



Program : **B.Tech**

Subject Name: **Analysis and Design of Algorithm**

Subject Code: **CS-402**

Semester: **4th**



LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in

Concept of dynamic programming, problems based on this approach such as 0/1 knapsack, multistage graph, reliability design, Floyd Warshall algorithm

Unit-3

Concept of Dynamic Programming

Dynamic programming is a name, coined by Richard Bellman in 1955. Dynamic programming, as greedy method, is a powerful algorithm design technique that can be used when the solution to the problem may be viewed as the result of a sequence of decisions. In the greedy method we make irrevocable decisions one at a time, using a greedy criterion. However, in dynamic programming we examine the decision sequence to see whether an optimal decision sequence contains optimal decision subsequence.

When optimal decision sequences contain optimal decision subsequences, we can establish recurrence equations, called dynamic-programming recurrence equations, that enable us to solve the problem in an efficient way.

Dynamic programming is based on the principle of optimality (also coined by Bellman). The principle of optimality states that no matter whatever the initial state and initial decision are, the remaining decision sequence must constitute an optimal decision sequence with regard to the state resulting from the first decision. The principle implies that an optimal decision sequence is comprised of optimal decision subsequences. Since the principle of optimality may not hold for some formulations of some problems, it is necessary to verify that it does hold for the problem being solved. Dynamic programming cannot be applied when this principle does not hold.

The steps in a dynamic programming solution are:

- Verify that the principle of optimality holds
- Set up the dynamic-programming recurrence equations
- Solve the dynamic-programming recurrence equations for the value of the optimal solution.
- Perform a trace back step in which the solution itself is constructed.

0/1 – KNAPSACK

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays $val[0..n-1]$ and $wt[0..n-1]$ which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of $val[]$ such that sum of the weights of this subset is smaller than or equal to W . You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

Dynamic-0-1-knapsack (v, w, n, W)

for $w = 0$ to W do

$c[0, w] = 0$

for $i = 1$ to n do

$c[i, 0] = 0$

```

for w = 1 to W do
  if  $w_i \leq w$  then
    if  $v_i + c[i-1, w-w_i]$  then
       $c[i, w] = v_i + c[i-1, w-w_i]$ 
    else  $c[i, w] = c[i-1, w]$ 
  else
     $c[i, w] = c[i-1, w]$ 

```

MULTI STAGE GRAPHS

A multistage graph is a graph

- $G=(V,E)$ with V partitioned into $K \geq 2$ disjoint subsets such that if (a,b) is in E , then a is in V_i , and b is in V_{i+1} for some subsets in the partition;
- and $|V_1| = |V_K| = 1$.

The vertex s in V_1 is called the source; the vertex t in V_K is called the sink.

G is usually assumed to be a weighted graph.


The cost of a path from node v to node w is sum of the costs of edges in the path.

The "multistage graph problem" is to find the minimum cost path from s to t .

Dynamic Programming solution:

Let $\text{path}(i,j)$ be some specification of the minimal path from vertex j in set i to vertex t ; $C(i,j)$ is the cost of this path; $c(j,t)$ is the weight of the edge from j to t .

To write a simple algorithm, assign numbers to the vertices so those in stage V_i have lower number those in stage V_{i+1} .



```

int[] MStageForward(Graph G)
{
  // returns vector of vertices to follow through the graph
  // let  $c[i][j]$  be the cost matrix of  $G$ 

  int n = G.n (number of nodes);
  int k = G.k (number of stages);
  float[] C = new float[n];
  int[] D = new int[n];
  int[] P = new int[k];
  for (i = 1 to n)  $C[i] = 0.0$ ;
  for j = n-1 to 1 by -1 {
    r = vertex such that  $(j,r)$  in  $G.E$  and  $c(j,r)+C(r)$  is minimum
     $C[j] = c(j,r)+C(r)$ ;
     $D[j] = r$ ;
  }
   $P[1] = 1$ ;  $P[k] = n$ ;
  for j = 2 to k-1 {
     $P[j] = D[P[j-1]]$ ;
  }
  return P;
}

```

Time complexity:

Complexity is $O(|V| + |E|)$. Where the $|V|$ is the number of vertices and $|E|$ is the number of edges.

Example

Consider the following example to understand the concept of multistage graph.

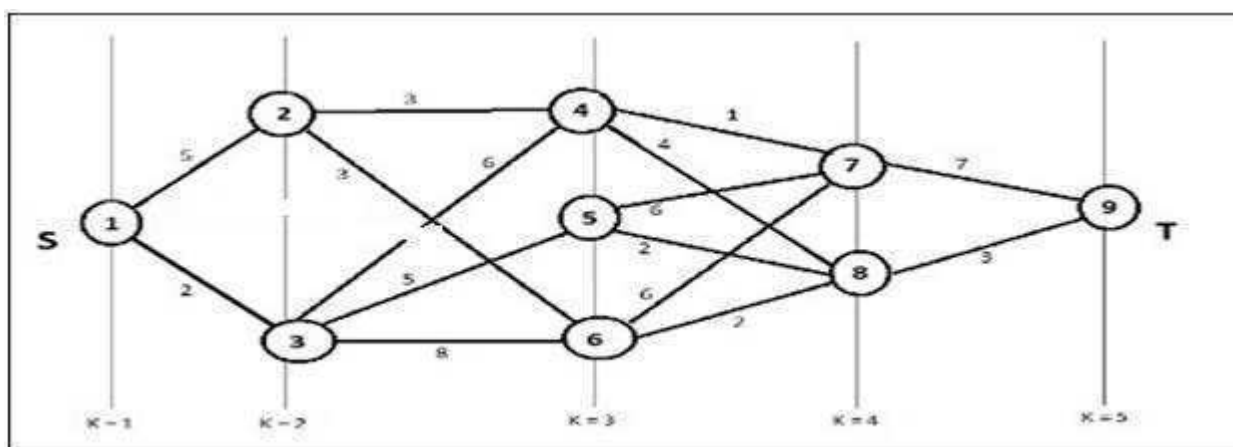


Figure 3.1 Multistage Graph

Step-1: Cost (K-2, j)

In this step, three nodes (node 4, 5, 6) are selected as j . Hence, we have three options to choose the minimum cost at this step.

$$\text{Cost}(3, 4) = \min \{c(4, 7) + \text{Cost}(7, 9), c(4, 8) + \text{Cost}(8, 9)\} = 7$$

$$\text{Cost}(3, 5) = \min \{c(5, 7) + \text{Cost}(7, 9), c(5, 8) + \text{Cost}(8, 9)\} = 5$$

$$\text{Cost}(3, 6) = \min \{c(6, 7) + \text{Cost}(7, 9), c(6, 8) + \text{Cost}(8, 9)\} = 5$$

Step-2: Cost (K-3, j)

Two nodes are selected as j because at stage $k - 3 = 2$ there are two nodes, 2 and 3. So, the value $i = 2$ and $j = 2$ and 3.

$$\text{Cost}(2, 2) = \min \{c(2, 4) + \text{Cost}(4, 8) + \text{Cost}(8, 9), c(2, 6) +$$

$$\text{Cost}(6, 8) + \text{Cost}(8, 9)\} = 8$$

$$\text{Cost}(2, 3) = \{c(3, 4) + \text{Cost}(4, 8) + \text{Cost}(8, 9), c(3, 5) + \text{Cost}(5, 8) + \text{Cost}(8, 9), c(3, 6) + \text{Cost}(6, 8) + \text{Cost}(8, 9)\} = 10$$

Step-3: Cost (K-4, j)

$$\text{Cost}(1, 1) = \{c(1, 2) + \text{Cost}(2, 6) + \text{Cost}(6, 8) + \text{Cost}(8, 9), c(1, 3) + \text{Cost}(3, 5) + \text{Cost}(5, 8) + \text{Cost}(8, 9)\} = 12$$

$$c(1, 3) + \text{Cost}(3, 6) + \text{Cost}(6, 8) + \text{Cost}(8, 9)\} = 13$$

Hence, the path having the minimum cost is $1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9$

RELIABILITY DESIGN

In **reliability design**, the problem is to design a system that is composed of several devices connected in series.

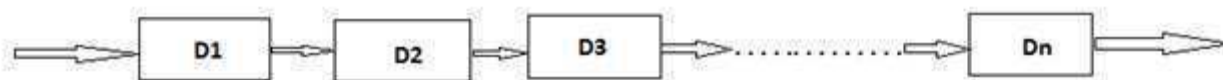


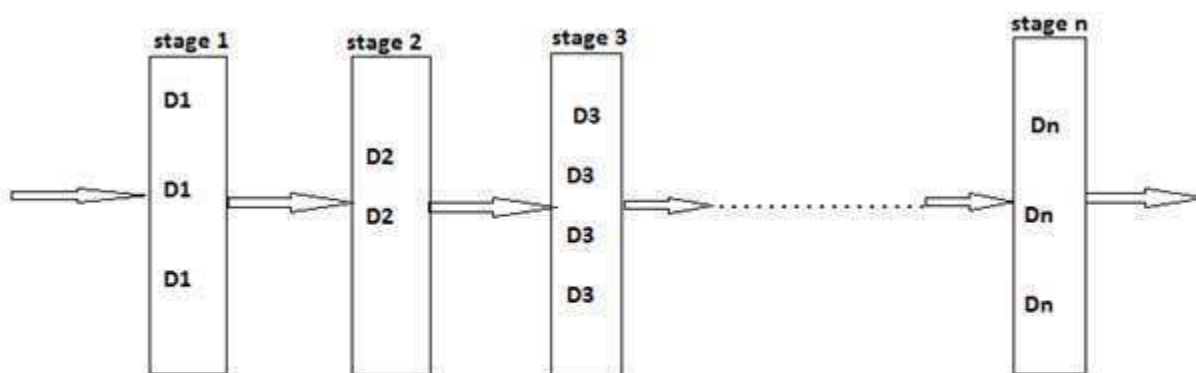
Figure 3.2 Reliability Design(Serial Connection)

If we imagine that r_1 is the reliability of the device. Then the reliability of the function can be given by $\prod r_i$. If $r_1 = 0.99$ and $n = 10$ that n devices are set in a series, $1 \leq i \leq 10$, then reliability of the whole system $\prod r_i$ can be given as: $\prod r_i = 0.904$

So, if we duplicate the devices at each stage then the reliability of the system can be increased.

It can be said that multiple copies of the same device type are connected in parallel through the use of switching circuits. Here, switching circuit determines which devices in any given group are functioning properly. Then they make use of such devices at each stage, that result is increase in reliability at each stage. If at each stage, there are m_i similar types of devices D_i , then the probability that all m_i have a malfunction is $(1 - r_i)^{m_i}$, which is very less.

And the reliability of the stage i becomes $(1 - (1 - r_i)^{m_i})$. Thus, if $r_i = 0.99$ and $m_i = 2$, then the stage reliability becomes 0.9999 which is almost equal to 1 . Which is much better than that of the previous case or we can say the reliability is little less than $1 - (1 - r_i)^{m_i}$ because of less reliability of switching circuits.



Multiple Devices Connected in Parallel in Each Stage

Figure 3.3 Reliability Design(Parallel Connection)

In reliability design, we try to use device duplication to maximize reliability. But this maximization should be considered along with the cost.

FLOYD WARSHALL ALGORITHM

The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

All pairs shortest paths

In the all pairs shortest path problem, we are to find a shortest path between every pair of vertices in a directed graph G . That is, for every pair of vertices (i, j) , we are to find a shortest path from i to j as well as one from j to i . These two paths are the same when G is undirected.

When no edge has a negative length, the all-pairs shortest path problem may be solved by using Dijkstra's greedy single source algorithm n times, once with each of the n vertices as the source vertex.

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 shortest Paths. Since each application of this procedure requires $O(n^2)$ time, the matrix A can be obtained in $O(n^3)$ time.

Algorithm All Paths (Cost, A , n)

// cost $[1:n, 1:n]$ is the cost adjacency matrix of a graph which

// n vertices; $A[i, j]$ is the cost of a shortest path from vertex

// i to vertex j . cost $[i, i] = 0.0$, for $1 < i < n$.

{

for $i := 1$ to n do

for $j := 1$ to n do

$A[i, j] := \text{cost}[i, j]$; // copy cost into A . for $k := 1$ to n do

for $i := 1$ to n do

for $j := 1$ to n do

$A[i, j] := \min(A[i, j], A[i, k] + A[k, j]);$

}

Complexity Analysis:

A Dynamic programming algorithm based on this recurrence involves in calculating $n+1$ matrices, each of size $n \times n$. Therefore, the algorithm has a complexity of $O(n^3)$.

Problem-

Consider the following directed weighted graph-

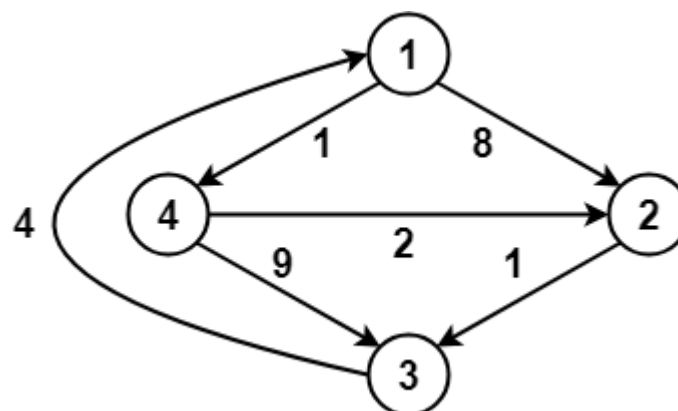


Figure 3.4 Floyd Warshall Example

Solution-

Step-01:

- Remove all the self loops and parallel edges (keeping the edge with lowest weight) from the graph if any.

- In our case, we don't have any self edge and parallel edge.

Step-02:

Now, write the initial distance matrix representing the distance between every pair of vertices as mentioned in the given graph in the form of weights.

- For diagonal elements (representing self-loops), value = 0
- For vertices having a direct edge between them, value = weight of that edge
- For vertices having no direct edges between them, value = ∞

$$D_0 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{matrix}$$

Step-03:

The four matrices are-

$$\begin{matrix} D_1 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{matrix} & D_2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{matrix} \\ D_3 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{matrix} & D_4 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{matrix} \end{matrix}$$

The last matrix D4 represents the shortest path distance between every pair of vertices.



RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at
<https://qp.rgpvnotes.in> .

If you have any queries or you want to submit your
study notes please write us at
rgpvnotes.in@gmail.com



LIKE & FOLLOW US ON FACEBOOK

facebook.com/rgpvnotes.in