados hacia atrás de O y P.

$S_O = 2$ $S_P = 1$

$2 + S_O = 4$

$S_L = 5 + S_O = 7$

$S_M = \min$

$8 + SP = 9$

$SH = 3 + SL = 10$ $SN = 4 + SP = 5$

$2 + 5M = 6$

$SK = 2 + 5N = 7$

$SJ = \text{Min}$

$2 + 5N = 7$

$3 + SL = 10$

$SI = \text{Min}$

$4 + SM = 8$

$2 + SH = 12$ $1 + SI = 9$

$SE = \text{Min}$ $SF = \text{Min}$

$1 + SI = 9$ $2 + SJ = 8$

$5 + SJ = 11$ $7 + SF = 15$

$SG = \text{Min}$ $SD = \text{Min}$

$4 + SK = 11$ $3 + 5G = 14$

$5 + SE = 14$ $1 + SC = 13$

$SC = \text{Min}$ $SA = \text{Min}$

$4 + 5F = 12$ $0 + SD = 14$

Punto óptimo próximo de cada punto inicial:

PO = B PP = B

PL = O PM = O PN = P

PH = L PI = M PJ = M PK = N

PE = I PF = J PG = J ó K

PA = C PD = G

PC = F

El camino óptimo es: A C F J M O B; 1+4+2+2+2+2 = 13

Se puede resolver este problema empleando la programación en retroceso o la programación en avance.

Programación en retroceso

Ejemplo 5

Se desea encontrar el mínimo de esfuerzo total para ir de la calle A a la calle B. Este problema se puede resolver también transformando el mapa de la ciudad en un procedimiento de ejes, donde el punto A tiene los ejes (0,0), B tiene (6,0) e I tiene (3,1), entre otros, todas las calles tienen un sólo sentido y el número representado en el mapa evidencia el esfuerzo requerido para atravesar cada bloque individualmente de la ciudad.

La función objetivo es ahora una función de un par de números (x,y).

S(x,y)= El valor del camino de mínimo-esfuerzo uniendo el vértice (x,y) con el vértice (6,0).

De esta forma se puede definir:

a_u(x,y) es el esfuerzo asociado con el arco, conectando el vértice de (x,y) a (x+1,y+1)

a_d (x,y) es el esfuerzo del arco que va diagonalmente hacia abajo de (x,y) a (x+1,y-1)

u significa flecha hacia arriba.

d significa hacia abajo.

a_u (x,y) o ad (x,y) =% ∞ si tal arco no existe en la red.

Ejemplo: a_u(7,8) = ∞ %, A_d(4,2)=% ∞

Principio de optimalidad:

$$S(x,y) = \min \begin{cases} a_u(x,y) + S(x+1, y+1) \\ a_d(x,y) + S(x+1, y-1) \end{cases}$$

Condición limitante: S (6,0) = 0

Programación de avance

Sea S(x,y)= El valor del camino de mínimo-esfuerzo uniendo el vértice (0.0) con el vértice (x,y) a (x,y) con el vértice (0,0).

Relación recursiva:

$$S(x,y) = \min \begin{matrix} a_u\,(x-1, y-1) + S(x-1, y-1) \\ a_d\,(x-1, y+1) + S(x-1, y+1) \end{matrix}$$

Condición limitante S (0,0) = 0

Ejemplo 6

Por medio de la programación dinámica resolver un problema, encontrando f6(s)*, f5(s)*, f4(s)*, f3(s)*,f2(s)* y f1(s)*, conociendo que por definición: Xn (para cualquier etapa n = 1, hasta 6) sean las variables de decisión en n. La ruta elegida es 1=>X1=>X2=>X3=> X4=> X5=>X6. Se está en s y se selecciona Xn como el destino inmediato dado s y ns, conociendo:

.fn (s,xn) el costo total de las etapas restantes.

Xn* el valor de Xn que minimiza fn(s,Xn).

Valor mínimo fn*(s)=fn(s,Xn*)

Relación recursiva:

$f_n(s,X_n)=d_S,X_n+f_{n+1}*(X_n)$

Condición limitante:
$f_6(s,X_6)=f(X_6)$

Decisión:

X_6 $f_6(s,X_6)=f(X_6)$

Variable de Estado

Decisión:

X_5 $f_5(s,X_5)=d_S x_5+f_6(X_5)*$

Decisión óptima - Distancia mínima óptima

S	O	P	$F_5(S)*$	X_5*
B	0	2	0	-
L	5+2=7	-	7	0
P	1	P	-	0
M	2+2=4	8+1=9	4	0
N	-	4+1=5	5	P

X_4 $f_4(s,X_4)=d_S X_4+f*5(X_4)$

S	L	M	N	F4 * (s)X4*
H	3+7=10	-	-	10L
I	3+7=10	4+4=8	-	8M
J	-	2+4=6	2+5=7	6M
K	-	-	2+5=7	7N

X_3 $f_3(s,X_3)=d_S X_3+f_4(x_3)*$

S	H	I	J	K	F3 (s)*	X3*
E	2+10=12	1+8=9	-	-	9	I
F	-	1+8=9	2+6=8	-	8	J
G	-	-	5+6=11	4+7=11	11	J o K

X_2 $f_2(s,X_2)=d_S X_2+f_3(X_2)*$

S	E	F	G	$F_2(s)*$	X_2
C	5+9=14	4+8=12	-	12	F
D	-	7+8=15	3+11=14	14	G

X_1 $f(s,X_1)=d_s X_1+f(X_1)*$

S	C	O	f1(S)*	X1
A	1+12=13	0+14=14	13	C

Ruta óptima A - C - F - J - M - O - B

Con un costo total de 13.

4.5.6 Asignación de recursos

Un problema simple de localización de recursos consiste en N tablas de datos $r_i(x)$ (para i=1,...N) y X=0,1,...X) que representan la llegada por localizar X unidades del recurso en la actividad i.

El problema consiste en que el retorno total sea máximo.

N

Σri (x)

i=1

Sujeto a:

N

ΣXi =X, El coeficiente inscrito a Xi no aparece y es la unidad.

i=1

Xi ≥$ 0 y Enteros

Habiendo elegido las actividades caprichosamente (números fijos) de 1 a N, se eligen X elementos del recurso y primero se sitúa X_1 elementos en la actividad 1. Luego se sitúan X_2 elementos de las restantes $X-X_1$ elementos del recurso a la actividad 2, entonces X_3 elementos del recurso a la actividad 3, etc.

4.5.7 Función de Valor Óptimo

Sea S(X,K)=/$_K$(X) el máximo reingreso obtenido desde las actividades K hasta N, dados X elementos del recurso remanente para ser asignados. La asignación total hecha hasta ahora será la variable de estado. Y K es igual al número de actividades hasta ahora trabajadas.

Relación recursiva:
$S(X,K)=/_k(X)=Max\ [r_K(X_K)+/_{K+1}(X- X_k\}]$

$S(X,K)=Max\ [r^K(X^K)+S(X-X^K, K+1)]$

Donde X_k= 0,1,..., X es la retribución en la actividad K y $f_k(x)$ debe ser calculada para X_k=0,1,...,X

La condición limitante $f_N(x)=r_N(x)$

Ejemplo 7

max $3x_1^2 + 4x_2^2 + 5x_3^2$

Sujeto a:

x1+ x2+ x3#≤4

x´s $≥ 0 y Enteros

Retorno:

$$r_1 = 3x_1^2$$

$r_2 = 4x_2^2$

$$r_3 = 5x_3^2$$

Etapa 1 (Variable X1)		
f1(4) =	0	0+f2(4)=80 ←
	1	3+f2(3)=48
	2	12+f2(2)=32
	3	27+f2(1)=32
	4	48+f2(0)=48
Etapa 2 (Variable X2)		
f2(4) =	0	0+f3(4)=80 ←
	1	4+f3(3)=49
	2	16+f3(2)=36
	3	36+f3(1)=41
	4	64+f3(0)=64
f2(3) =	0	0+f3(3)=45 ←
	1	4+f3(2)=24

	2	16+f3(1)=21
	3	36+f3(0)=36
f2(2) =	0	0+f3(2)=20 ←
	1	4+f3(1)=9
	2	16+f3(0)=16
f2(2) =	0	0+f3(1)=5 ←
	1	4+f3(0)=4
Etapa 3 (Variable X3)		
r3(4)=f3(4)=80 r3(3)=f3(3)=45 r3(2)=f3(2)=20 r3(1)=f3(1)=5 r3(0)=f3()=0		
Asignación óptima: x1 y x2=0, x3=4 con un óptimo de 80.		

Tabla 4.5. Etapas asignación de recursos

Ejemplo 8

Resolver:

$$\text{Max Z} = 13\,x_1 + x_1^2 + + 5x_2^2 + 30.2x_2 + 10\,x_3 - 2.5x_3^2$$

Sujeto a:

$X_1 + X_2 \leq 5$

es equivalente a:

$2X_1 + 2X_2 \leq 10$

$2X_1 + 4X_2 + 5X_3 \# \leq 10$

Los valores de las variables son:

$x_1 = 0,1,,....,5$

$x_2 = 0,1,2$

$x_3 = 0,1,2$

$R_1(x_1) = 13x_1 - x_1^2 \quad R_2(x_2) = -5\,x_2^2 + 30.2x_2 \quad R_3(x_3) = 10x_3 - 2.5\,x_3^2$

f1(10) = Max	**0**	**0+/2(1,0)=40.2**	**30+ f2(10- Max{1(3),2(3)}=30+ f2(10-6)=30+ f2(4)** **30+ f2(10- Max{1(4),2(4)}=30+ f2(8-6)=36+ f2(2)**
	1	12+/2(8)=52.2	
	2	22+/2(6)=47.2	
	3	30+/2(4)=55.2←	
	4	36+/2(2)=36	
	5	40+/2(0)=40	
f2(10) = Max	0	0+/3 (10)=10	
	1	25.2+/3(6)=32.7	
	2	40.4+/3(2)=40.2←	
f2(8) = Max	0	0+/3(8)=7.5	
	1	25.2+/3(4)=25.2	
	2	40.4+/3(0)=40.2←	
f2(6) = Max	0	0+/3(6)=7.5	
	1	1+/3(2)=25.2←	
f2(4) = Max	0	0+/3(4)=0	
	1	25.2+/3(0)=25.2←	
f2(2) = 0			
f3(10) = Max	0	0	
	1	7.5	
	2	10←	5X3=10 X3=1
f3(8) = Max	0	0	
	1	7.5←	
f3(6) = Max	0	0	
	1	7.5←	5X3=6 X3=1
f3(4) =	2	0←	5X3=8 X3=1
f3(2) =		0←	
f3(0) =		0←	
X1=3, X2=1, X3=0			

Tabla 4.6. Asignación de recursos con varias restricciones.

4.5.8 Multiplicación de una secuencia de matrices

Calcular el producto matricial: $M = M1, M2, \dots Mn$

Existen varias maneras de realizar dicho cálculo. Se debe tener en cuenta que el algoritmo resultante de la definición del producto de dos matrices *pxq* y *qxr* necesita *pqr* multiplicaciones de escalares.

Ejemplo 9

Se quiere calcular el producto $ABCD$ de las matrices A(13x5), B(5x89), C(89x3) y D(3x34).

	No. multiplicadores
((AB)C) D)	10582
(AB) (CD)	54201
(A(BC))D	2856
A((BC)D)	4055
A(B(CD))	26418

Tabla 4. 7. No. multiplicadores

Metodología:

- Insertar los paréntesis en las matrices de todas las formas posibles.
- Calcular para cada matriz el número de multiplicaciones escalares necesarios.

Posibles formas $T(n)$ de insertar paréntesis:

Si es entre la i y la (i+1)-ésima:

$$M_1\ M_{2\ldots}\ M_i\ y\ M_{i+1}\ M_{i+2}\ \ldots M_n$$

Entonces se logran $T(i)T(n\text{-}i)$ formas distintas.

Con i entre 1 y n-1:

$$T\ (n) = \sum_{i=1}^{n-1} T\ (i)T\ (n-i), \qquad para\ n > 1$$

$$T\ (1) = 1$$

4.5.9 Aplicación del principio de optimalidad

Para realizar el producto se exige dividir, inicialmente, entre las matrices i e (i+1)-ésima, los productos:

$$M_1\ M_{2\ldots}\ M_i\ y\ M_{i+1}\ M_{i+2}\ \ldots M_n$$

Para que el total sea óptimo deberán ser realizados de forma óptima.

Método:

- Levantar la matriz $[m_{ij}]$, $1 \le i \le j \le n$, donde m_{ij} da como resultado el óptimo (el número de multiplicaciones escalares requeridas) para la parte $M_i\ M_{i+1}\ \ldots\ M_j$ del producto total.
- La solución final será definida por m_{1n}.
- Construcción de $[m_{ij}]$, $1 \le i \le \le n$:

– En un vector d, de $0\ldots n$ elementos, de manera que M_i presenta dimensiones d_{i-1}xd_i. se deben guardar las dimensiones de las M_i, $1 \le i \le n$,

– Los m_{ij} tales que $i\,j- = s$ están contenidos en la diagonal s de $[m_{ij}]$

$$s = 0: \quad m_{i,i} = 0, \quad para\ i = 1,2,\ldots,n$$

$$s = 1 \quad m_{i,i+1} = d_{i-1}d_i d_{i+1,} \quad para\ i = 1,2,\ldots,n-1$$

$$1 < s < n: \quad m_{i,i+s=} \min_{i \le k \le i+s-1}\left(m_{ik} + m_{k+1,i+s} + d_{i-1}d_k d_{i+s}\right),$$

– $para\ i = 1,2,\ldots,n-s$

– Intentar todas las posibilidades $(M_i\ M_{i+1}\ \ldots\ M_k)(M_{k+1}\ M_{k+2}\ \ldots\ M_{i+s})$ y escoger la mejor para calcular $M_i\ M_{i+1}\ \ldots\ M_{j+s}$
Una forma más trabajada:

$$m_{ij} = \begin{cases} 0, & si\ i = j \\ \min\limits_{i \le k \le j}\{m_{ik} + m_{k+1,j} + d_{i-1}d_k d_j\} & si\ i < j \end{cases}$$

Para el ejemplo anterior:

✓ A(13x5), B(5x89), C(89x3) y D(3x34)

- ✓ Se tiene d=(13,5,89,3,34).
- ✓ Para s=1: m_{12}=5785, m_{23}=1335, m_{34}=9078.
- ✓ Para s=2:

m_{13} = min (m_{11} + m_{23} + 13x5x3, m_{12} + m_{33} + 13x89x3)

= min (1530, 9256) = 1530

m_{24} = min (m_{22} + m_{34} + 5x89x34, m_{23} + m_{44} + 5x3x34)

= min (24208, 1845) = 1845

- ✓ **Para s=3:**

m_{14} = min ((k=1) m_{11} + m_{24} + 13x5x34,

(k=2) m_{12} + m_{34} + 13x89x34,

(k=3) m_{13} + m_{44} + 13x3x34)

= min (4055, 54201, 2856) = 2856

La matriz es:

	$j=1$	2	3	4	
$i=1$	0	5785	1530	2856	
					$s=3$
2		0	1335	1845	
					$s=2$
3			0	9078	
					$s=1$
4				0	
					$s=0$

Gráfica 4.5. Matriz

4.5.10 Planificación de trabajos

En un sistema para la realización de un grupo de trabajos se requiere la elaboración por parte de un equipo de operadores de una serie de tareas diferentes para cada trabajo.

Los trabajos n requieren cada uno m tareas:

$$T_{1i}, T_{2i}, \ldots, T_{mi}, 1 \leq i \leq n$$

Se requiere un tiempo t_{ji} para que el operador P_j, $1 \leq j \leq m$, realice la tarea T_{ji}

Programación para los n trabajos:

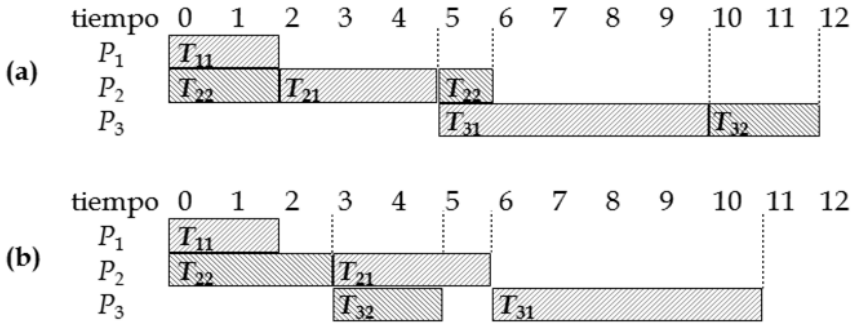
- En los operadores se realiza la asignación de tareas a intervalos de tiempo.
- A P_j se le debe asignar la tarea T_{ji}
- En cada instante de tiempo un operador r no puede tener más de una tarea asignada.
- Para todo trabajo i, hasta que $T_{j-1,i}$ haya terminado el procesamiento de T_{ji}, $j>1$, no puede empezar.

Ejemplo 10

Se desea programar la realización de dos trabajos en tres operadores, de forma que los tiempos de cada tarea vienen dados por:

$$T = \begin{matrix} 2 & 0 \\ 3 & 3 \\ 5 & 2 \end{matrix}$$

Dos programaciones factibles:



Gráfica4. 6. Planificación de trabajos

La programación (b) no es apropiada porque la operación de una tarea no se obstaculiza hasta que ésta haya terminado.

La programación (a) es apropiada porque el trabajo 1 se apropia de la operación 2 antes de que ésta termine con el trabajo 2.

Cuando todas las tareas del trabajo *i* han terminado, es el tiempo de culminación del trabajo *i* en la programación $f_i(S)$,

Programación (a), $f_1(S_a)$=10 y $f_2(S_a)$=12.

Programación (b), $f_1(S_b)$=11 y $f_2(S_b)$=5.

El tiempo de terminación, $f(S)$, de la programación S es:

$$F\ (S) = \max_{1 \le I \le N} \{f_i\ (S)\}$$

El tiempo medio de terminación, MFT(S), se puede definir como:

$$MFT\ (S)) = \frac{1}{n} \sum_{1 \le i \le n} f_i\ (S)$$

- Una programación no apropiada, S, para la que $F(S)$ es mínimo entre todas las programaciones no apropiadas, es una programación con tiempo de terminación óptimo (OFT) para un conjunto de trabajos.
- Una programación apropiada, S, para la que $F(S)$ es mínimo entre todas las planificaciones apropiadas, es una programación apropiada y con tiempo de terminación óptimo (POFT).
- Una programación no apropiada, S, para la que MFT(S) es mínimo entre todas las programaciones no apropiadas, es una programación con tiempo medio de terminación óptimo (OMFT).
- Una programación apropiada, S, para la que MFT(S) es mínimo entre todas las programaciones apropiadas, es una programación apropiada y con tiempo medio de terminación óptimo (POMFT).

Mediante programación dinámica se hallará el cálculo de OFT para m=2.

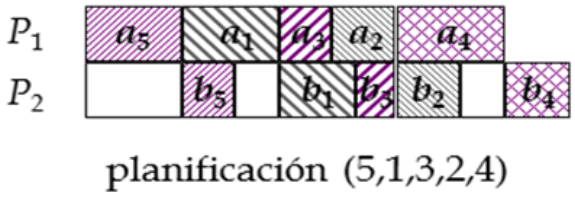
***m*=2:**

- T_{1i} como a_i
- T_{2i} como b_i.

Una programación está completamente especificada reglamentando una permuta de los trabajos en uno de los operadores (coincidirá con el otro operador). Cada tarea empezará tan pronto como sea posible.

Ejemplo 11

Realizar la planificación para 5 trabajos.



Gráfica 4.7. Planificación de 5 trabajos

Suponiendo que $a_i \neq 0$,

Se elabora inicialmente la planificación óptima para las actividades con $a_i \neq 0$ y después se agregan delante las actividades con $a_i = 0$.

El procesador 2 no estará disponible hasta el instante t, hasta cuando sea $g(S,t)$ la duración de una planificación óptima para el grupo de actividades S.

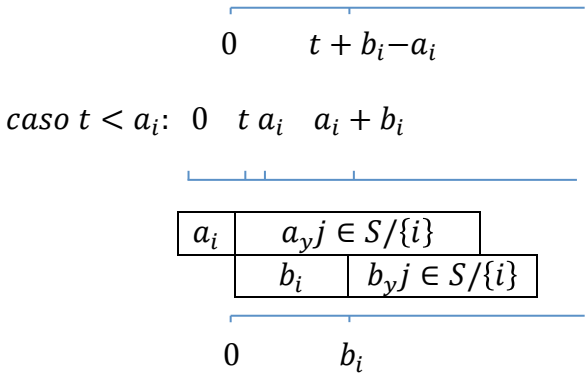
Entonces:

$$g(S,t) = \min_{i \in S}\{a_i + g(S \therefore \{i\}, b_i + max\{t - a_i, 0\})\}$$

Con $g(\emptyset,t) = \max\{t,0\}$ y $a_i \neq 0$, $1 \leq i \leq n$.

$$caso \geq a_i: \quad 0 \quad a_i \quad t \quad t + b_i =$$

a_i	$a_y j \in S/\{i\}$
b_i	$b_y j \in S/\{i\}$



Gráfica 8. Comparativos de planificación

Podría solucionarse de manera similar a la del problema del viajero de comercio la ecuación recursiva resultante, pero de una mejor manera.

Sean i y j los dos primeros trabajos, en ese orden en la planificación óptima del subconjunto S, entonces:

$$g'(S,t) = a_i + g(S \therefore \{i\}, b_i + max\{t - a_{i,}0\}) =$$

$$= a_i + a_j + g(S \therefore \{i,j\}, b_j + max\{b_i + max\{t - a_{i,}0\} - a_{j,}0\})$$

$$t_{ij}$$

$$t_{ij} = b_j + max\{t - a_{i,}0\} - a_{j,}0 =$$

$$= b_j + b_i - a_j + max\{max\{t - a_{i,}0\}, a_j - b_i\} =$$

$$= b_j + b_i - a_j + max\{t - a_{i,}a_{j,} - b_{i,}0\} =$$

$$= b_j + b_i - a_j - a_i + max\{t, a_{i,} + a_j - b_{i,}a_i\} =$$

Si los dos primeros trabajos fueran *j* e *i*:

$$g'(S,t) = a_j + a_i + g'(S \therefore \{j,i\}, b_i + b_j - a_i - a_j + max\{t, a_j + a_i - b_j, a_j\}$$

Entonces:

$$g(S,t) \leq g'(S,t) =$$

$$= \max\{t, a_i + a_j - b_i, a_i\} \leq \max\{t, a_j + a_i - b_j, a_j\}$$

De esta manera:

$$\max\{a_i + a_j - b_i, a_i\} \leq \max\{a_j + a_i - b_j, a_j\}$$

Es decir:

$$a_i + a_j + \max\{-b_i, -a_j\} \leq a_j + a_i + \max\{-b_i, -a_j\}$$

O sea:

$$\min\{b_i, a_j\} \geq \min\{b_i, a_j\} \quad (*)$$

Seguidamente se halla una planificación óptima en la que cada par de trabajos adyacentes (*i,j*) se verifica (*).

Todas las planificaciones que se verifican (*) tienen la misma longitud por lo que se pueden demostrar.

Por tanto, basta crear una permuta para que se cumpla la verificación (*) para todo par de trabajos adyacentes (*i,j)*.

Repitiendo el proceso, se puede construir la planificación óptima.

Por tanto, la solución es:

- En orden no decreciente se ordenan los a_i y b_i
- Si el trabajo *i* no ha sido planeado aún y el siguiente número de la secuencia es a_i planear el trabajo *i* en la posición más a la izquierda de entre los que restan;
  si en b_j el trabajo *j* no ha sido planeado todavía y el siguiente número es, planear el trabajo *j* en la posición más a la derecha de entre los que restan.

Resumen

Frente a una serie de problemas cuyas soluciones pueden ser expresadas recursivamente en términos matemáticos, posiblemente la manera más natural de resolverlos es mediante un método recursivo. Sin embargo, el tiempo de ejecución de una solución, normalmente de orden exponencial, puede mejorarse mediante la programación dinámica.

Para resolver un problema se pueden hacer muchas divisiones y obtener subproblemas independientes, de esta manera se puede llegar más fácil a la solución del problema original, sin embargo no todos los problemas se pueden resolver de esta forma, ya que cuando los subproblemas obtenidos no son independientes sino que existe solapamiento entre ellos, la solución no resulta ser eficiente por la repetición de cálculos que conlleva. En estos casos es cuando la programación dinámica ofrece una solución aceptable.

Bibliografía

- Bronson, R. (1993). Investigación de operaciones, México, Editorial McGraw-Hill.
- Chediak, F. (2005). Investigación de operaciones, Colombia Ibagué, Editorial El Poira.
- Izar, J.(2012). Investigación de operaciones, México, Editorial Trillas.
- Roscoe, D.(1984). Modelos cuantitativos para administración, México, Editorial Iberoamérica.
- Lieberman ,G.(2002). Investigación de operaciones. México, Editorial McGraw-Hill.
- Taha, H.(2008). Investigación de operaciones, México, Editorial Alfaomega.
- Winston, W. (2005). Investigación de operaciones, México, Editorial Thomson.
- http://www.lcc.uma.es/~av/Libro/CAP5.pdf
- http://www.andrew.cmu.edu/user/mgoic/files/documents/optimization/pdinamica.pdf
- http://cms.dm.uba.ar/materias/1ercuat2009/optimizacion/Maurette_Ojea.pdf
- http://www.angelfire.com/oz/rubincelis/Pdinamica.pdf
- http://webdiis.unizar.es/asignaturas/EDA/ea/slides/4-Programacion%20dinamica.pdf
- http://www.lcc.uma.es/~av/Libro/CAP5.pdf
- http://webdiis.unizar.es/asignaturas/EDA/ea/slides/4-Programacion%20dinamica.pdf
- http://www.angelfire.com/oz/rubincelis/Pdinamica.pdf
- http://www.angelfire.com/oz/rubincelis/Pdinamica.pdf
- http://cms.dm.uba.ar/materias/1ercuat2009/optimizacion/Maurette_Ojea.pdf