3. Dynamic Programming

(*) General Method: Dynamic Programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of the sequence of decisions.

Dynamic Programming is used to find optimal solution to the given problem. In this method, solution to a problem is obtained by making sequence of decisions In dynamic programming an optimal sequence of decisions is obtained by using principal of optimality.

Principle of Optimality: the principle of optimality states that " in an optimal sequence of decisions or choices each subsequence must also be optimal"

(*) Differences between Divide and Conquer and Dynamic Programming:

Divide and Conquer:

1. the problem is divided into smaller subproblems. these subproblems are solved independantly. Finally, all the solutions of subproblems are collected together to get the solution to the given problem.
2. In this method duplications in subsolutions are ignored.
3. this method is less efficient.
4. this method uses top down approach of problem solving.
5. this method splits its input at specific points usually in the middle.

Dynamic Programming:

① In this method, many decision sequences are generated and all the overlapping subinstances are considered.

② In this method, duplications are not allowed.

③ This method is efficient than Divide and Conquer

④ In this method, bottom up approach is used.

⑤ Dynamic Programming splits its input at every possible split point. then it determines which split point is optimal.

(*) Differences between Greedy method and Dynamic Programming

Greedy Method:

① Greedy method is used for obtaining optimum solution.

② In Greedy method, from a set of feasible solutions, a solution which satisfies the constraints is optimal solution.

③ In Greedy method, Optimal solution is obtained without revising previously generated solutions.

④ In Greedy method, there is no guarantee of getting optimal sol.

Dynamic Programming:

① Dynamic programming is used for obtaining optimal solution.

② there is no special set of feasible solutions in this method.

③ Dynamic Programming considers all possible sequences in order to obtain the optimal solution.

④ It is guaranteed that the dynamic programming will generate optimal solution using principle of optimality.

⊛ Single Source Shortest Paths: General Weights:

Bellman-Ford algorithm is used to find the shortest path from one source vertex to all other vertices in a graph. Unlike, Dijkstra's algorithm Bellman-ford algorithm is capable of handling negative edges in the graph.

This algorithm initializes the distance to the source to 0 and all other nodes to ∞. then for all edges, if the distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value. At each iteration i that the edges are scanned, the algorithm finds all shortest paths of at most length i edges. Since the longest path without a cycle can be V-1 edges, the edges must be scanned V-1 times to ensure the shortest path has been found for all nodes.

Let $dist^{l}[u]$ be the length of a shortest path from source vertex v to u under the constraint that the shortest path contains at most l edges. Then $dist^{1}[u] = cost[v,u]$, $1 \le u \le n$. Hence $dist^{n-1}[u]$ is the length of an unrestricted shortest path from v to u. Our aim is to find $dist^{n-1}[u]$ for all u. This can be done using the following notations.

(i) If the shortest path from v to u with at most k, k>1, edges has no more than k-1 edges, then $dist^k[u] = dist^{k-1}[u]$.

(ii) If the shortest path from v to u with at most k, k>1, edges has exactly k edges, then it is made up of a shortest path from v to some vertex j following by the edge <j,u>. the path from v to j has k-1 edges, and its length is $dist^{k-1}[j]$. then $dist^{k-1}[j]$ is $dist^{k-1}[i] + cost[i,u]$

$$\therefore dist^k[u] = \min\{dist^{k-1}[u], \min_i\{dist^{k-1}[i] + cost[i,u]\}\}.$$

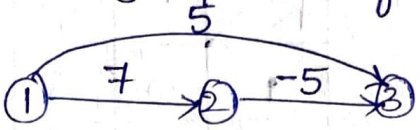
Algorithm: Algorithm BellmanFord (v, cost, dist, n)

```
{
    for i := 1 to n do
        dist [i] := cost [v, i];

    for k = 2 to n-1 do
        for each u such that u ≠ v and u has
            at least one incoming edge do
            for each <i, u> in the graph do
                if dist [u] > dist[i] + cost[i, u] then
                    dist [u] := dist[i] + cost[i, u];

}
```

Example ①: Find the shortest path from node 1 to every other node in the graph using Bellman-Ford algorithm.



Sol: Initially $dist^k[1] = 0 \ \forall \ k \geq 1$

i.e $dist^1[1] = dist^2[1] = 0$

K \ M	1	2	3
1	0	7	5
2	0	7	2

and $dist^1[2] = cost[1,2] = 7$

$dist^1[3] = cost[1,3] = 5$

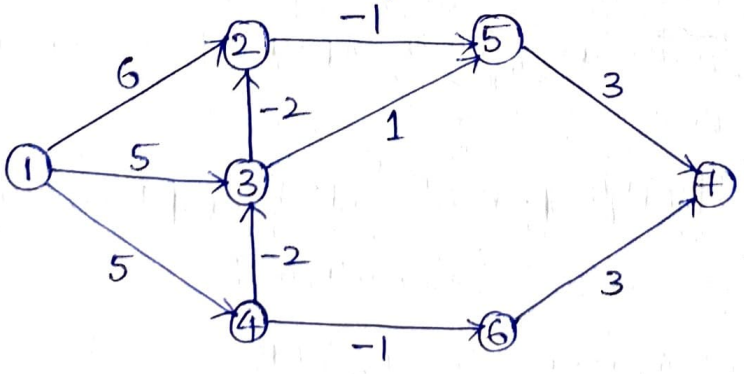
$$dist^2[2] = \min\left\{dist^1[2], \min_{i \neq 3}\{dist^1[i] + cost[i,2]\}\right\}$$

$$= \min\left\{7, \min\{\cancel{\infty+7}, 5+\infty\}\right\} = 7 //$$

$$dist^2[3] = \min\left\{dist^1[3], \min_{i \neq 2}\{dist^1[i] + cost[i,3]\}\right\}$$

$$= \min\left\{5, \min\{\cancel{\infty+\infty}, 7-5\}\right\} = 2 //$$

Example ②: Find the shortest path from 1 to every other node in the graph using Bellman-Ford algorithm



Sol: Initially $dist^k[1] = 0 \ \forall$ values of k

i.e $dist^1[1] = dist^2[1] = dist^3[1] = dist^4[1] = dist^5[1] = dist^6[1] = dist^7[1] = 0$.

For (k=1)

$dist^1[2] = cost(1,2) = 6$

$dist^1[3] = cost(1,3) = 5$

$dist^1[4] = cost(1,4) = 5$

$dist^1[5] = cost(1,5) = \infty$

$dist^1[6] = cost(1,6) = \infty$

$dist^1[7] = cost(1,7) = \infty$

K \ u	1	2	3	4	5	6	7
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

For (k=2)

$$dist^2[2] = \min\{dist^1[2], \min_{i=3,\ldots}\{dist^1[i] + cost[i,2]\}\}$$

$$= \min\{6, \min\{\ldots, 5-2, \ldots\}\}$$

$$= 3$$

$$dist^2[3] = \min\{dist^1[3], \min_{i=\ldots,4,\ldots}\{dist^1[i] + cost[i,3]\}\}$$

$$= \min\{5, \min\{\ldots 5-2, \ldots\}\}$$

$$= 3.$$

$$dist^2[4] = \min\{dist^1[4], \min_{i=\ldots}\{dist^1[i] + cost[i,4]\}\}$$

$$= \min\{5, \min\{\ldots\}\}$$

$$= 5$$

$$dist^2[5] = \min\{dist^1[5], \min_{i=2,3,\ldots}\{dist^1[i] + cost[i,5]\}\}$$

$$= \min\{\infty, \min\{6-1, 5+1, \ldots\}\}$$

$$= 5$$

$dist^2[6] = \min\{dist^1[6], \min\{dist^1[i] + cost[i,6]\}\}$

$i = 4, [redacted]$

$= \min\{\infty, \min\{[redacted], 5-1, [redacted]\}\}$

$= 4.$

$dist^2[7] = \min\{dist^1[7], \min\{dist^1[i] + cost[i,7]\}\}$

$i = [redacted] 5,6$

$= \min\{\infty, \min\{[redacted], \infty+3, \infty+3\}\}$

$= \infty.$

For (k=3)

$dist^3[2] = \min\{dist^2[2], \min\{dist^2[i] + cost[i,2]\}\}$

$i = 3, [redacted]$

$= \min\{3, \min\{3+(-2), [redacted]\}\}$

$= 1$

$dist^3[3] = \min\{dist^2[3], \min\{dist^2[i] + cost[i,3]\}\}$

$i = 2, 4[redacted]$

$= \min\{3, \min\{[redacted], 5+(-2), [redacted]\}\}$

$= 3.$

$dist^3[4] = \min\{dist^2[4], \min\{dist^2[i] + cost[i,4]\}\}$

$i = [redacted]$

$= \min\{5, \min\{[redacted]\}\}$

$= 5$

$dist^3[5] = \min\{dist^2[5], \min\{dist^2[i] + cost[i,5]\}\}$

$i = 2, 3, [redacted]$

$= \min\{5, \min\{3+(-1), 3+1, [redacted]\}\}$

$= 2$

$dist^3[6] = \min\{dist^2[6], \min\{dist^2[i] + cost[i,6]\}\}$

$i = [redacted], 4, [redacted]$

$= \min\{4, \min\{[redacted], 5+(-1), [redacted]\}\}$

$= 4$

$dist^3[7] = \min\{dist^2[7], \min\{dist^2[i] + cost[i,7]\}\}$

$i = [redacted], 5, 6$

$= \min\{\infty, \min\{[redacted] 5+3, 4+3\}\}$

$= 7$

For $k=4$

$dist^4[2] = \min\{dist^3[2], \min\{dist^3[i] + cost[i,2]\}\}$

$i = 3, [redacted]$

$= \min\{1, \min\{3+(-2), [redacted]\}\}$

$= 1$

$dist^4[3] = \min\{dist^3[3], \min\{dist^3[i] + cost[i,3]\}\}$

$i = [redacted] 4, [redacted]$

$= \min\{3, \min\{[redacted], 5+(-2), [redacted]\}\}$

$= 3$

$dist^4[4] = \min\{dist^3[4], \min\{dist^3[i] + cost[i,4]\}\}$

$i = [redacted]$

$= \min\{5, \min\{[redacted]\}\}$

$= 5$

$dist^4[5] = \min\{dist^3[5], \min\{dist^3[i] + cost[i,5]\}\}$

$i = 2, 3, [redacted]$

$= \min\{2, \min\{1+(-1), 3+1, [redacted]\}\}$

$= 0$

$dist^4[6] = \min\{dist^3[6], \min\{dist^3[i] + cost[i,6]\}\}$

$i = [redacted], 4, [redacted]$

$= \min\{4, \min\{[redacted], 5+(-1), [redacted]\}\}$

$= 4$

$dist^4[7] = \min\{dist^3[7], \min\{dist^3[i] + cost[i,7]\}\}$

$i = [redacted], 5, 6$

$= \min\{7, \min\{[redacted], \underline{2+3}, 4+3\}\}$

$= 5$

for (K=5)

$dist^5[2] = .\min\{dist^4[2], \min\{dist^4[i] + cost[i,2]\}\}$

$i = 3, [redacted]$

$= \min\{1, \min\{3+(-2), [redacted]\}\}$

$= 1$

$dist^5[3] = \min\{dist^4[3], \min\{dist^4[i] + cost[i,3]\}\}$

$i = [redacted]4[redacted]$

$= \min\{3, \min\{[redacted], 5+(-2), [redacted]\}\} = 3$

$dist^5[4] = \min\{dist^4[4], \min\{dist^4[i] + cost[i,4]\}\}$

$i = [redacted]$

$= \min\{5, \min\{[redacted]\}\} = 5$

$dist^5[5] = \min\{dist^4[5], \min\{dist^4[i] + cost[i,5]\}\}$

$i = 2, 3, [redacted]$

$= \min\{0, \min\{1+(-1), 3+1, [redacted]\}\} = 0$

$dist^5[6] = \min\{dist^4[6], \min\{dist^4[i] + cost[i,6]\}\}$

$i = [redacted], 4[redacted]$

$= \min\{4, \min\{[redacted], 5+(-1), [redacted]\}\} = 4$

$dist^5[7] = \min\{dist^4[7], \min\{dist^4[i] + cost[i,7]\}\}$

$i = [redacted], 5, 6$

$= \min\{5, \min\{[redacted], 0+3, 4+3\}\} = 3$

For (k=6)

$dist^6[2] = \min\{dist^5[2], \min\{dist^5[i] + cost[i,2]\}\}$

$i = 3, [redacted]$

$= \min\{1, \min\{3+(-2), [redacted]\}\} = 1$

$dist^6[3] = \min\{dist^5[3], \min\{dist^5[i] + cost[i,3]\}\}$

$i = [redacted], 4, [redacted]$

$= \min\{3, \min\{[redacted], 5+(-2), [redacted]\}\} = 3$

$dist^6[4] = \min\{dist^5[4], \min\{dist^5[i] + cost[i,4]\}\}$

$i = [redacted]$

$= \min\{5, \min\{[redacted]\}\} = 5$

$dist^6[5] = \min\{dist^5[5], \min\{dist^5[i] + cost[i,5]\}\}$

$i = 2, 3, [redacted]$

$= \min\{0, \min\{1+(-1), 3+1, [redacted]\}\} = 0$

$dist^6[6] = \min\{dist^5[6], \min\{dist^5[i] + cost[i,6]\}\}$

$i = [redacted], 4, [redacted]$

$= \min\{4, \min\{[redacted], 5+(-1), [redacted]\}\} = 4$

$dist^6[7] = \min\{dist^5[7], \min\{dist^5[i] + cost[i,7]\}\}$

$i = [redacted], 5, 6$

$= \min\{3, \min\{[redacted], 0+3, 4+3\}\} = 3$ //

③ 0/1 Knapsack Problem: Consider n no: of objects for $i=1,2\ldots n$ having their corresponding profits P_i and weights w_i. Consider a knapsack or bag with a capacity m. If a fraction x_i, $0 \le x_i \le 1$, of object i is placed into the knapsack, then a profit of $P_i x_i$ is earned. The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Since the knapsack capacity is m, total weight of all chosen objects to be at most m. formally, the problem can be states as

$$\text{maximize} \sum_{1 \le i \le n} P_i x_i \quad \text{subject to} \sum_{1 \le i \le n} w_i x_i \le m$$

$$\text{and} \quad 0 \le x_i \le 1, \quad 1 \le i \le n.$$

A solution to the knapsack can be obtained by making a sequence of decisions on the variables $x_1, x_2 \ldots x_n$. To solve this problem using dynamic programming use the following notations.

① Compute $S^1, S^2 \ldots S^i$. Initially $S^0 = \{(0,0)\}$

② Use the following formulae to compute S^i

$$S^i = S^{i-1} \cup S_1^{i-1}$$

Where $S_1^{i-1} = S^{i-1} + (P_i, w_i)$

③ After computing every S^i value apply purging rule (or) Dominance rule: If S^i contains (P_j, w_j) and (P_k, w_k) such that $P_j \le P_k$ and $w_j \ge w_k$ then eliminate the tuple (P_j, w_j) from S^i.

(*) Ex: Solve the following knapsack instance $m=6, n=3$, $(P_1, P_2, P_3) = (1, 2, 5)$ and $(w_1, w_2, w_3) = (2, 3, 4)$ using dynamic programming.

Sol: Given that $n=3, m=6$, $(P_1, w_1) = (1, 2)$, $(P_2, w_2) = (2, 3)$

$(P_3, w_3) = (5, 4)$.

Compute S^1, S^2, S^3 $\boxed{\because n=3}$

Initially $S^0 = \{(0,0)\}$

$S^i = S^{i-1} \cup S_1^{i-1}$ where $S_1^{i-1} = S^{i-1} + (P_i, w_i)$.

for $i=1$ $S^1 = S^0 \cup S_1^0$

$\therefore S_1^0 = S^0 + (P_1, w_1) = \{(0,0)\} + (1,2) = \{(1,2)\}$

$\Rightarrow S^1 = \{(0,0)\} \cup \{(1,2)\} = \{(0,0), (1,2)\}$

After applying purging rule S^1 is remains unchanged

for $i=2$ $S^2 = S^1 \cup S_1^1$

$\therefore S_1^1 = S^1 + (P_2, w_2) = \{(0,0), (1,2)\} + (2,3)$

$= \{(2,3), (3,5)\}$

$\Rightarrow S^2 = \{(0,0), (1,2)\} \cup \{(2,3), (3,5)\}$

$S^2 = \{(0,0), (1,2), (2,3), (3,5)\}$

After applying purging rule S^2 is uncharged.

for $i=3$ $\quad S^3 = S^2 \cup S_1^2$

$$\therefore S_1^2 = S^2 + (P_3, W_3)$$

$$= \{(0,0), (1,2), (2,3), (3,5)\} + (5,4)$$

$$= \{(5,4), (6,6), (7,7), (8,9)\}$$

$$\Rightarrow S^3 = \{(0,0), (1,2), (2,3), (3,5)\} \cup \{(5,4), (6,6), (7,7), (8,9)\}$$

$$= \{(0,0), (1,2), (2,3), \underbrace{(3,5)}_{P_j, W_j}, (5,4), \underbrace{(6,6)}_{P_k, W_j}, (7,7), (8,9)\}$$

By applying purging rule $\because P_j \leq P_k$ and $W_j \geq W_k$. the tuple $(3,5)$ is eliminated and since capacity of the knapsack is 6 eliminate the tuples having weight greater than 6. Hence

$$S^3 = \{(0,0), (1,2), (2,3), (5,4), (6,6)\}$$

Since $m=6$, search for the tuple whose weight is 6 or closer. Now the selected tuple (P_i, W_i) is $(6,6)$.

if $(P_i, W_i) \in S^i$ and $(P_i, W_i) \notin S^{i-1}$ then $x_i = 1$ else $x_i = 0$.

$(6,6) \in S^3$ and $(6,6) \notin S^2$ then $x_3 = 1$.

Now search for $(P_i - P_3, W_i - W_3) = (6-5, 6-4) = (1,2)$.

$(1,2) \in S^2$ and $(1,2) \notin S^1$ is false then $x_2 = 0$

$(1,2) \in S^1$ and $(1,2) \notin S^0$ is true then $x_1 = 1$

Now search for $(P_i - P_1, W_i - W_1) = (1-1, 2-2) = (0,0)$. Hence optimal solution is $(x_1, x_2, x_3) = (1, 0, 1)$ //