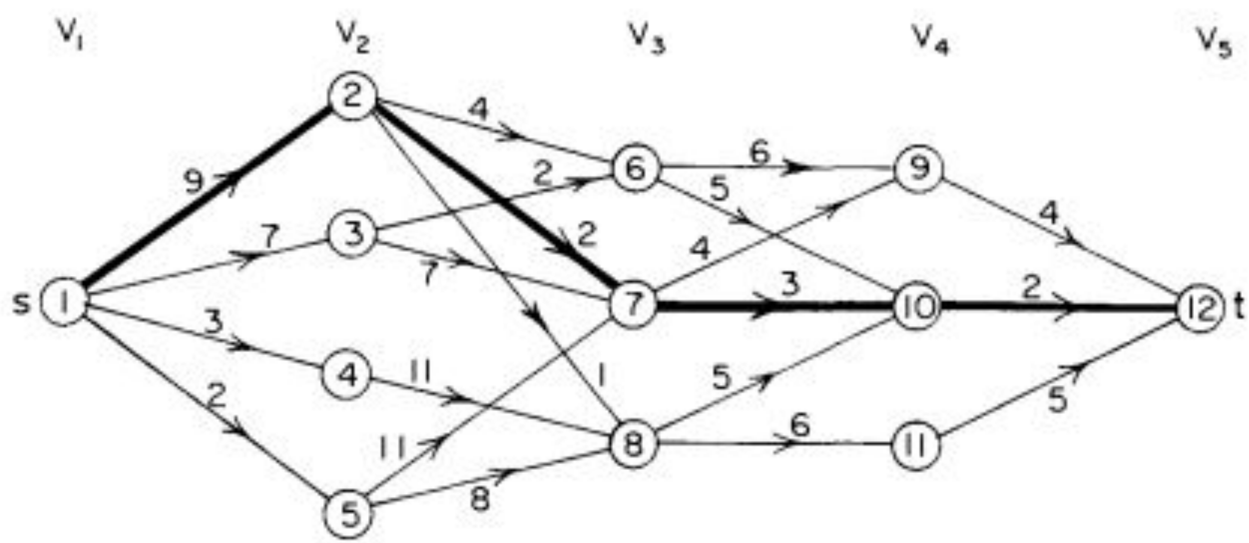
DYNAMIC PROGRAMMING
MULTISTAGE GRAPHS

MULTISTAGE GRAPH

- A **Multistage graph** is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only
- If there is an edge (u,v), then $u \in V_i$ and $v \in V_{i+1}$



v1,v2,v3,v4,v5 are stages

MULTISTAGE GRAPH

- A multistage graph $G=(V,E)$ is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets V_i, $1 \leq i \leq k$ (K stage graph)
- $|V_1| = |V_k| = 1$ (Only one vertex in first and last stage)
- The vertex V_1 is source (s) and V_k is the sink (t)
- Cost of a path is the sum of cost of the edges on the path
- Goal : to find a minimum-cost path from s to t
- Resource allocation problems are represented as multi stage graphs

Dynamic Programming Approach

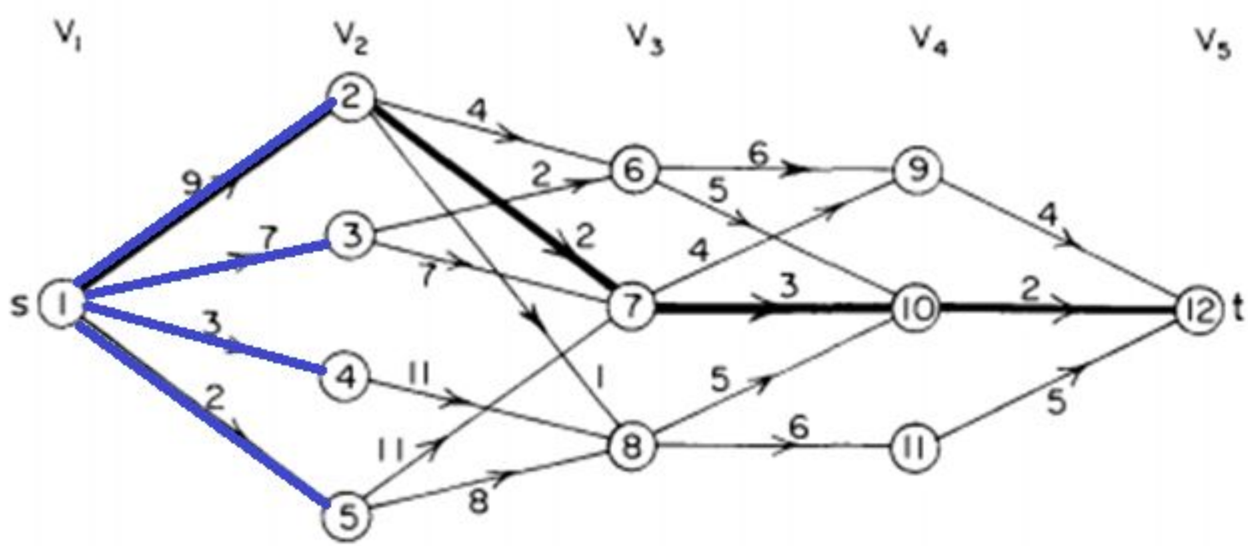
- A dynamic programming formulation for a k-stage graph problem is obtained by first noticing that every s to t path is the result of a sequence of k-2 decisions.
- The i^{th} decision involves determining which vertex in V_{i+1}, $1 \le i \le k-2$ is on the path.
- Cost(i, j) = cost (i, l) +cost(l,j)
- For cost(i, j) to be minimum, cost(l, j) should be minimum
- Hence Principal of optimality holds
- Multi stage graph problem can be solved using either Forward Method or Backward Method

FORWARD METHOD

- Let P(i,j) be a minimum-cost path from vertex j in V_i to vertex t
- Let cost(i,j) be the cost of this path
- Using forward approach to find cost of the path .

$$\textbf{cost(i,j)} = \min_{\substack{l \in V_{i+1} \\ \langle j,l \rangle \in E}} \{ c(j,l) + cost(i+1,l) \}$$

Cost(i, j) is the cost of the shortest path from vertex j (in stage V_i) to t



1. $Cost(k-1,j) = c(j,t)$ if $(j,t) \in E$

   $= \infty$ otherwise

2. Find $cost(k-2, j)$ for all $j \in V_{k-2}$

3. Find $cost(k-3, j)$ for all $j \in V_{k-3}$ and so on

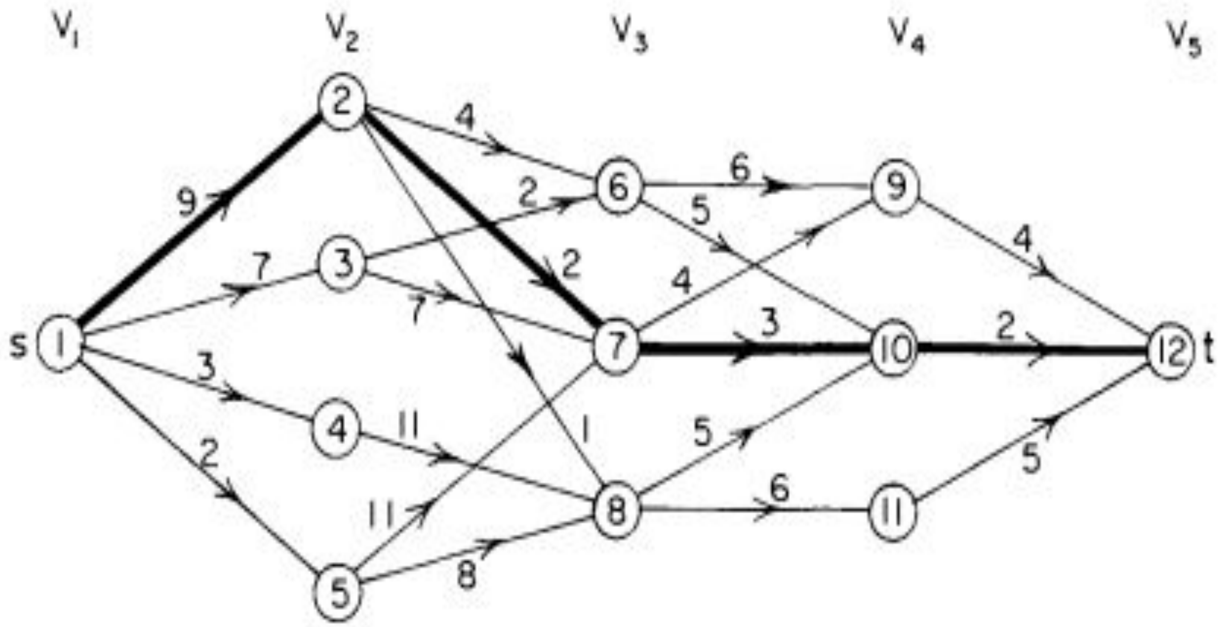
4. Finally find $cost(1,s)$

Note: At every step, record the value of j that gave the minimum.

This is required to determine the path from s to t

EXAMPLE : Find the minimum cost path using

FORWARD Method

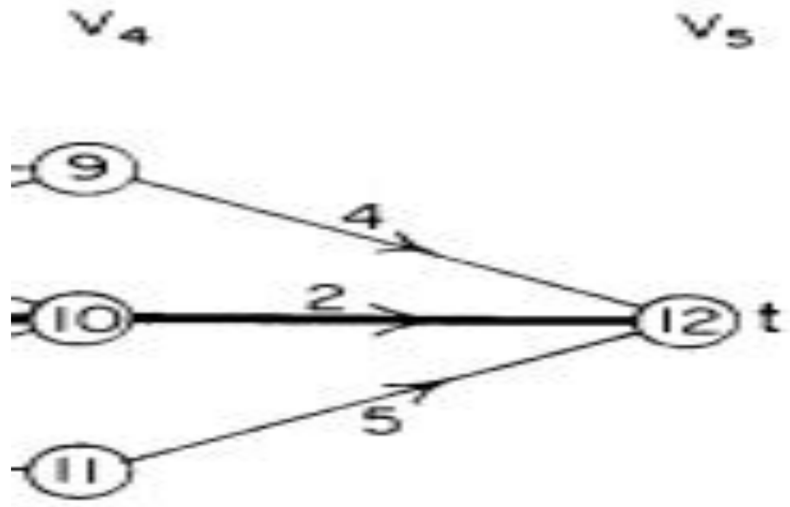


Vertices in V_4 = { 9, 10, 11}

Cost(4,9) = 4

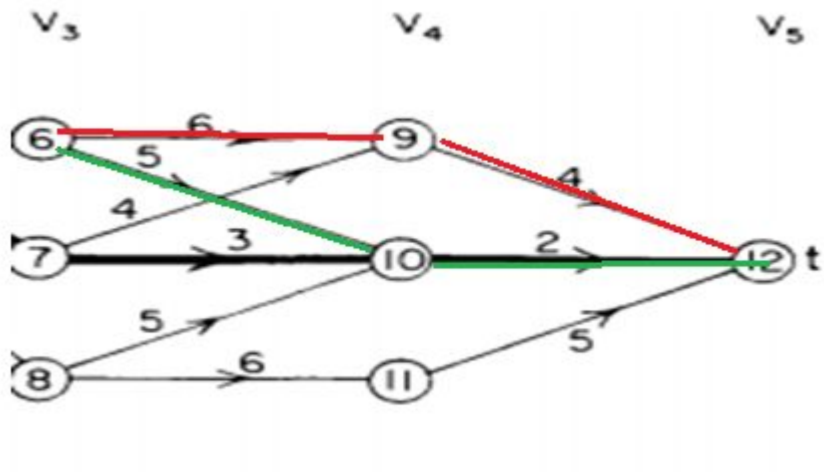
Cost(4,10) = 2

Cost(4,11) = 5



STEP 2:

Vertices in V3 {6, 7, 8}



Cost(3, 6) = min{ 6 + Cost(4, 9), 5 + Cost(4, 10)} = 7

Min l=10

Cost(3, 7) = min{ 4 + Cost(4, 9), 3 + Cost(4, 10)} = 5

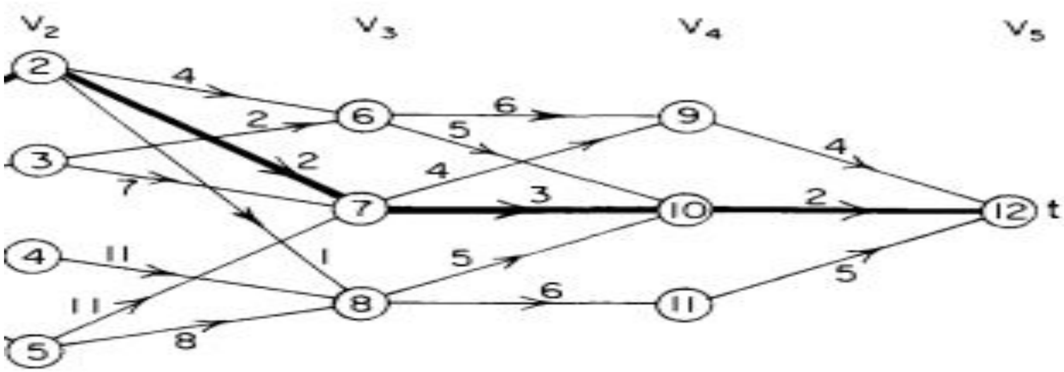
Min l=10

Cost(3, 8) = min(5+Cost(4,10), 6+Cost(5,11) = 7

Min l=11

STEP 3:

Vertices in V_2 {2,3,4,5}



Cost(2, 2) = min{ 4 + Cost (3, 6), 2 + Cost (3, 7), 1 + Cost (3, 8)} =7 *min l=7*

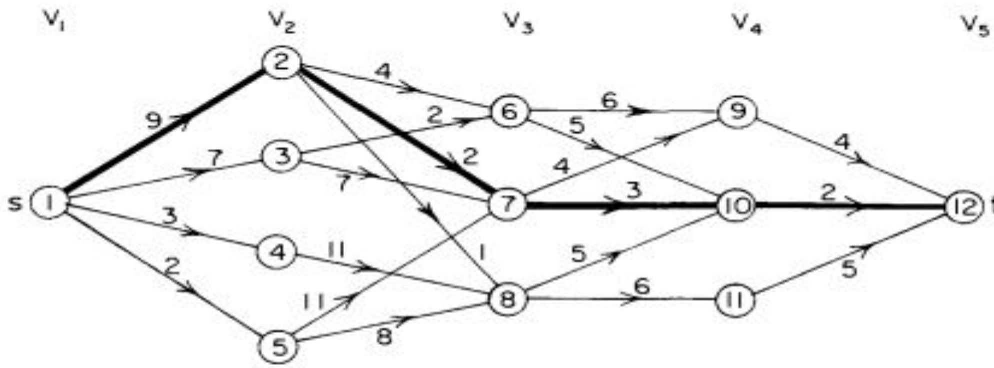
Cost(2, 3) =min{ 2 + Cost (3, 6), 7 + Cost (3, 7)}= 9 *min l=6*

Cost(2, 4) = 11 + Cost (3, 8)=18 min *l=8*

Cost(2, 5) = min{ 11 + Cost (3,7), 8 + Cost (3, 8)}=15 *min l=8*

STEP 4:

Vertex in V1 is 1



COST(l, 1) = min{ 9 + COST(2, 2), 7 + COST(2, 3), 3 + COST(2, 4), 2 + COST(2, 5)} = 16

The minimum cost is 16

min l = 2

Hence, the shortest path is **1⟶ 2 ⟶ 7 ⟶ 10 ⟶ 12**

Algorithm for FORWARD APPROACH:

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

Algorithm Multistage graph pseudocode corresponding to the forward approach

The time complexity of this forward method is **O(n+ E)**

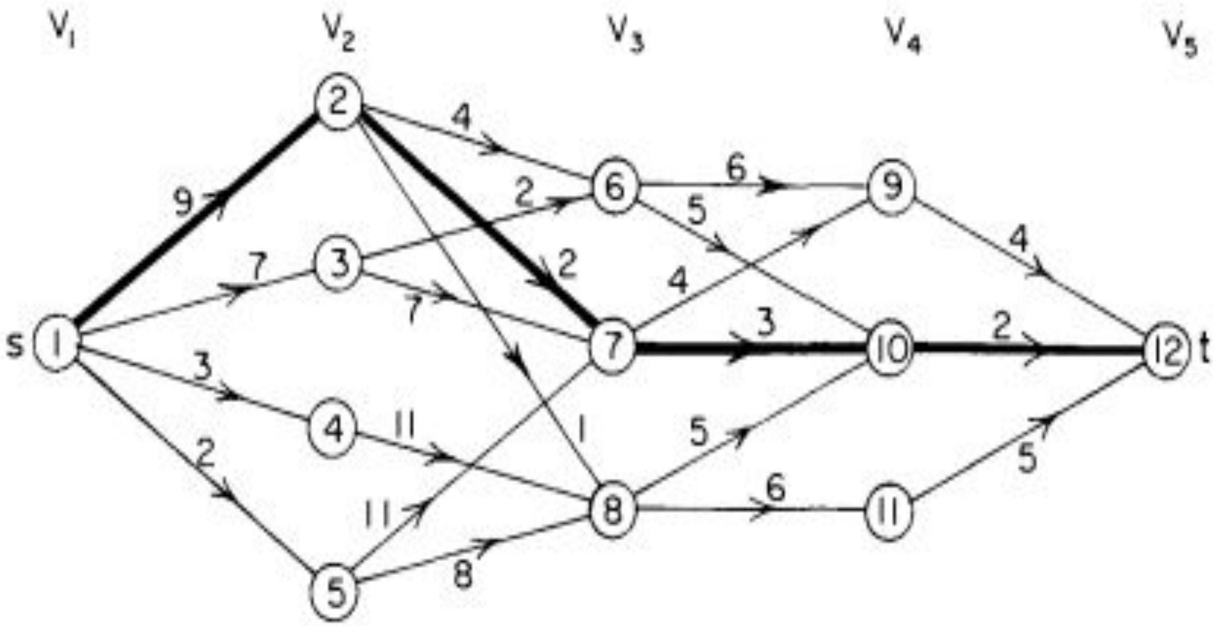
BACKWARD APPROACH

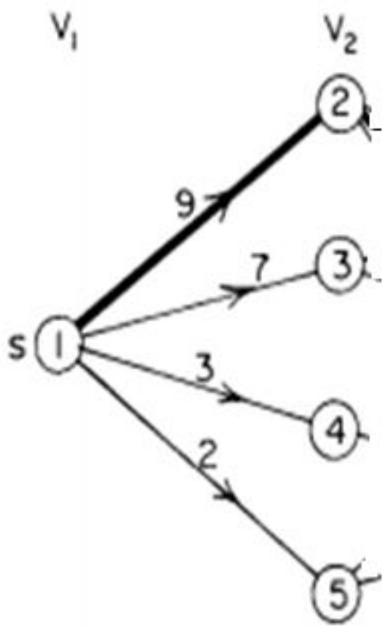
- In Backward Approach, the minimum cost path found from destination to source (i.e)[from stage k to stage 1].
- Let bp(i,j) be a minimum-cost path from vertex s to vertex j in Vi .
- Let bcost(i,j) be cost of bp(i,j).
- The backward approach to find minimum cost is

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

- Since bcost(2,j) = c(1,j) if E and bcost(2,j) =∞, if <i,j> does not belong to E
- bcost(i,j) can be computed using above formula

EXAMPLE : Find the minimum cost path using Backward Method



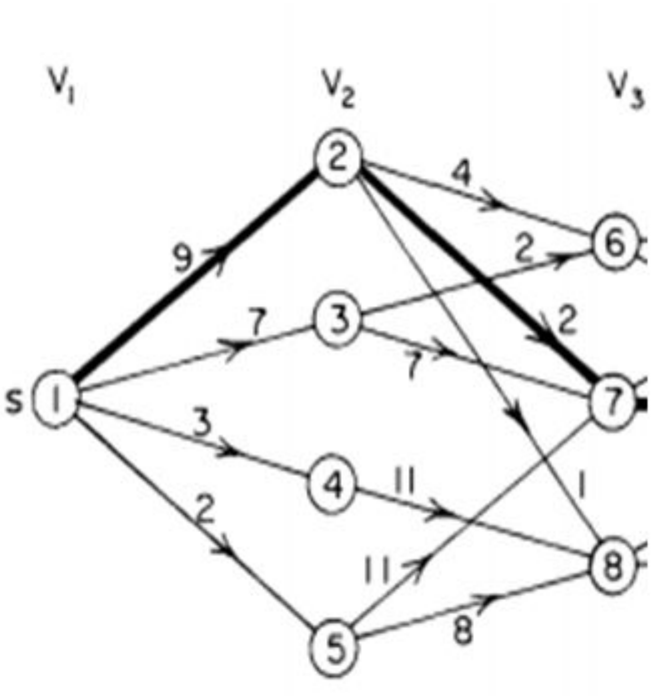


Bcost(2,2)=9

Bcost(2,3)=7

Bcost(2,4)=3

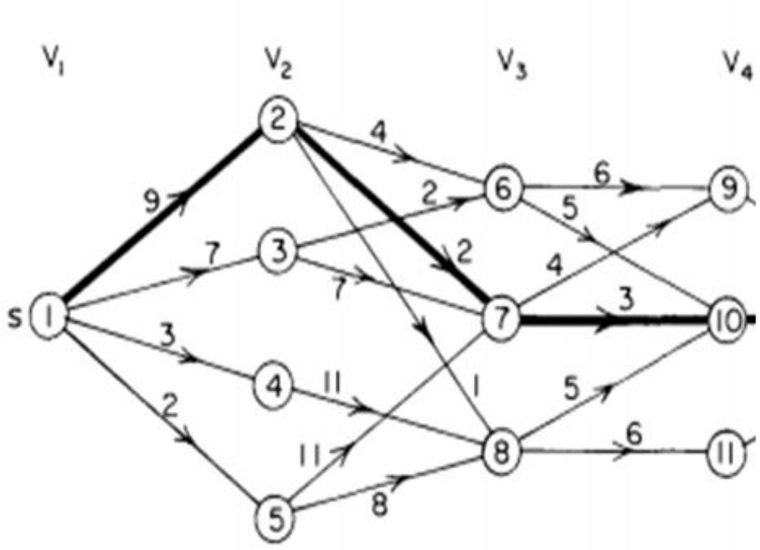
Bcost(2,5)=2



Bcost(3,6)=min(4+bcost(2,2),
2+bcost(2,3)) =9
min l = 3

Bcost(3,7)=min(2+bcost(2,2),
7+bcost(2,3),
11+bcost(2,5))= 11
min l = 2

Bcost(3,8)=min(1+bcost(2,2),
11+bcost(2,4),
8+bcost(2,5))= 10
min l = 2



Bcost(4,9)=min(6+bcost(3,6),

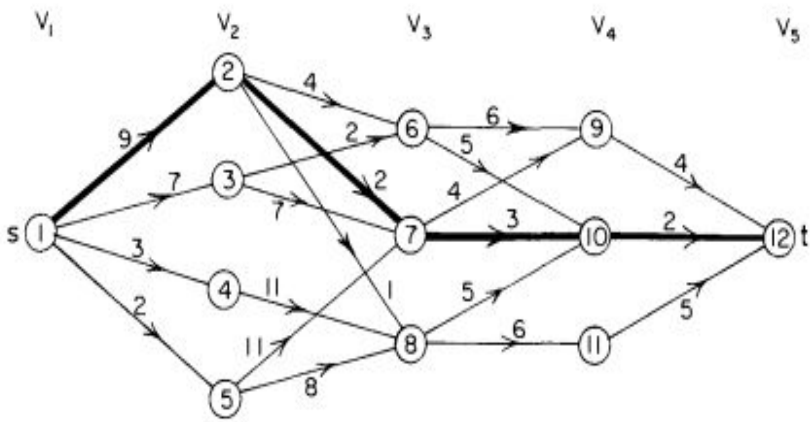
4+bcost(3,7))

= 15   (min 7)

Bcost(4,10) = min(5+bcost(3,6), 3+bcost(3,7), 5+bcost(3,8)) = 14 (min 7)

Bcost (4,11)= 6+bcost(3,8)= 16

(min 8)



Bcost(5,12) =

min(4+bcost(4,9),

2+bcost(4,10),

5+bcost(4,11))

= 16 (min 10)

Cost of Shortest path from s to t = 16
Shortest Path **1**⟶ **2** ⟶ **7** ⟶ **10** ⟶ **12**

ALGORITHM FOR 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 minimum;
        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]];
}
```

Algorithm Multistage graph pseudocode corresponding to backward approach

Exercise : Find the shortest path from source to sink