Dynamic Programming is about optimization on multi-staged (multi-layered) directed graphs. Let $k = 1, \ldots, n$ denote the kth stage (layer) of the graph, and $s_k \in V_k$ denote the state (node/vertex) variable from the set, V_k, of the beginning states (nodes/vertices) of stage k, and similarly, $s_{k+1} \in V_{k+1}$ denote the state (node/vertex) variable from the set, V_{k+1}, of the destination states (nodes/vertices) of stage k. Thus, by definition, V_{k+1} is the set of beginning states for the $(k+1)$th stage, and so on. All state/node/vertex sets, V_k, contain no more than M many elements each. Let $p^{(k)}_{s_k s_{k+1}}$ denote the payoff or penalty from state s_k to state s_{k+1} that is represented as a weight over the directed edge/path from s_k to s_{k+1}. The superscript, (k), is necessary for stage k because it is often the case that the states of a node set V_i are coded by natural numbers shared by other node sets V_j. (The node set V_k and V_{k+1} together with the directed edges between them with nonzero weights $p^{(k)}_{s_k s_{k+1}}$ make up a so-called directed bipartite graph.) An objective function is a function of directed paths of the graph whose value is defined by the payoff or penalty weights/parameters $p^{(k)}_{ij}$. Let opt denote either $\min$ or $\max$ operation on the objective function. The goal for such a dynamic programming problem is to determine a path from V_1 to V_{n+1} such that the objective function value along the path is optimal and the corresponding path is therefore the optimal solution to the problem. For this notes we will assume the objective function is a summation of the edge weight $p^{(k)}_{ij}$ along (directed) paths from V_1 to V_{n+1}, but the method can be adapted to other objective functional forms, such as products of the weights which may represent some kind of likelihood in probability distributions, etc. Many practical problems can be modeled by such multi-layered directed graphs even though there are no literal paths or stages in them.

Backward Iterative Method. Define

$$f_n(s_n, s_{n+1}) = p^{(n)}_{s_n s_{n+1}} \quad \text{for } s_n \in V_n \text{ and } s_{n+1} \in V_{n+1}.$$

For each $s_n \in V_n$, find

$$f^*_n(s_n) = \text{opt}_{s_{n+1} \in V_{n+1}} f_n(s_n, s_{n+1}).$$

This can be done since V_{n+1} is a finite set. (For most cases, the last state set, V_{n+1} is a singleton.)

For $1 \le k \le n-1$, suppose $f^*_{k+1}(s_{k+1})$ is found, which is the case for $k = n-1$. Define

$$f_k(s_k, s_{k+1}) = p^{(k)}_{s_k s_{k+1}} + f^*_{k+1}(s_{k+1})$$

which is the suboptimal objective value from state $s_k \in V_k$ to $s_{k+1} \in V_{k+1}$ and then onwards to V_{n+1}. Then, the kth stage optimal value from state s_k is defined inductively as

$$f^*_k(s_k) = \text{opt}_{s_{k+1} \in V_{k+1}} f_k(s_k, s_{k+1}) = \text{opt}_{s_{k+1} \in V_{k+1}} [p^{(k)}_{s_k s_{k+1}} + f^*_{k+1}(s_{k+1})] = f_k(s_k, s^*_{k+1}) \text{ for some } s^*_{k+1} \in V_{k+1}$$

which must exist since V_{k+1} is a finite set. That is, this is an optimization over all possible destination states of V_{k+1} for the suboptimal function f_k.

Finally, for $k = 1$, the optimal solution is found so that for $s^*_1 \in V_1$, which is a singleton set for many cases,

$$f^*_1(s^*_1) = \text{opt}_{s_1 \in V_1} f^*_1(s_1).$$

Now, starting from s^*_1 we can find the corresponding s^*_2 from $f^*_1(s^*_1) = f_1(s^*_1, s^*_2)$. From s^*_2 we can find s^*_3 from $f^*_2(s^*_2) = f_2(s^*_2, s^*_3)$, and so on and so forth, until we find the last state s^*_{n+1}. That is, $s^* = s^*_1 \to s^*_2 \to \cdots \to s^*_n \to s^*_{n+1}$ is the optimal path, with the optimal value $f^*_1(s^*_1)$.

In terms of a programming pseudo code, we have the following

```
**********************************************************************
Define f_n(s_n, s_{n+1}) = p^{(n)}_{s_n s_{n+1}} for s_n ∈ V_n and s_{n+1} ∈ V_{n+1}.
For each s_n ∈ V_n, find f*_n(s_n) = opt_{s_{n+1} ∈ V_{n+1}} f_n(s_n, s_{n+1}).
For k = n − 1 to 1
  For each s_k ∈ V_k
    For each s_{k+1} ∈ V_{k+1}
      Find f_k(s_k, s_{k+1}) = p^{(k)}_{s_k s_{k+1}} + f*_{k+1}(s_{k+1})
    End
    Find s*_{k+1} and f*_k(s_k) = opt_{s_{k+1} ∈ V_{k+1}} f_k(s_k, s_{k+1}) = f_k(s_k, s*_{k+1})
  End
End
Find s*_1 ∈ V_1 so that f*_1(s*_1) = opt_{s_1 ∈ V_1} f*_1(s_1).
Output the optimal path (s*_1, s*_2, ..., s*_n, s*_{n+1}) and the optimal value f*_1(s*_1).
**********************************************************************
```

Forward Iterative Method. The method has its symmetrical version in forward iteration. A programming pseudo code looks like this.

****

Define $g_1(s_1, s_2) = p^{(1)}_{s_1 s_2}$ for $s_1 \in V_1$ and $s_2 \in V_2$.

For each $s_2 \in V_2$, find $g^*_1(s_2) = \text{opt}_{s_1 \in V_1} g_1(s_1, s_2)$. (For most cases, the initial set of state, V_1 is a singleton.)

For $k = 2$ to n

  For each $s_{k+1} \in V_{k+1}$

    For each $s_k \in V_k$

      Find $g_k(s_k, s_{k+1}) = g^*_{k-1}(s_k) + p^{(k)}_{s_k s_{k+1}}$

    End

    Find s^*_k and $g^*_k(s_{k+1}) = \text{opt}_{s_k \in V_k} g_k(s_k, s_{k+1}) = g_k(s^*_k, s_{k+1})$

  End

End

Find $s^*_{n+1} \in V_{n+1}$ so that $g^*_n(s^*_{n+1}) = \text{opt}_{s_{n+1} \in V_{n+1}} g_n(s_{n+1})$.

Output the optimal path $(s^*_1, s^*_2, \ldots, s^*_n, s^*_{n+1})$ and the optimal value $g^*_n(s^*_{n+1})$.

****