

13 : 4. DYNAMIC PROGRAMMING ALGORITHM

The computational procedure for solving a problem by dynamic programming approach can be summarized in the following steps :

Step 1. Identify the decision variables and specify objective function to be optimized under certain limitations, if any.

Step 2. Decompose (or divide) the given problem into a number of smaller sub-problems (or stages). Identify the state variables at each stage and write down the transformation function as a function of the state variables and decision variables at the next stage.

Step 3. Write down a general recursive relationship for computing the optimal policy. Decide whether forward or backward method is to follow to solve the problem.

Step 4. Construct appropriate stages to show the required values of the return function at each stage.

Step 5. Determine the overall optimal policy or decisions and its value at each stage. There may be more than one such optimal policy.

Remarks 1. Generally the solution of a recursive equation involves two types of computations, according as the system is continuous or discrete. In the first case the optimal decision at each stage is obtained by using the usual classical methods of optimization. In the second case, a tabular computational scheme is followed.

2. If the dynamic programming problem is solved by using the recursive equation starting from the first through the last stage, i.e., obtaining the sequence $f_1 \rightarrow f_2 \rightarrow \dots \rightarrow f_N$, the computation involved is called the *forward* computational procedure. If the recursive equation is formulated in a different way so as to obtain the sequence $f_N \rightarrow f_{N-1} \rightarrow \dots \rightarrow f_1$, then the computation is known as the *backward* computational procedure.

SAMPLE PROBLEM

1301. (Optimal Sub-division Problem) Divide a positive quantity c into n parts in such a way that their products is a maximum. [IAS 1994]

Or

Maximize $z = y_1 \cdot y_2 \cdot \dots \cdot y_n$ subject to the constraints :

$$y_1 + y_2 + \dots + y_n = c \text{ and } y_j \geq 0; \quad j = 1, 2, \dots, n$$

[IAS 1994; Meerut M.Sc. (Math.) 1993]

Solution. Let y_j be the j th part of the positive quantity c ($j = 1, 2, \dots, n$), then each j corresponding to part y_j may be regarded as a stage. Now, since y_j may assume any non-negative value satisfying the constraint

$$y_1 + y_2 + \dots + y_n = c,$$

the alternatives at each stage are infinite. This means that y_j may be considered to be continuous.

Let $f_n(c)$ denote the maximum attainable product when the quantity c is divided into n parts.

If we regard c as a fixed quantity and n as the number of stages, which varies over positive integers, then a recursive equation connecting $f_n(c)$ and $f_{n-1}(c)$ is

$$f_n(c) = \max_{0 < x \leq c} \{ x \cdot f_{n-1}(c-x) \}$$

For $n = 1$ (i.e., one-stage problem), we write

$$f_1(c) = \max_{y_1=c} \{ y_1 \} = c \text{ (initially true)}$$

For $n = 2$ (i.e., two-stage problem), the quantity c is divided into two parts, say $y_1 = x$ and $y_2 = c - x$. Then

$$\begin{aligned} f_2(c) &= \max \{ y_1 \cdot y_2 \} \\ &= \max_{0 < x \leq c} \{ x \cdot (c-x) \} \\ &= \max_{0 < x \leq c} \{ x \cdot f_1(c-x) \}, \text{ since } f_1(c-x) = c-x. \end{aligned}$$

Similarly, for $n = 3$, the quantity c is divided into three parts, given the initial choices of x which leaves $c - x$ to be further divided into two parts. Denote the maximum possible product for $(c - x)$ into two parts by $f_2(c - x)$. Then using the principle of optimality, we have

$$f_3(c) = \max_{0 < x \leq c} \{x \cdot f_2(c - x)\}.$$

Continuing in a similar manner, the recursive equation for general value of x is given by

$$f_n(c) = \max_{0 \leq x \leq c} \{x \cdot f_{n-1}(c - x)\}$$

We now solve the recurrence equation formulated above.

For $n = 2$, the function $x \cdot (c - x)$ attains its maximum value at $x = c/2$ satisfying the condition $0 < x \leq c$. Thus

$$f_2(c) = \frac{c}{2} \left(c - \frac{c}{2}\right) = \left(\frac{c}{2}\right)^2$$

\therefore The optimal policy is $(c/2, c/2)$ and $f_2(c) = (c/2)^2$.

For $n = 3$, $f_3(c) = \max_{0 \leq x \leq c} \left\{x \cdot \left(\frac{c-x}{2}\right)^2\right\}$, since $f_2(c - x) = \left(\frac{c-x}{2}\right)^2$.

Now, since the maximum value of $x \left(\frac{c-x}{2}\right)^2$ is attained for $x = c/3$ satisfying the condition $0 < x \leq c$; therefore

$$f_3(c) = \left\{\frac{c}{3} \cdot \frac{1}{4} \left(c - \frac{c}{3}\right)^2\right\} = \left(\frac{c}{3}\right)^3$$

Thus, for $n = 3$, we have

Optimal policy : $\left(\frac{c}{3}, \frac{c}{3}, \frac{c}{3}\right)$ and $f_3(c) = \left(\frac{c}{3}\right)^3$

In general, for n -stage problem, we assume that

Optimal policy : $\left(\frac{c}{n}, \frac{c}{n}, \dots, \frac{c}{n}\right)$ and $f_n(c) = \left(\frac{c}{n}\right)^n$, for $n = 1, 2, \dots, m$.

Now, for $n = m + 1$, the recursive equation is

$$\begin{aligned} f_{m+1}(c) &= \max_{0 < x \leq c} \{x \cdot f_m(c - x)\} = \max_{0 < x \leq c} \left\{x \cdot \left(\frac{c-x}{m}\right)^m\right\} \\ &= \left(\frac{c}{m+1}\right)^{m+1}, \end{aligned}$$

as the maximum value of $x \left(\frac{c-x}{m}\right)^m$ is attained at $x = \frac{c}{m+1}$, i.e., the result is also true for $n = m + 1$.

Hence, by mathematical induction, the optimal policy is

$$\left(\frac{c}{n}, \frac{c}{n}, \dots, \frac{c}{n}\right) \text{ and } f_n^0(c) = \left(\frac{c}{n}\right)^n.$$

1302. Use dynamic programming to show that

$z = p_1 \log p_1 + p_2 \log p_2 + \dots + p_n \log p_n$ subject to the constraints :

$$p_1 + p_2 + \dots + p_n = 1 \quad \text{and} \quad p_j \geq 0 \quad (j = 1, 2, \dots, n)$$

is a minimum when $p_1 = p_2 = \dots = p_n = 1/n$.

[Nagarjuna M.Sc. (Stat.) 1989; Meerut M.Sc. (Math.) 2000, 1995; M.S. Tirunvelli B.Sc. (Math.) 1993]

Solution. The problem here is to divide unity into n parts so as to minimize the quantity $\sum p_i \log p_i$.

Let $f_n(1)$ denote the minimum attainable sum of $p_i \log p_i$ ($i = 1, 2, \dots, n$).

For $n = 1$ (stage I), we have

$$f_1(1) = \min_{0 < x \leq 1} \{p_1 \log p_1\} = 1 \log 1$$

as unity is divided only into $p_1 = 1$ part.

For $n = 2$, the unity is divided into two parts p_1 and p_2 , such that $p_1 + p_2 = 1$.

If $p_1 = x$ and $p_2 = 1 - x$, then

$$\begin{aligned} f_2(1) &= \min_{0 < x \leq 1} \{p_1 \log p_1 + p_2 \log p_2\} \\ &= \min_{0 < x \leq 1} \{x \log x + (1-x) \log (1-x)\} \\ &= \min_{0 < x \leq 1} \{x \log x + f_1(1-x)\} \end{aligned}$$

In general, for an n -stage problem, the recursive equation is

$$\begin{aligned} f_n(1) &= \min_{0 < x \leq 1} \{p_1 \log p_1 + p_2 \log p_2 + \dots + p_n \log p_n\} \\ &= \min_{0 < x \leq 1} \{x \log x + f_{n-1}(1-x)\} \end{aligned}$$

We now solve this recursive equation.

For $n = 2$ (stage 2), the function $x \log x + (1-x) \log (1-x)$ attains its minimum value at $x = 1/2$ satisfying the condition $0 < x \leq 1$. Thus,

$$f_2(1) = \frac{1}{2} \log \frac{1}{2} + \left(1 - \frac{1}{2}\right) \log \left(1 - \frac{1}{2}\right) = 2 \left(\frac{1}{2} \log \frac{1}{2}\right).$$

Similarly, for stage 3, the minimum value of the recursive equation is obtained as

$$\begin{aligned} f_3(1) &= \min_{0 < x \leq 1} \{x \log x + f_2(1-x)\} \\ &= \min_{0 < x \leq 1} \left\{ x \log x + 2 \left(\frac{1-x}{2}\right) \log \left(\frac{1-x}{2}\right) \right\} \end{aligned}$$

Now, since the minimum value of

$$x \log x + 2 \left(\frac{1-x}{2}\right) \log \left(\frac{1-x}{2}\right)$$

is attained at $x = 1/3$ satisfying $0 < x \leq 1$, we have

$$f_3(1) = \frac{1}{3} \log \frac{1}{3} + 2 \left(\frac{1}{3} \log \frac{1}{3}\right) = 3 \left(\frac{1}{3} \log \frac{1}{3}\right).$$

\therefore Optimal policy is : $p_1 = p_2 = p_3 = \frac{1}{3}$.

In general, for n -stage problem we assume that

$$\text{Optimal policy : } p_1 = p_2 = \dots = p_n = \frac{1}{n} \quad \text{and} \quad f_n'(1) = n \left\{ \frac{1}{n} \log \frac{1}{n} \right\}.$$

This can be shown easily using mathematical induction.

For $n = m + 1$, the recursive equation is

$$\begin{aligned} f_{m+1}(1) &= \min_{0 < x \leq 1} \{x \log x + f_m(1-x)\} \\ &= \min_{0 < x \leq 1} \left[x \log x + m \left\{ \frac{1-x}{m} \log \left(\frac{1-x}{m} \right) \right\} \right] \\ &= \frac{1}{m+1} \log \frac{1}{m+1} + m \left\{ \frac{1}{m+1} \log \frac{1}{m+1} \right\} \\ &= (m+1) \left\{ \frac{1}{m+1} \log \frac{1}{m+1} \right\} \end{aligned}$$

since minimum of $x \log x + \frac{1-x}{m} \log \frac{1-x}{m}$ is attained at $x = \frac{1}{m+1}$.

Hence the required optimal policy is

$$\left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n} \right) \quad \text{with} \quad f_n^0(1) = n \left(\frac{1}{n} \log \frac{1}{n} \right).$$

1303. Use dynamic programming to solve the following problem :

Minimize $z = y_1^2 + y_2^2 + y_3^2$ subject to the constraints :

$$y_1 + y_2 + y_3 \geq 15 \quad \text{and} \quad y_1, y_2, y_3 \geq 0.$$

[IAS 1995]

Solution. Since the decision variables are y_1, y_2 and y_3 , the given problem is a three-stage problem defined as follows :

$$s_3 = y_1 + y_2 + y_3 \geq 15, \quad s_2 = y_1 + y_2 = s_3 - y_3 \quad \text{and} \quad s_1 = y_1 = s_2 - y_2$$

Therefore, the functional (recurrence) relation is

$$f_1(s_1) = \min_{0 \leq y_1 \leq s_1} y_1^2 = (s_1 - y_1)^2$$

$$f_2(s_2) = \min_{0 \leq y_2 \leq s_2} \{y_1^2 + y_2^2\} = \min_{0 \leq y_2 \leq s_2} \{y_2^2 + f_1(s_1)\}$$

and

$$f_3(s_3) = \min_{0 \leq y_3 \leq s_3} \{y_1^2 + y_2^2 + y_3^2\} = \min_{0 \leq y_3 \leq s_3} \{y_3^2 + f_2(s_2)\}$$

\therefore

$$f_2(s_2) = \min_{0 \leq y_2 \leq s_2} \{y_2^2 + (s_2 - y_2)^2\}$$

$$= \left(\frac{1}{2}s_2\right)^2 + \left(s_2 - \frac{1}{2}s_2\right)^2 = \frac{1}{2}s_2^2;$$

since the function $y_2^2 + (s_2 - y_2)^2$ attains its minimum value at $y_2 = \frac{1}{2}s_2$.

Again

$$f_3(s_3) = \min_{0 \leq y_3 \leq s_3} \{y_3^2 + f_2(s_2)\} = \min_{0 \leq y_3 \leq s_3} \left\{y_3^2 + \frac{1}{2}(s_3 - y_3)^2\right\}$$

or

$$f_3(15) = \min_{0 \leq y_3 \leq 15} \left\{y_3^2 + \frac{1}{2}(15 - y_3)^2\right\}, \text{ since } s_3(y_1 + y_2 + y_3) \geq 15.$$

Since the minimum value of the function $y_3^2 + \frac{1}{2}(15 - y_3)^2$ occurs at $y_3 = 5$; we have

$$f_3(15) = \left\{5^2 + \frac{1}{2}(15 - 5)^2\right\} = 75$$

Thus

$$s_3 = 15 \Rightarrow y_3^0 = 5$$

$$s_2 = s_3 - y_3 = 15 - 5 = 10 \Rightarrow y_2^0 = \frac{1}{2}s_2 = 5$$

$$s_1 = s_2 - y_2 = 10 - 5 = 5 \Rightarrow y_1^0 = s_1 = 5.$$

Hence the optimal policy is

$$(5, 5, 5) \text{ with } f_3^0(15) = 75.$$

PROBLEMS

Use dynamic programming to find the value of

1304. Maximum $z = y_1 \cdot y_2 \cdot y_3$ subject to the constraints :

$$y_1 + y_2 + y_3 = 5, y_1, y_2, y_3 \geq 0.$$

[Andhra B.E. (Mech. & Ind.) 1996; Purvanchal MCA 1996]

1305. Minimum $z = y_1 + y_2 + y_3 + \dots + y_n$ subject to the constraints :

$$y_1 y_2 \dots y_n = d; \quad y_j \geq 0 \quad \text{for } j = 1, 2, \dots, n.$$

[IAS 1992; Bharthiar M.Sc. (Math.) 1992]

1306. Use the principle of optimality to find the maximum value of

$$b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots + b_n x_n$$

when

$$x_1 + x_2 + x_3 + \dots + x_n = c$$

$$x_1, x_2, x_3, \dots, x_n \geq 0.$$

[Meerut M.Sc. (Math.) 2001]

1307. Find the minimum value of

$$z = y_1^2 + y_2^2 + \dots + y_n^2 \text{ subject to the constraints :}$$

$$y_1 \cdot y_2 \dots y_n = c$$

$$y_j \geq 0$$

for $j = 1, 2, \dots, n$

[Meerut M.Sc. (Math.) 1992]

1308. (a) Use dynamic programming to solve :

Maximize $z = x_1^2 + x_2^2 + \dots + x_{10}^2$ subject to the constraints :

$$x_1 \cdot x_2 \cdot \dots \cdot x_{10} = 8$$

$$x_1, x_2, \dots, x_{10} \geq 0.$$

(b) Minimise $z = x_1^3 + x_2^3 + x_3^3$ subject to the constraints :

$$x_1 x_2 x_3 = 27 \text{ and } x_i \geq 0, i = 1, 2, 3. \quad [\text{Madras M.E. (Structural) 2000}]$$

1309. Use dynamic programming to solve :

Minimize $z = y_1^2 + y_2^2 + y_3^2$ subject to the constraints :
 $y_1 + y_2 + y_3 = 10$ and

- (a) y_1, y_2 and y_3 are non-negative,
 (b) y_1, y_2 and y_3 are non-negative integers.

1310. Find the minimum value of

$x_1^2 + 2x_2^2 + 4x_3$ subject to the constraints :
 $x_1 + 2x_2 + x_3 \geq 8$
 $x_1, x_2, x_3 \geq 0$.

1311. Find the maximum value of $z = x_1^2 + 2x_2^2 + 4x_3$ subject to the constraints :
 $x_1 + 2x_2 + x_3 \leq 8, x_1, x_2, x_3 \geq 0$. [Meerut M.Sc. (Math.) 1989]

1312. Find the maximum value of $z = -x_1^2 - 2x_2^2 + 3x_2 + x_3$ subject to the conditions :
 $x_1 + x_2 + x_3 \leq 1, x_1, x_2, x_3 \geq 0$. [Meerut M.Sc. (Math.) 1999]

13 : 5. SOLUTION OF DISCRETE D.P.P.

Many problems such as production allocation, long-term planning, equipment replacement, multistage chemical processes, etc. can be solved by Dynamic Programming, using convenient tabular computations. This is best illustrated with the help of some sample problems.

SAMPLE PROBLEMS

1313. (Product Allocation Problem) The owner of a chain of four grocery stores has purchased six crates of fresh strawberries. The estimated probability distribution of potential sales of the strawberries before spoilage differ among the four stores. The following table gives the estimated total expected profit at each store, when it is allocated various numbers of crates :

	Store			
	1	2	3	4
0	0	0	0	0
1	4	2	6	2
2	6	4	8	3
3	7	6	8	4
4	7	8	8	4
5	7	9	8	4
6	7	10	8	4

For administrative reasons, the owner does not wish to split crates between stores. However, he is willing to distribute zero crates to any of his stores.

Find the allocation of six crates to four stores so as to maximize the expected profit.

Solution. Let the four stores be considered as four stages in a dynamic programming formulation. The decision variables x_j ($j = 1, 2, 3, 4$) denote the number of crates allocated as the j th stage from the previous one.

Now let $P_j(x_j)$ be the expected profit from allocation of x_j crates to store j . Then the problem can be formulated as an L.P.P. as follows :

Maximize $z = P_1(x_1) + P_2(x_2) + P_3(x_3) + P_4(x_4)$ subject to the constraints :

$$x_1 + x_2 + x_3 + x_4 = 6, x_1, x_2, x_3, x_4 \geq 0.$$

Let there be s crates available for j remaining stores and x_j be the initial allocation. Define $f_j(x_j)$ as the value of the optimal allocation for stores 1 through 4 both inclusive. Thus for stage $j = 1$,

$$f_1(s, x_1) = \{P_1(x_1)\}$$

If $f_j(s, x_j)$ be the profit associated with the optimum solution $f_j^*(s)$ ($j = 1, 2, 3, 4$) then

$$f_1^*(s) = \max_{0 \leq x_1 \leq s} P_1(x_1).$$

Thus the recurrence relation is

$$f_1(s, x_j) = P_j(x_j) + f_{j+1}^*(s - x_j)$$

for $j = 1, 2, 3, 4$

and

$$f_j^*(s) = \max_{0 \leq x_j \leq s} \{P_j(x_j) + f_{j+1}^*(s - x_j)\}.$$

The solution to this problem starts with $f_4^*(s)$ and is completed when $f_1^*(s)$ is obtained.

The computations for one-stage problem (i.e., for $j = 1$) are as follows :

s	$f_4^*(s)$	x_4^*
0	0	0
1	2	1
2	3	2
3	4	3
4	4	3, 4
5	4	3, 4, 5
6	4	3, 4, 5, 6

For $j = 2$, we have a two-stage problem. The computations are as follows :

$$f_3(s, x_3) = P_3(x_3) + f_4^*(s - x_3)$$

Optimum Sol.

s	x_3	0	1	2	3	4	5	6	$f_3^*(s)$	x_3^*
0	0 + 0								0	0
1	0 + 2	6 + 0							6	1
2	0 + 3	6 + 2	8 + 0						8	1, 2
3	0 + 4	6 + 3	8 + 2	8 + 0					10	2
4	0 + 4	6 + 4	8 + 3	8 + 2	8 + 0				11	2
5	0 + 4	6 + 4	8 + 4	8 + 3	8 + 2	8 + 0			12	2
6	0 + 4	6 + 4	8 + 4	8 + 4	8 + 3	8 + 2	8 + 0		12	2, 3

For $j = 3$, we have three-stage problem. So, we have

$$f_2(s, x_2) = P_2(x_2) + f_3^*(s - x_2)$$

Optimum Sol.

s	x_2	0	1	2	3	4	5	6	$f_2^*(s)$	x_2^*
0	0 + 0								0	0
1	0 + 6	2 + 0							6	0
2	0 + 8	2 + 6	4 + 0						8	0, 1
3	0 + 10	2 + 8	4 + 6	6 + 0					10	0, 1, 2
4	0 + 11	2 + 10	4 + 8	6 + 6	8 + 0				12	1, 2, 3
5	0 + 12	2 + 11	4 + 10	6 + 8	8 + 6	9 + 0			14	2, 3, 4
6	0 + 12	2 + 12	4 + 11	6 + 10	8 + 8	9 + 6	10 + 0		16	3, 4

For $j = 4$, we have the required four-stage problem :

$$f_1(s, x_1) = P_1(x_1) + f_2^*(s - x_1)$$

Optimum Sol.

s	x_1	0	1	2	3	4	5	6	$f_1^*(s)$	x_1^*
6	0 + 16	4 + 14	6 + 12	7 + 10	7 + 8	7 + 6	7 + 0		18	1, 2

From the above computations it is observed that the maximum profit of Rs. 18 can be obtained by choosing the following eight alternative solutions :

Store 1	Store 2	Store 3	Store 4
x_1^*	x_2^*	x_3^*	x_4^*
1	2		1
1	3	2	1
1	3	1	0
1	4	2	0
		1	0

2	1	2	1
2	2	1	1
2	2	2	0
2	3	1	0

1314. (Cargo Load Problem) A vessel is to be loaded with stocks of 3 items. Each unit of item i has a weight w_i and value r_i . The maximum cargo weight the vessel can take is 5 and the details of the three items are as follows :

i	w_i	r_i
1	1	30
2	3	80
3	2	65

Develop the recursive equation for the above case and find the most valuable cargo load without exceeding the maximum cargo weight by using dynamic programming.

Solution. We have to determine how many units of three items are to be loaded. So it is a three-stage problem. Let x_j ($j = 1, 2, 3$) denote the three decisions. Let $f_j(x_j)$ denote the value of the optimal allocation for the three types of items.

If $f_j(s, x_j)$ be the value associated with the optimum solution $f_j^*(s)$, $j = 1, 2, \dots, n$, then we have

$$f_1^*(s) = \max_{0 \leq x_1 \leq s} f_1(s, x_1)$$

and

$$f_j^*(s) = \max_{0 \leq x_j \leq s} \{p_j(x_j) + f_{j+1}^*(s - x_j)\}, \quad j = 1, 2, 3$$

where $p_j(x_j)$ denotes the expected value obtained from allocation of x_j units of weight to the item j .

Now, for one-stage problem (i.e., for one item cargo loading)

$$f_1^*(s) = \max_{x_1} \{30x_1\},$$

where the largest value of x_1 is $[W/w_1] = [5/1] = 5$.

We have the following tabular computations :

Value of $30x_1$							Optimum Solution	
$s \backslash x_1$	0	1	2	3	4	5	$f_1^*(s)$	x_1^*
0	0						0	0
1	0	30					30	1
2	0	30	60				60	2
3	0	30	60	90			90	3
4	0	30	60	90	120		120	4
5	0	30	60	90	120	150	150	5

For 2-stage problem, the largest value of x_2 is $[5/3] = 1$ and

$$f_2^*(s) = \max_{x_2} \{80x_2 + f_1(s - 3x_2)\}$$

Value of $80x_2 + f_1^*(s - 3x_2)$

Optimum Solution

$s \backslash x_2$	0	1	$f_2^*(s)$	s_2^*
0	0 + 0 = 0		0	0
1	0 + 30 = 30		30	0
2	0 + 60 = 60		60	0
3	0 + 90 = 90	80 + 0 = 80	90	0
4	0 + 120 = 120	80 + 30 = 110	120	0
5	0 + 150 = 150	80 + 60 = 140	150	0

For 3-stage problem, the largest value of x_3 is $[5/2] = 2$ and

$$f_3^*(s) = \max_{x_3} \{65x_3 + f_2^*(s - 2x_3)\}$$

Value of $65x_3 + f_2^*(s - 2x_3)$					Optimum Solution	
s	x_2	0	1	2	$f_3^*(s)$	x_3^*
0	0 + 0				0	0
1	0 + 30				30	0
2	0 + 60		$65 + 0 = 65$		65	1
3	0 + 90		$65 + 30 = 95$		95	1
4	0 + 120		$65 + 60 = 125$	$130 + 0 = 130$	130	2
5	0 + 150		$65 + 90 = 155$	$130 + 30 = 160$	160	2

Given $W = 5$, the optimum solution, therefore, is

$$x_3^* = 2, x_2^* = 0 \text{ and } x_1^* = 1$$

with $f_3^*(s) = 160 = \text{maximum value of load.}$

PROBLEMS

1315. A student has to take an examination in three courses x , y and z . He has three days available for study. He feels that it would be better to devote a whole day to study of the same course, so that he may study a course for one day, two days or three days or not at all. His estimates of grades he may get according to days of study he puts in are as follows :

Study days	Course		
	x	y	z
0	1	2	1
1	2	2	2
2	2	4	4
3	4	5	4

1316. Seven units of capital can be invested in four activities with return from each activity given in the accompanying table. Find the allocation of capital to each activity that will maximize the total return :

Q	$g^1(Q)$	$g^2(Q)$	$g^3(Q)$	$g^4(Q)$
0	0	0	0	0
1	2	3	2	1
2	4	5	3	3
3	6	7	4	5
4	7	9	5	6
5	8	10	5	7
6	9	11	5	8
7	9	12	5	8

1317. A chairman of a certain political party is making plans for his election to Parliament. He has engaged the services of six volunteer workers and wishes to assign them four districts in such a way as to maximize their effectiveness. He feels that it would be inefficient to assign a worker to more than one district but he is also willing to assign no workers to any one of the districts judging by what the workers can accomplish in other districts.

The following table gives the estimated increase in the number of votes in favour of the party's candidate if it were allocated various number of workers :

Number of workers	Districts		
	1	2	3
0	0	0	0
1	25	20	33
2	42	38	43
3	55	54	47
4	63	65	50
5	69	73	52
6	74	80	53

How many of the workers should be assigned to each of the three districts in order to maximize the total number of votes in his favour?

5.3 ALL-PAIRS SHORTEST PATHS

Let $G = (V, E)$ be a directed graph with n vertices. Let $cost$ be a cost adjacency matrix for G such that $cost(i, i) = 0$, $1 \leq i \leq n$. Then $cost(i, j)$ is the length (or cost) of edge $\langle i, j \rangle$ if $\langle i, j \rangle \in E(G)$ and $cost(i, j) = \infty$ if $i \neq j$ and $\langle i, j \rangle \notin E(G)$. The *all-pairs shortest-path problem* is to determine a matrix A such that $A(i, j)$ is the length of a shortest path from i to j . The matrix A can be obtained by solving n single-source problems using the algorithm ShortestPaths of Section 4.8. Since each application of this procedure requires $O(n^2)$ time, the matrix A can be obtained in $O(n^3)$ time. We obtain an alternate $O(n^3)$ solution to this problem using the principle of optimality. Our alternate solution requires a weaker restriction on edge costs than required by ShortestPaths. Rather than require $cost(i, j) \geq 0$, for every edge $\langle i, j \rangle$, we only require that G have no cycles with negative length. Note that if we allow G to contain a cycle of negative length, then the shortest path between any two vertices on this cycle has length $-\infty$.

Let us examine a shortest i to j path in G , $i \neq j$. This path originates at vertex i and goes through some intermediate vertices (possibly none) and terminates at vertex j . We can assume that this path contains no cycles for if there is a cycle, then this can be deleted without increasing the path length (no cycle has negative length). If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j , respectively. Otherwise, the i to j path is not of minimum length. So, the principle of optimality holds. This alerts us to the prospect of using dynamic programming. If k is the intermediate vertex with highest index, then the i to k path is a shortest i to k path in G going through no vertex with index greater than $k - 1$. Similarly the k to j path is a shortest k to j path in G going through no vertex of index greater than

$k - 1$. We can regard the construction of a shortest i to j path as first requiring a decision as to which is the highest indexed intermediate vertex k . Once this decision has been made, we need to find two shortest paths, one from i to k and the other from k to j . Neither of these may go through a vertex with index greater than $k - 1$. Using $A^k(i, j)$ to represent the length of a shortest path from i to j going through no vertex of index greater than k , we obtain

$$A(i, j) = \min \left\{ \min_{1 \leq k \leq n} \{A^{k-1}(i, k) + A^{k-1}(k, j)\}, \text{cost}(i, j) \right\} \quad (5.7)$$

Clearly, $A^0(i, j) = \text{cost}(i, j)$, $1 \leq i \leq n$, $1 \leq j \leq n$. We can obtain a recurrence for $A^k(i, j)$ using an argument similar to that used before. A shortest path from i to j going through no vertex higher than k either goes through vertex k or it does not. If it does, $A^k(i, j) = A^{k-1}(i, k) + A^{k-1}(k, j)$. If it does not, then no intermediate vertex has index greater than $k - 1$. Hence $A^k(i, j) = A^{k-1}(i, j)$. Combining, we get

$$A^k(i, j) = \min \{A^{k-1}(i, j), A^{k-1}(i, k) + A^{k-1}(k, j)\}, \quad k \geq 1 \quad (5.8)$$

The following example shows that (5.8) is not true for graphs with cycles of negative length.

Example 5.14 Figure 5.5 shows a digraph together with its matrix A^0 . For this graph $A^2(1, 3) \neq \min\{A^1(1, 3), A^1(1, 2) + A^1(2, 3)\} = 2$. Instead we see that $A^2(1, 3) = -\infty$. The length of the path

$$1, 2, 1, 2, 1, 2, \dots, 1, 2, 3$$

can be made arbitrarily small. This is so because of the presence of the cycle $1 \ 2 \ 1$ which has a length of -1 . \square

Recurrence (5.8) can be solved for A^n by first computing A^1 , then A^2 , then A^3 , and so on. Since there is no vertex in G with index greater than n , $A(i, j) = A^n(i, j)$. Function AllPaths computes $A^n(i, j)$. The computation is done in-place so the superscript on A is not needed. The reason this computation can be carried out in-place is that $A^k(i, k) = A^{k-1}(i, k)$ and $A^k(k, j) = A^{k-1}(k, j)$. Hence, when A^k is formed, the k th column and row do not change. Consequently, when $A^k(i, j)$ is computed in line 11 of Algorithm 5.3, $A(i, k) = A^{k-1}(i, k) = A^k(i, k)$ and $A(k, j) = A^{k-1}(k, j) = A^k(k, j)$. So, the old values on which the new values are based do not change on this iteration.

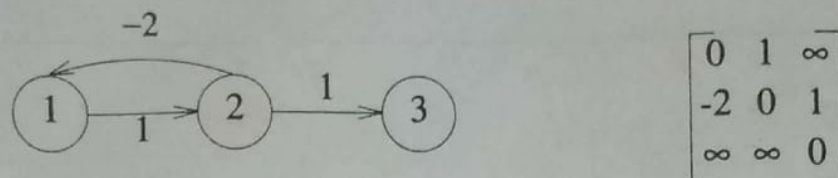


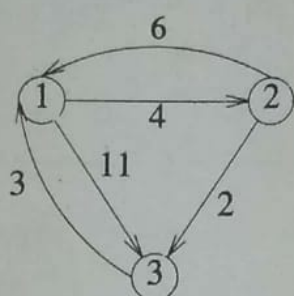
Figure 5.5 Graph with negative cycle

```

0  Algorithm AllPaths(cost, A, n)
1  // cost[1 : n, 1 : n] is the cost adjacency matrix of a graph with
2  // n vertices; A[i, j] is the cost of a shortest path from vertex
3  // i to vertex j. cost[i, i] = 0.0, for  $1 \leq i \leq n$ .
4  {
5      for i := 1 to n do
6          for j := 1 to n do
7              A[i, j] := cost[i, j]; // Copy cost into A.
8          for k := 1 to n do
9              for i := 1 to n do
10                 for j := 1 to n do
11                     A[i, j] := min(A[i, j], A[i, k] + A[k, j]);
12  }
```

Algorithm 5.3 Function to compute lengths of shortest paths

Example 5.15 The graph of Figure 5.6(a) has the cost matrix of Figure 5.6(b). The initial A matrix, $A^{(0)}$, plus its values after 3 iterations $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are given in Figure 5.6. \square



(a) Example digraph

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

(b) A^0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

(c) A^1

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

(d) A^2

A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

(e) A^3

Figure 5.6 Directed graph and associated matrices

Let $M = \max \{cost(i, j) \mid \langle i, j \rangle \in E(G)\}$. It is easy to see that $A^n(i, j) \leq (n-1)M$. From the working of AllPaths, it is clear that if $\langle i, j \rangle \notin E(G)$ and $i \neq j$, then we can initialize $cost(i, j)$ to any number greater than $(n-1)M$ (rather than the maximum allowable floating point number). If, at termination, $A(i, j) > (n-1)M$, then there is no directed path from i to j in G . Even for this choice of ∞ , care should be taken to avoid any floating point overflows.

The time needed by AllPaths (Algorithm 5.3) is especially easy to determine because the looping is independent of the data in the matrix A . Line 11 is iterated n^3 times, and so the time for AllPaths is $\Theta(n^3)$. An exercise examines the extensions needed to obtain the i to j paths with these lengths. Some speedup can be obtained by noticing that the innermost for loop need be executed only when $A(i, k)$ and $A(k, j)$ are not equal to ∞ .