Dynamic Programming

OBJECTIVES

After studying this chapter, the student will be able to

- Understand the concept of dynamic programming
- Recognize the difference between dynamic programming and divide and conquer approaches
- Apply dynamic programming to solve Multistage Graph problem

1- INTRODUCTION

As stated in the earlier chapters, the greedy approach does not give us a correct solution in many cases. The divide and conquer approach can only be applied if the sub-problems are symmetric and independent. For other problems, the above approaches would not work. Dynamic programming helps us to find the optimal solution of many such problems.

There is another reason to use the dynamic approach. It was stated earlier that for every iterative procedure, we can have a corresponding recursive procedure. For example, the recursive procedure for calculating the nth Fibonacci term (given below), though easy to understand, results in calculating a particular value many times.

```
fib(int n)
        {
        if (n==1)
                return 1;
        else if (n==2)
                return 1;
        else
                return (fib (n-1) + fib (n-2));
        }
```

The calculation of 7^{th} term, for instance, is done as shown in Fig. 1. From the figure, it can be seen that the third term, fib(3), is calculated 5 times. The approach is a top-down approach and is not capable of using values calculated earlier, so ends up evaluating the same value many times. As a matter of fact, the calculation of n^{th} term would require $O(2^n)$ calculations, most of which would have been calculated earlier.

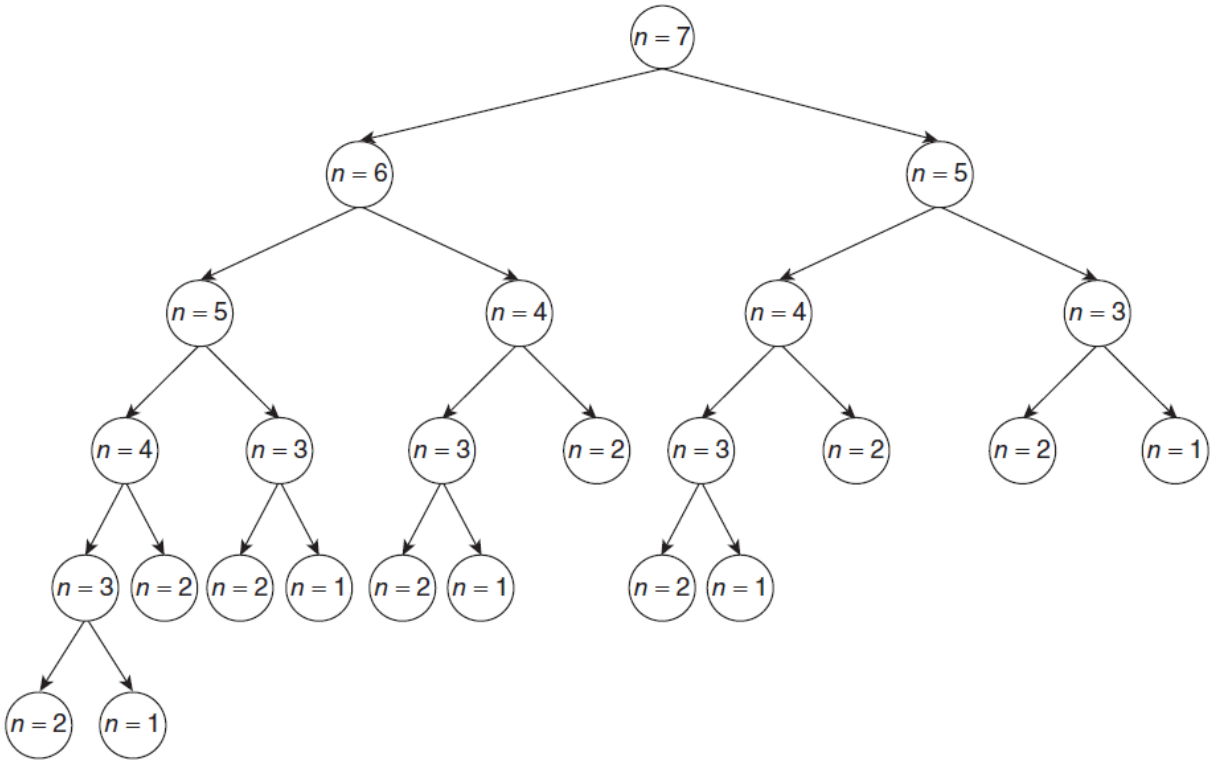


Figure 1: Calculating the n^{th} Fibonacci term

The above recursive method can be made efficient by storing the earlier calculated values in a table. The following non-recursive procedure calculates the n^{th} Fibonacci term in a bottom-up fashion.

```
fib (n)// non-recursive procedure to calculate n Fibonacci terms
{
//a[] is the global array
a[1] = 1;
a [2] = 1;
for ( i=3; i<n ; i++)
            {
                    a[i] = a[i-1] + a[i-2];
            }
}
```

This method of storing the values in a global array would make the calculation of the subsequent terms easy, as the values stored in the table would be used. Note that the complexity of the above algorithm is just O(n) as against $O(2^n)$ of the previous algorithm.

This approach of first designing a recursive procedure and then converting it into a form which makes the calculations easy is the gist of dynamic programming. This chapter explores the idea of dynamic programming and applies it to solve one of the most important problems, which is finding the minimum path in a Multistage Graph.

2- CONCEPT OF DYNAMIC PROGRAMMING

Dynamic programming uses the results obtained in the previous steps to get the final answer. The idea is to use the sub-solutions obtained in solving larger sub-problems. This programming is different from the procedural or the object-oriented programming techniques. Though it is a difficult approach, the problems dealt within this chapter will enable us to understand it.

Dynamic programming might appear similar to the divide and conquer approach, but there is a fundamental difference in the two approaches. Both the divide and conquer approach and dynamic programming require division of the problems; in the former, the sub-problems are independent, whereas in the latter, they are not

(Fig. 2). Moreover, the divide and conquer evaluates generally in a top-down fashion. The dynamic programming generally uses the bottom-up approach. These two approaches have been discussed as follows.

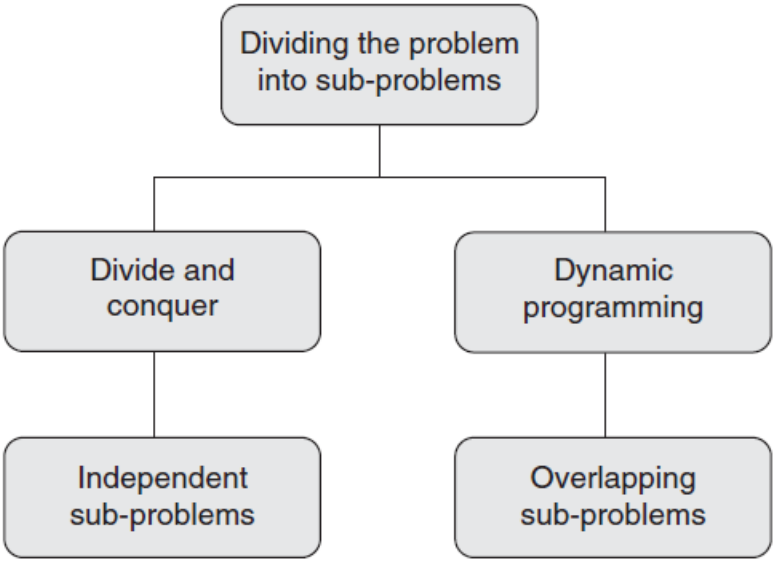


Figure 2: Dynamic versus divide and conquer

The use of the calculated values at a later stage is facilitated by storing the calculated values in a table. The paradigm, therefore, requires memorization as well. The approach can work in two ways: bottom-up and top-down. In the bottom-up approach, the solution starts with the basic input value and builds up the larger solution; the top-down approach, on the other hand, breaks the bigger problem into smaller sub-problems and then solves them (Fig. 3).

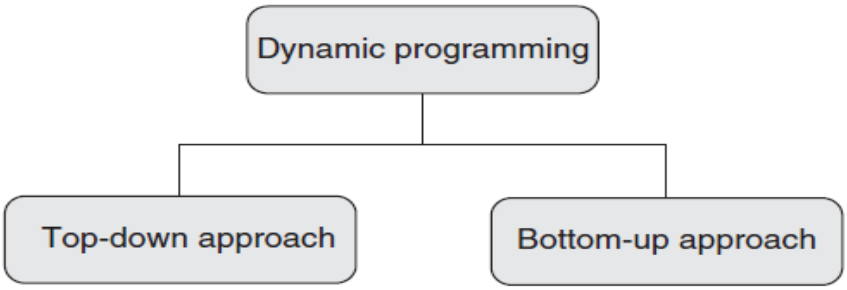


Figure 3: Classification of dynamic programming

3- Implementing the Dynamic Approach

The development of a dynamic algorithm for a problem requires the following steps, which are depicted in Fig. 4:

- Conversion of the given problem into a search problem. The search would be from a large search space.
- The search space is generally segregated into various sub-spaces. This requires the development of a partition algorithm. This can be accomplished by developing a recursive algorithm.
- In the final step, the recursive calls are characterized and a non-recursive procedure is developed for filling the table. The values of the table are filled in a way so that the calculations can be used at a later stage.

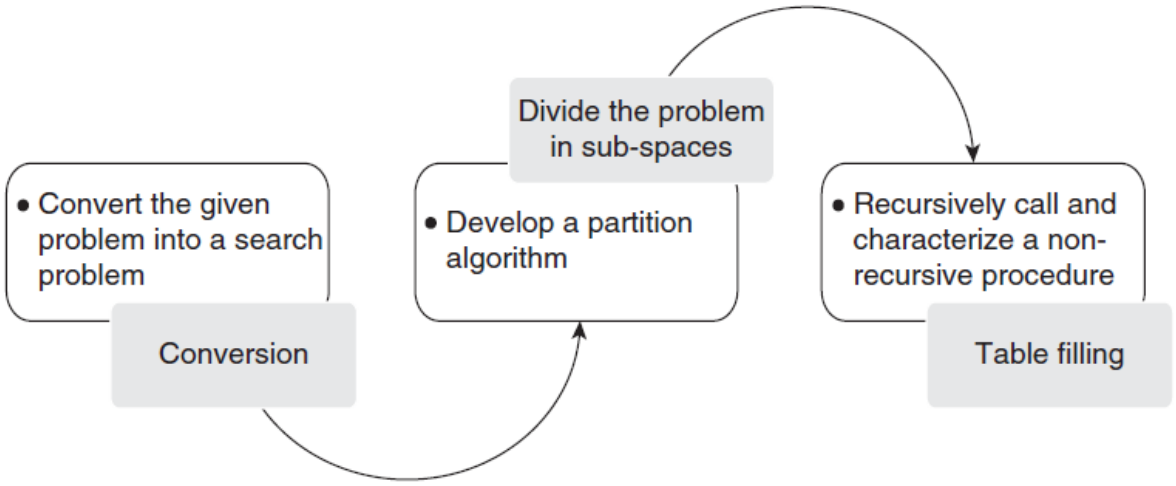


Figure 4: Implementing dynamic algorithms

4- MULTISTAGE GRAPH

A multistage graph G = (V,E) is a weighted directed graph. Let 's' and 't' nodes be the source and destination respectively. The cost of a path from source (s) to destination (t) is the sum of the costs of the edges on the path. The MULTISTAGE GRAPH problem is to find a minimum cost path from 's' to 't'.

The MULISTAGE GRAPH problem can be solved in two ways:

a) Forward Method. b) Backward Method.

- Note that if the recurrence relations are formulated using the ***forward approach*** then the relations are solved ***backwards***, i.e., beginning with the last decision
- On the other hand, if the relations are formulated using the ***backward approach***, they are solved ***forwards***.
- To sum up, In forward approach we will find the path from destination to source, in backward approach we will find the path from source to destination.

a) FORWARD METHOD

1. Assume that there are 'k' stages in a graph.
2. In the *FORWARD* approach, we will find out the cost of each and every node starling from the k^{th} stage to the 1^{st} stage.
3. We will find out the minimum cost path from source to the destination (Stage-1 to Stage-k).

PROCEDURE:

- Maintain a cost matrix cost (n) which stores the distance from any vertex to the destination.
- If a vertex is having more than one path, then we have to choose the minimum distance path and the intermediate vertex, which gives the minimum distance path, will be stored in the distance array 'D'.

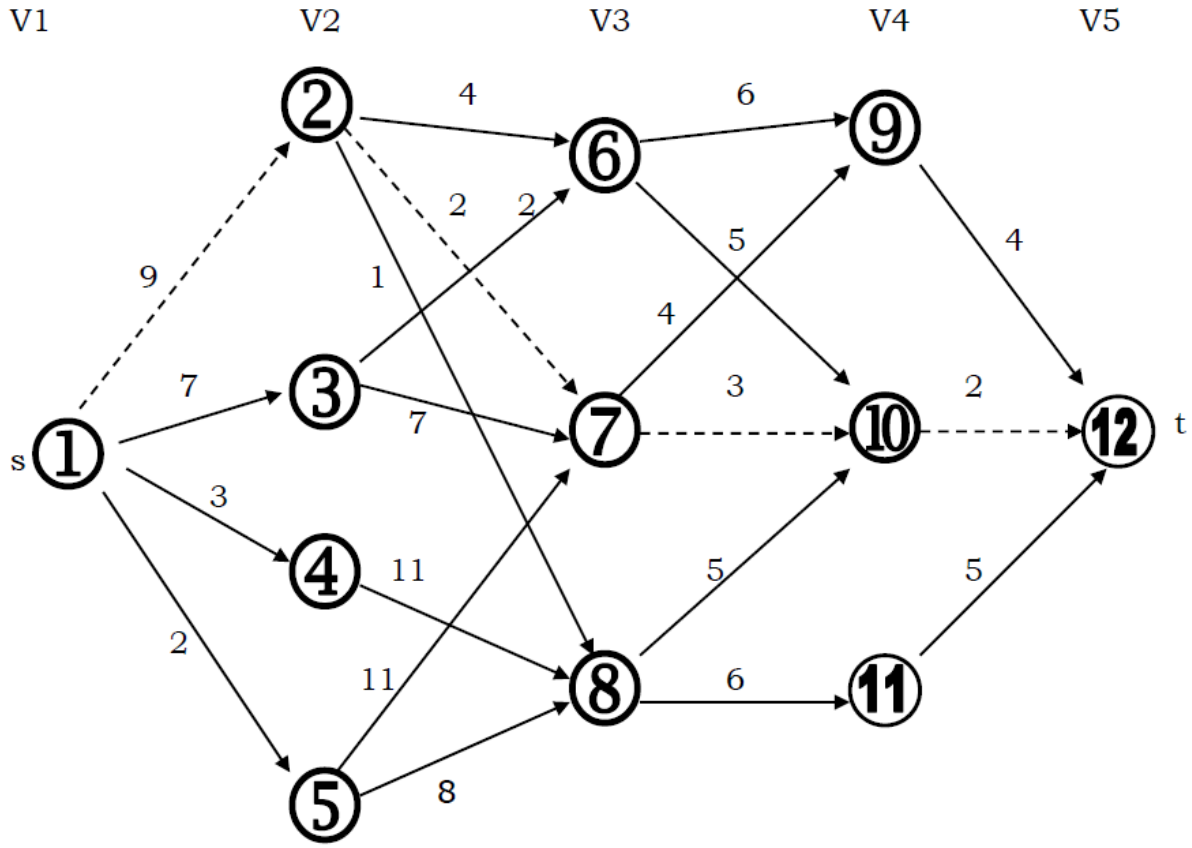
- In this way we will find out the minimum cost path from each and every vertex.

```
1   Algorithm FGraph(G, k, n, p)
2   // The input is a k-stage graph G = (V, E) with n vertices
3   // indexed in order of stages. E is a set of edges and c[i, j]
4   // is the cost of ⟨i, j⟩. p[1 : k] is a minimum-cost path.
5   {
6       cost[n] := 0.0;
7       for j := n − 1 to 1 step −1 do
8       { // Compute cost[j].
9           Let r be a vertex such that ⟨j, r⟩ is an edge
10          of G and c[j, r] + cost[r] is minimum;
11          cost[j] := c[j, r] + cost[r];
12          d[j] := r;
13      }
14      // Find a minimum-cost path.
15      p[1] := 1; p[k] := n;
16      for j := 2 to k − 1 do p[j] := d[p[j − 1]];
17  }
```

complexity:

The time complexity of this forward method is $O(|V|+|E|)$

Example: Find the minimum cost path from 's' to 't' using the forward method of dynamic programming.



solution:

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost									4	2	5	0
D									12	12	12	12

❖ For forward approach,

$$\text{Cost}(i,j) = \min_{\substack{l \in V_{i+1} \\ (j,l) \in E}} \{C(j,l) + \text{Cost}(i+1,l)\}$$

Cost(8) = min {C (8,10) + Cost (10), C (8,11) + Cost (11) }
= min (5 + 2, 6 + 5)
= min (7,11)
= 7
cost(8) =7 =>D(8)=10
cost(7) = min(c (7,9)+ cost(9),c (7,10)+ cost(10))
(4+4,3+2)
= min(8,5)
= 5
cost(7) = 5 =>D(7) = 10
cost(6) = min (c (6,9) + cost(9),c (6,10) +cost(10))
= min(6+4 , 5 +2)
= min(10,7)
= 7
cost(6) = 7 =>D(6) = 10

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost						7	5	7	4	2	5	0
D						10	10	10	12	12	12	12

cost(5) = min (c (5,7) + cost(7),c (5,8) +cost(8))

= min(11+5 , 8 +7)
= min(16,15)
= 15
cost(5) = 15 =>D(5) = 8

cost(4) = min (c (4,8) + cost(8))
= min(11+7)
= 18
cost(4) = 18 =>D(4) = 8
cost(3) = min (c (3,6) + cost(6),c (3,7) +cost(7))
= min(2+7 , 7 +5)
= min(9,12)
= 9

cost(3) = 9 =>D(3) = 6
cost(2) = min (c (2,6) + cost(6),c (2,7) +cost(7) ,c (2,8) +cost(8))
= min(4+7 , 2+5 , 1+7)
= min(11,7,8)
= 7
cost(2) = 7 =>D(2) = 7

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost		7	9	18	15	7	5	7	4	2	5	0
D		7	6	8	8	10	10	10	12	12	12	12

cost(1) = min (c (1,2)+cost(2) ,c (1,3)+cost(3) ,c (1,4)+cost(4) ,c(1,5)+cost(5))
= min(9+7 , 7 +9 , 3+18 , 2+15)
= min(16,16,21,17)
= 16
cost(1) = 16 =>D(1) = 2

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	5	0
D	2	7	6	8	8	10	10	10	12	12	12	12

The path through which you have to find the shortest distance.

Start from vertex - 2
D (1) = 2
D (2) = 7
D (7) = 10
D (10) = 12
So, the minimum –cost path is,

The cost is 9+2+3+2+=16

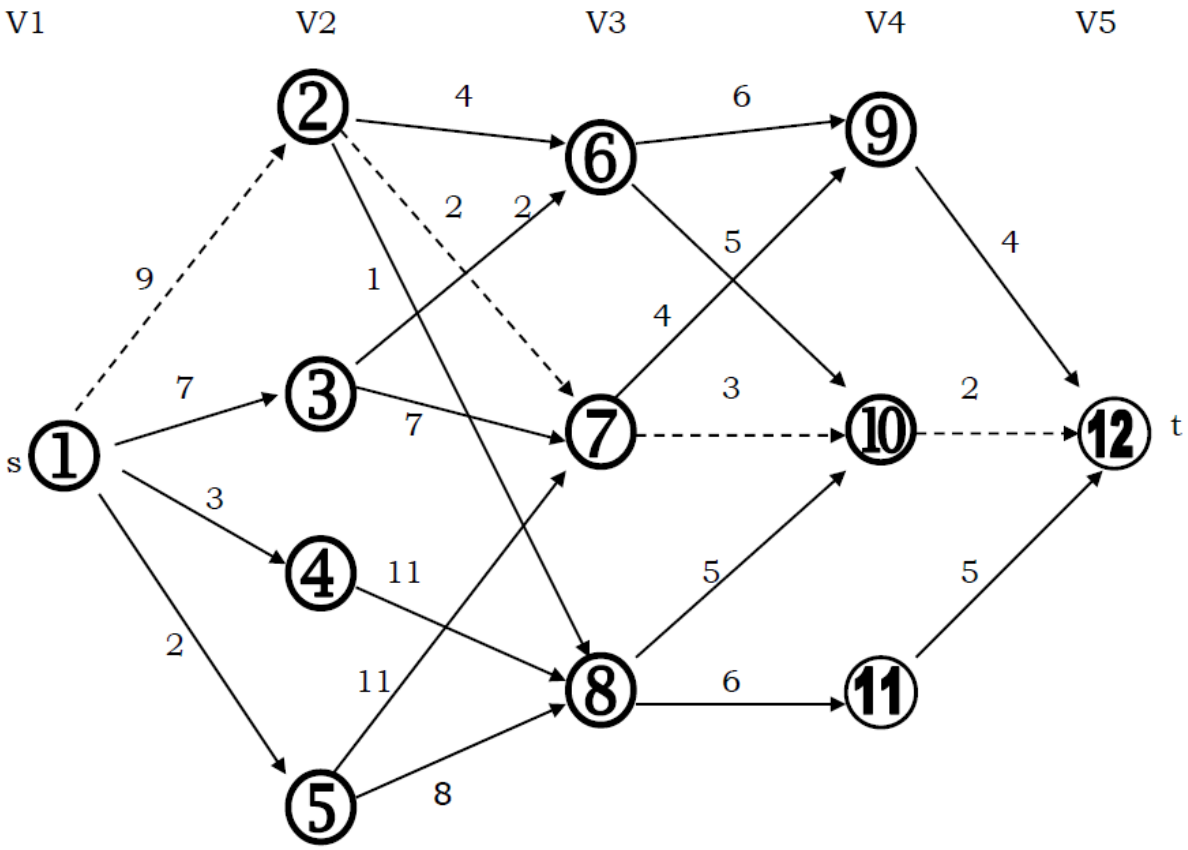
b) BACKWARD METHOD

The multistage graph problem can be solved using backward approach.

- Algorithm Bgraph(G,k,n,p)

```
//same function as Fgraph
{
bcost[1]:=0.0;
For j:=2 to n do
{ // compute bcost[j].
Let r be such that <r,j> is an edge of G and bcost[r] + c[r,j] is
   mimimum;
bcost[j]:=bcost[r]+c[r,j];
d[j]:=r;
}
//Find a minimum-cost path
P[1]:=1;p[k]:=n;
For j:=k-1 to 2 do p[j]:= d[p[j+1]];
}
```

Example: Find the minimum cost path from 's' to 't' using the backward method of dynamic programming.



Solution:

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	0	9	7	3	2							
D	0	1	1	1	1							

The backward apporach to find min. cost is

$$bcost(i,j) = \min_{\substack{l \in V_{i+1} \\ <j,l> \in E}} \{ bcost(i-1, l) + c(l,j) \}$$

Cost(6) =min(c (2,6) + cost(2),c (3,6) + cost(3))
=min(13,9)
cost(6) = 9 =>D(6)=3
Cost(7) =min(c (3,7) + cost(3),c (5,7) + cost(5) ,c (2,7) + cost(2))
=min(14,13,11)
cost(7) = 11 =>D(7)=2
Cost(8) =min(c (2,8) + cost(2),c (4,8) + cost(4) ,c (5,8) +cost(5))
=min(10,14,10)
cost(8) = 10 =>D(8)=2

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	0	9	7	3	2	9	11	10				
D	0	1	1	1	1	3	2	2				

Cost(9) =min(c (6,9) + cost(6),c (7,9) + cost(7))
=min(15,15)
cost(9) = 15 =>D(9)=6
Cost(10)=min(c(6,10)+cost(6),c(7,10)+cost(7)),c (8,10)+cost(8)) =min(14,14,15)
cost(10)= 14 =>D(10)=6
Cost(11) =min(c (8,11) + cost(8))
cost(11) = 16 =>D(11)=8

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	0	9	7	3	2	9	11	10	15	14	16	
D	0	1	1	1	1	3	2	2	6	6	8	

cost(12)=min(c(9,12)+cost(9),c(10,12)+cost(10),c(11,12)+cost(11))
=min(19,16,21)
cost(12) = 16 =>D(12)=10

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	0	9	7	3	2	9	11	10	15	14	16	16
D	0	1	1	1	1	3	2	2	6	6	8	10

Start from vertex-12
D(12) = 10
D(10) = 6
D(6) = 3
D(3) = 1
So the minimum cost path is,

$$1 \xrightarrow{7} 3 \xrightarrow{2} 6 \xrightarrow{5} 10 \xrightarrow{2} 12$$

The cost is 16.
H.W: Find the minimum cost path from S to T using the forward and backward methods of dynamic programming.