Dynamic Programming

References:

[1] Bellman, R.E. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ. Republished 2003: Dover, ISBN 0-486-42809-5.

[2] Dreyfus (2002), “Richard Bellman on the birth of dynamic programming”, Archived January 10, 2005, at the Wayback Machine Operations Research 50 (1), pp. 48–51.

[3] R Bellman, “On the Theory of Dynamic Programming”, *Proceedings of the National Academy of Sciences*, 1952

[4] S. E. Dreyfus and A. M. Law, *The art and theory of dynamic programming*, Academic Press, 1977.

Richard Bellman

(August 26, 1920 – March 19, 1984)

Participated in the Manhattan project during 2nd World War [2]
Completed PhD (Princeton) 1949
Introduced dynamic programming in 1952 [3]
Fellow in the American Academy of Arts and Sciences, a member of the National Academy of Engineering, and a member of the National Academy of Sciences.
John von Neumann Theory Prize (1976)
IEEE Medal of Honor (1979)
Richard E. Bellman Control Heritage Award (1984)
Introduced the expression “curse of dimensionality”.

Source: Wikipedia

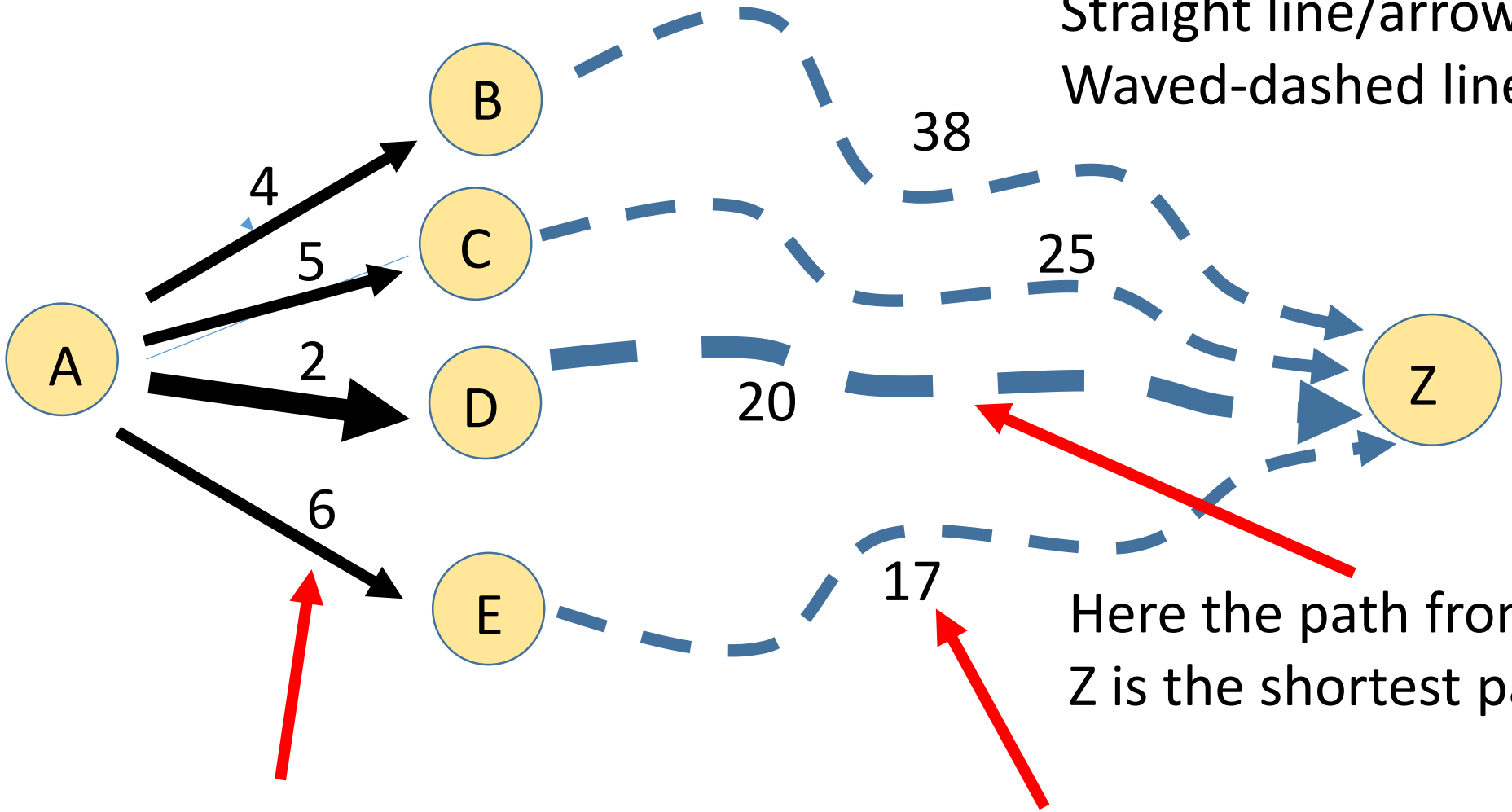
Dynamic Programming is applicable to Multi-stage Decision process problems [2]

- Consider a process that at any stage changes from state to state based on our decision.
- A “state” is all the information we need, have and use at each stage of the process to make the next decision.
- Our aim is to optimize an (additive) objective function, e.g. minimize distance (as in shortest path problems) or maximize wealth.

Bellman’s Principle of Optimality (a necessary condition)

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (See Bellman, 1957 [1], Chap. III.3.) See also [2][3].

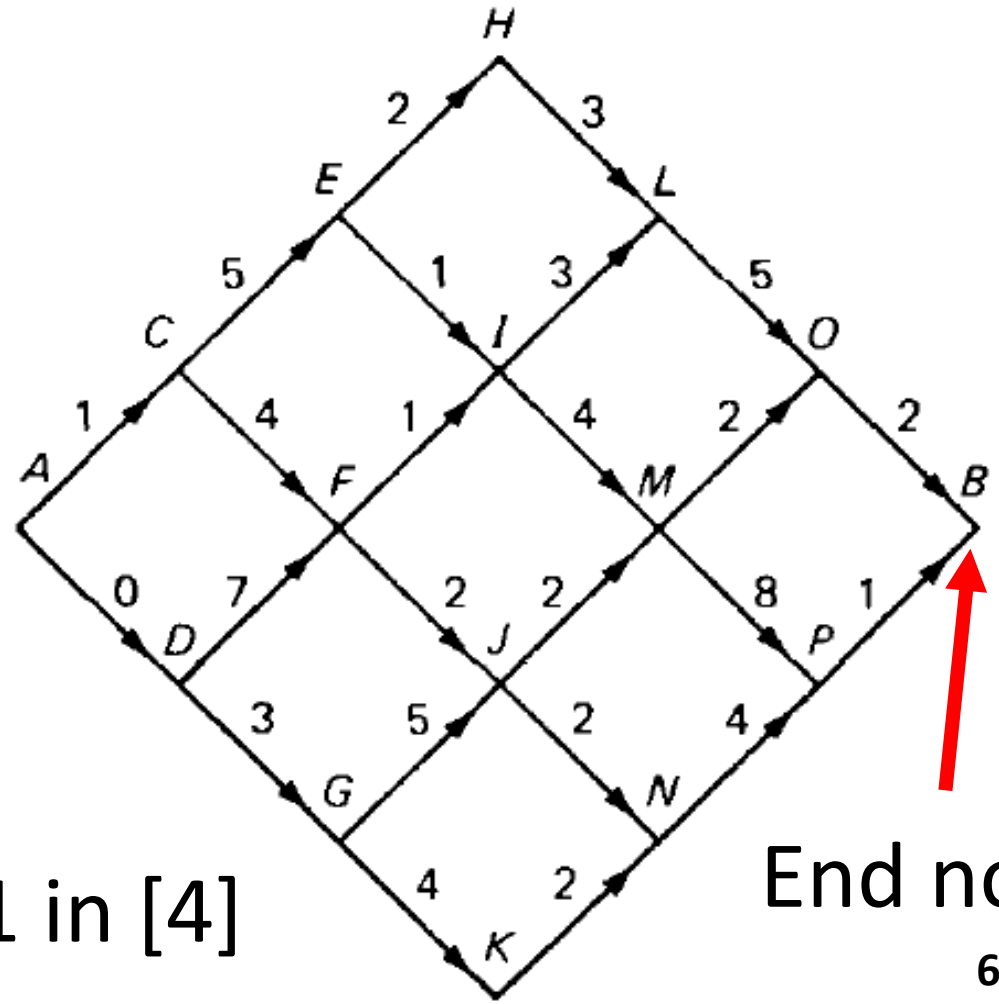
Source: Wikipedia
https://en.wikipedia.org/wiki/Bellman_equation



Credit: https://en.wikipedia.org/wiki/Dynamic_programming

Example 1:
A Simple Shortest Path Problem [4]

Find the shortest path from A to B



Start node

End node

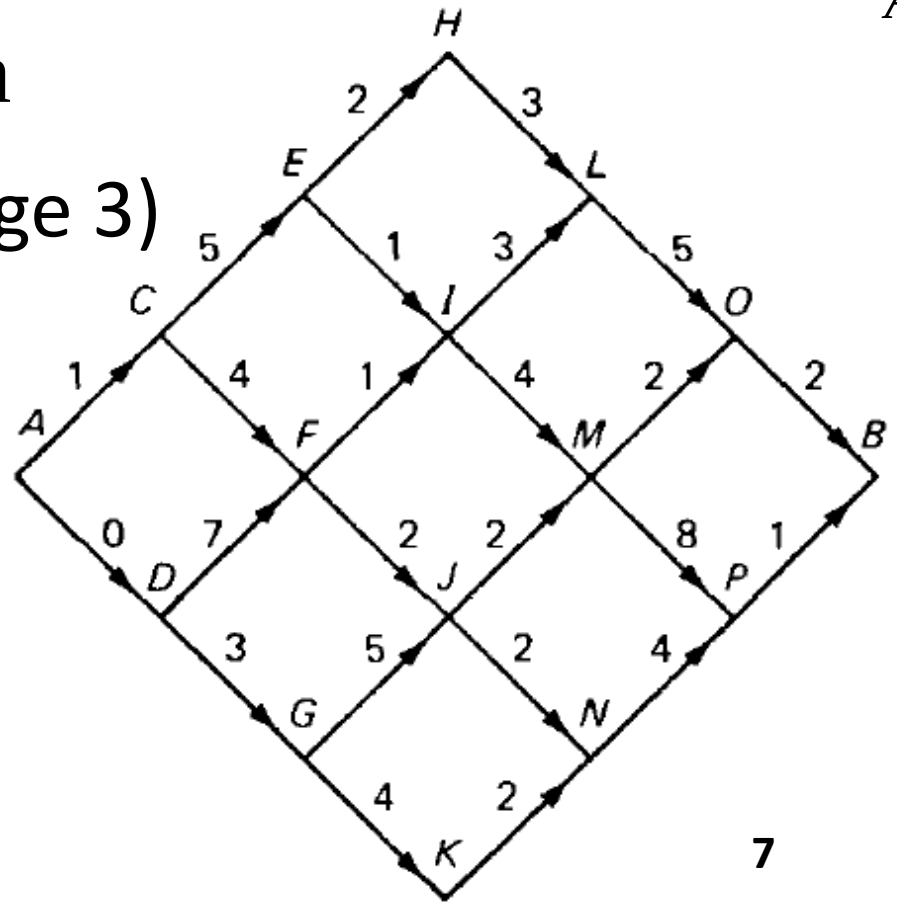
Fig. 1.1 in [4]

- Define: S_X = The minimal distance from X to B.
- The function S_X is called *optimal value function* which is a rule that assign value to various *subproblems* associated with the different values of X.
- We want to find the shortest path from A to B that is of distance S_A.
- By the principle of optimality, we obtain

([4] page 3)

$$S_A = \min\begin{bmatrix} 1 + S_C \\ 0 + S_D \end{bmatrix},$$

where $\min\begin{bmatrix} x \\ y \end{bmatrix}$ = minimum of x and y



Source: [4]

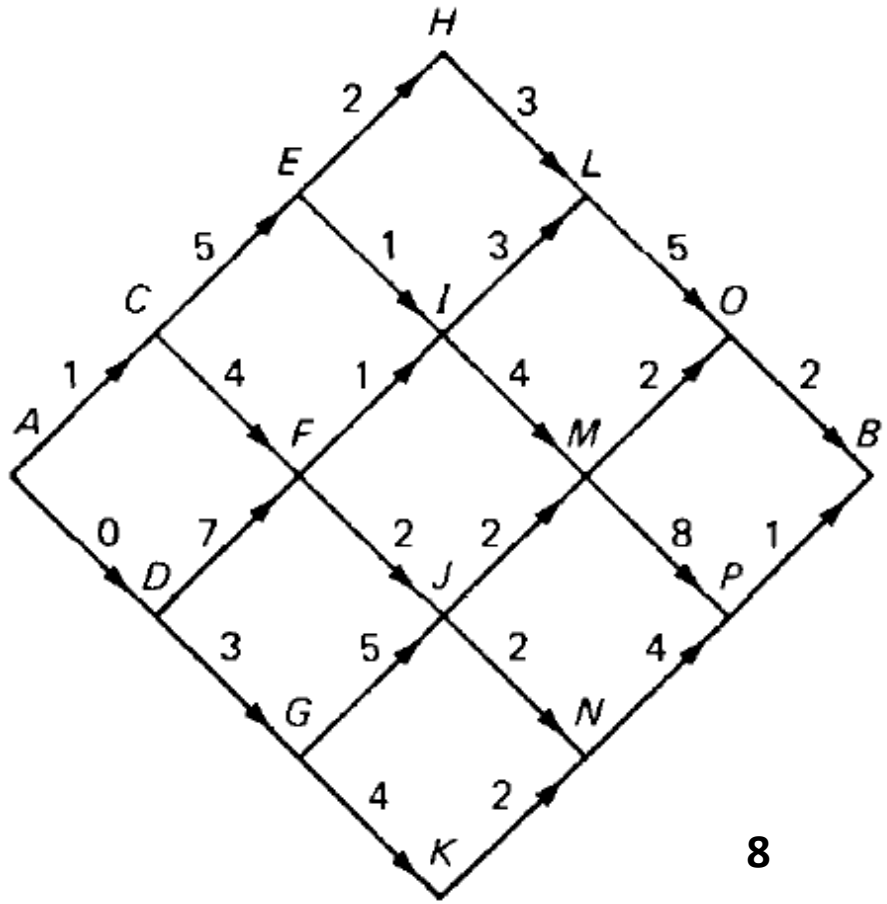
Again, by the principle of optimality, we obtain

$$S_C = \min \begin{bmatrix} 5 + S_E \\ 4 + S_F \end{bmatrix}.$$

$$S_D = \min \begin{bmatrix} 7 + S_F \\ 3 + S_G \end{bmatrix}.$$

etc.

([4] page 3)



Source: [4]

Again, by the principle of optimality, going backwards we obtain

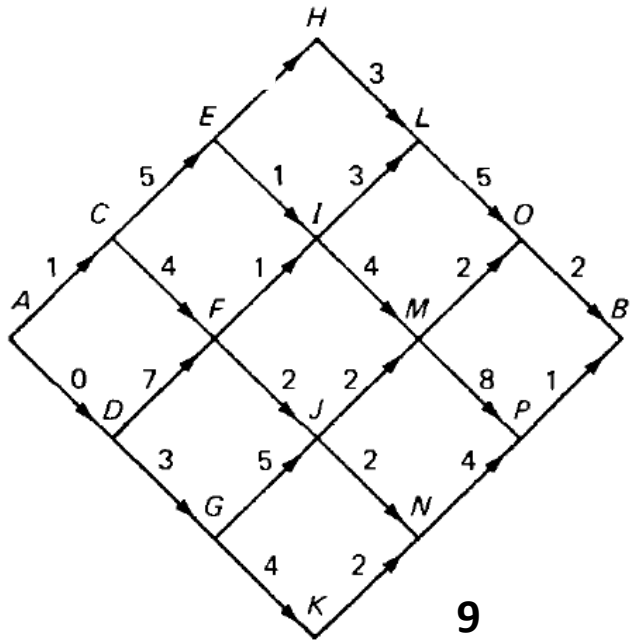
$S_O = 2$ $S_P = 1$ These two are called *boundary conditions.*

$$S_L = 5 + S_O = 7, \qquad S_M = \min\begin{bmatrix} 2 + S_O \\ 8 + S_P \end{bmatrix} = 4, \qquad S_N = 4 + S_P = 5;$$

$$S_H = 3 + S_L = 10, \qquad S_I = \min\begin{bmatrix} 3 + S_L \\ 4 + S_M \end{bmatrix} = 8, \qquad S_J = \min\begin{bmatrix} 2 + S_M \\ 2 + S_N \end{bmatrix} = 6,$$

$$S_K = 2 + S_N = 7;$$

([4] page 3)



Source: [4]

Again and again, by the principle of optimality, we obtain

$$S_E = \min\begin{bmatrix} 2 + S_H \\ 1 + S_I \end{bmatrix} = 9, \qquad S_F = \min\begin{bmatrix} 1 + S_I \\ 2 + S_J \end{bmatrix} = 8,$$

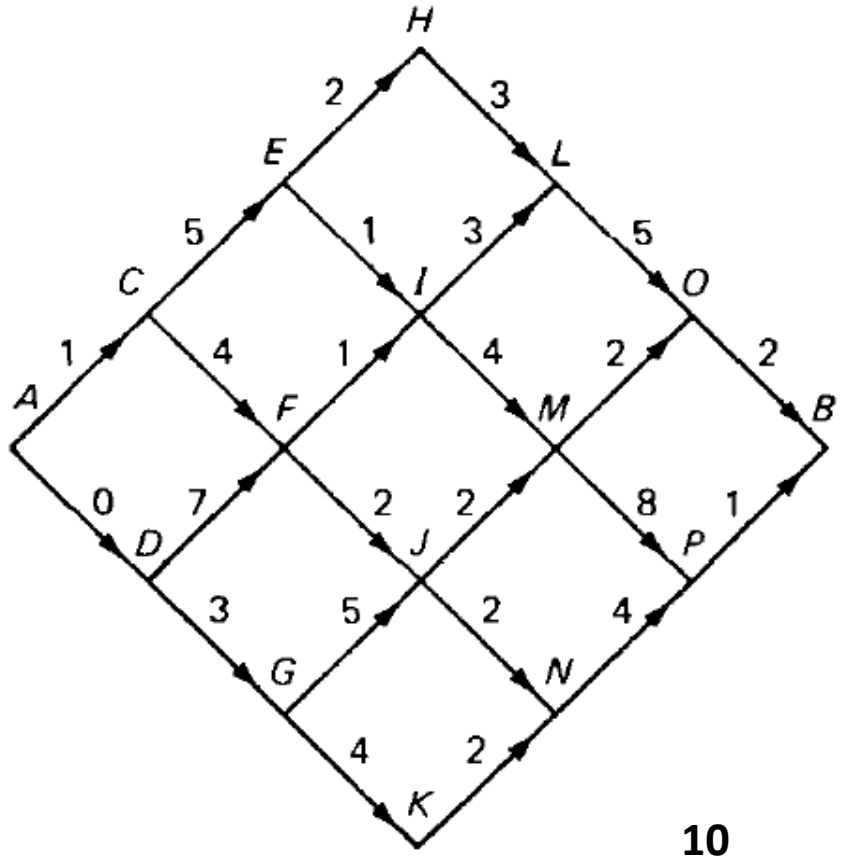
$$S_G = \min\begin{bmatrix} 5 + S_J \\ 4 + S_K \end{bmatrix} = 11;$$

$$S_C = \min\begin{bmatrix} 5 + S_E \\ 4 + S_F \end{bmatrix} = 12, \qquad S_D = \min\begin{bmatrix} 7 + S_F \\ 3 + S_G \end{bmatrix} = 14;$$

$$S_A = \min\begin{bmatrix} 1 + S_C \\ 0 + S_D \end{bmatrix} = 13.$$

([4] page 4)

$S_I = 8, \; S_J = 6$



$$S_A = 13$$

Source: [4]

Now let's find the shortest path.

Define P_X = the node after node X on the shortest path from X to B. P_X is called the *optimal policy function*.

Table 1.1 in [4]: The optimal next point for each initial point

$P_O = B$, $P_P = B$;

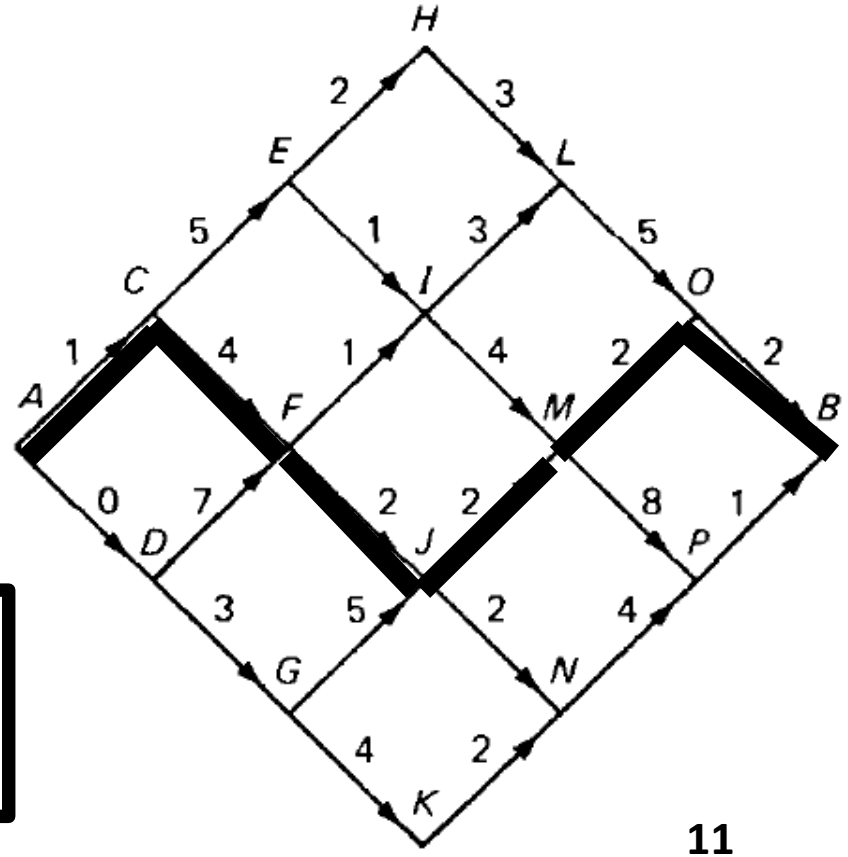
$P_L = O$, $P_M = O$, $P_N = P$;

$P_H = L$, $P_I = M$, $P_J = M$, $P_K = N$;

$P_E = I$, $P_F = J$, $P_G = J$ or K;

$P_C = F$, $P_D = G$;

$P_A = C$.



so the shortest path is:

$A \rightarrow C \rightarrow F \rightarrow J \rightarrow M \rightarrow O \rightarrow B$

Source: [4]

What have we done? – Three steps

1. Use the *Principle of Optimality* to write a set of equations for the optimal value function S_X for $X = A, C, D, E, F, G, H, I, J, K, L, M, N, O, P$, by going forward from A to B. These equations define what is called *recurrence relations*.
2. Solve these equations by going backwards from *B* to *A*. Starting with the *boundary conditions* $S_o = 2$ and $S_P = 1$.
3. Decide the shortest path by going forward using the *optimal policy function* P_X given in Table 1.1 of [4].

Credit: [4]

Computation Requirements (Dynamic Programming) [4]

Using **dynamic programming**, we performed:

- one addition at each of the nodes *H*, *L*, *O*, *K*, *N*, *P* = 6 additions,
- two additions and one comparisons in the remaining nine nodes = 18 additions + 9 comparisons.

In total we need: **24 additions + 9 comparisons**.

Computation Requirements (Brute Force) [4]

Using **Brute Force,** we have 20 different paths (why?) to compare.

- For each path, we need 5 additions to add up 6 numbers, so we need 20 x 5 = 100 additions.
- To compare 20 paths, we need 19 comparisons.

In total we need: **100 additions + 19 comparisons**.

Computation Requirements for *N*-Stage Problem [4]

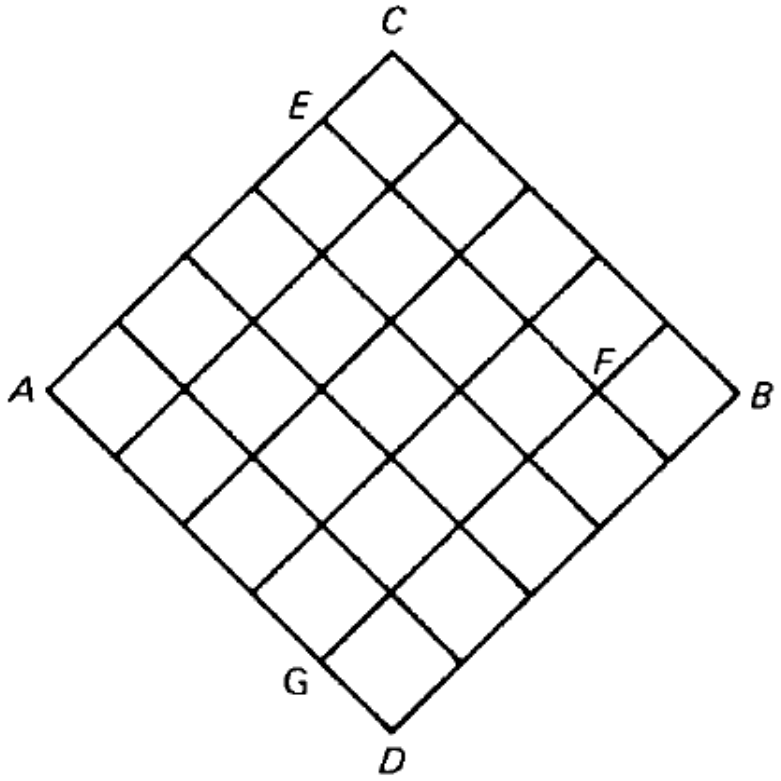


Fig. 1.14 in [4] for a 10-Stage Problem

Different labels are used here for the nodes than before. Here only selected nodes have labels for evaluation of computation requirements.

Computation Requirements for N-Stage Problem (Dynamic Programming) [4]

- As observed in the figure for the 10-Stage Problem, in the lines CB and DB there are 10 nodes excluding B. Steps in these nodes do not require comparisons and require one addition in each of these nodes.
- In the N-Stage Problem, these N nodes require N additions and no comparisons.
- The other $N^2/4$ nodes (in the diamond AEFG), each requires two additions and one comparison.
- In total, we require $N^2/2+N$ additions and $N^2/4$ comparisons.

Computation Requirements (Brute Force) for N-Stage Problem [4]

Using **Brute Force,** we have $\binom{N}{N/2}$ different paths (why?) to compare.

- For each path, we need $N-1$ additions to add up N numbers, so we need $(N-1)\binom{N}{N/2}$ additions.
- To compare $\binom{N}{N/2}$ paths, we need $\binom{N}{N/2}-1$ comparisons.

In total we need: $(N-1)\binom{N}{N/2}$ additions and $\binom{N}{N/2}-1$ comparisons.

For N = 20:

Dynamic Programming requires 220 additions and 100 comparisons.

Brute Force requires more than 3,000,000 additions and 184,000 comparisons. [Calculate the exact number of additions.]

Another advantage of Dynamic Programming over Brute Force

Using **Brute Force,** we obtain a solution only for the shortest path between *A* and *B*. But with **Dynamic Programming**, we obtain solutions for various other subproblems in addition to the shortest path between *A* and *B*.

The Same Simple Shortest-Path Problem in a Coordinate System for a More Systematic Representation

Now we use (x, y) coordinates
A has coordinates (0, 0),
B has coordinates (6, 0),
J has coordinates (3, -1),
L has coordinates (4, 2),
K has coordinates (3, -3),
etc.

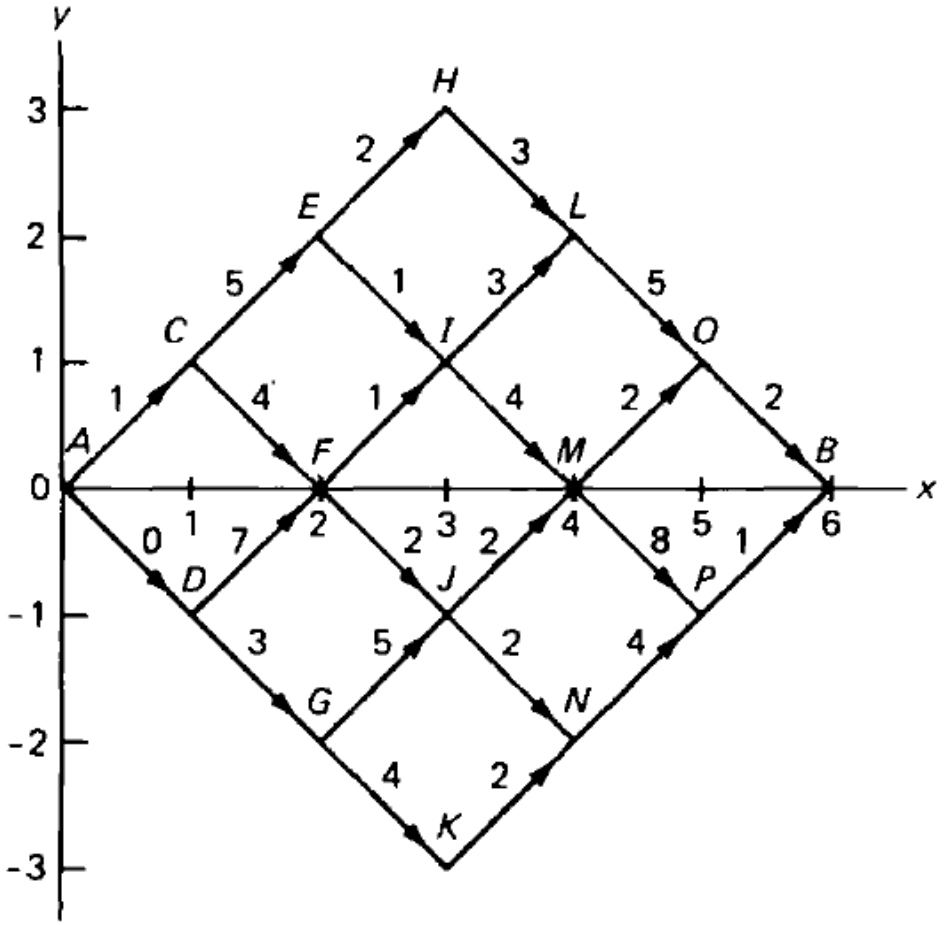


Figure 1.2 in [4]

Credit: [4]

The Shortest-Path Problem in a Coordinate System (cont'd)

The optimal value function is:

$S(x, y)$ = the value of the minimum-effort path connecting the vertex (x, y) with the terminal vertex (6, 0).

Eq. (1.1) in [4]

$a_u(x, y)$ = the cost (distance) of the link between node (x, y) and $(x+1, y+1)$. The u indicates the move "up" from (x, y).

$a_d(x, y)$ = the cost (distance) of the link between node (x, y) and $(x+1, y-1)$. The d indicates the move "down" from (x, y).

Credit: [4]

The Shortest-Path Problem in a Coordinate System (cont'd)

We set $a_u(x, y) = \infty$ if there is no link from node (x, y) to $(x+1, y+1)$. For example: $a_u(4, 2) = \infty$.

We set $a_d(x, y) = \infty$ if there is no link from node (x, y) to $(x-1, y-1)$. For example: $a_d(4, -2) = \infty$.

Credit: [4]

The Shortest-Path Problem in a Coordinate System (cont'd)

Considering all the above notations and definitions, and using the principle of optimality, we obtain the following recurrence relation for the optimal value function $S(x, y)$.

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

Eq. (1.2) in [4]

with the boundary condition $S(6, 0) = 0$.

Credit: [4]

The Shortest-Path Problem in a Coordinate System (cont'd)

A Simple Exercise

Show that the boundary conditions we used before, namely,

$S(5,1) = 2$

and

$S(5,-1) = 1$

are obtained from the recurrent relation and the boundary condition considering the above definitions and notations including the cases where we set

$a_u(x,y) = \infty$ and $a_d(x,y) = \infty$.

Credit: [4]

When you are asked to provide a Dynamic Programming formulation of a given problem, the intention is that you provide the following.

1. Definition of the appropriate optimal value function including definition of the function and its arguments;
2. An appropriate recurrence relation;
3. Appropriate boundary conditions.

Source: [4]

Assignment

Consider again Fig. 1.1 in [4] and the problem of finding the shortest path from *A* to *B* with the following difference. The link costs are now based on your unique number. If the link cost is *j* in the figure, it changes to the *j*th digit of your unique number. For example, the cost of link *CE* was 5. Change it to the value of your 5th digit. For link *AD*, the cost remains 0.

Provide a Dynamic Programming (DP) formulation for the new problem and solve it completely using DP (find the shortest path and its cost). Show all steps.

Upload your solution to Canvas/Discussions.

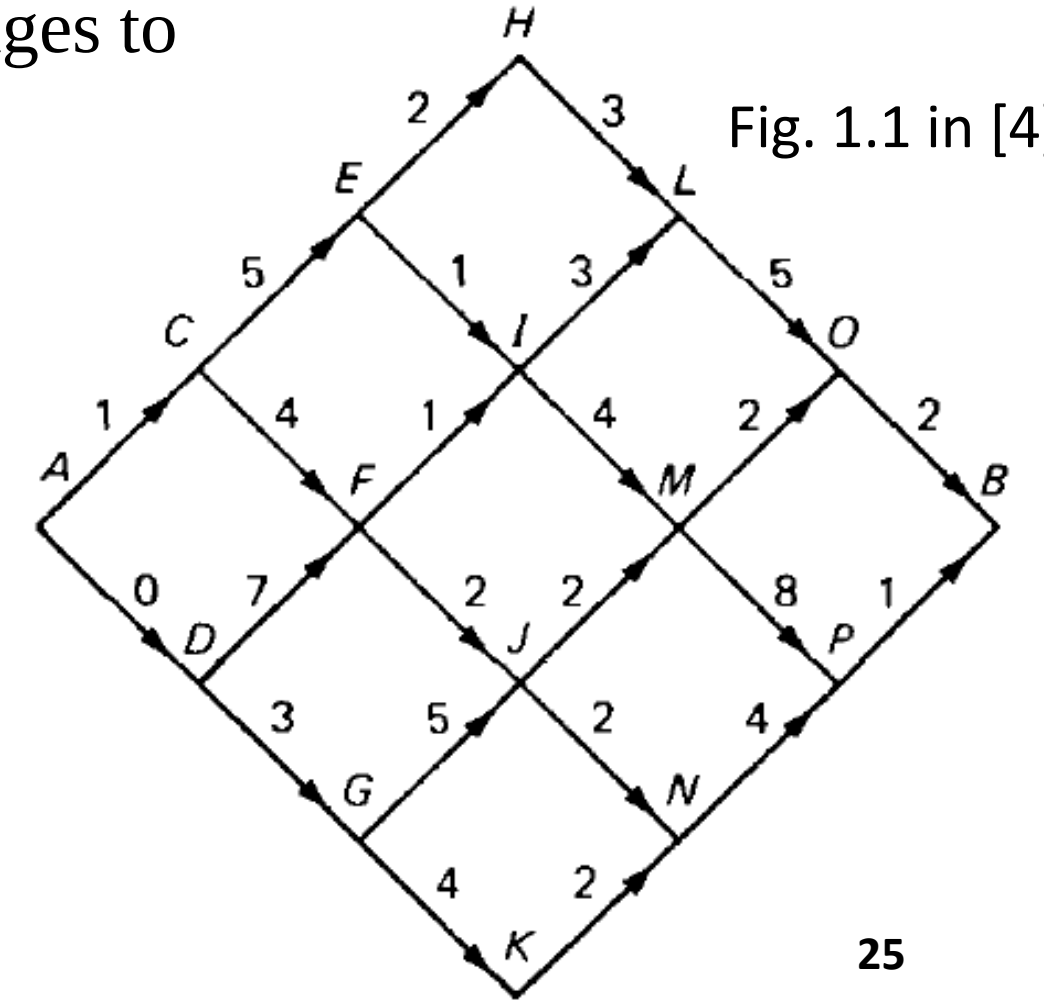


Fig. 1.1 in [4]

More Exercises

Problem 1.1. On the network shown in Figure 1.3, we seek that path connecting A with any point on line B which minimizes the sum of the four arc numbers encountered along the path. (There are 16 admissible paths.) Give the dynamic-programming formulation of this problem.

Problem 1.2. Solve the above problem using dynamic programming, with the additional specification that there is a rebate associated with each terminal point; ending at the point (4, 4) has a cost of -2 (i.e., 2 is subtracted from the path cost), (4, 2) has cost -1, (4, 0) has cost -3, (4, -2) has cost -4, and (4, -4) has cost -3.

Source: [4]

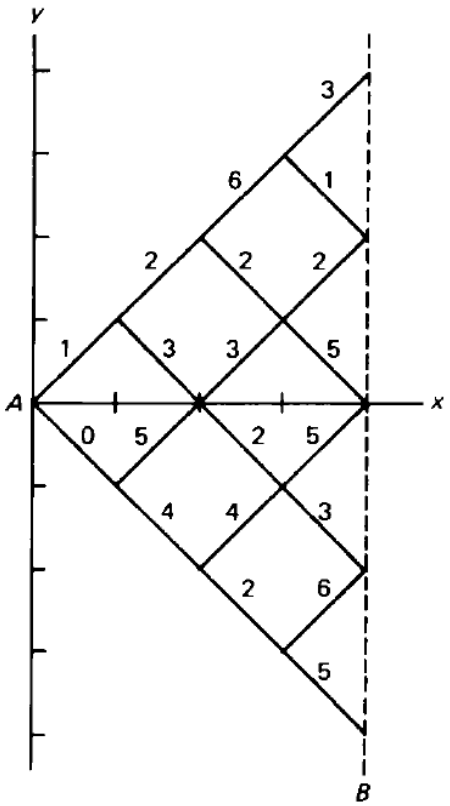


Figure 1.3 in [4]

Exercise on Computation Requirements

Problem 1.3. How many additions and how many comparisons are required in the dynamic-programming solution and in brute-force enumeration for an N-stage problem involving a network of the type shown in Figure 1.3? Evaluate your formulas for $N = 20$.

Source: [4]

Forward Dynamic Programming

A reversed version of Bellman's Principle of Optimality for the shortest path problem

For the shortest path from *A* to *B*, let *C* be the node before *B*. Then the path from *A* to *C* which is part of the shortest path from *A* to *B* is the shortest path between *A* to *C*.

Source: [4]

Forward Dynamic Programming (Cont'd)

Now notice also that the shortest path from *A* to *B* can be obtained if we know the shortest path from *A* to *O* and the shortest path from *A* to *P*.
Accordingly, we define the **new** optimal value function as follows.

$S(x, y)$ = the value of the minimum-effort path connecting the initial vertex (0, 0) with the vertex (x, y).

Then, the appropriate recurrence relation is:

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

And the boundary condition is $S(0, 0) = 0$

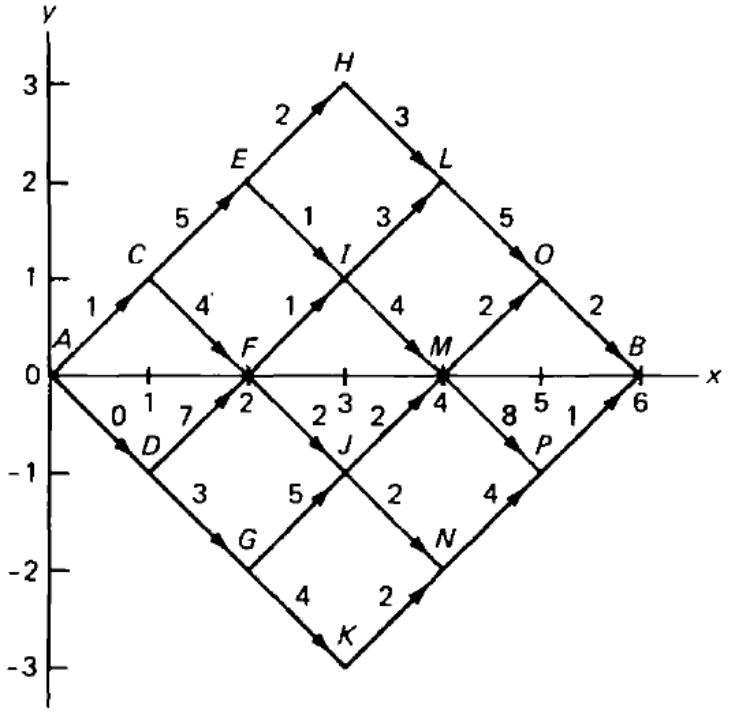


Figure 1.2 in [4]

Source: [4]

Homework

1. Implement the forward dynamic programming procedure for the problem of Fig. 1.2 in [4] and find the shortest path from *A* to *B*.
2. Show that the computation requirements (number of additions and of comparisons) of the forward dynamic programming for the problem of Fig. 1.2 in [4] are the same as those for the original (backward) Dynamic Programming.

Credit: [4]

What is similar and different between the information provided by the forward dynamic programming solutions and the information provided by the backward dynamic programming solutions for the problem of Fig. 1.2 in [4].

Both provide the shortest path from *A* to *B* (and of course the total cost of the shortest path).

The backward procedure provides the shortest path from every node to node *B*, but does not provide any information on other paths, while the forward procedure provides the shortest path from node *A* to every node, but does not provide any information on other paths.

Credit: [4]

Equipment Replacement (ER) [4]

1. We need to have a certain machine, say, a car, for the next N years.
2. Every year (from 1 to N) we make a decision either to keep the old machine or to replace it for a new machine with the objective to minimize the total costs in N years.

Source: motor1.com

ER (cont'd) [4]

y = age of the machine at the beginning of year 1.

$c(i)$ = cost of operating for one year a machine which is of age i at the start of the year,

p = price of a new machine (of age 0),

$t(i)$ = trade-in value received when a machine which is of age i at the start of a year is traded for a new machine at the start of the year,

$s(i)$ = salvage value received for a machine that has just turned age i at the end of year N.

From Page 25 in [4]

Problem 2.1. Define the appropriate optimal value function for the backward dynamic-programming solution of the above problem and write the recurrence relation and boundary condition that solve the problem.

ER (cont'd) [4]

2.1. The optimal value function is defined by

$S(x, k)$ = the minimum cost of owning a machine from year k through N, starting year k with a machine just turned age x.

The recurrence relation is

$$S(x, k) = \min \begin{bmatrix} \text{Buy:} & p - t(x) + c(0) + S(1, k+1) \\ \text{Keep:} & c(x) + S(x+1, k+1) \end{bmatrix}. \qquad \text{(S2.1)}$$

The boundary condition is

$$S(x, N+1) = -s(x). \qquad \text{(S2.2)}$$

The recurrence relation (S2.1) evaluates, first, the "buy decision" by adding the cost during year k of buying a new machine (the purchase price plus the year's operating cost less the trade-in value) to the minimum attainable cost for the remaining process, which starts year $k+1$ with a one-year-old machine. Then it evaluates the decision "keep the current machine for at least one more year" by adding the cost of operating the old machine for a year to the minimum cost of the remaining process, which in this case starts year $k+1$ with a machine of age $x+1$ (one year older than it was at the start of year k). The better of these two alternatives is recorded as the value of $S(x, k)$ since the principle of optimality assures us that no nonoptimal continuation from year $k+1$ to N need be considered.

One should also note the optimal decision in this situation, writing $P(x, k) = B$ (buy) if line 1 on the right of (S2.1) yields the minimum, and $P(x, k) = K$ (keep) if line 2 yields the minimum.

From Page 225 in [4]

ER (cont'd) ([4], Page 226)

The boundary condition states that a rebate (negative cost) of $s(x)$ occurs if the machine has just turned age x at the start of year $N + 1$, i.e., at the end of year N.

Naturally, our use of the particular letters S, x, and k to denote certain quantities is quite arbitrary, and any symbols, if appropriately defined, are equally acceptable. An equivalent and equally acceptable boundary condition is

$$S(x, N) = \min \begin{bmatrix} p - t(x) + c(0) - s(1) \\ c(x) - s(x + 1) \end{bmatrix}.$$

If y denotes the age of the starting (incumbent) machine, (S2.2) must be tabulated for $x = 1, 2, \ldots, N$ and $y + N$ since various decisions during the first N years could result in each of these terminal ages. (Age $y + N$ corresponds to the prior decisions never to replace the incumbent machine.) Then $S(x, N)$ is computed, using (S2.1) for $x = 1, \ldots, N - 1$ and $y + N - 1$. (The incumbent machine, if never replaced, will be of age $y + N - 1$ starting year N.) Next compute $S(x, N - 1)$ for $x = 1, \ldots, N - 2$ and $y + N - 2$, etc. until $S(x, 2)$ is computed for $x = 1$ and $y + 1$. Finally compute $S(y, 1)$, the minimum cost for the given problem.

ER (cont'd) ([4], Page 25)

Now, assuming that you have read and understood the solution of Problem 2.1, consider the problem given by the following data:

$N = 5$;

y (the age of the incumbent machine at the start of year 1) = 2;

$c(0) = 10, \quad c(1) = 13, \quad c(2) = 20, \quad c(3) = 40,$

$c(4) = 70, \quad c(5) = 100, \quad c(6) = 100;$

$p = 50;$

$t(1) = 32, \quad t(2) = 21, \quad t(3) = 11, \quad t(4) = 5, \quad t(5) = 0, \quad t(6) = 0;$

$s(1) = 25, \quad s(2) = 17, \quad s(3) = 8, \quad s(4) = 0, \quad s(5) = 0, \quad s(6) = 0;$

Problem 2.2. Use the solution of Problem 2.1 to solve the above problem numerically. What is the optimal sequence of decisions?

ER (Cont'd) - Solution of Problem 2.2 ([4] page 226)

2.2. According to the boundary condition (S2.2)

$$S(1, 6) = -25, \quad S(2, 6) = -17, \quad S(3, 6) = -8, \quad S(4, 6) = S(5, 6) = S(7, 6) = 0.$$

Now, using (S2.1) to compute $S(x, 5)$:

$$S(1, 5) = \min\begin{bmatrix} 50 - 32 + 10 + S(1, 6) \\ 13 + S(2, 6) \end{bmatrix} = -4, \qquad P(1, 5) = \text{keep};$$

$$S(2, 5) = \min\begin{bmatrix} 50 - 21 + 10 + S(1, 6) \\ 20 + S(3, 6) \end{bmatrix} = 12, \qquad P(2, 5) = \text{keep};$$

$$S(3, 5) = \min\begin{bmatrix} 24 \\ 40 \end{bmatrix} = 24, \qquad P(3, 5) = \text{buy};$$

$$S(4, 5) = \min\begin{bmatrix} 30 \\ 70 \end{bmatrix} = 30, \qquad P(4, 5) = \text{buy};$$

$$S(6, 5) = \min\begin{bmatrix} 35 \\ 100 \end{bmatrix} = 35, \qquad P(6, 5) = \text{buy}.$$

ER - Solution of Problem 2.2 ([4] page 226) (cont'd)

Next, use (S2.1) and the above results to determine $S(x, 4)$:

$$S(1, 4) = \min\begin{bmatrix} 28 + S(1, 5) \\ 13 + S(2, 5) \end{bmatrix} = 24, \qquad P(1, 4) = \text{buy};$$

$$S(2, 4) = \min\begin{bmatrix} 35 \\ 44 \end{bmatrix} = 35, \qquad P(2, 4) = \text{buy};$$

$$S(3, 4) = \min\begin{bmatrix} 45 \\ 70 \end{bmatrix} = 45, \qquad P(3, 4) = \text{buy};$$

$$S(5, 4) = \min\begin{bmatrix} 56 \\ 135 \end{bmatrix} = 56, \qquad P(5, 4) = \text{buy}.$$

Next, determine $S(x, 3)$:

$$S(1, 3) = \min\begin{bmatrix} 52 \\ 48 \end{bmatrix} = 48, \qquad P(1, 3) = \text{keep};$$

$$S(2, 3) = \min\begin{bmatrix} 63 \\ 65 \end{bmatrix} = 63, \qquad P(2, 3) = \text{buy};$$

$$S(4, 3) = \min\begin{bmatrix} 79 \\ 126 \end{bmatrix} = 79, \qquad P(4, 3) = \text{buy}.$$

ER-***Problem 2.3.*** For a problem of the above type, approximately how many additions and how many comparisons, as a function of the duration of the process N, are required for the dynamic-programming solution? **(Page 26 in [4])**

Solution (Page 227 in [4])

2.3. Each evaluation of (S2.1) requires four additions and a comparison. Now imagine that you are actually performing the computation and count the number of values of S that you would need to calculate. At the start of year N, there are N values of S required ($x = 1, \ldots, N-1$ and $y + N - 1$). Verify this for Problem 2.2, where $N = 5$ and $y = 2$. At the start of year $N-1$, $N-1$ values of S are needed, etc., until at the start of year 1, one value must be computed. Hence $\sum_{i=1}^{N} i = N(N+1)/2$ values of S must be computed, so a total of $2N(N+1)$ additions and $N(N+1)/2$ comparisons are needed.

When we ask for the approximate number of calculations, we really want an idea of how the answer grows with N and how it depends on the range of values that the state variables can take on (which in this example depends on N, but generally is an independent choice in the model formulation). For this problem we would consider KN^2 with K any number between 1 and, say, 5 an appropriate answer for the total number of calculations. Hence the reader should feel free to be fairly approximate in counting additions or comparisons, or adding up the number of values of S required.

ER (cont'd) - Computation Requirements ([4], Page 26)

We shall assume henceforth that a modern digital computer takes 10^{-5} seconds to perform either an addition or comparison (i.e., it can add 100,000 numbers/second). Actually machines are somewhat faster than this, but several machine operations are required for the logic of locating the appropriate numbers and other bookkeeping procedures.

For the above problem, even if we make monthly decisions over a 20-year horizon (i.e., $N = 12 \cdot 20 = 240$), the computer solution by dynamic programming would require only about $\frac{5}{2} \cdot 240 \cdot 241 \cdot 10^{-5}$ seconds, or approximately $1\frac{1}{2}$ seconds.

While we shall not continue to belabor the advantage of dynamic programming over brute-force enumeration, we cannot resist one more comparison. There are 2^N possible decision sequences for an N-period problem, so for $N = 240$, the solution would take about $2^{240} \cdot 240 \cdot 10^{-5}$ seconds which is more than 10^{69} seconds or about 10^{62} years. This is much larger than the age of our solar system.

This is in 1977. Today it could be $<10^{-9}$ second.

Resource Allocation (RA) ([4], page 33)

The simplest resource allocation problem is as follows. You are given X units of a resource and told that this resource must be distributed among N activities. You are also given N data tables $r_i(x)$ (for $i = 1, \ldots, N$ and $x = 0, 1, \ldots, X$) representing the return realized from an allocation of x units of resource to activity i. (We assume, unless stated otherwise, that all return functions are nondecreasing functions of their arguments.) The problem is to allocate all of the X units of resource to the activities so as to maximize the total return, i.e., to choose N nonnegative integers x_i, $i = 1, \ldots, N$, that maximize

$$\sum_{i=1}^{N} r_i(x_i) \tag{3.1}$$

subject to the constraint

$$\sum_{i=1}^{N} x_i = X. \tag{3.2}$$

RA(cont'd) ([4], page 34)

Define the optimal value function as follows.

$f_k(x)$ = the maximum return obtainable from activities k through N, given x units of resource remaining to be allocated. **(3.3) in [4]**

By the principle of optimality, the recurrence relation appropriate to definition (3.3) is

$$f_k(x) = \max_{x_k = 0, 1, \ldots, x} [r_k(x_k) + f_{k+1}(x - x_k)], \qquad (3.4)$$

where x_k is the allocation to activity k and $f_k(x)$ must be computed for $x = 0, \ldots, X$. The boundary condition is

$$f_N(x) = r_N(x). \qquad (3.5)$$

The answer is $f_1(X)$.

Define $p_k(x)$ = the value of x_k that maximizes (3.4)

RA (cont'd) ([4], page 35)

Example: $X = 8$, $N = 4$

x_1	$r_1(x_1)$	x_2	$r_2(x_2)$	x_3	$r_3(x_3)$	x_4	$r_4(x_4)$
0	0	0	0	0	0	0	0
1	3	1	1	1	2	1	1
2	7	2	2	2	4	2	3
3	10	3	4	3	6	3	6
4	12	4	8	4	8	4	9
5	13	5	13	5	10	5	12
6	14	6	17	6	12	6	14
7	14	7	19	7	14	7	16
8	14	8	20	8	16	8	17

$f_4(0) = 0$, $f_4(1) = 1$, $f_4(2) = 3$, $f_4(3) = 6$, $f_4(4) = 9$,
$f_4(5) = 12$, $f_4(6) = 14$, $f_4(7) = 16$, $f_4(8) = 17$.

$p_4(x) = x$,
$x=0, 1, \ldots, 8.$

RA (cont'd) ([4], page 35)

Recurrence relation (3.4) yields

$f_3(0) = 0, \qquad p_3(0) = 0;$

$f_3(1) = \max[0 + 1, 2 + 0] = 2, \qquad p_3(1) = 1;$

$f_3(2) = \max[0 + 3, 2 + 1, 4 + 0] = 4, \qquad p_3(2) = 2;$

$f_3(3) = \max[0 + 6, 2 + 3, 4 + 1, 6 + 0] = 6, \qquad p_3(3) = 0 \text{ or } 3;$

$f_3(4) = \max[0 + 9, 2 + 6, 4 + 3, 6 + 1, 8 + 0] = 9, \qquad p_3(4) = 0;$

$f_3(5) = \max[0 + 12, 2 + 9, 4 + 6, 6 + 3, 8 + 1, 10 + 0] = 12, \qquad p_3(5) = 0;$

$f_3(6) = 14, \qquad p_3(6) = 0 \text{ or } 1;$

$f_3(7) = 16, \qquad p_3(7) = 0, 1, \text{ or } 2;$

$f_3(8) = 18, \qquad p_3(8) = 1, 2, \text{ or } 3.$

RA (cont'd) ([4], page 35)

$f_2(0) = 0, \qquad p_2(0) = 0;$

$f_2(1) = \max[0 + 2, 1 + 0] = 2, \qquad p_2(1) = 0;$

$f_2(2) = \max[0 + 4, 1 + 2, 2 + 0] = 4, \qquad p_2(2) = 0;$

$f_2(3) = \max[0 + 6, 1 + 4, 2 + 2, 4 + 0] = 6, \qquad p_2(3) = 0;$

$f_2(4) = 9, \qquad p_2(4) = 0;$

$f_2(5) = 13, \qquad p_2(5) = 5;$

$f_2(6) = 17, \qquad p_2(6) = 6;$

$f_2(7) = 19, \qquad p_2(7) = 6 \text{ or } 7;$

$f_2(8) = 21, \qquad p_2(8) = 6 \text{ or } 7.$

$f_1(8) = \max[0 + 21, 3 + 19, 7 + 17, 10 + 13, 12 + 9, 13 + 6, 14 + 4,$
$14 + 2, 14 + 0] = 24, \qquad p_1(8) = 2.$

RA Assignment

Consider again the table in [4], Page 35, and the problem of solving the RA Problem for $X = 8$, $N = 4$ with the following difference. The $r_i(x_i)$ functions now based on your unique number (repeated after the 16th digit – i.e. the 17th digit = the 1st digit, the 18th digit = 2nd digit, etc.). To calculate the new $r_i(x_i)$ functions, follow the following two steps.

Step 1: If the $r_i(x_i)$ value in the table is j, it changes to the jth digit of your unique number. For example, in the table $r_1(1) = 3$. Change it to the value of your 3-rd digit. If it is 0 in the table, it remains 0.

Step 2: For each new column you have in the new table, rearrange the numbers so that $r_i(x_i)$ are all non-decreasing functions. Also, make sure that $r_i(x_i) \geq x_i$ for all i.

Provide a Dynamic Programming (DP) formulation for the new problem and solve it completely using DP (find the optimal RA policy).

Show all steps. Upload your solution to Canvas/Discussions.

A Dynamic Programming Formulation of Dijkstra Shortest path algorithm

Now that you have learnt Dynamic Programming and you also learnt Dijkstra shortest path algorithm, please provide a Dynamic Programming Formulation of Dijkstra Shortest path algorithm, i.e., please provide:

(1) Optimal value function

(2) Recurrence relation

(3) Boundary conditions

Please upload your solution on Canvas Discussions and provide it also in Chat.

Question 1 (25 marks)

You were given a unique 16-digit number **for this test** that corresponds to a unique 10-node network. The 10 nodes are labelled 0, 1, 2, …, 9. Every pair of consecutive digits *i*,*j* corresponds to a directed link from Node *i* to Node *j* with weight |*i*-*j*|. An example for the network associated with the number 7298261503910431 is provided below. **This is not your number, so you have a different network.**

(1) (1 Mark) Write down your 16-digit number.

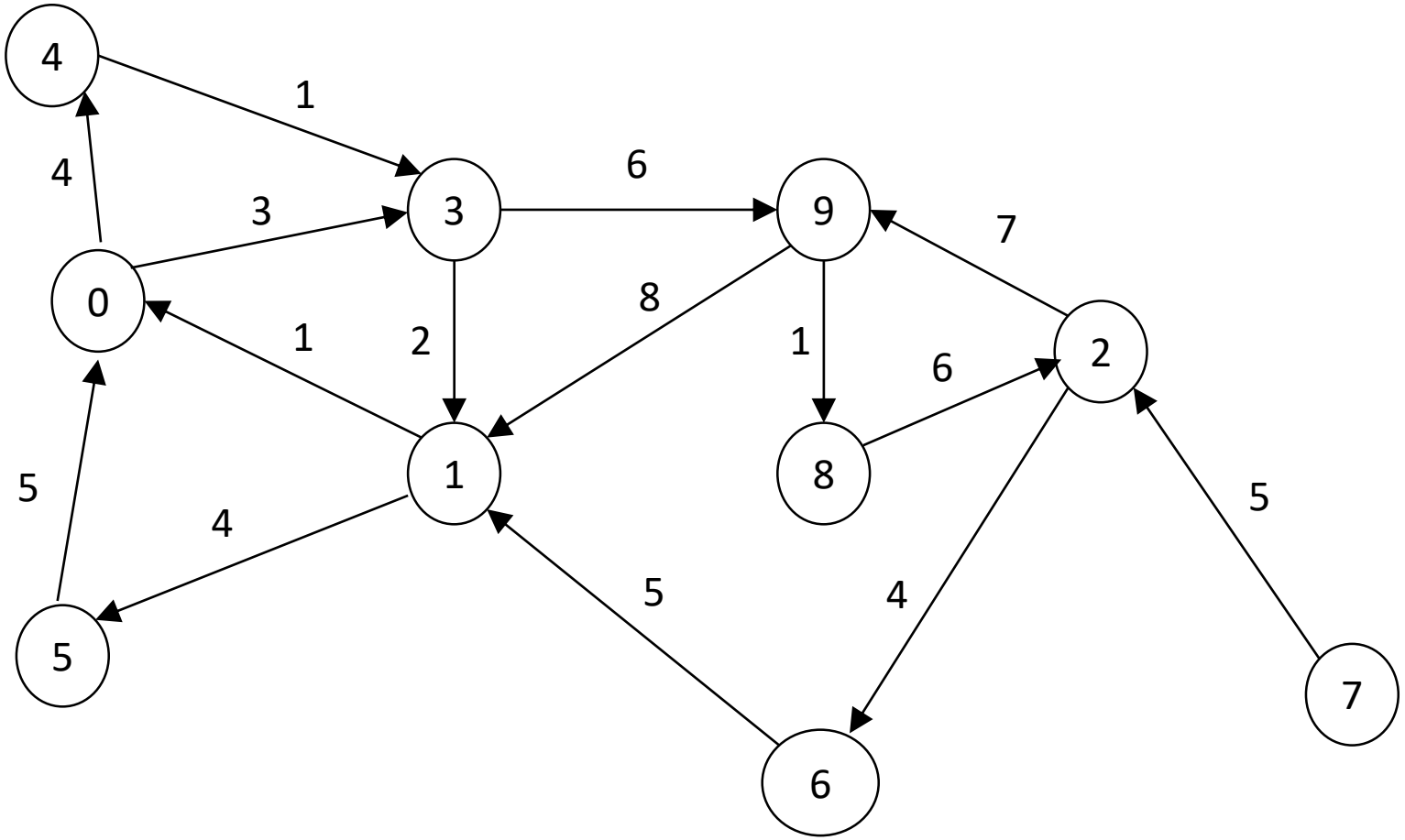
(2) (2 Marks) Plot your network (a directed and weighted graph) based on your 16-digit number.

(3) (2 Marks) Select Node *i* to be the start node with the objective to maximize the number of nodes that are reachable from Node *i*. If more than one node satisfies this requirement, choose one arbitrarily. If every node is reachable from any other node, the choice of Node *i* can be arbitrary.

(4) (**20 Marks)** Implement Dijksra's algorithm to find the shortest path between Node *i* and every other reachable node. Also provide the cost/distance of all these shortest paths. Show all steps.

In the example below, *i* = 7 because all other nodes are reachable from Node 7 but Node 7 is not reachable from any other node, so Node 7 achieves the objective that the maximal number of nodes are reachable from it.

7298261503910431
Weighted Directed Graph

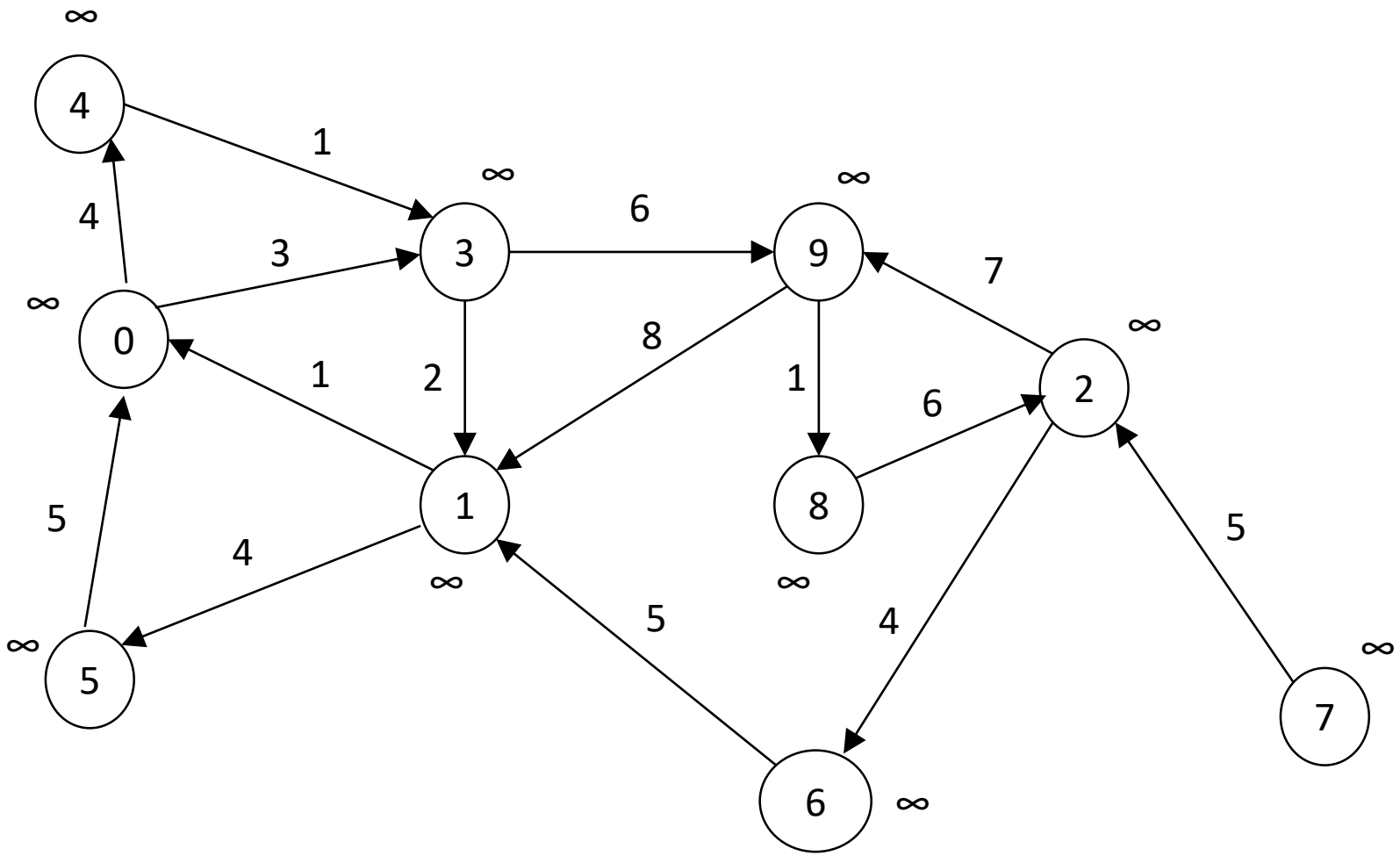


Because all nodes are reachable from Node 7 but Node 7 is not reachable from any other node, choosing Node 7 as the start node maximizes the number of reachable nodes. Therefore we choose Node 7 as the start node and we will find the shortest paths from Node 7 to all other nodes.

S = set of visited nodes.
Q = the queue = set of unvisited nodes.
At the beginning, S is an empty set and Q includes all nodes.

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

7298261503910431
Weighted Directed Graph



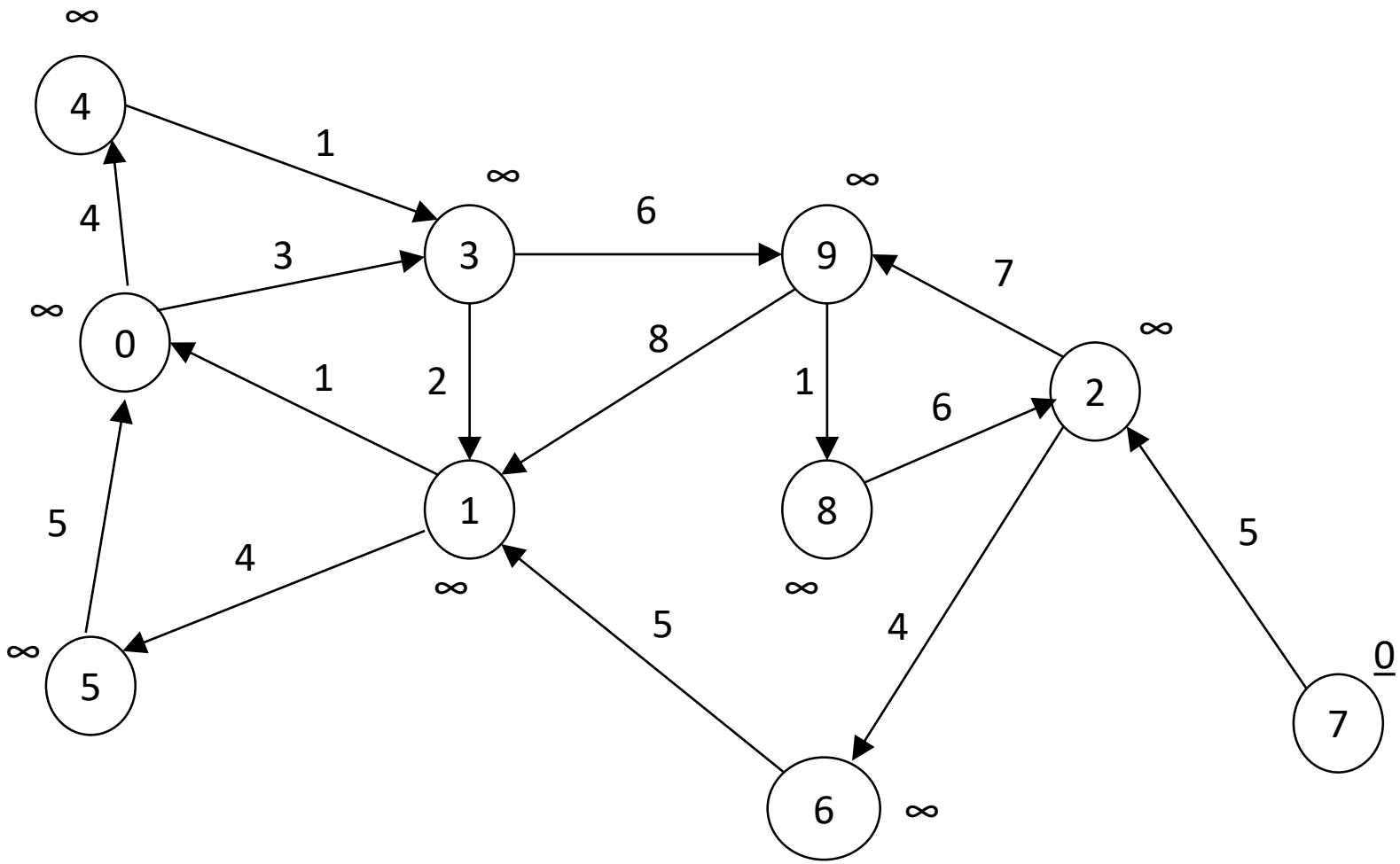
In the first step where we consider that Node 7 has the minimal distance from node 7 which is equal to 0, the labels of the nodes have the following values:

0	1	2	3	4	5	6	7	8	9
∞	∞	∞	∞	∞	∞	∞	<u>0</u>	∞	∞

(The minimal value of the shortest path length is underlined)
Now Node 7 is included in S, and Q includes all other nodes.

S = {7}, and Q = {0,1,2,3,4,5,6,8.9}

7298261503910431
Weighted Directed Graph

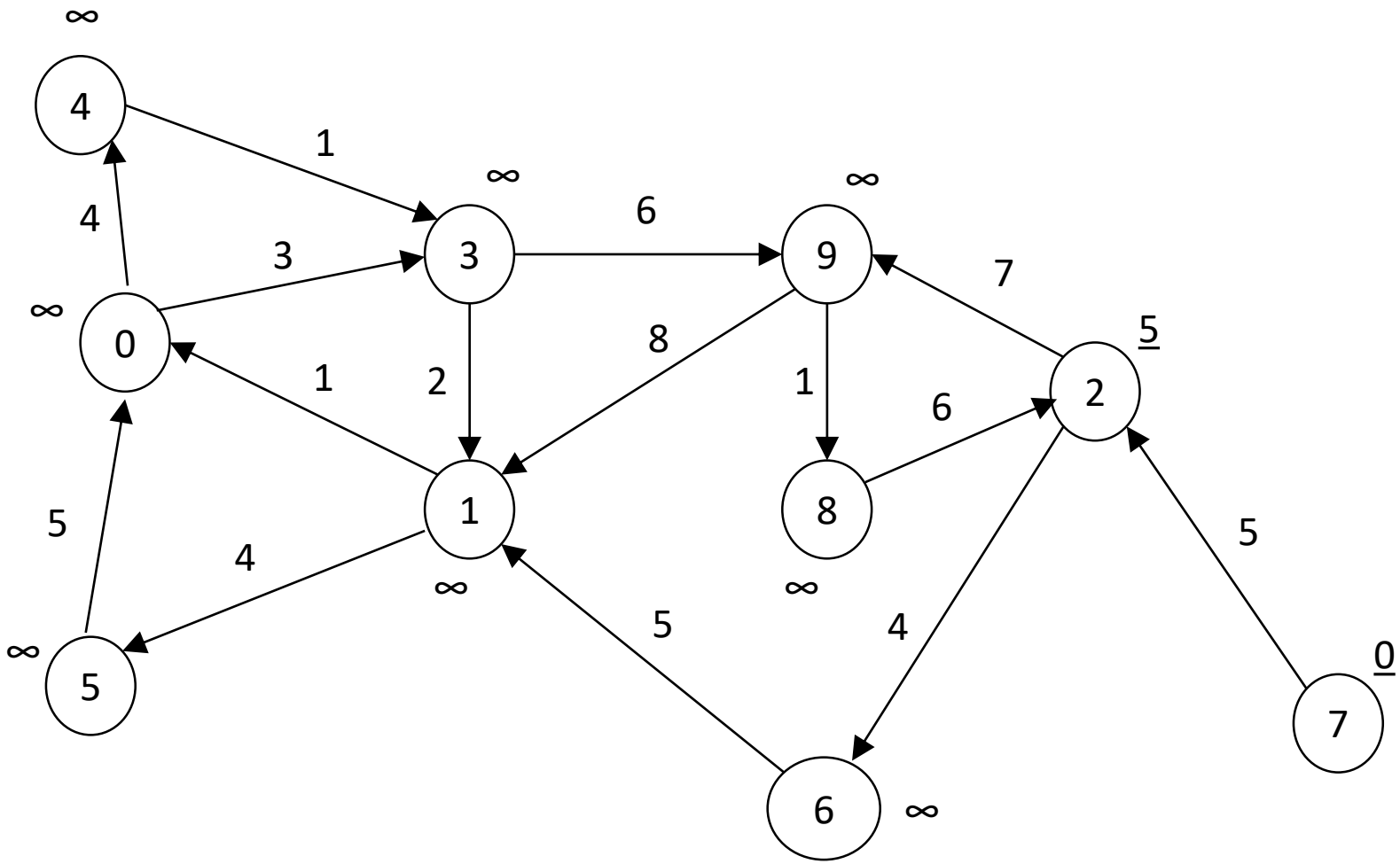


The next node to join the set of visited nodes S is Node 2 with label (cost/distance) = 5, so Node 2 joins the set S. Updating the labels, we obtain:

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

S = {2,7}, and Q = {0,1,3,4,5,6,8,9}

7298261503910431
Weighted Directed Graph

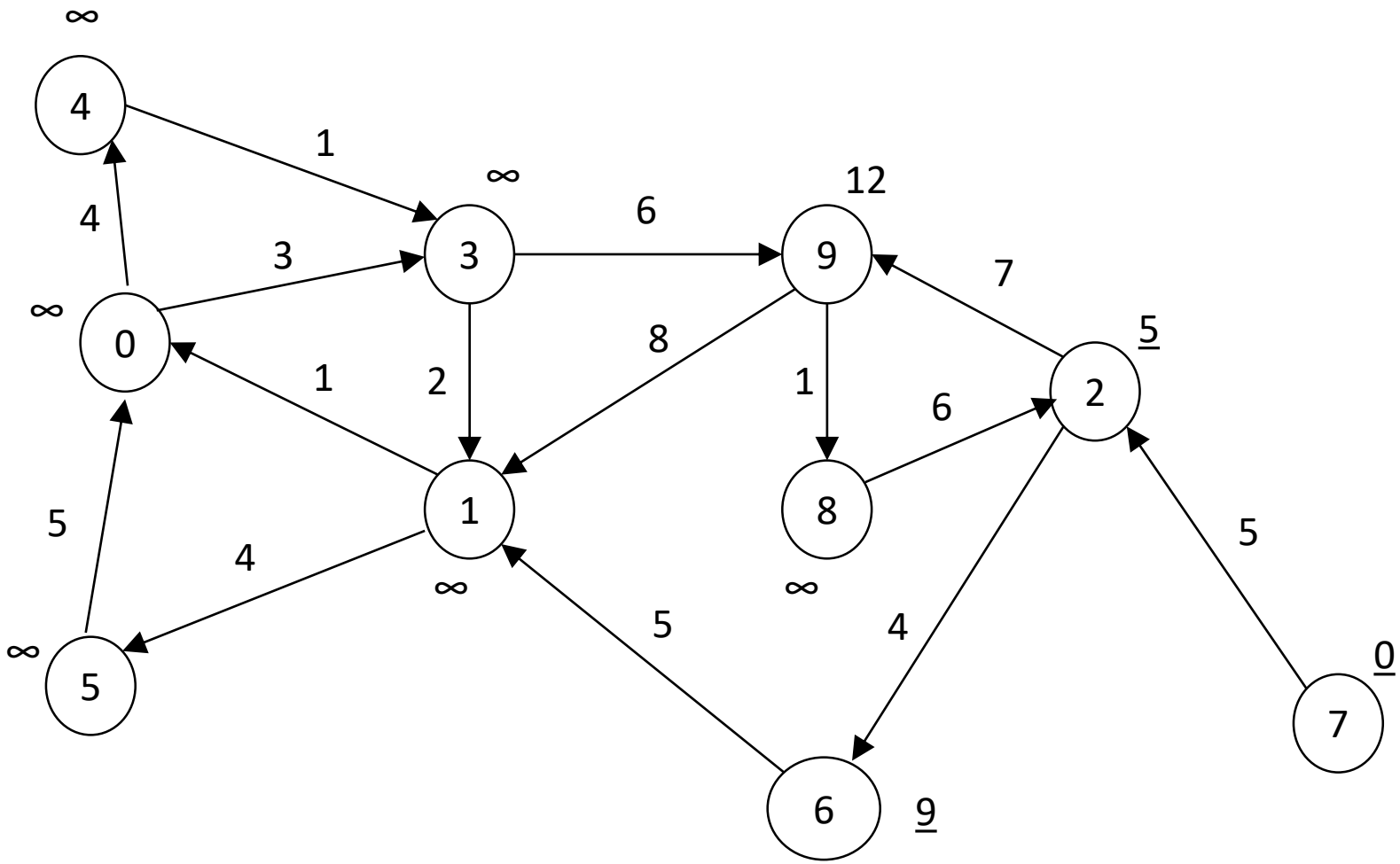


Next, the nodes that have their labels updated are Node 6 with label = 5 + 4 = 9 (through Node 2) and Node 9 with label = 5 + 7 = 12 (through Node 2). Because min(9,12) = 9, Node 6 joins the set of visited nodes S label (cost/distance) = 9. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
∞	∞	5	∞	∞	∞	9	0	∞	12

S = {2,6,7}, and Q = {0,1,3,4,5,8,9}

7298261503910431
Weighted Directed Graph

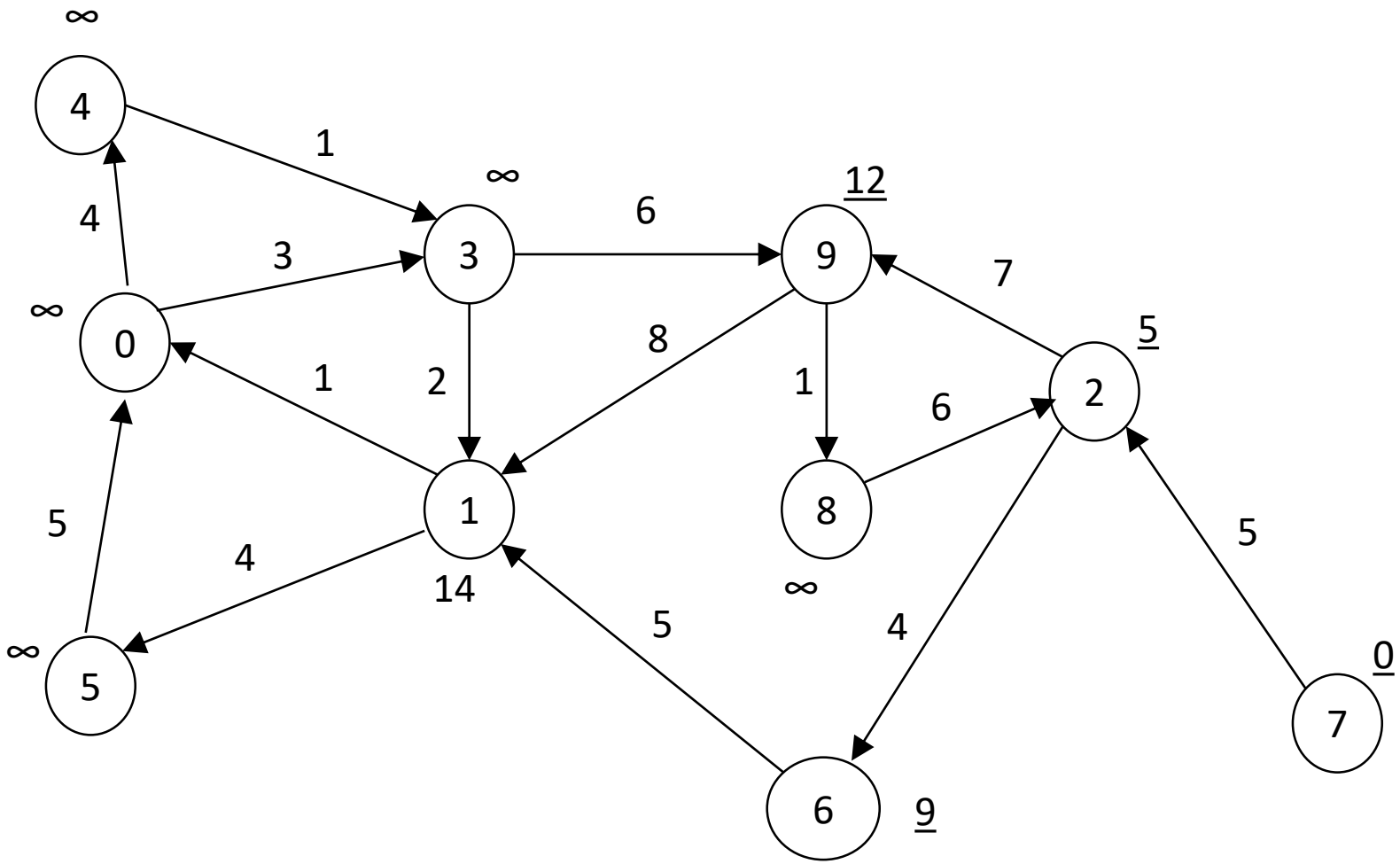


Next, the Node 1's label is updated to 9 + 5 = 14 (through Node 6) , but since Node 9 label = 12, Node 9 joins S. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
∞	14	5	∞	∞	∞	9	0	∞	12

S = {2,6,7,9}, and Q = {0,1,3,4,5,8}

7298261503910431
Weighted Directed Graph

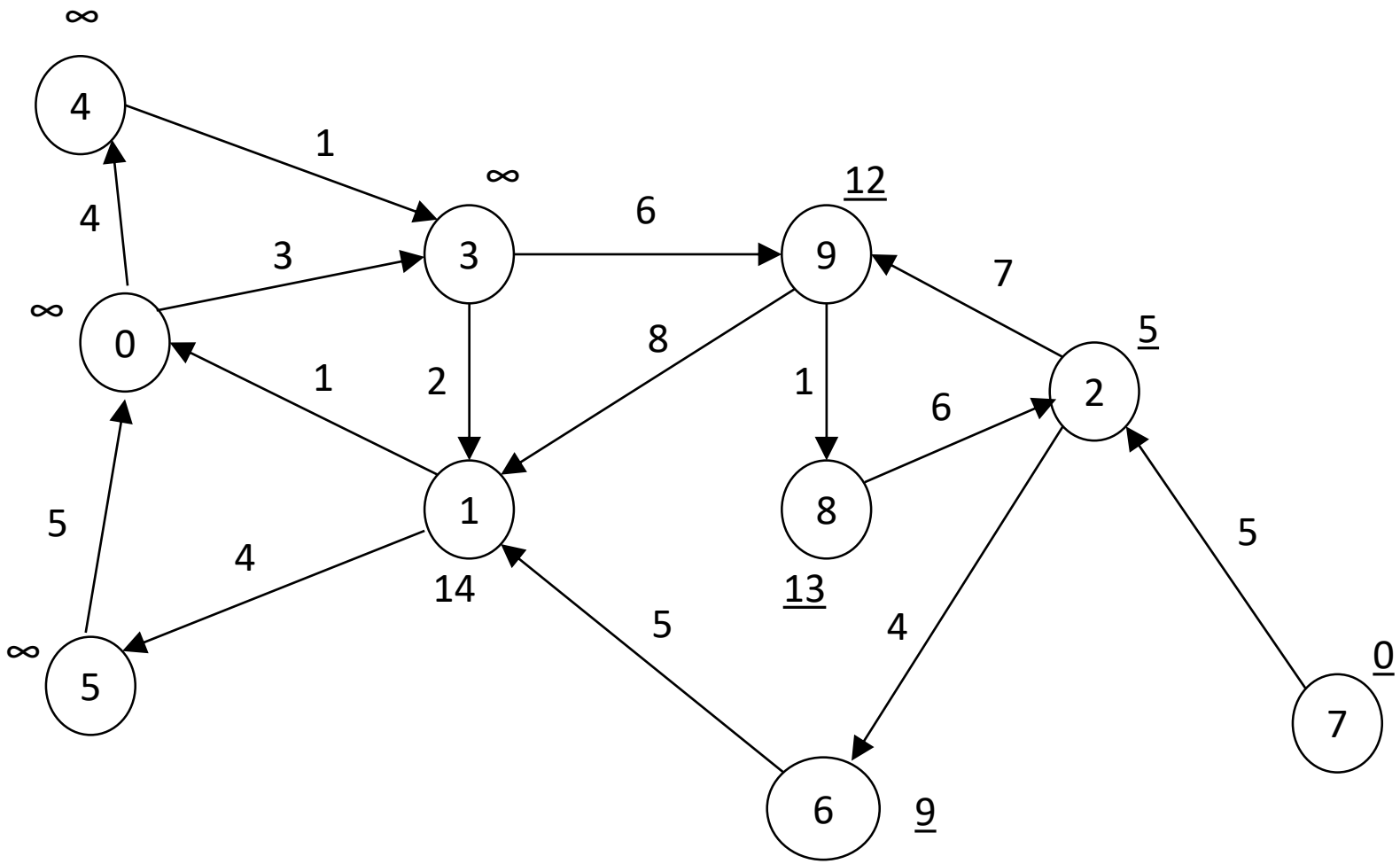


Next, Node 1 retains its label = 14 because 12 + 8 = 20 (through Node 9) is higher, and Node 8's label is updated to 12 +1 =13 (through Node 9). Because min(13,14) = 13, Node 8 joins S. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
∞	14	5	∞	∞	∞	9	0	13	12

S = {2,6,7,8,9}, and Q = {0,1,3,4,5}

7298261503910431
Weighted Directed Graph

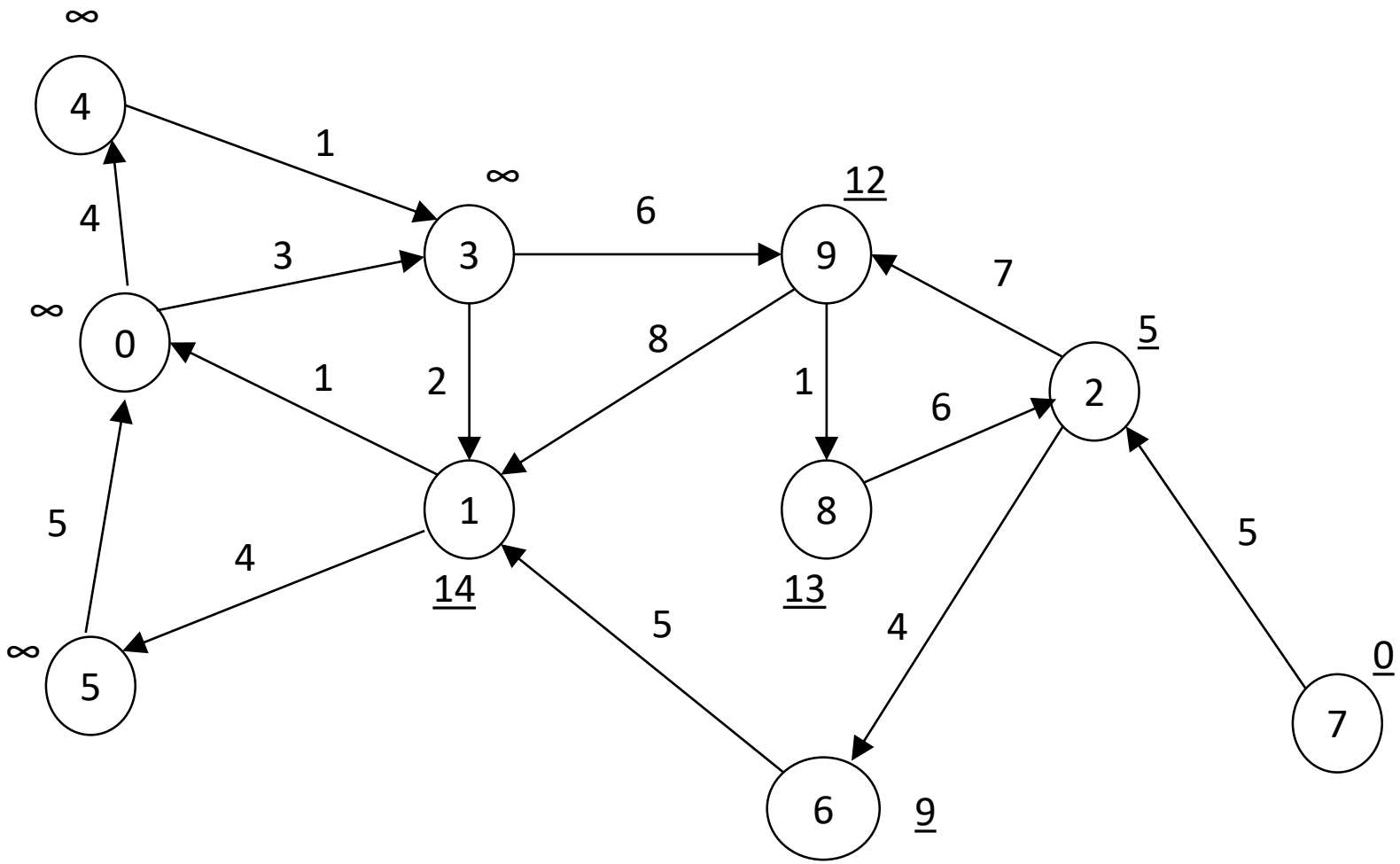


Next, Node 1 with label = 14 (through Node 6) joins S. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
∞	14	5	∞	∞	∞	9	0	13	12

S = {1,2,6,7,8,9}, and Q = {0,3,4,5}

7298261503910431
Weighted Directed Graph

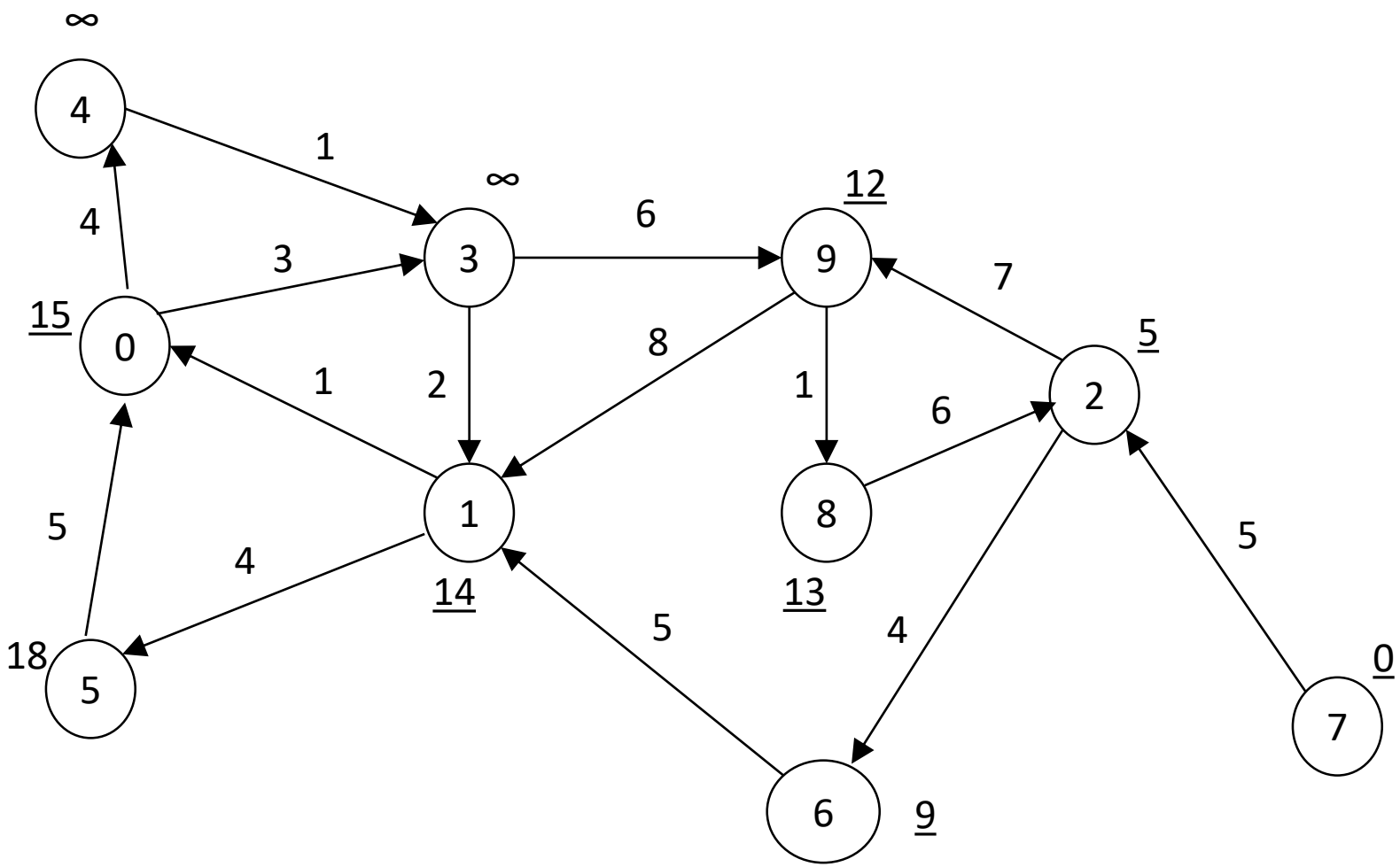


Next, the two nodes Node 0 and Node 5 update their labels Node 0's label = 14 + 1 = 15 (through Node 1) and Node's 5 label = 14 + 4 = 18 (through Node 1). Since min(15,18)=15, Node 0 joins S. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
15	14	5	∞	∞	18	9	0	13	12

S = {0,1,2,6,7,8,9}, and Q = {3,4,5}

7298261503910431
Weighted Directed Graph

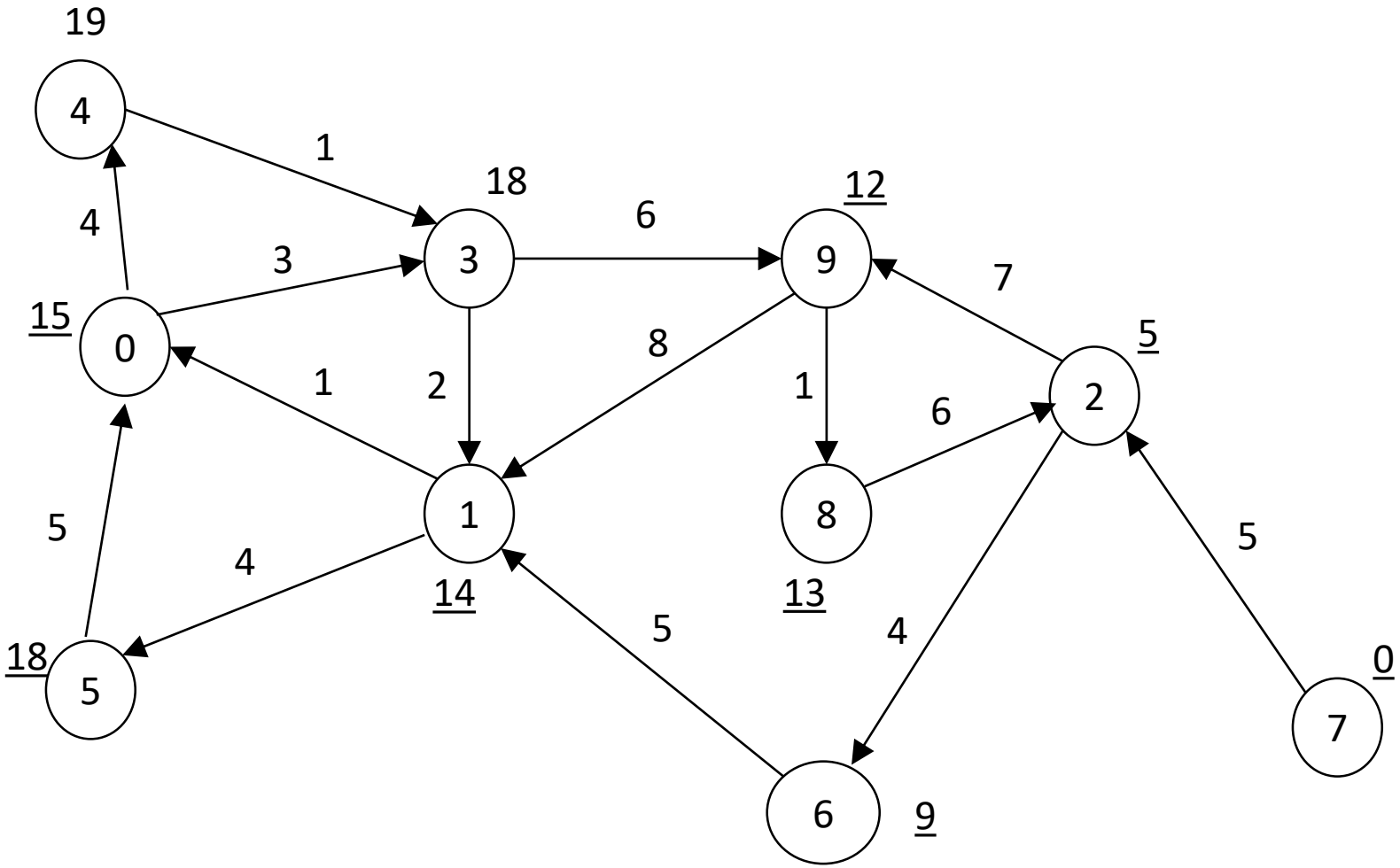


Next, the two nodes Node 4 and Node 3 update their labels through Node 0. For Node 4, label = 15 + 4 = 19, and for Node 3, label = 15 + 3 = 18. Node 5 retains its label = 18. Since min(19,18,18)=18, Node 5 joins S. This is an arbitrary choice between Node 5 and Node 3. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
15	14	5	18	19	18	9	0	13	12

S = {0,1,2,5,6,7,8,9}, and Q = {3,4}

7298261503910431
Weighted Directed Graph

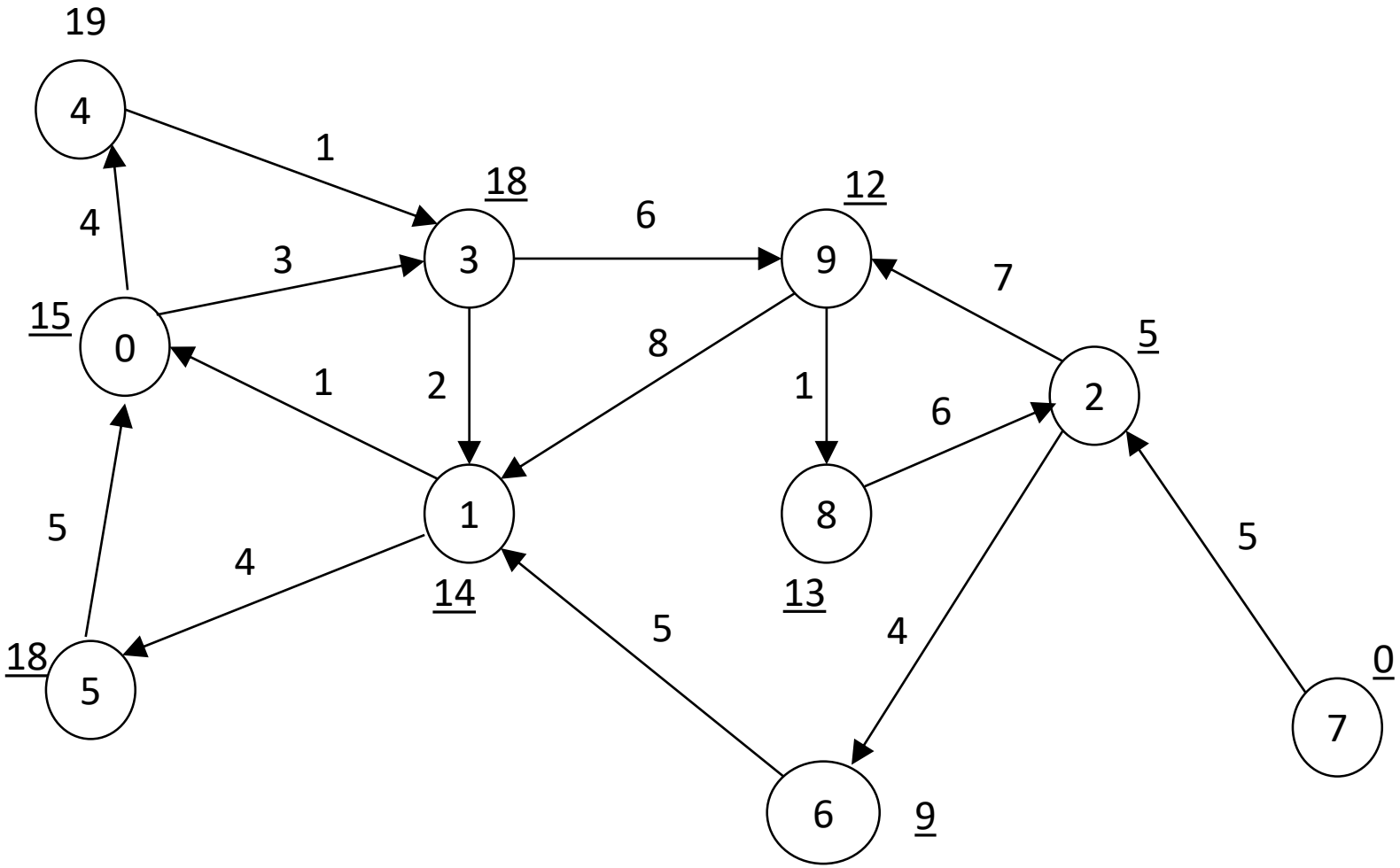


Next, Node 3 joins S because min(18,19)-18. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
15	14	5	18	19	18	9	0	13	12

S = {0,1,2,3,5,6,7,8,9}, and Q = {4}

7298261503910431
Weighted Directed Graph

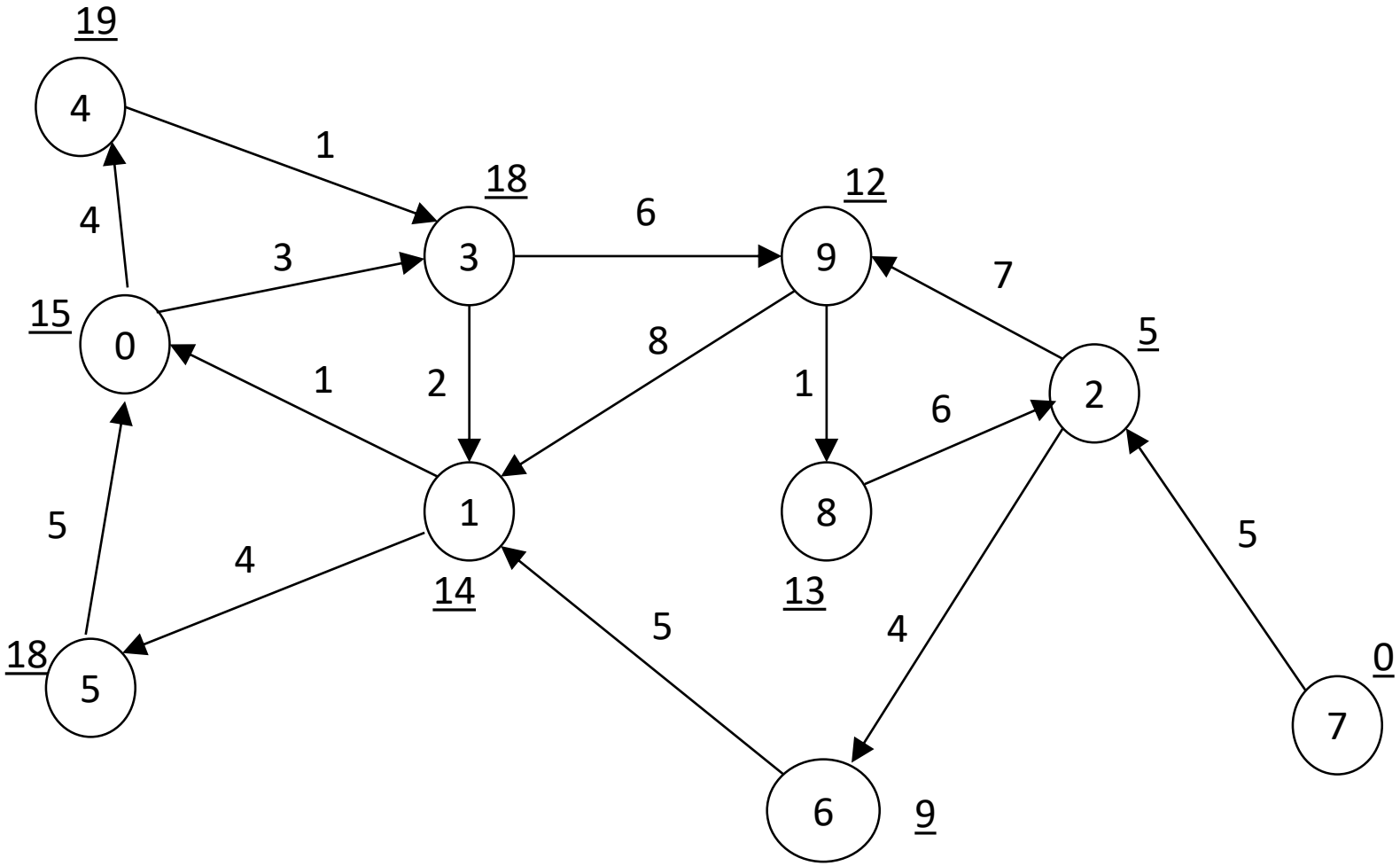


Finally, Node 4 joins S. Updating the labels, we obtain:

0	1	2	3	4	5	6	7	8	9
15	14	5	18	19	18	9	0	13	12

S = {0,1,2,3,4,5,6,7,8,9}, and Q = {}

7298261503910431
Weighted Directed Graph



The costs/distances from Node 7 to all reachable nodes are given by their final labels.

The shortest paths from Node 7 to each reachable node are given by

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

A Dynamic Programming (DP) Formulation of Dijkstra Shortest path algorithm ([4], page 56)

We now show that Dijkstra's procedure can be given a dynamic-programming formulation. Let $N_i(1)$ be the set composed of the i closest nodes (by shortest path) to node 1, where $N_1(1) = \{1\}$. (In case this set is not well defined due to ties, $N_i(1)$ is a set of i nodes such that no node not in $N_i(1)$ is closer to 1 than **any** node in $N_i(1)$.) Define the optimal value function $f_i(j)$ as follows:

$f_i(j) =$ the length of the shortest path from node 1 to node j when we are restricted to use paths such that all nodes preceding node j belong to $N_i(1)$.

A DP Formulation of Dijkstra's Shortest path algorithm (Cont'd) ([4], page 57)

Then the dynamic-programming formulation is as follows:

At stage i $(i = 2, 3, \ldots, N)$ determine a node, k_i, such that $k_i \notin N_{i-1}(1)$ and $f_{i-1}(k_i) = \min_{j \notin N_{i-1}(1)} f_{i-1}(j)$. Set $N_i(1) = N_{i-1}(1) \cup \{k_i\}$. Then compute $f_i(j)$ from

recurrence relation:

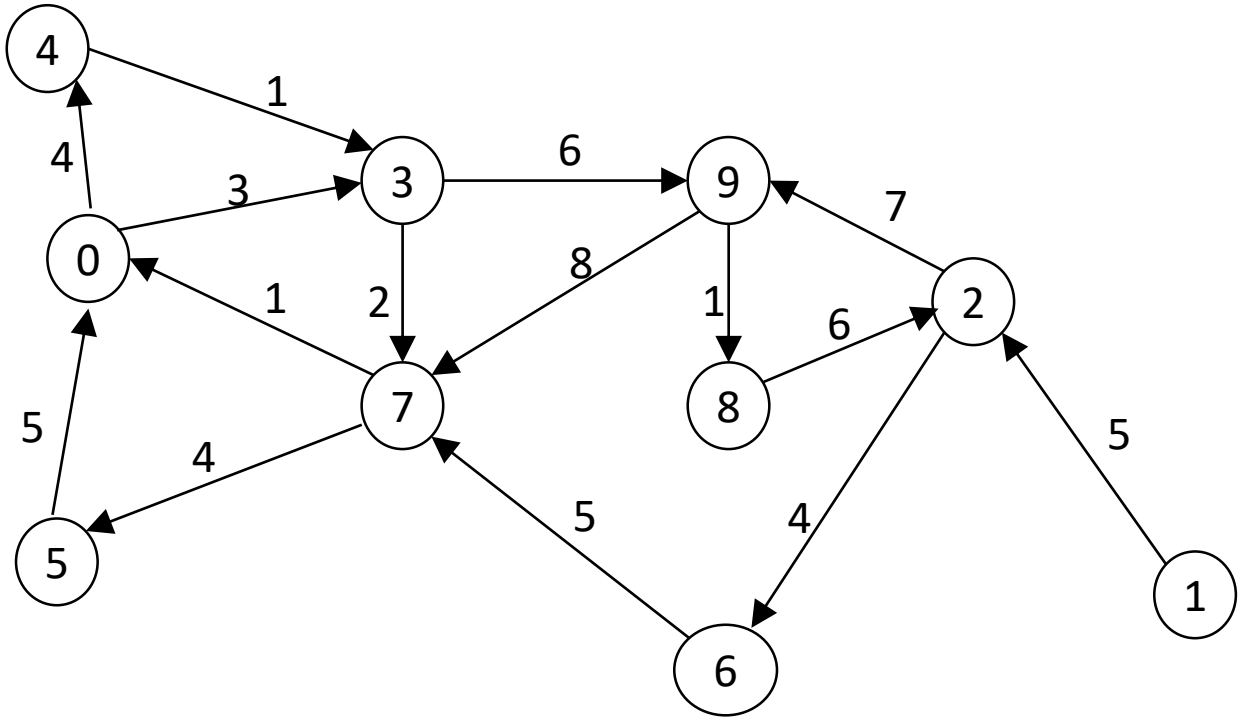
$$f_i(j) = \begin{cases} f_{i-1}(j) & \text{if} \quad j \in N_i(1), \\ \min\{ f_{i-1}(j), f_{i-1}(k_i) + d_{k_i, j} \} & \text{if} \quad j \notin N_i(1); \end{cases} \tag{4.6}$$

boundary conditions: $N_1(1) = \{1\}$, $k_1 = 1$, and $f_1(j) = d_{1j}$; (4.7)

answers: $f_i(k_i)$ for $i = 1, 2, \ldots, N$. (4.8)

DP/Dijkstra Assignment 1

Consider the following network which was created by swapping nodes 1 and 7 in the previous network. Use the above described DP formulation to find the shortest path between Node 1 and any other reachable node.



DP/Dijkstra Assignment 2

Repeat Assignment 1 to your network in Question 1 in Test 1 with the following change: if you did not start the implementation of Dijkstra Algorithm in Node 1, swap the node labeled 1 with the node you chose to start the implementation of Dijkstra Algorithm and start the DP algorithm with the new Node 1.

The secretary problem

There are N candidates to be interviewed in a sequence, before the interviews, they are considered to be all equally qualified. After every interview, a decision on hiring is made and cannot be changed later.
The aim is to maximize the expected probability to choose the best among N candidates.

The world is not that simple

An implied assumption here is that although before the interviews, all candidates are considered equal, after m interviews ($m = 1, 2, 3, \ldots N$), **it is possible to tell who is the best among the m candidates that were interviewed.** This is a strong assumption. The world is not that simple. The other abovementioned assumption that one cannot change a decision once made, and the objective of maximizing the probability of selecting the best among the N candidates may also not apply in many practical situations.

Mathematics (Slide 6 in Graph Theory)

“Many people think that mathematics is so complicated, this is wrong. Mathematics is an extremely simple approximation of the world. The world itself is more complicated that anything we can think of.”

PROFESSOR CÉDRIC VILLANI, FIELDS MEDAL WINNER

http://www.abc.net.au/radionational/programs/scienceshow/5014540

Observation

Our objective is to maximize the probability of choosing the best among the N candidates.

Therefore, under the optimal policy, after the mth interview ($m = 1, 2, 3, \ldots N$), we will never select the mth candidate unless this candidate is the best among the first m candidates. If we select a candidate who is not the best of the first m candidates, then the probability of selecting the best of N candidates is zero. If a candidate is the best of the first m candidates, then we need to decide whether or not this candidate is selected.

Definitions [5]

$v(m)$ = After the mth interview, the probability to choose the best among N candidates if the mth candidate is not selected.

$u(m)$ = After the mth interview, the probability to choose the best among N candidates under optimal policy if the mth candidate is the best among the first m candidates.

$v(m)$ and $u(m)$ are two optimal value functions.

[5] M. J. Beckman, "Dynamic Programming and the secretary problem" *Computer. Math. Applic.* vol. 19., no. 11, pp. 25-28, 1990.

Recurrent Relation [5]

$$v(m) = \frac{m}{m+1} v(m+1) + \frac{1}{m+1} u(m+1) \qquad \text{(R1)}$$

Explanation: Given that the mth candidate is not chosen, the m + l th candidate will be either not the best among the first m+1 candidates, with probability $\frac{m}{m+1}$, or will be the best among the first m+1 candidates, with probability $\frac{1}{m+1}$.

If the m + l candidate is not the best among the first m+1 candidates, s/he is definitely not selected, then the probability of selecting the best candidate among the N candidates is $v(m+1)$. If the m + l th candidate is the best among the first m+1 candidates, then the probability of selecting the best candidate among the N candidates is $u(m+1)$.

Recurrent Relation (Cont'd) [5]

$$u\,(m) = \max\left[\frac{m}{N}, v(m)\right] \qquad \text{(R2)}$$

Explanation: Given that the *m* th candidate is the best among the first *m* candidates, then s/he is either chosen or not. This is a decision that needs to be made whenever we have the best among the first *m* candidates.
Recall that if the *m* th is not the best among the first *m* candidates, s/he is never selected.

This probability of choosing the best among the *N* candidates is equal to $\frac{m}{N}$ if candidate m is chosen, and it is equal to $v(m)$ if candidate *m* is not chosen. By choosing the maximum of the two we make the optimal decision.

Boundary Condition [5]

$$u\,(N) = 1, \qquad v(N) = 0$$

Explanation: Given that candidate N is the best among the first N candidates, then s/he is chosen for sure and then the probability to choose the best candidate among the N candidates is equal to 1. Therefore, $u\,(N) = 1$.

After the Nth interview if the Nth candidate is not chosen, then the probability of choosing the best candidate among the N candidates is equal to zero. Therefore, $v(N) = 0$.

Now with only one optimal value function $v(m)$

Substituting R2 in R1, we obtain the recurrence relation:

$$v(m) = \frac{m}{m+1} v(m+1) + \frac{1}{m+1} \max\left[\frac{m+1}{N}, v(m+1)\right]$$

with the boundary condition $v(N) = 0$

Then, $v(N-1) = \frac{1}{N}$ Why?

P(m) = the optimal policy function defined as the decision Accept/reject of the *m*th candidate who is the best among the first *m* candidates. Recall that if a candidate is not the best among the first *m* candidates, s/he is always rejected.

Example: $N = 10$

$v(10) = 0$ $\qquad$ $P(10) = \text{Accept}$

To find $P(m)$ we look at $\max\left[\frac{m}{N}, v(m)\right]$.

If $\frac{m}{N} > v(m)$, the decision is Accept; otherwise, Reject.

$v(9) = \frac{1}{10}$; $\quad$ $P(9) = \text{Accept}$, because $\frac{9}{10} > \frac{1}{10}$.

Example: $N = 10$ (Cont'd)

$$v(8) = \frac{8}{9} \times \frac{1}{10} + \frac{1}{9} \max\left[\frac{9}{10}, \frac{1}{10}\right] = \frac{8}{90} + \frac{9}{90} = 0.188889$$

$$P(8) = \text{Accept, because } \frac{8}{10} > 0.188889.$$

$$v(7) = \frac{7}{8} \times 0.188889 + \frac{1}{8} \max\left[\frac{8}{10}, 0.188889\right] = 0.265278$$

$$P(7) = \text{Accept, because } \frac{7}{10} > 0.265278.$$

Example: $N = 10$ (Cont'd)

$$v(6) = 0.327381$$

$$P(6) = \text{Accept}, \quad \text{because} \quad \frac{6}{10} > 0.327381$$

$$v(5) = 0.3728175$$

$$P(5) = \text{Accept}, \ \text{because} \ \frac{5}{10} > 0.3728175.$$

$$v(4) = 0.398254$$

$$P(4) = \text{Accept}, \ \text{because} \ \frac{4}{10} > 0.398254.$$

Example: $N = 10$ (Cont'd)

$$v(3) = \frac{3}{4} \times 0.398254 + \frac{1}{4}\max\left[\frac{4}{10}, 0.398254\right] = 0.39869$$

$$P(3) = \text{Reject}, \qquad \text{because} \quad \frac{3}{10} < 0.39869$$

$$v(2) = 0.39869; \quad P(2) = \text{Reject}, \qquad \text{because} \quad \frac{2}{10} < 0.39869$$

$$v(1) = 0.39869; \quad P(1) = \text{Reject}, \qquad \text{because} \quad \frac{1}{10} < 0.39869$$

The critical m (m*)

From the case N = 10, we observe the following.
For $m < 4$, according to the optimal policy, we reject any candidate even if s/he is the best candidate so far.
From m = 4 and onwards, we select any candidate who is the best among the m candidates that have been interviewed. The critical stage m where we start selecting any best candidate so far is denoted m*.
For the case N = 10 case, m* = 4.

A simple approximation for a large N

For large *N*, the critical m^* value can be approximated by

$m^* = \left\lceil \frac{N}{e} \right\rceil$, that is, divide *N* by *e* and round the result up.

Then the probability of selecting the best among *N* candidates

Is approximated by $v(1) = \frac{1}{e}$.

Assignment

Consider the cases $N = 10$, $N = 50$, $N = 100$, $N = 1000$, and use excel sheet to calculate m^* and $v(1)$.

Then, compare your results with the relevant approximations, and comment of the approximation accuracy.

The traveler salesperson problem (TSP or TSP Problem) ([4], page 69)

We have a general network consisting of N cities (or nodes) such as the one described at the beginning of Chapter 4. The cities are arbitrarily numbered from 1 to N, and d_{ij} is the distance (or travel time, or cost, etc.) from city i to city j. A salesman would like to begin at city 1, visit each of the other cities once and only once, and then return to city 1 (such a path is called a *tour*). In what order should he visit the cities so that the total distance he travels is minimized?

Dynamic Programming (DP) Formulation of TSP ([4], P. 69-70)

Let $N_j = \{2, 3, \ldots, j-1, j+1, \ldots, N\}$

and let S be a subset of N_j containing i members. Define the optimal value function $f_i(j, S)$ as

$f_i(j, S) =$ the length of the shortest path from city 1 to city j via the set of i intermediate cities S (the stage variable i indicates the number of cities in S). (5.1)

Please write down now the recurrence relation and boundary condition, upload your solution on Canvas Discussions and provide it also in Chat.

Dynamic Programming (DP) Formulation of TSP ([4], P. 69-70) (cont'd)

Recurrence relation and boundary condition

recurrence relation:

$$f_i(j, S) = \min_{k \in S} \left[f_{i-1}(k, S - \{k\}) + d_{kj} \right]$$
$$(i = 1, 2, \ldots, N-2; \quad j \neq 1; \quad S \subseteq N_j); \qquad (5.2)$$

boundary condition: $f_0(j, -) = d_{1j}$; (5.3)

answer: $\min_{j=2, 3, \ldots, N} \left[f_{N-2}(j, N_j) + d_{j1} \right]$. (5.4)

DP-TSP: Intuitive justification ([4], page 70)

Let us intuitively justify the recurrence relation. Suppose that we would like to compute $f_i(j, S)$. Consider the shortest path from city 1 to city j via S which has city k (a member of S) immediately preceding city j. Since the cities in $S - \{k\}$ must be visited in an optimal order, the length of this path is $f_{i-1}(k, S - \{k\}) + d_{kj}$. Furthermore, since we are free to choose city k optimally, it is clear that $f_i(j, S)$ is correctly given by (5.2).

Therefore, to compute the length of the shortest tour, we begin by computing $f_1(j, S)$ (for every j, S pair) from the values of f_0. We then compute $f_2(j, S)$ (for every j, S pair) from the values of f_1. We continue in the same manner until $f_{N-2}(j, N_j)$ has been computed for every city j. The length of the shortest tour is then given by (5.4), since some city, j, must be the last city visited before returning to city 1. The corresponding shortest tour can be obtained by recording, for each stage and state, that city, k, which minimizes the r.h.s. of (5.2).

DP-TSP: Example [4]

Table 5.1. *The distance from city i to city j, d_{ij}, for a network of five cities*

i \ j	1	2	3	4	5
1	0	3	1	5	4
2	1	0	5	4	3
3	5	4	0	2	1
4	3	1	3	0	3
5	5	2	4	1	0

(the optimal policy function is given in parentheses):

$$f_0(2, \_) = d_{12} = 3(1),$$

$$f_0(3, \_) = 1(1),$$

DP-TSP: Example (cont'd) [4]

$$f_0(4, \_) = 5(1),$$
$$f_0(5, \_) = 4(1).$$

$$f_1(2, \{3\}) = f_0(3, \_) + d_{32} = 1 + 4 = 5(3),$$
$$f_1(2, \{4\}) = 5 + 1 = 6(4),$$
$$f_1(2, \{5\}) = 4 + 2 = 6(5),$$
$$f_1(3, \{2\}) = 3 + 5 = 8(2),$$
$$f_1(3, \{4\}) = 5 + 3 = 8(4),$$
$$f_1(3, \{5\}) = 4 + 4 = 8(5),$$
$$f_1(4, \{2\}) = 3 + 4 = 7(2),$$
$$f_1(4, \{3\}) = 1 + 2 = 3(3),$$
$$f_1(4, \{5\}) = 4 + 1 = 5(5),$$
$$f_1(5, \{2\}) = 3 + 3 = 6(2),$$
$$f_1(5, \{3\}) = 1 + 1 = 2(3),$$
$$f_1(5, \{4\}) = 5 + 3 = 8(4).$$

$$f_2(2, \{3, 4\}) = \min[f_1(3, \{4\}) + d_{32}, f_1(4, \{3\}) + d_{42}]$$
$$= \min[8 + 4, 3 + 1] = 4(4),$$
$$f_2(2, \{3, 5\}) = \min[8 + 4, 2 + 2] = 4(5),$$
$$f_2(2, \{4, 5\}) = \min[5 + 1, 8 + 2] = 6(4),$$
$$f_2(3, \{2, 4\}) = \min[6 + 5, 7 + 3] = 10(4),$$
$$f_2(3, \{2, 5\}) = \min[6 + 5, 6 + 4] = 10(5),$$
$$f_2(3, \{4, 5\}) = \min[5 + 3, 8 + 4] = 8(4),$$
$$f_2(4, \{2, 3\}) = \min[5 + 4, 8 + 2] = 9(2),$$
$$f_2(4, \{2, 5\}) = \min[6 + 4, 6 + 1] = 7(5),$$
$$f_2(4, \{3, 5\}) = \min[8 + 2, 2 + 1] = 3(5),$$
$$f_2(5, \{2, 3\}) = \min[5 + 3, 8 + 1] = 8(2),$$
$$f_2(5, \{2, 4\}) = \min[6 + 3, 7 + 3] = 9(2),$$
$$f_2(5, \{3, 4\}) = \min[8 + 1, 3 + 3] = 6(4).$$

DP-TSP: Example (cont'd) [4]

$$f_3(2, \{3, 4, 5\}) = \min[\, f_2(3, \{4, 5\}) + d_{32},\ f_2(4, \{3, 5\}) + d_{42},\ f_2(5, \{3, 4\}) + d_{52}\,]$$
$$= \min[8 + 4,\ 3 + 1,\ 6 + 2] = 4(4),$$
$$f_3(3, \{2, 4, 5\}) = \min[6 + 5,\ 7 + 3,\ 9 + 4] = 10(4),$$
$$f_3(4, \{2, 3, 5\}) = \min[4 + 4,\ 10 + 2,\ 8 + 1] = 8(2),$$
$$f_3(5, \{2, 3, 4\}) = \min[4 + 3,\ 10 + 1,\ 9 + 3] = 7(2).$$

$$\text{answer} = \min_{j=2,\ldots,5} [\, f_3(j, \{2, 3, 4, 5\} - \{j\}) + d_{j1}\,]$$
$$= \min[4 + 1,\ 10 + 5,\ 8 + 3,\ 7 + 5] = 5(2).$$

Therefore, the length of the shortest tour is 5 and the corresponding shortest tour is 1–3–5–4–2–1.

But computation and storage requirements are significant for large *N*.

You are welcome to write a program and try for increasing N values.

DP-TSP: Doubling up procedure for the case of symmetric distances [4]

Valid when $d_{ij} = d_{ji}$ for all i and j

We shall assume that N is even (minor modifications are required when N is odd). Suppose that $f_1, f_2, \ldots, f_{(N-2)/2}$ have been computed from (5.2) and (5.3). Consider any tour that begins and ends at city 1. Some city, say j, will be visited at the halfway point. The length of the shortest path from city 1 to city j via any set of $(N-2)/2$ intermediate cities is given by $f_{(N-2)/2}$. Therefore, the length of the shortest tour is given by

$$\min_{j=2,3,\ldots,N} \left[\min_{S \subseteq N_j} \{ f_{(N-2)/2}(j, S) + f_{(N-2)/2}(j, N_j - S) \} \right]. \qquad (5.5)$$

Assignment

Consider again Fig. 5.1 in [4] and the problem of solving the TSP Problem for 5 cities with the following difference. The link costs are now based on your unique number. If the link cost is *j* in the table, it changes to the *j*th digit of your unique number. For example, the cost of link 1-2 was 3. Change it to the value of your 3-rd digit. For link 1-1, the cost remains 0. Provide a Dynamic Programming (DP) formulation for the new problem and solve it completely using DP (find the shortest path to start in node 1 and return to node 1 and its cost).
Show all steps.
Upload your solution to Canvas/Discussions.