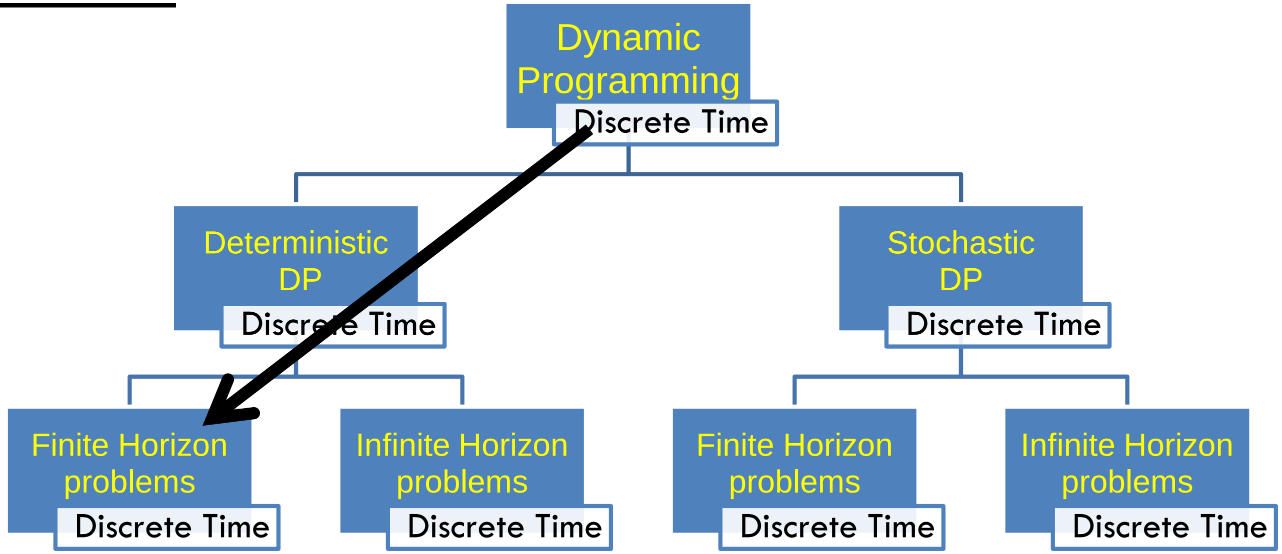
OR 674 DYNAMIC PROGRAMMING

Rajesh Ganesan,
Associate Professor
Systems Engineering and Operations Research
George Mason University

Ankit Shah
Ph.D. Candidate

Dynamic Programming

- What is Dynamic Programming (DP)?
  - An optimization method that finds the shortest path (ex: minimize cost) or the longest path (ex: maximize reward) in decision making problems that are solved sequentially over time.



Linear Programming Example

Example: Sailco corporation sells sailboats.

- Sailco's objective is to find an optimal production strategy for its sailboats for each day for the next 4 days.
- Demand is deterministic:
  - D1 = 40, D2 = 60, D3 = 75, D4 = 25
- Cost of making a boat:
  - with regular labor hours = $400/boat
  - with overtime labor hours = $450/boat
- Holding cost of a boat in inventory = $20/boat
- Maximum number of boats that can be produced using regular labor hours = 40
- Starting inventory = 10 boats on day 1
- All demand must be met.

Linear Programming Example

- Boats could be produced by regular labor and overtime labor.
  - Let x_t be number of boats produced by regular labor during day t.
  - Let y_t be number of boats produced by overtime labor during day t.
- Let i_t be the inventory remaining at the end of the day t.
  - Inventory at the end of day 1:
    - $i_1 = i_0 + x_1 + y_1 - D_1$
  - Inventory at the end of day 2:
    - $i_2 = i_1 + x_2 + y_2 - D_2$
  - Inventory at the end of day n:
    - $i_n = i_{n-1} + x_n + y_n - D_n$

Linear Programming Example

Objective:

- Minimize production costs (regular labor and overtime labor) and holding costs.
  - Minimize $\sum_{t=1}^{T}(400 * x_t + 450 * y_t + 20 * i_t)$

Constraints:

- Demand on each day must be met.
  - i_t>= 0
- Up to 40 boats per day can be produced with regular labor hours.
  - x_t<=40

Linear Programming Example

$$Min\ Z = 400x_1+400x_2+400x_3+400x_4+450y_1+450y_2+450y_3+450y_4+20i_1+20i_2+20i_3+20i_4 \quad (1)$$

Constraint: Inventory at the end of day t = Inventory at the end of day $(t-1)$+production during day t by regular labor+ production during day t by overtime labor - demand during day t

$$\begin{aligned} i_1 &= 10 + x_1 + y_1 - 40 && (2) \\ i_2 &= i_1 + x_2 + y_2 - 60 && (3) \\ i_3 &= i_2 + x_3 + y_3 - 75 && (4) \\ i_4 &= i_3 + x_4 + y_4 - 25 && (5) \\ x_t &\leq 40, \quad \forall\, t && (6) \\ i_t &\geq 0, \quad \forall\, t && (7) \\ y_t &\geq 0, \quad \forall\, t && (8) \\ x_t &\geq 0, \quad \forall\, t && (9) \end{aligned}$$

Challenges

How complex would be the problem if the following occur:

- If the problem had to be solved for 1 year?
  - 365*3 = 1095 variables, 365*2 = 730 constraints.
- If the problem had to be solved for 10 years?
  - 10950 variables, 7300 constraints.
- If the demand was probabilistic?
  - Modeling the problem may not be possible.

Dynamic Programming is an approach for solving such problems.

Sequential Decision Making (Dynamic Programming)

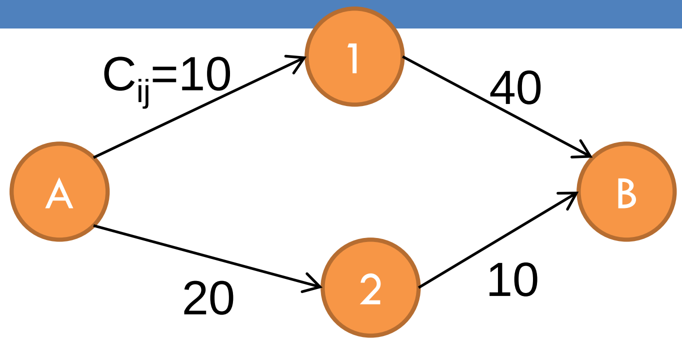
- Dynamic decisions **over time** and **uncertainty** (stochastic behavior)

on top of

- Big data
- Complex non-linear system
- Computational difficulty (state space and dimensionality)
- Time between decisions too short

Dynamic Programming Approach

C_{ij}= cost on the arc



Myopic policy: $V(A) = \min (C_{ij})$

= min of (10 or 20)

leads to solution of 50 from A to 1 to B

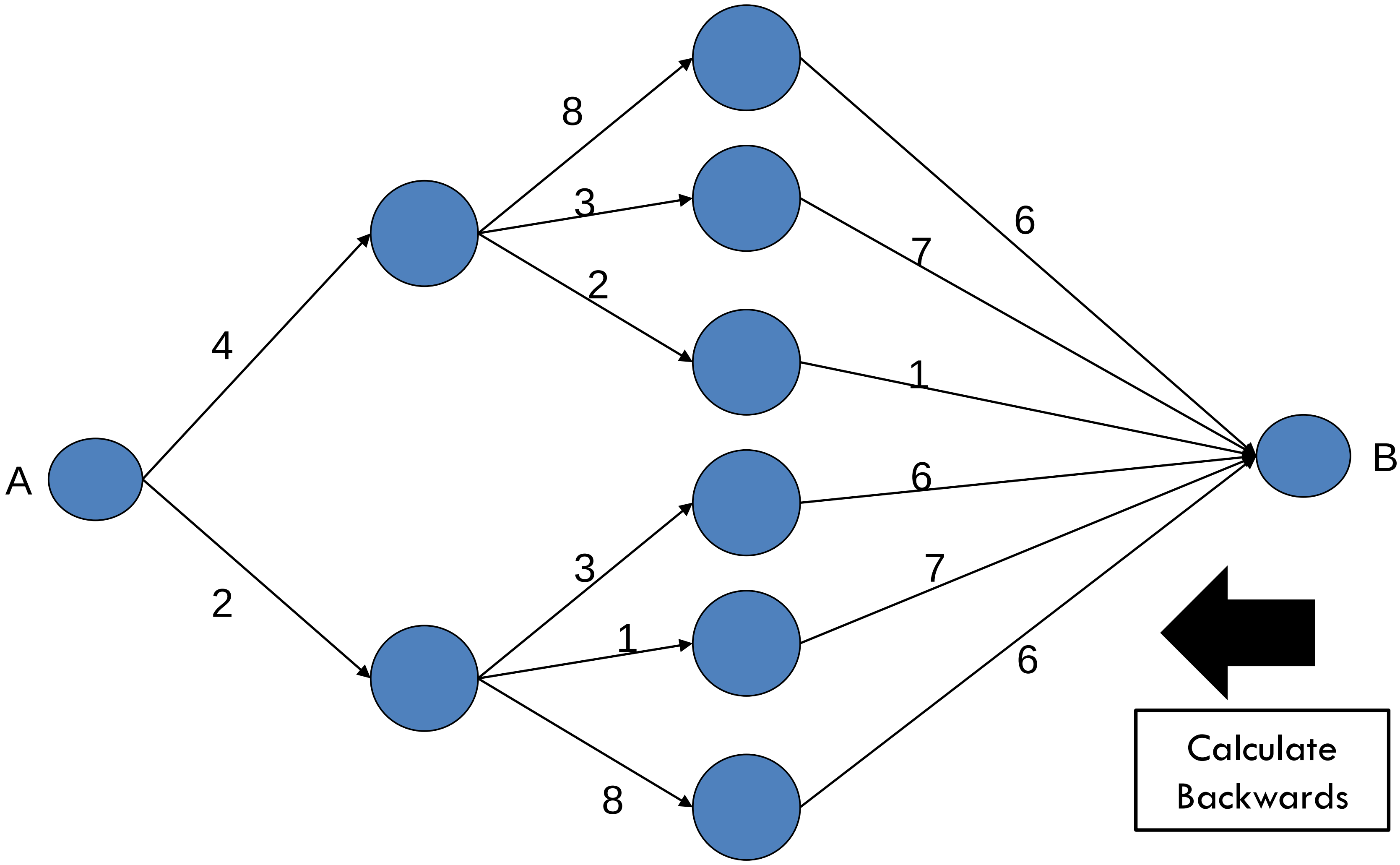
DP policy: $V(A) = \min (C_{ij} + V(\text{next node}))$

= min (10 + 40, 20+10) = 30

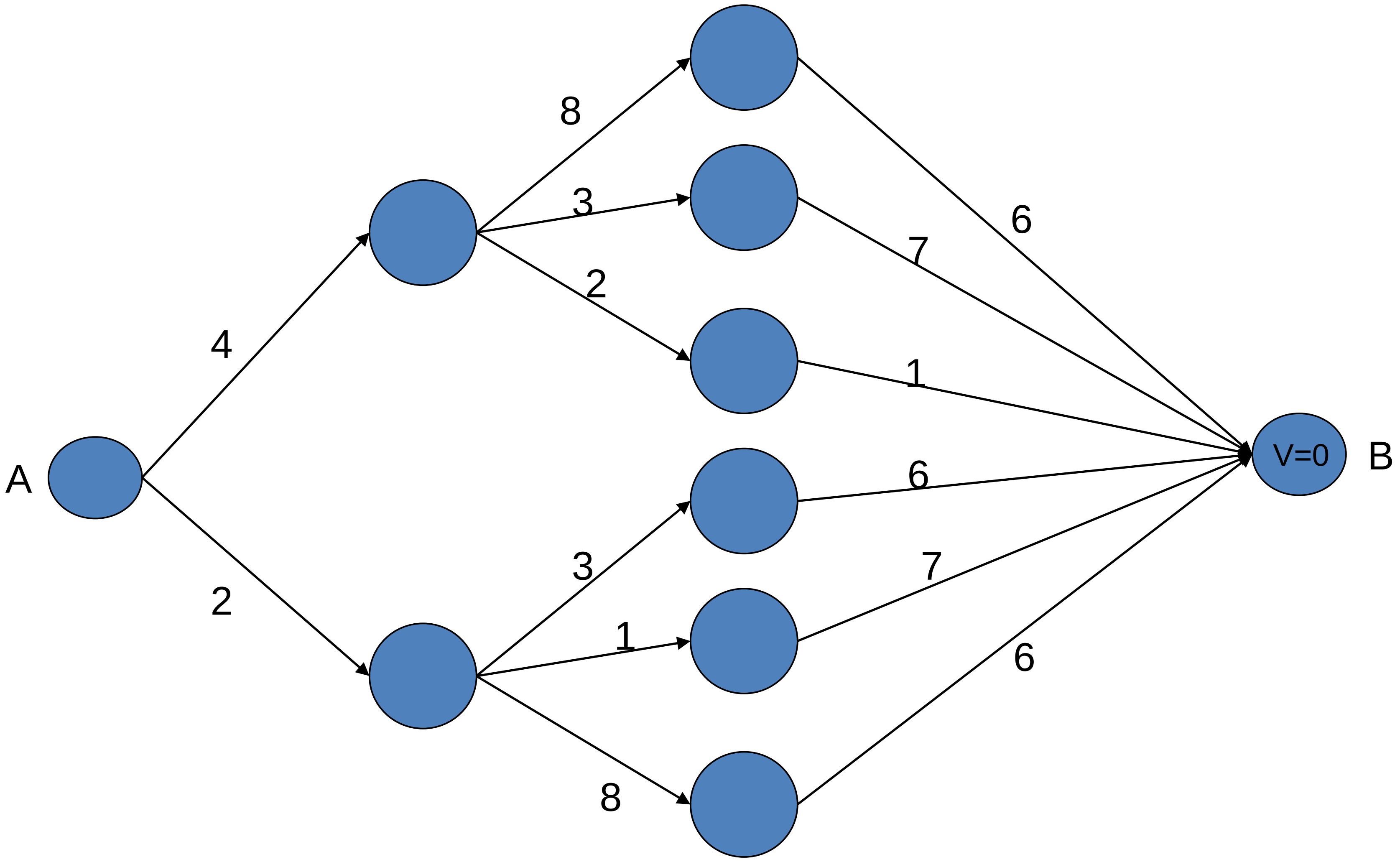
leads to solution of 30 from A to 2 to B

Key is to find the values of node 1 and 2

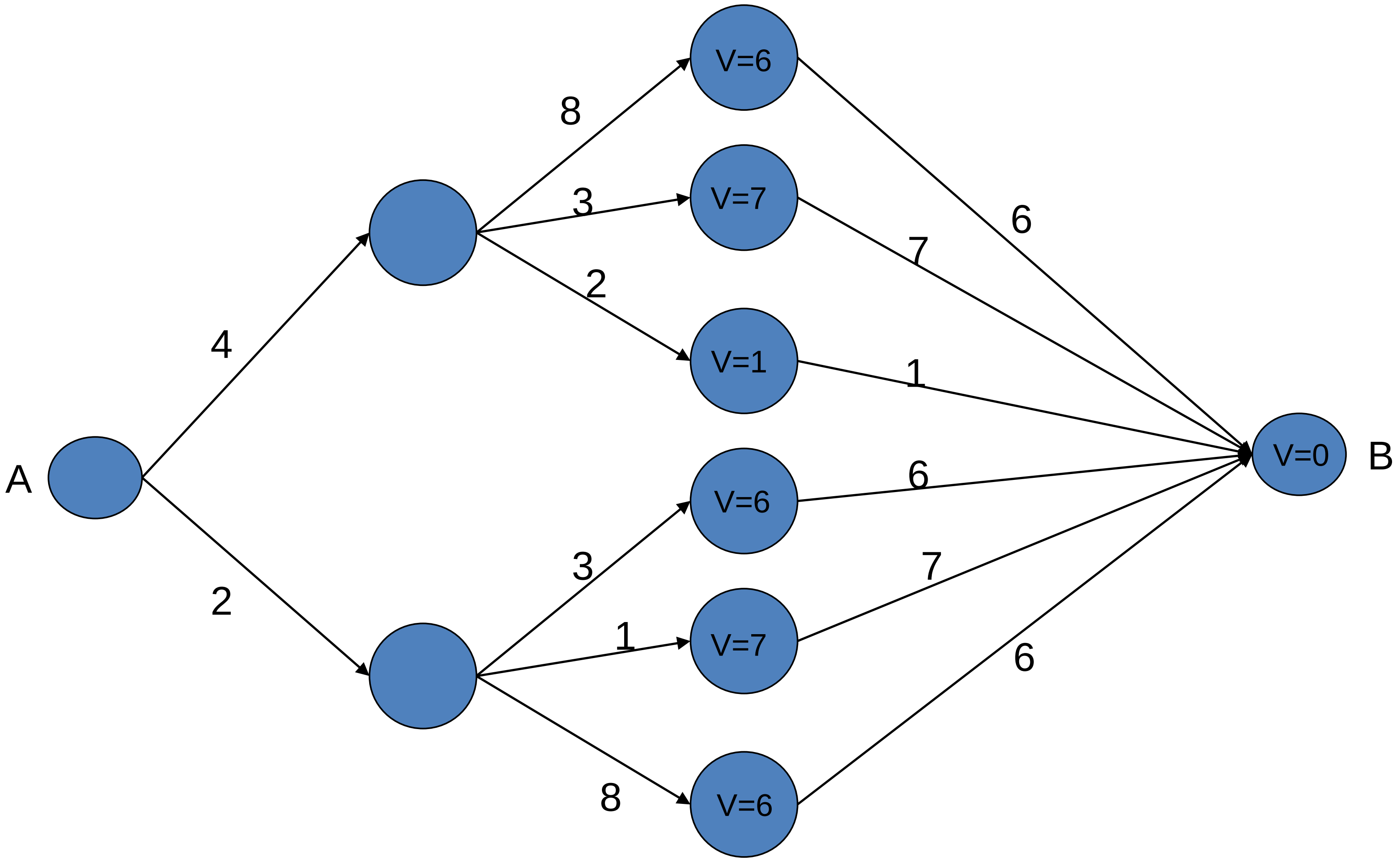
Find Shortest Path from A to B (using DP)



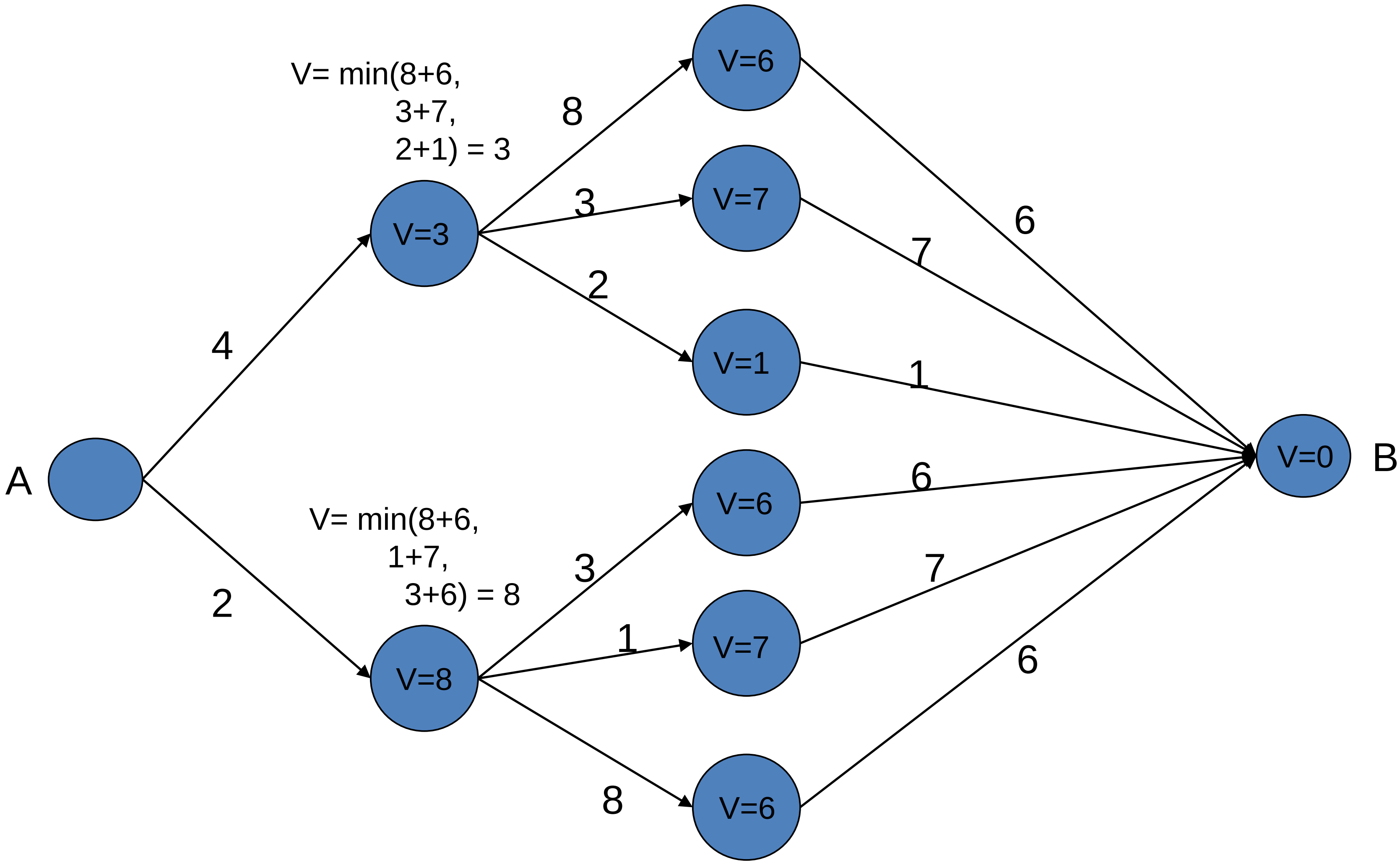
Find Shortest Path from A to B (using DP)



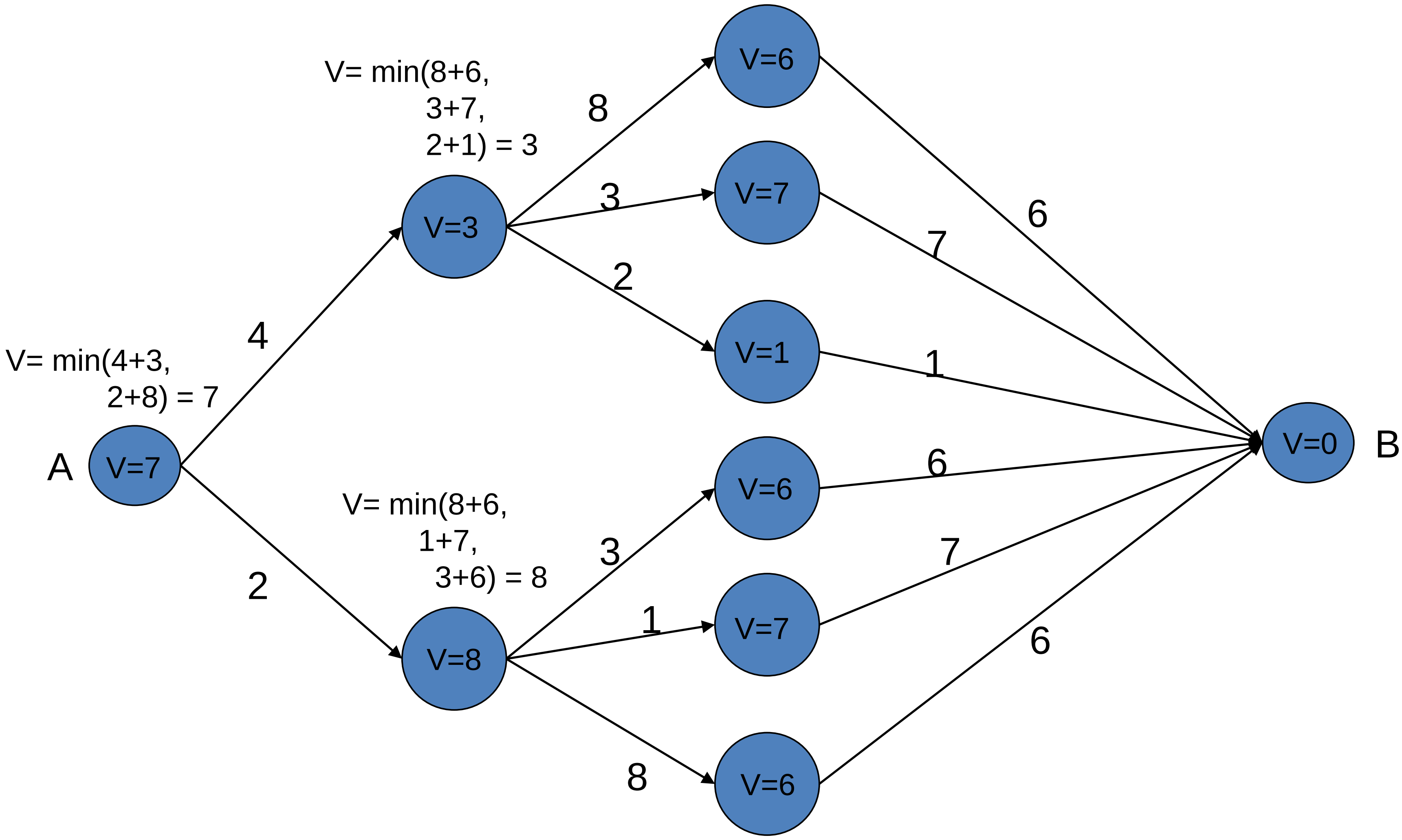
Find Shortest Path from A to B (using DP)



Find Shortest Path from A to B (using DP)

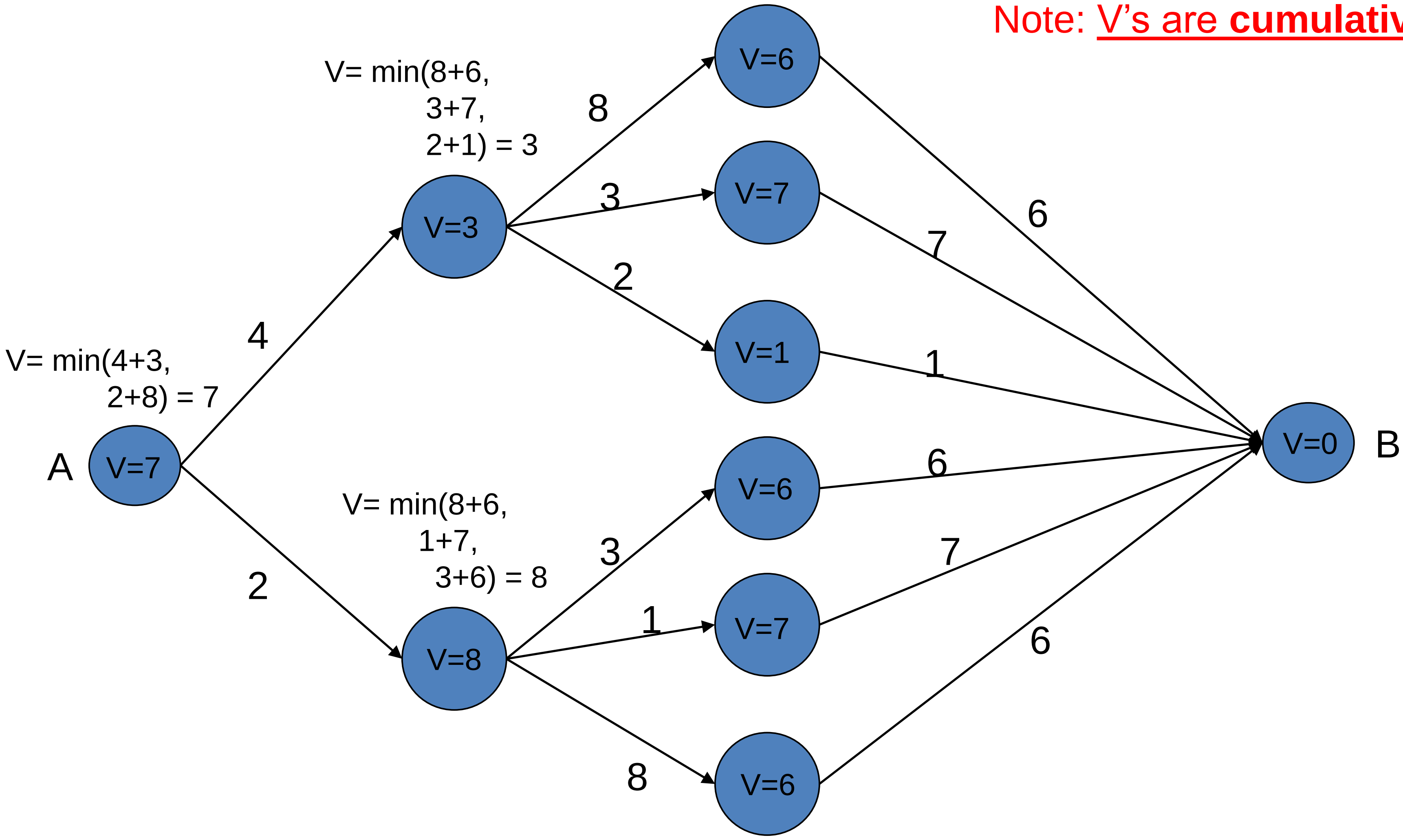


Find Shortest Path from A to B (using DP)



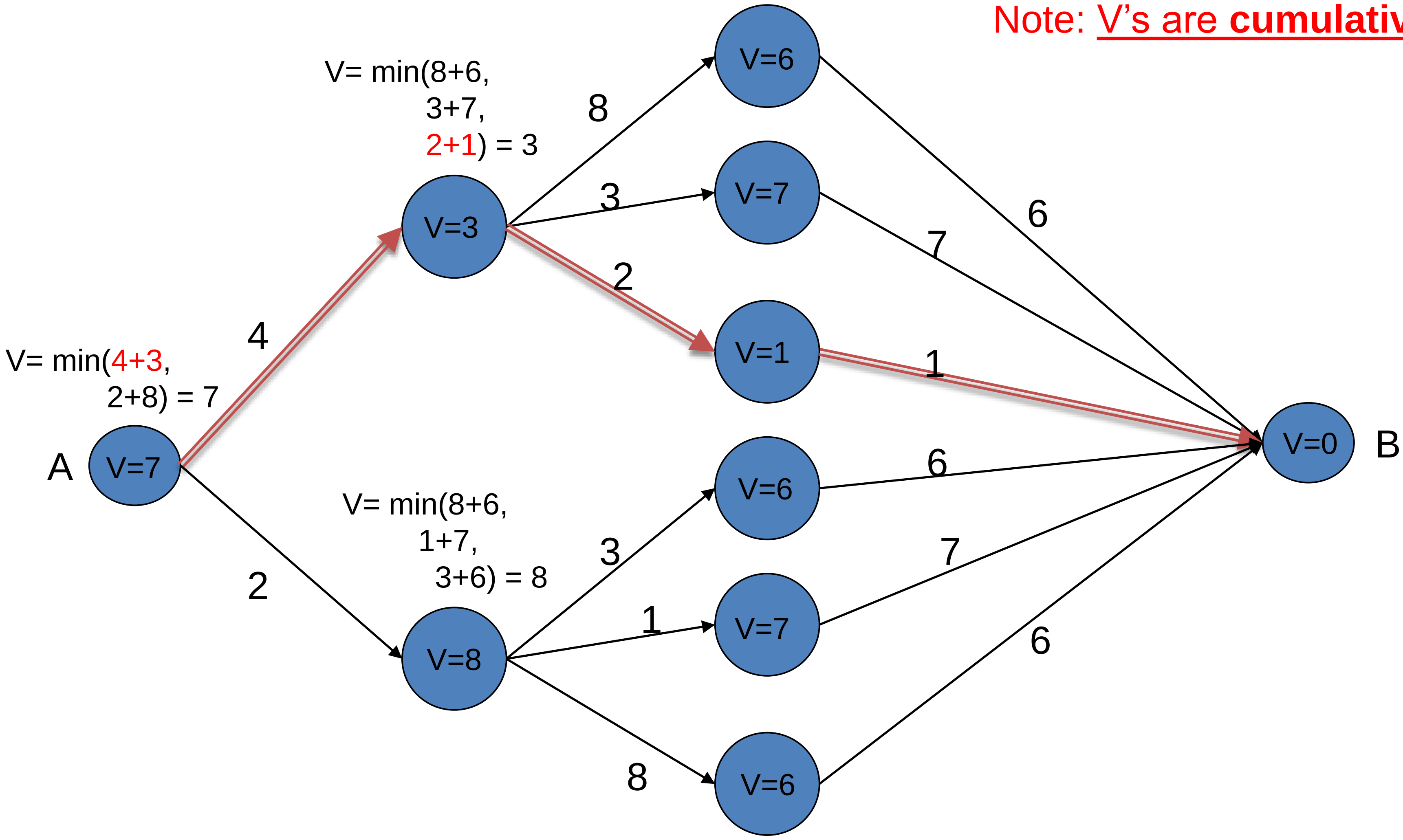
Find Shortest Path from A to B (using DP)

Note: V's are **cumulative**



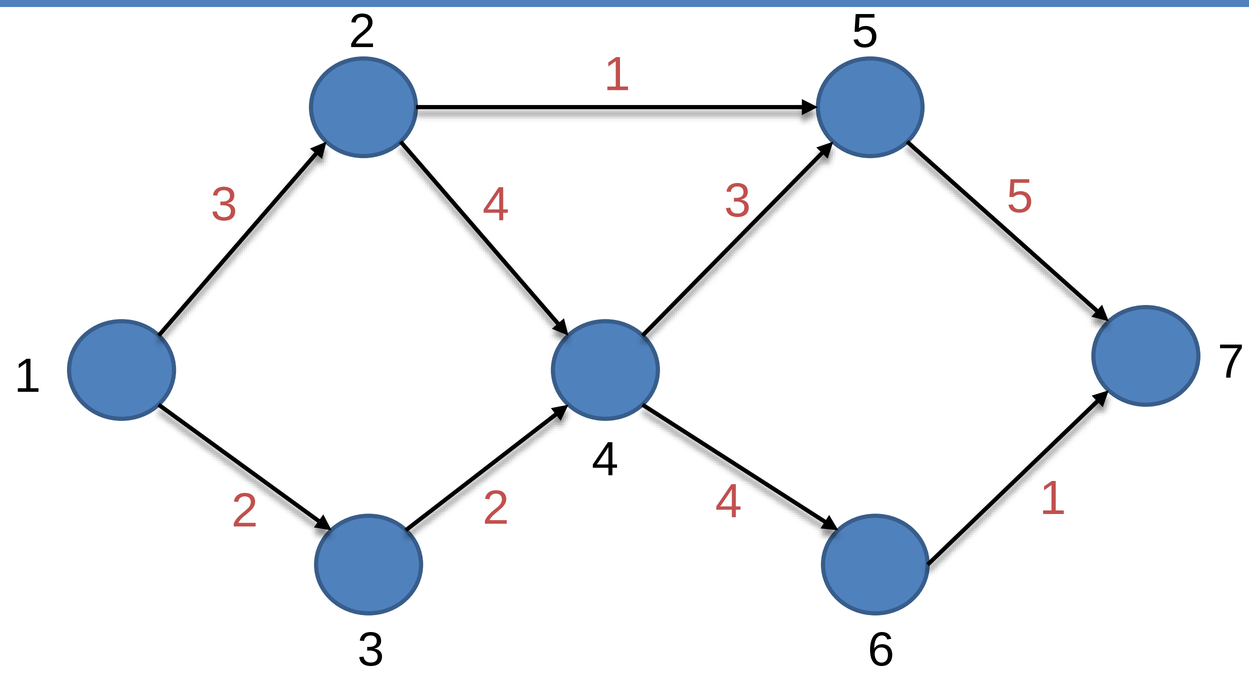
Find Shortest Path from A to B (using DP)

Note: V's are **cumulative**



Another Example

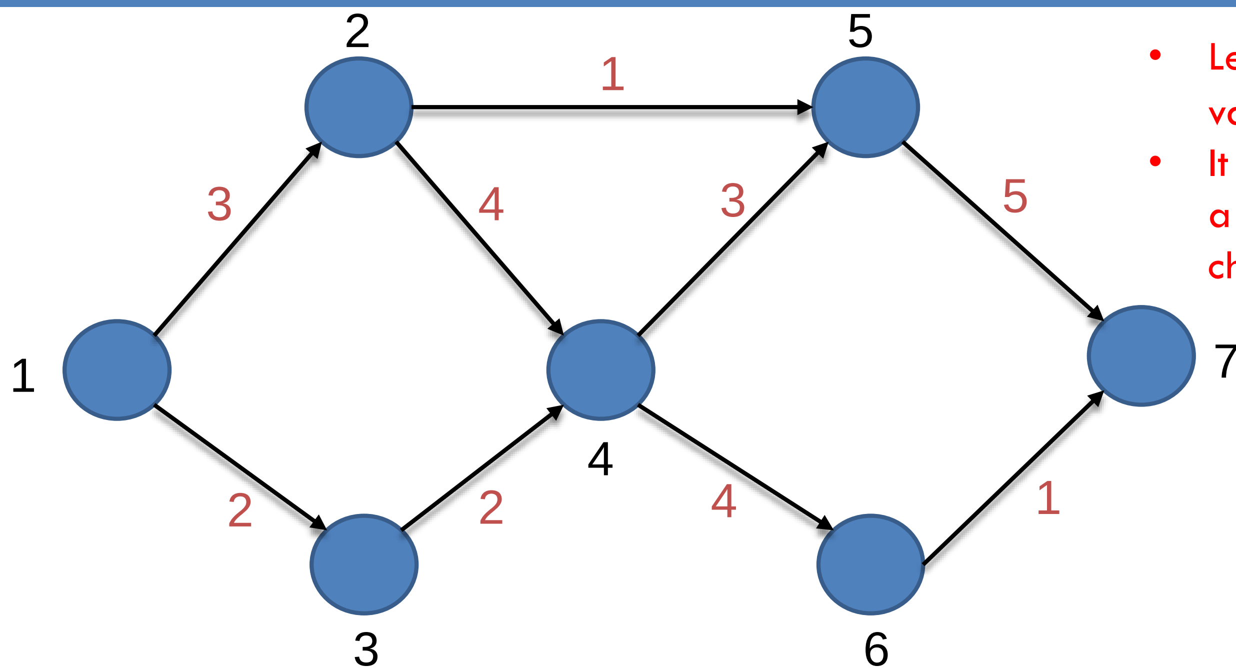
Find Shortest Path from Node 1 to 7



- First, we will model the problem using Integer Programming approach.
- Next, we will show the Dynamic Programming approach.

Integer Programming Approach

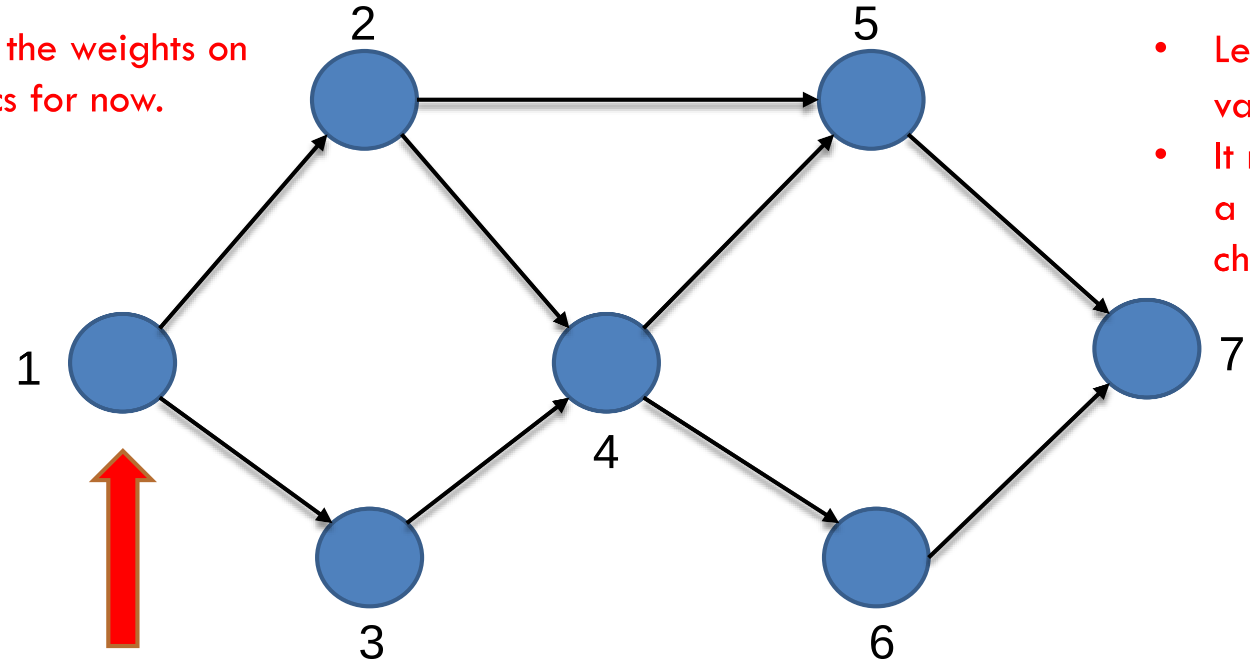
Integer Programming Approach



- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.

Integer Programming Approach

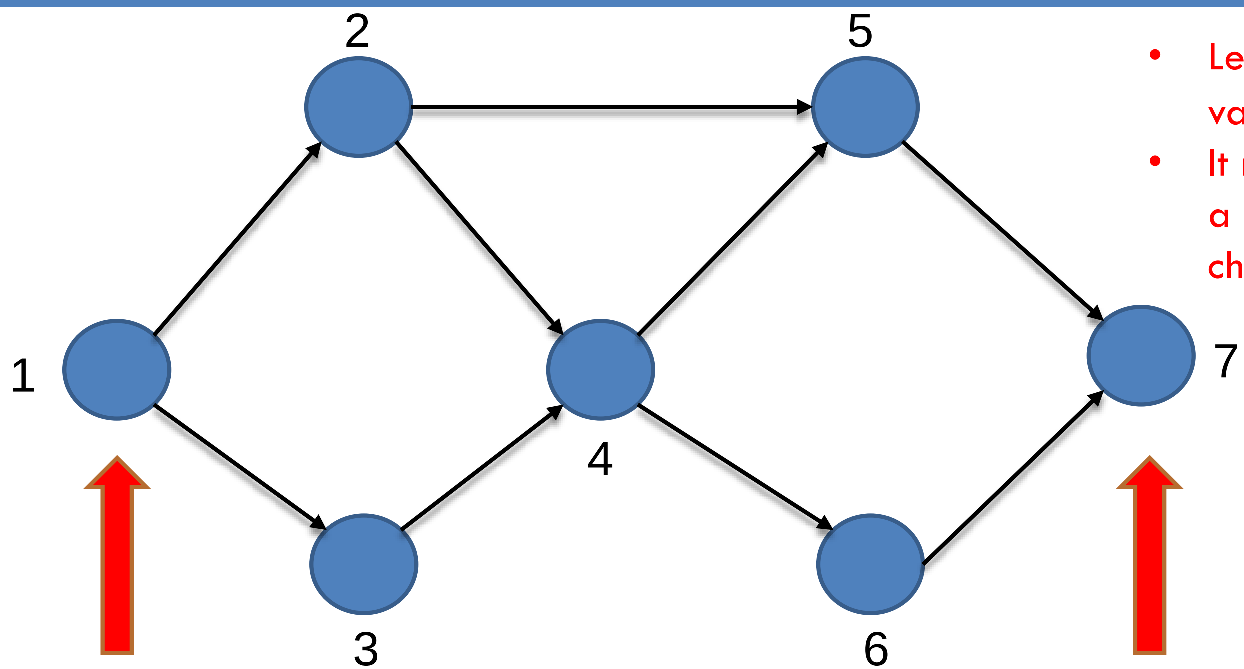
- Ignore the weights on the arcs for now.



- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.

$$x_{12} + x_{13} = 1$$

Integer Programming Approach



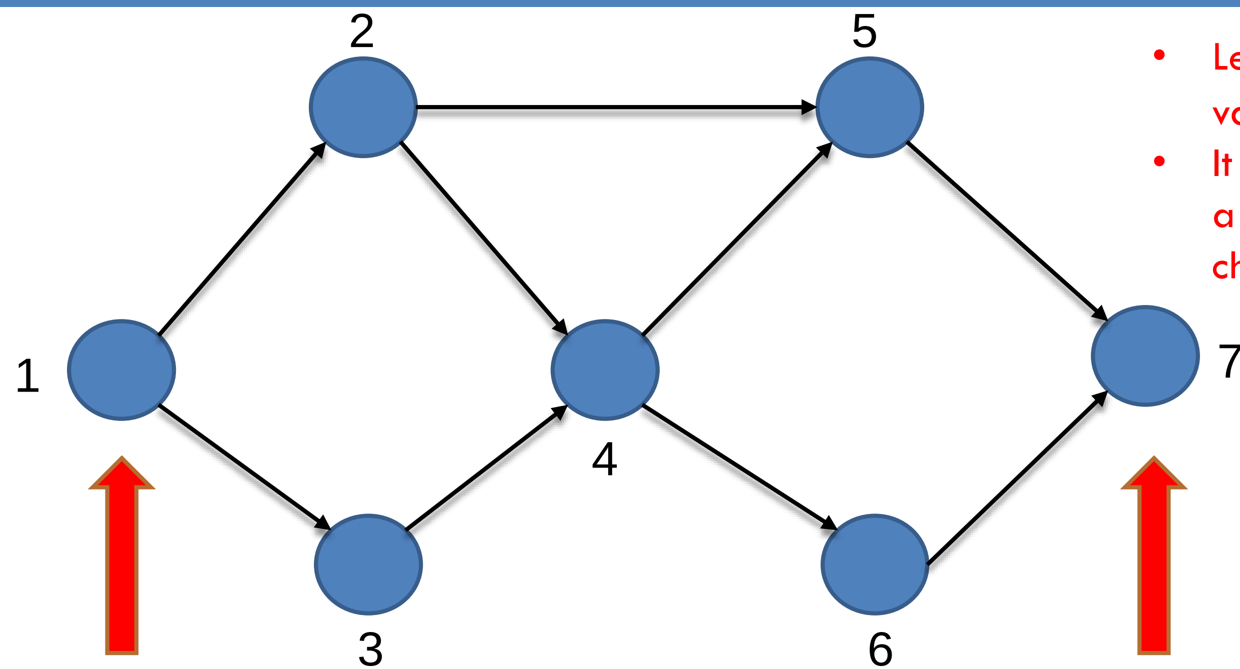
- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.

$x_{12} + x_{13} = 1$

$x_{57} + x_{67} = 1$

Integer Programming Approach

- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.



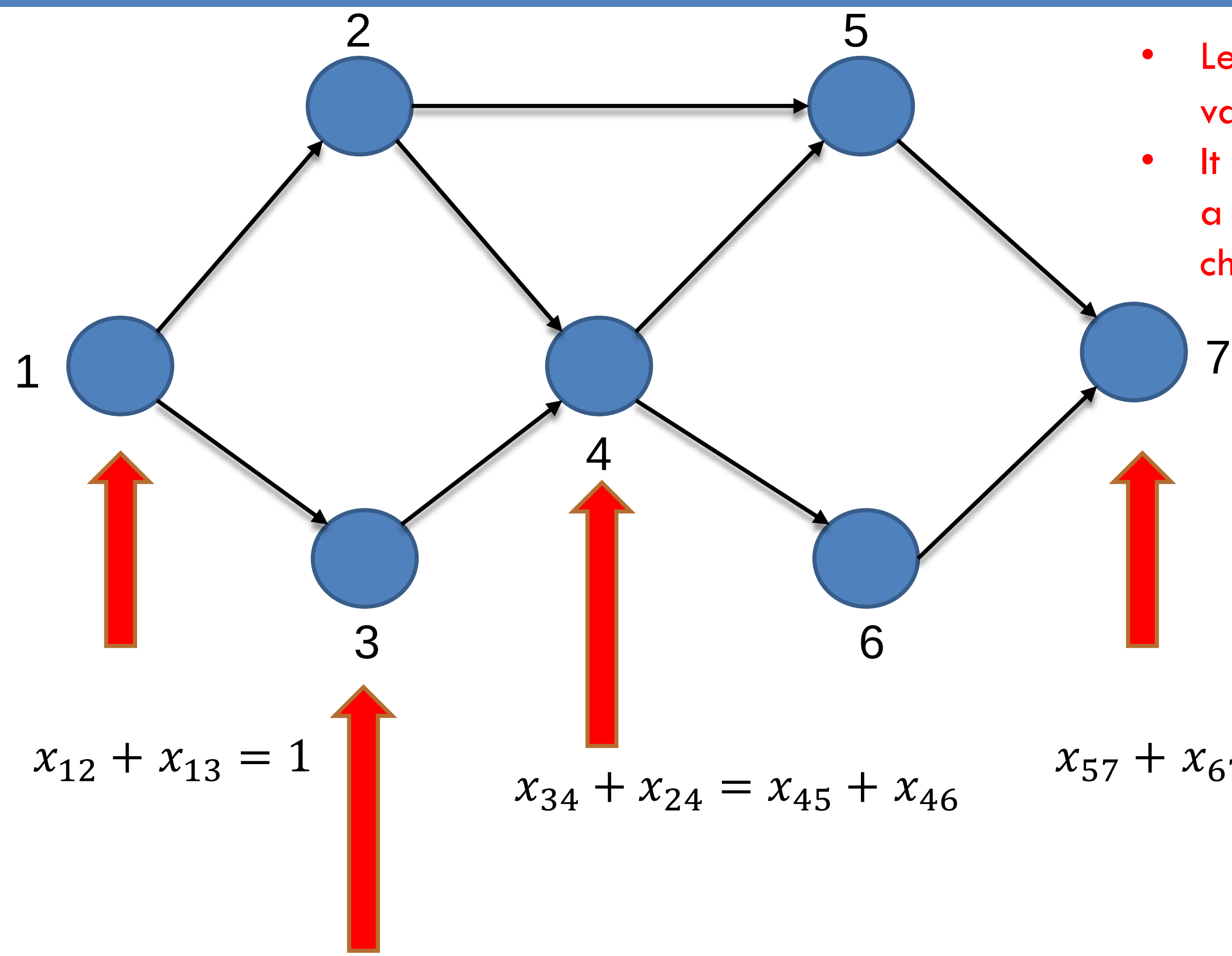
$x_{12} + x_{13} = 1$

$x_{57} + x_{67} = 1$

$x_{13} = x_{34}$
(Flow in = Flow out)

Integer Programming Approach

- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.



$x_{12} + x_{13} = 1$

$x_{34} + x_{24} = x_{45} + x_{46}$

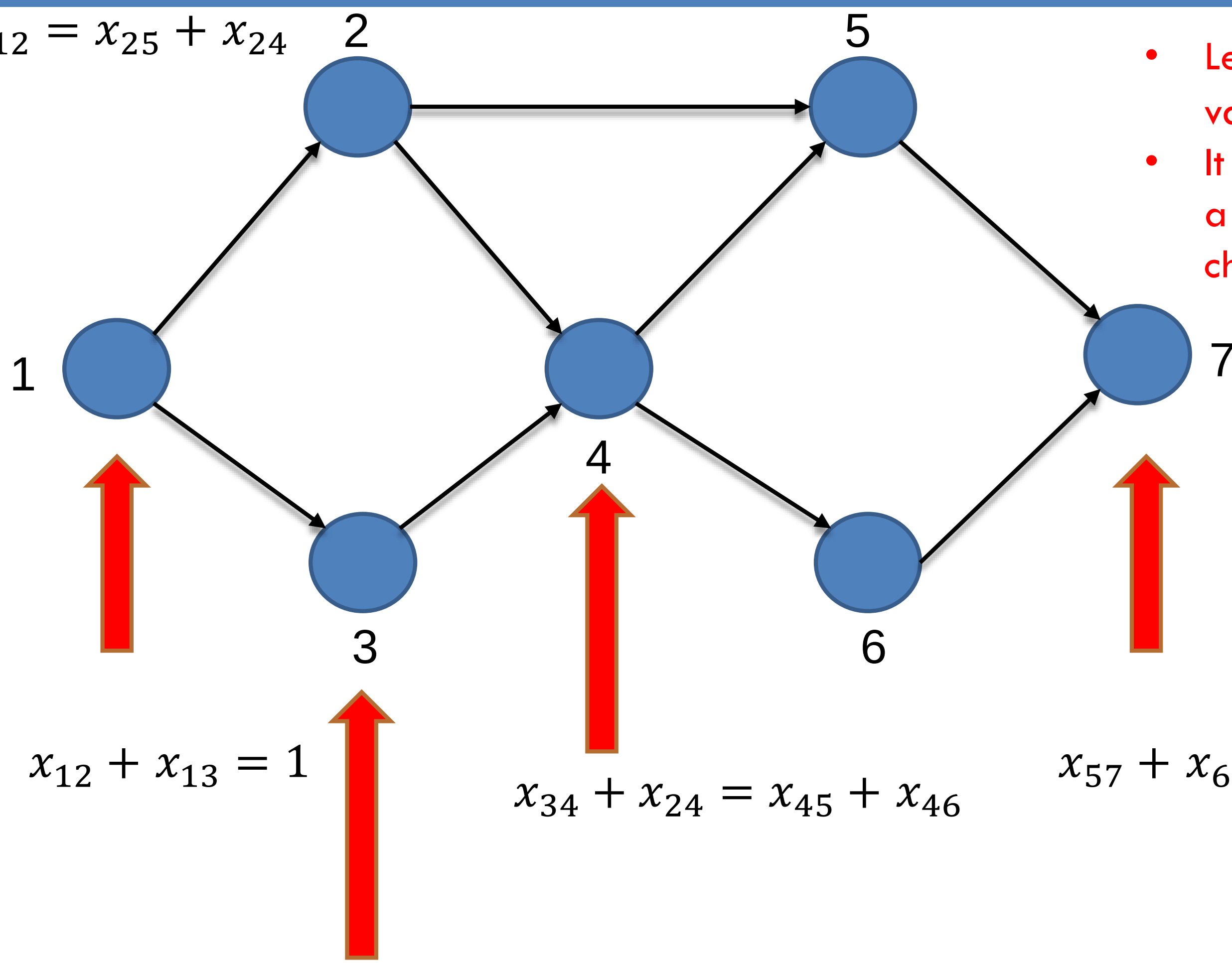
$x_{57} + x_{67} = 1$

$x_{13} = x_{34}$
(Flow in = Flow out)

Integer Programming Approach

- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.

$x_{12} = x_{25} + x_{24}$



$x_{12} + x_{13} = 1$

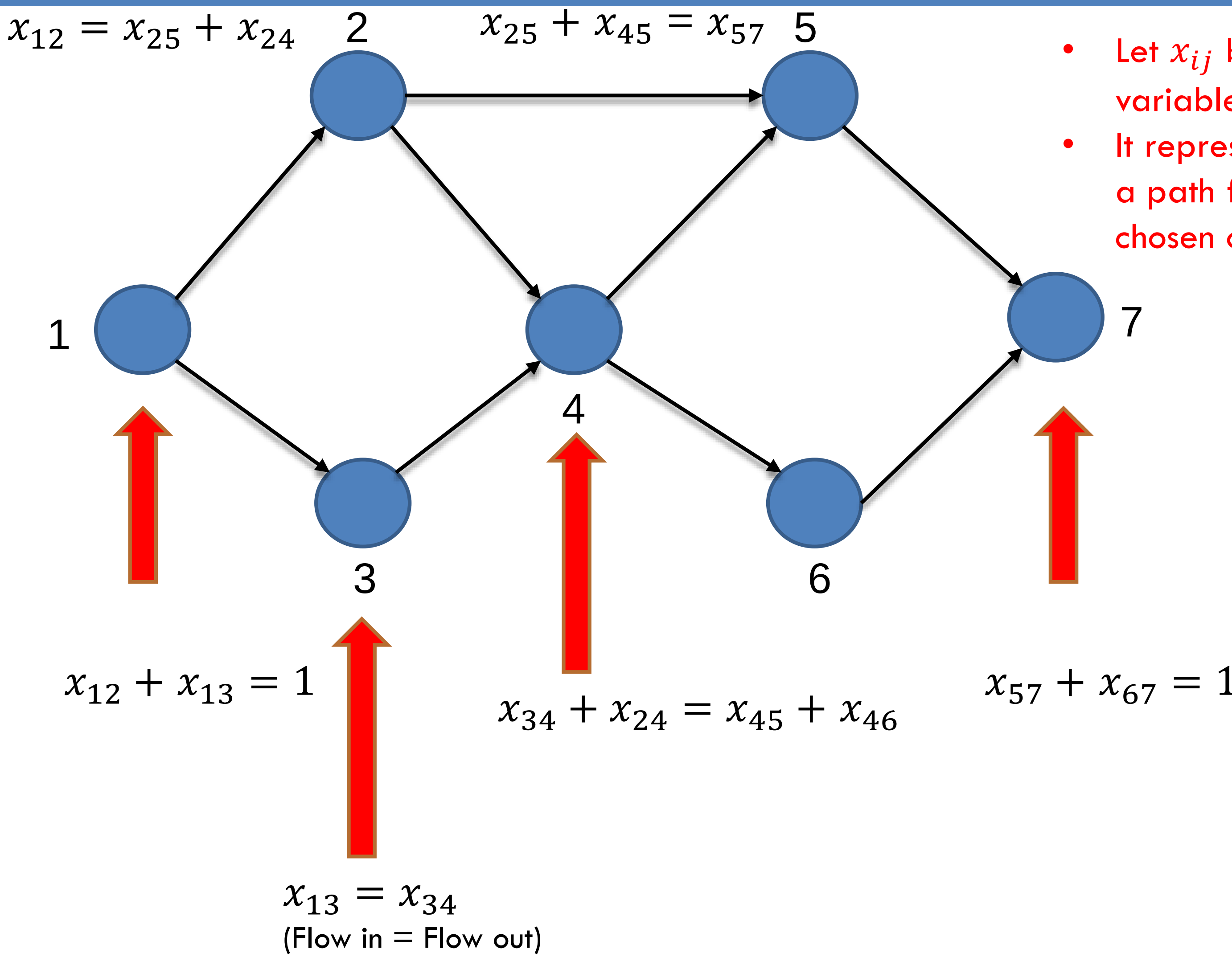
$x_{34} + x_{24} = x_{45} + x_{46}$

$x_{57} + x_{67} = 1$

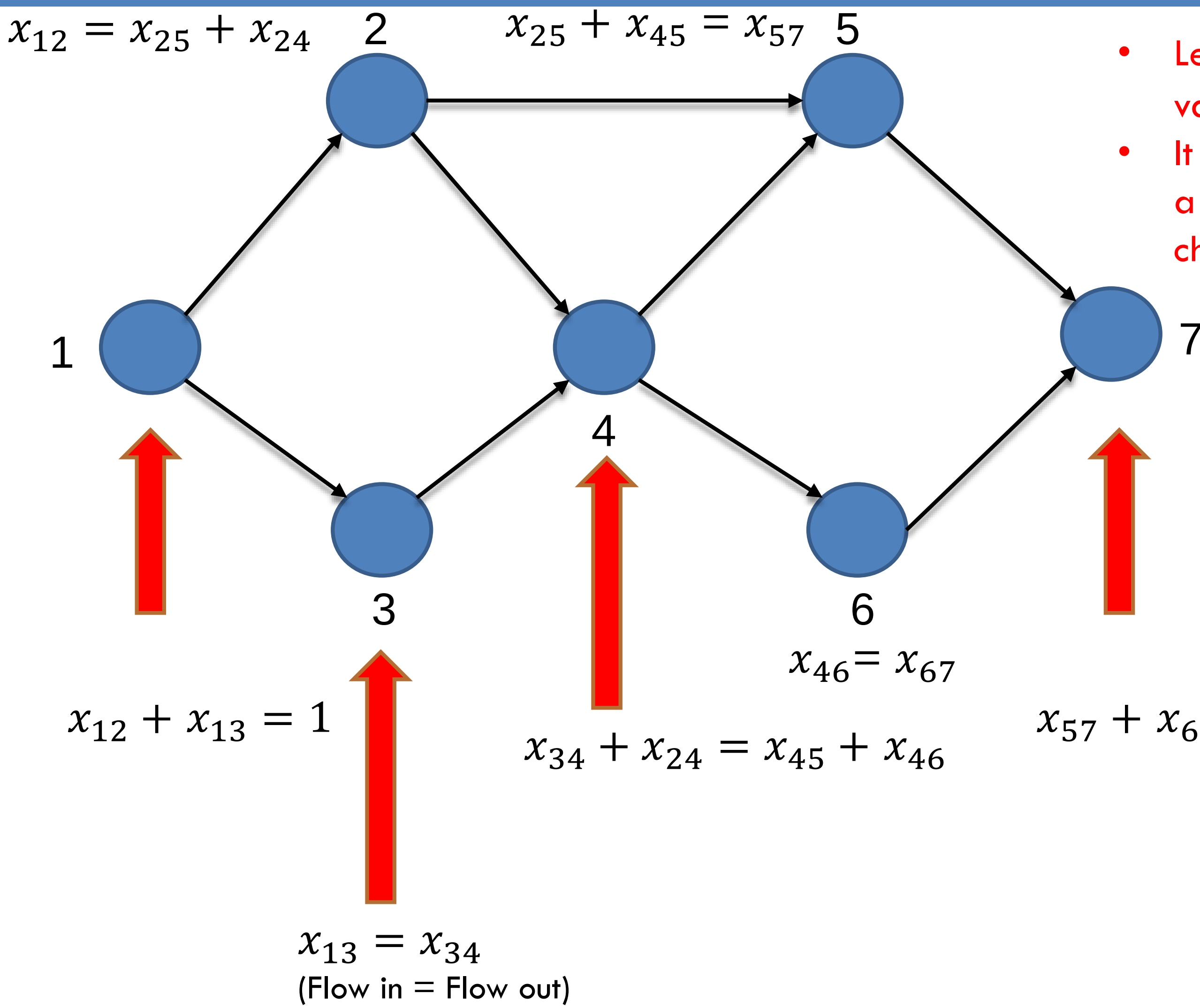
$x_{13} = x_{34}$
(Flow in = Flow out)

Integer Programming Approach

- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.



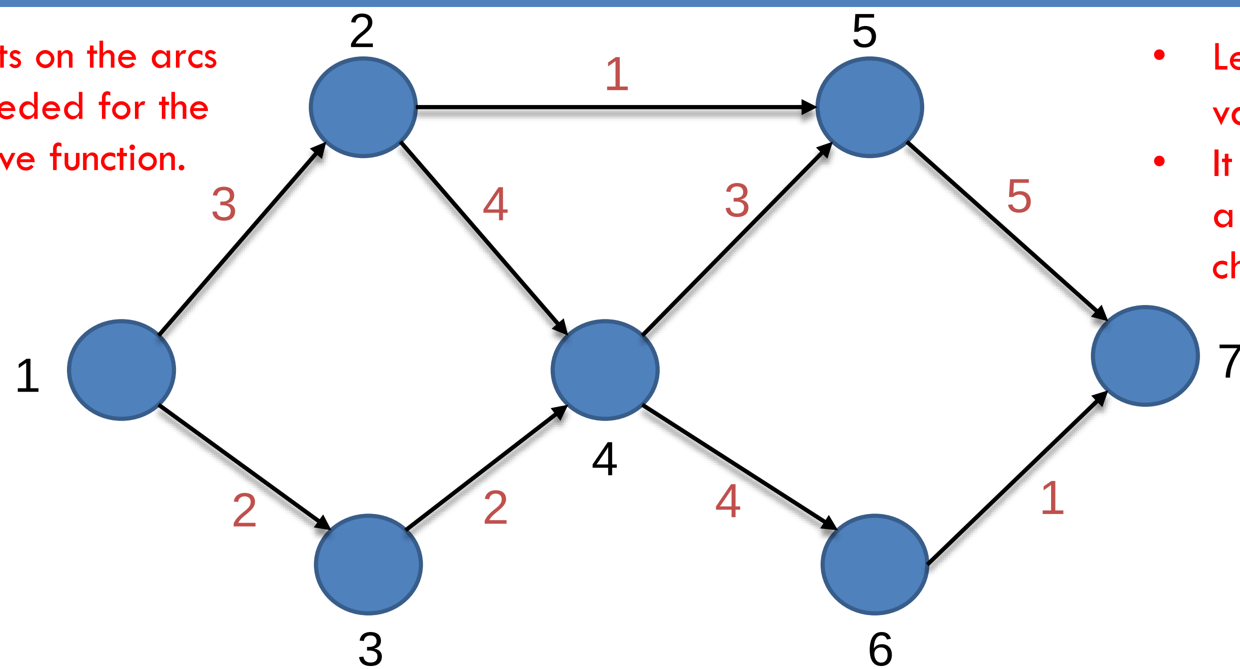
Integer Programming Approach



- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.

Integer Programming Approach

- Weights on the arcs are needed for the objective function.



- Let x_{ij} be a binary variable.
- It represents whether a path from i to j is chosen or not.

Objective: $Min\ Z = 3x_{12} + 2x_{13} + 1x_{25} + 4x_{24} + 2x_{34} + 3x_{45} + 4x_{46} + 5x_{57} + 1x_{67}$

Integer Programming Approach

Objective: $Min\ Z = 3x_{12} + 2x_{13} + 1x_{25} + 4x_{24} + 2x_{34} + 3x_{45} + 4x_{46} + 5x_{57} + 1x_{67}$

Constraints:

$x_{12} = x_{25} + x_{24}$

$x_{25} + x_{45} = x_{57}$

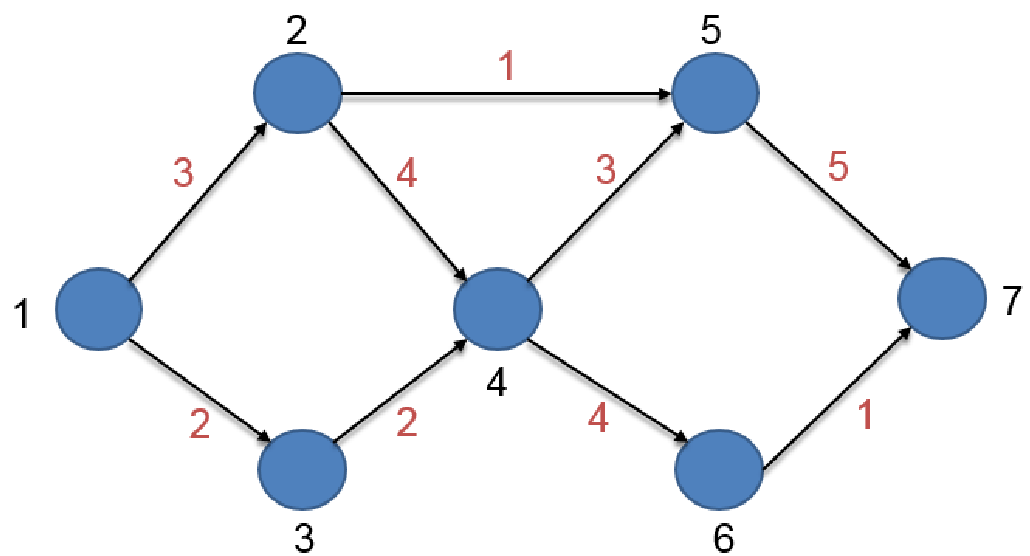
$x_{12} + x_{13} = 1$

$x_{46} = x_{67}$

$x_{57} + x_{67} = 1$

$x_{34} + x_{24} = x_{45} + x_{46}$

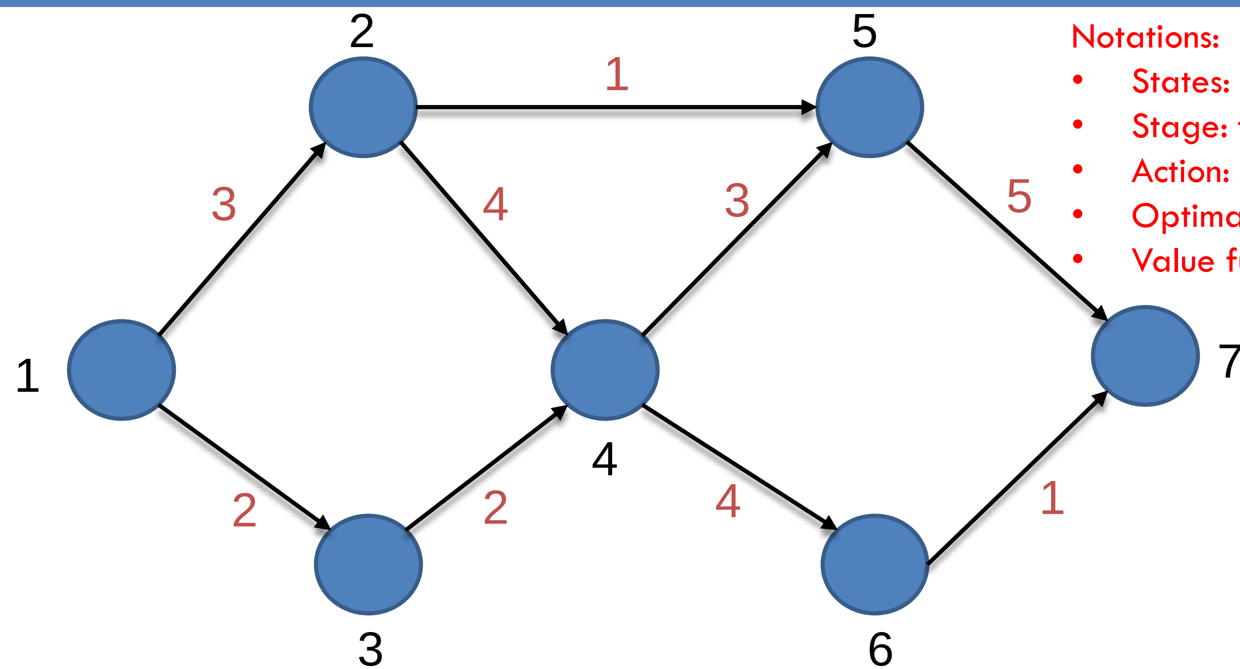
$x_{13} = x_{34}$



- As the number of states (nodes) increases, the number of equations will also increase.
- With Integer/Linear Programming, all variables and constraints are typically added and solved together as one BIG problem, which makes it computationally infeasible for large problems.

Dynamic Programming Approach

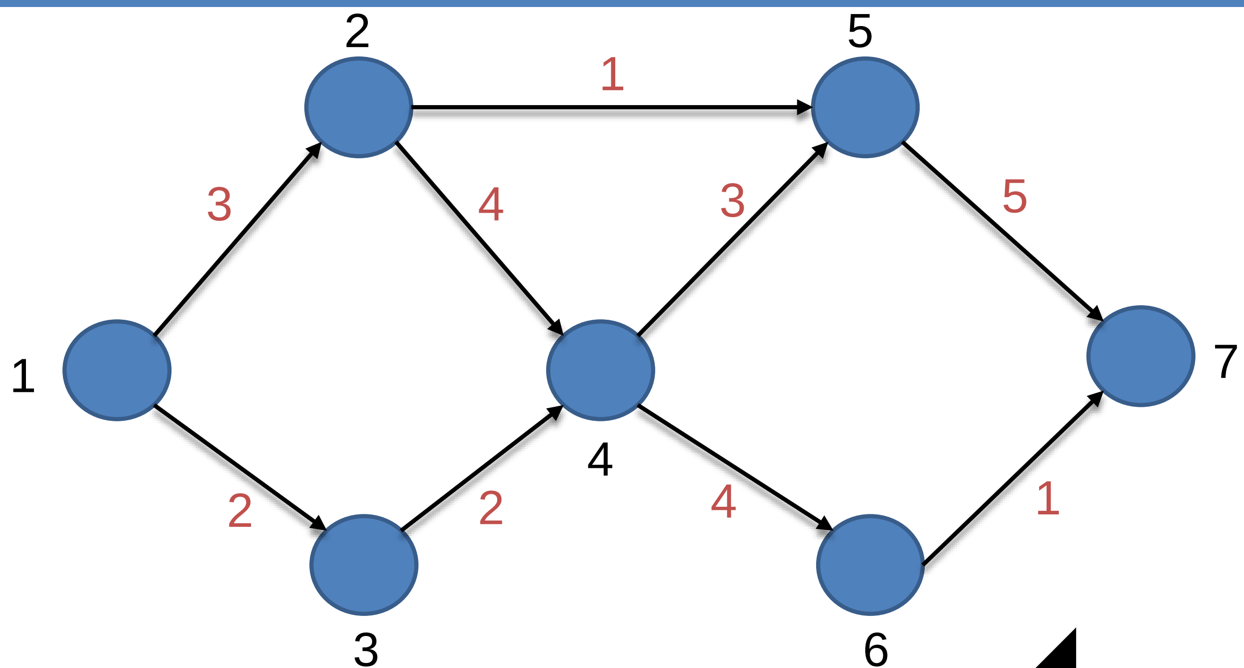
Dynamic Programming Approach



Notations:

- States: i, j
- Stage: t
- Action: k
- Optimal policy: R
- Value function of state: f(i)/V(i)

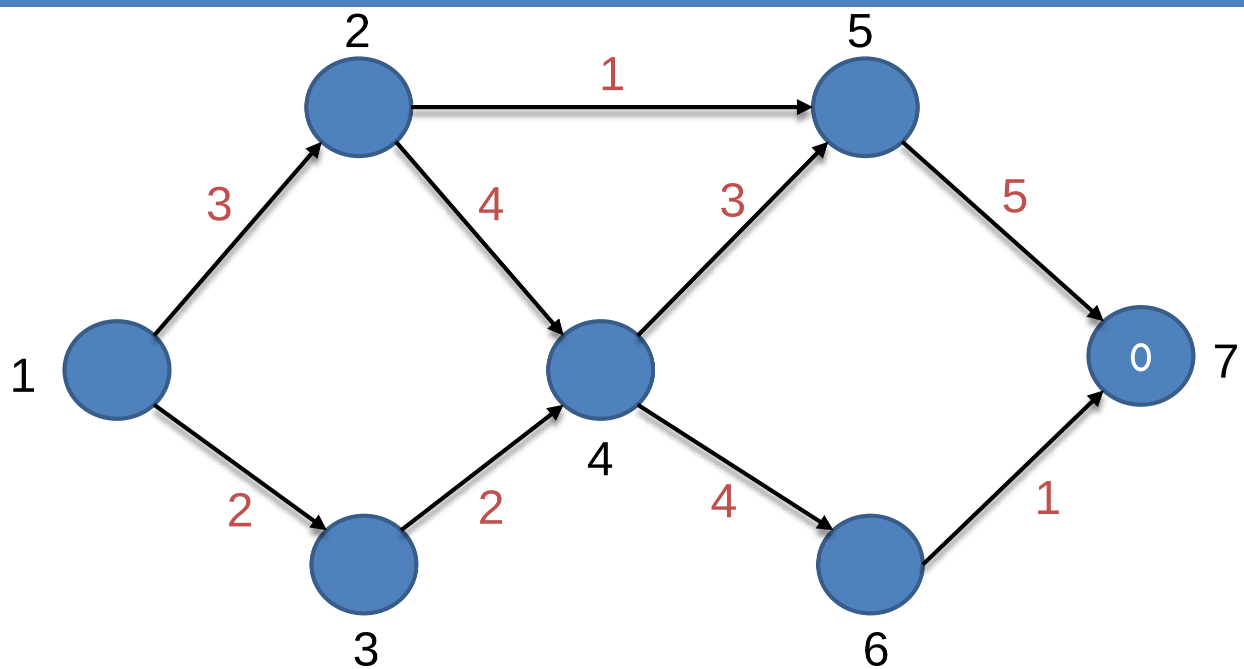
Dynamic Programming Approach



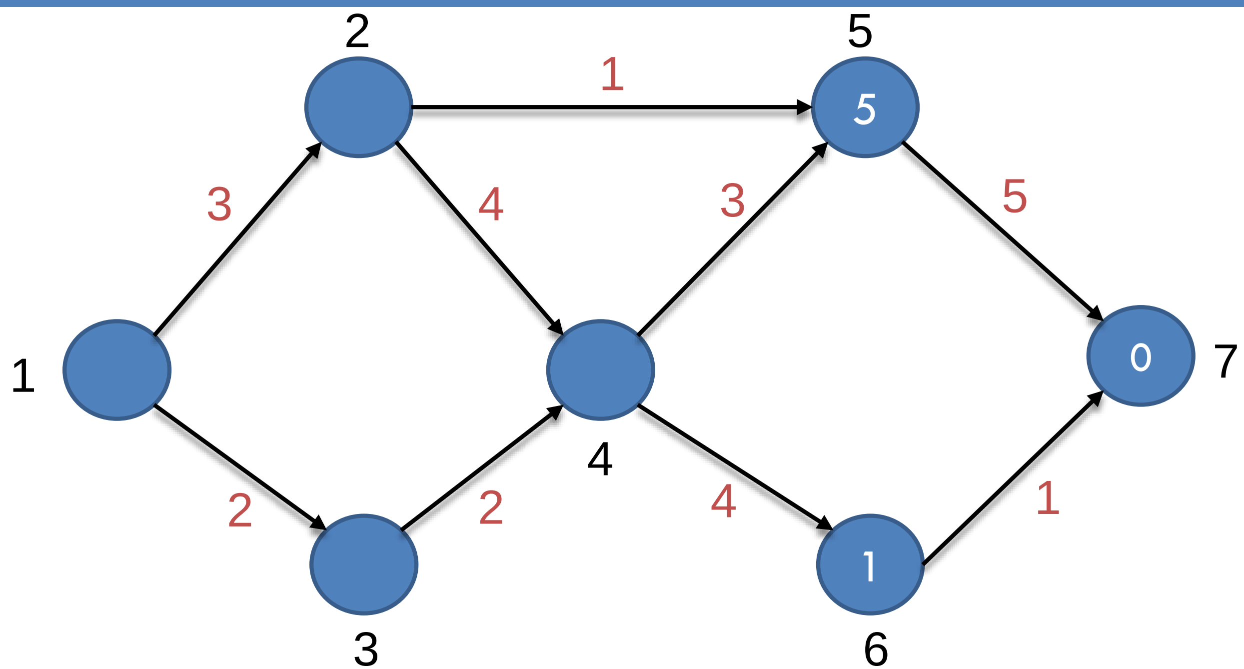
$$f_i = \min_j [c_{ij} + f_j],\ i < j$$

Backward Recursion

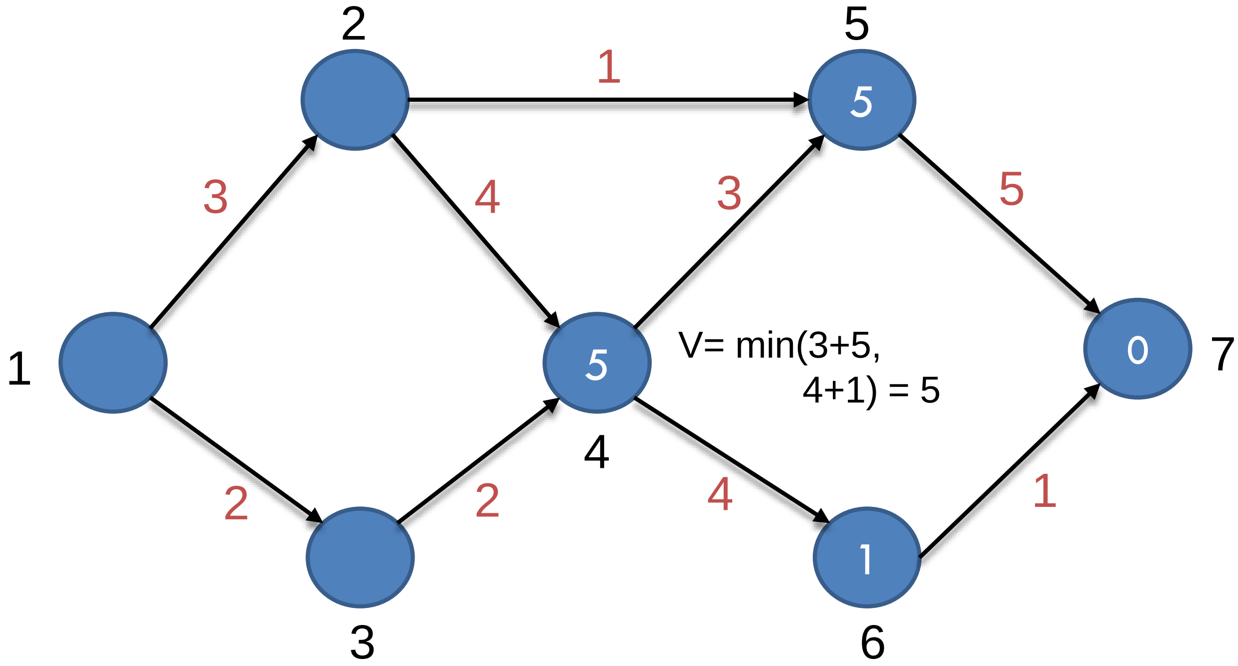
Dynamic Programming Approach



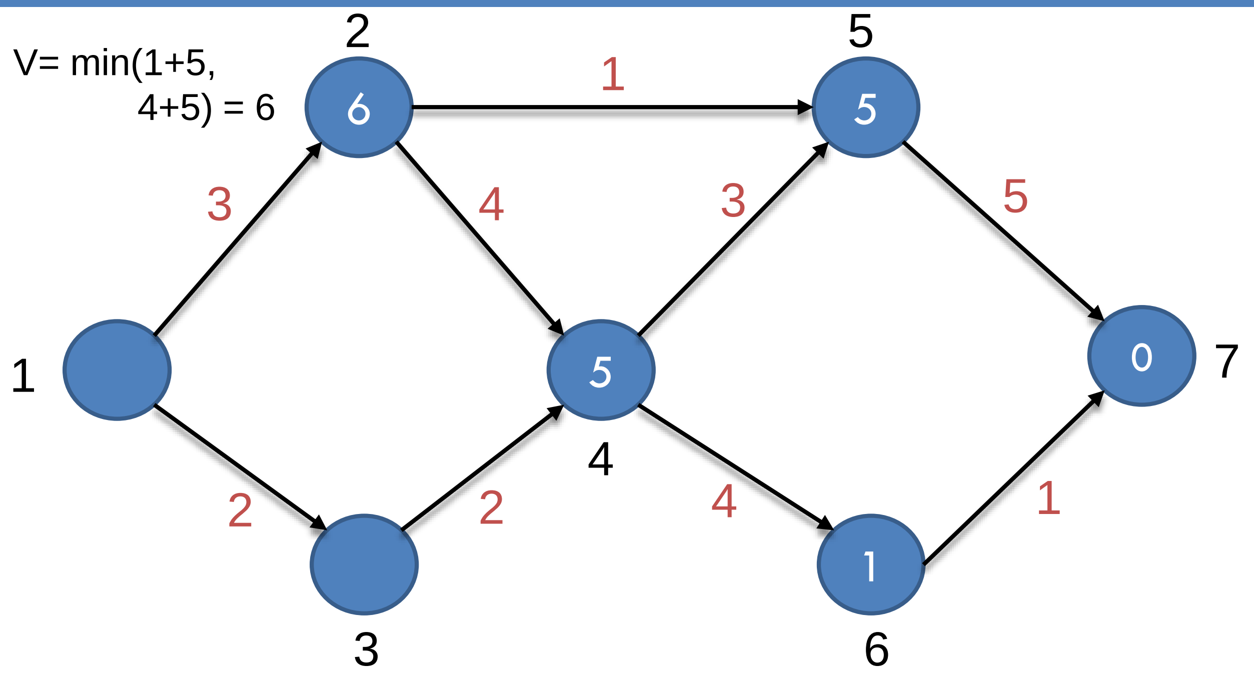
Dynamic Programming Approach



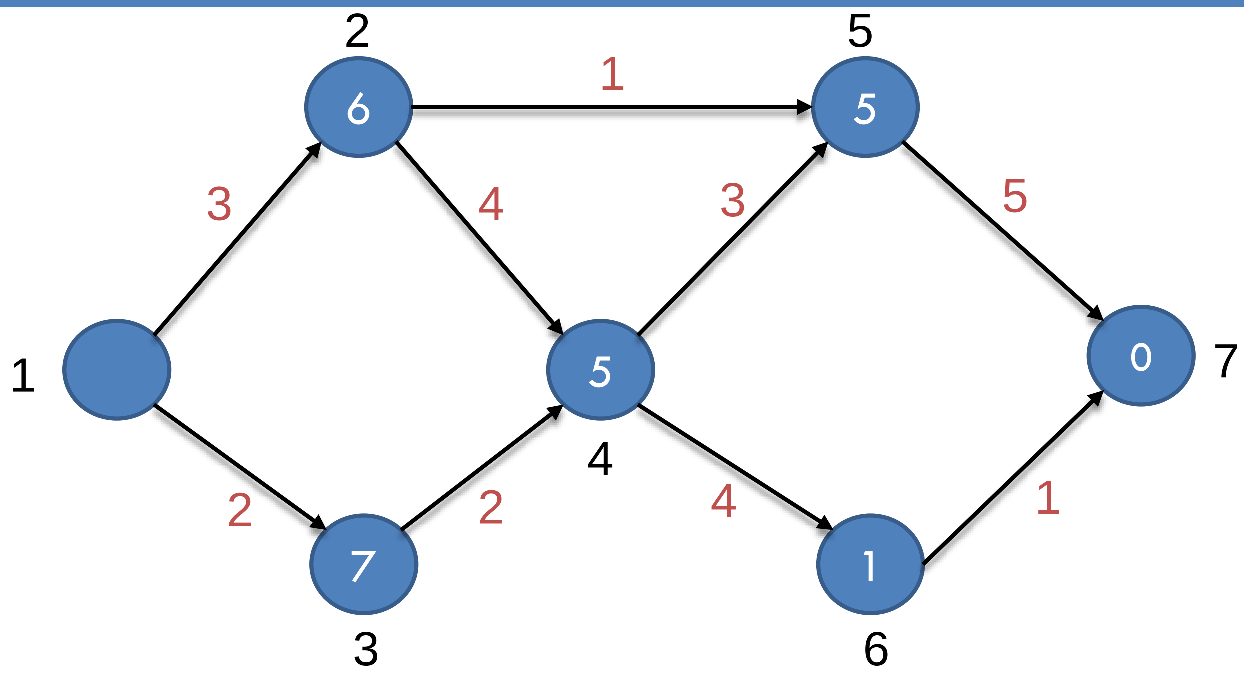
Dynamic Programming Approach



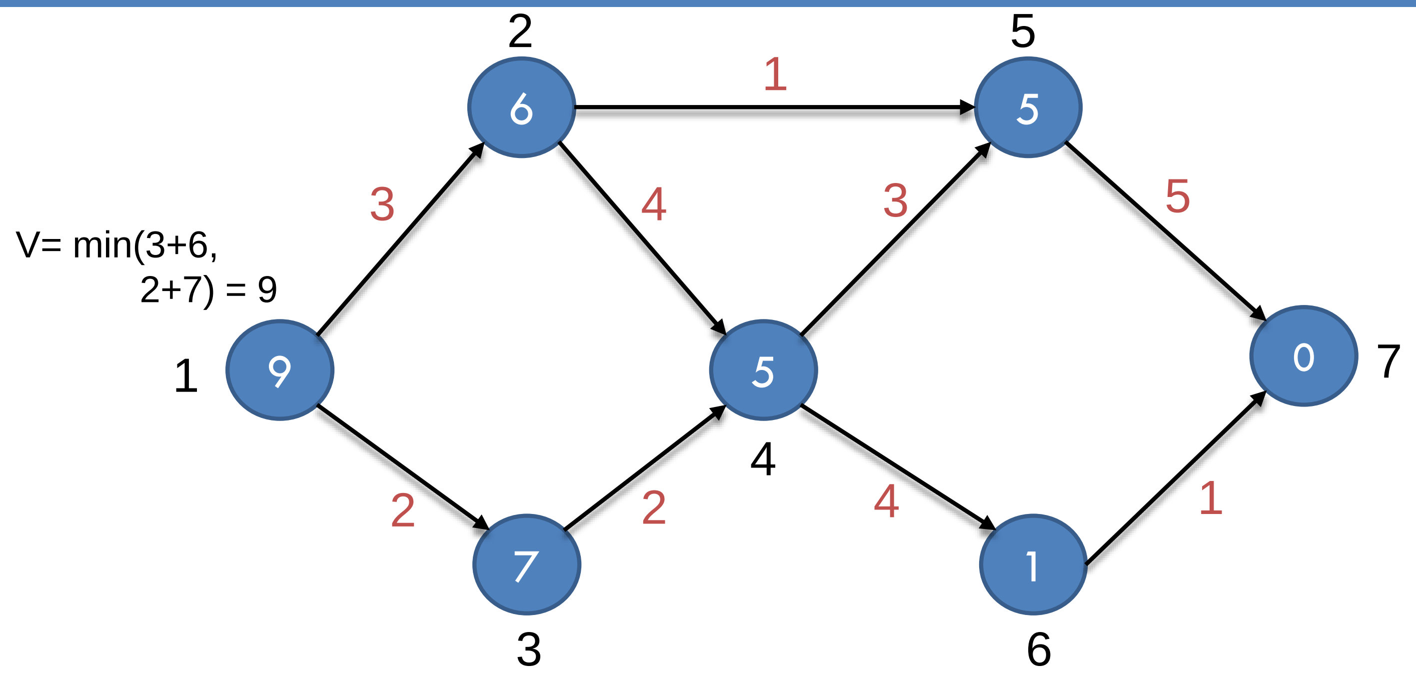
Dynamic Programming Approach



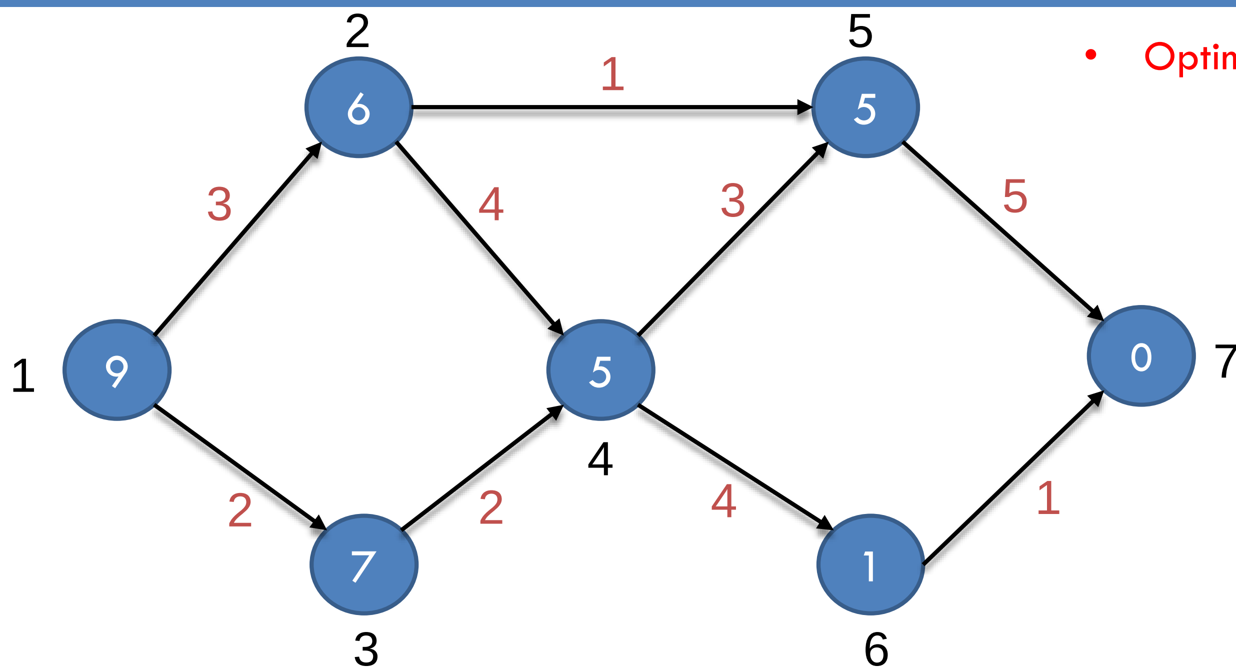
Dynamic Programming Approach



Dynamic Programming Approach



Dynamic Programming Approach



- Optimal Path Length = 9

- With a Dynamic Programming approach:
  - 2 solutions are found: (1-2-5-7) and (1-3-4-6-7)

Excel Demonstration

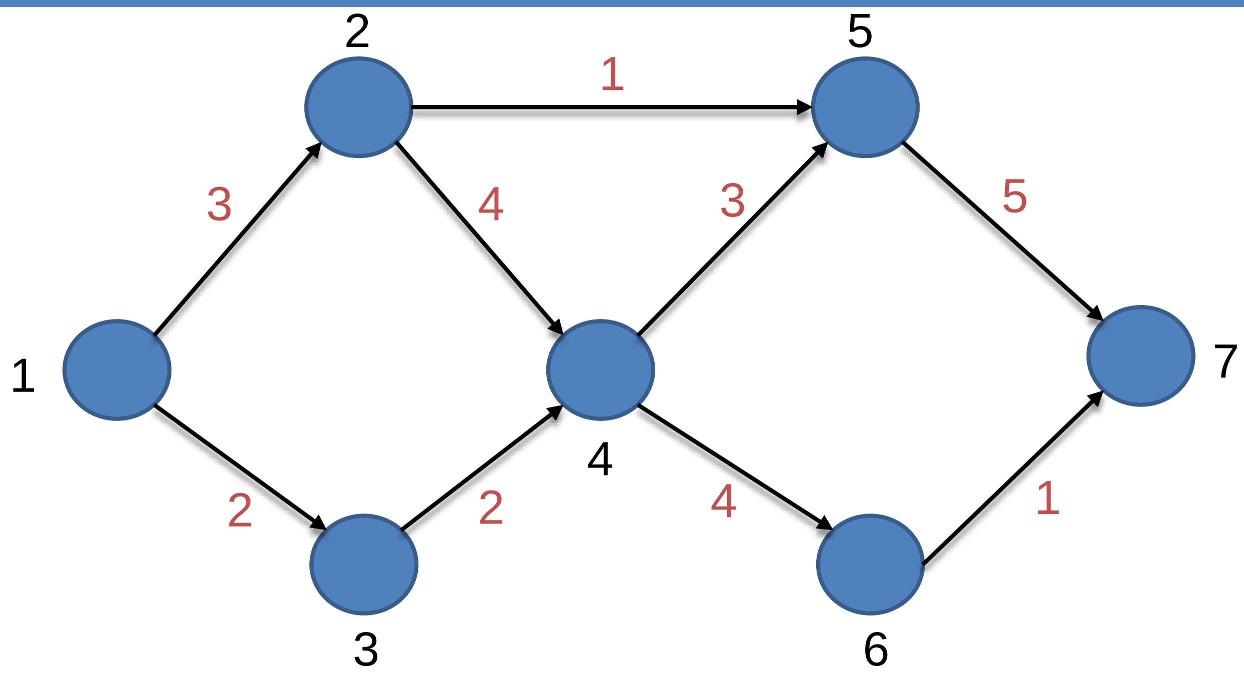
Deterministic Dynamic Programming – Finite Horizon

- We will deal with problems in which:
  - there are no cycles, and
  - the arcs are unidirectional.
- Real-world problems are sequential decision making problems over time, which is unidirectional.
  - Examples: Google Maps for directions, Inventory control, Equipment replacement, and so on.
- Recursion definition for minimization problems:
  - Value of a state i at stage t = minimum (cost of an action in state i at stage t which takes you to stage t+1 and the value of being in state j at stage t+1).

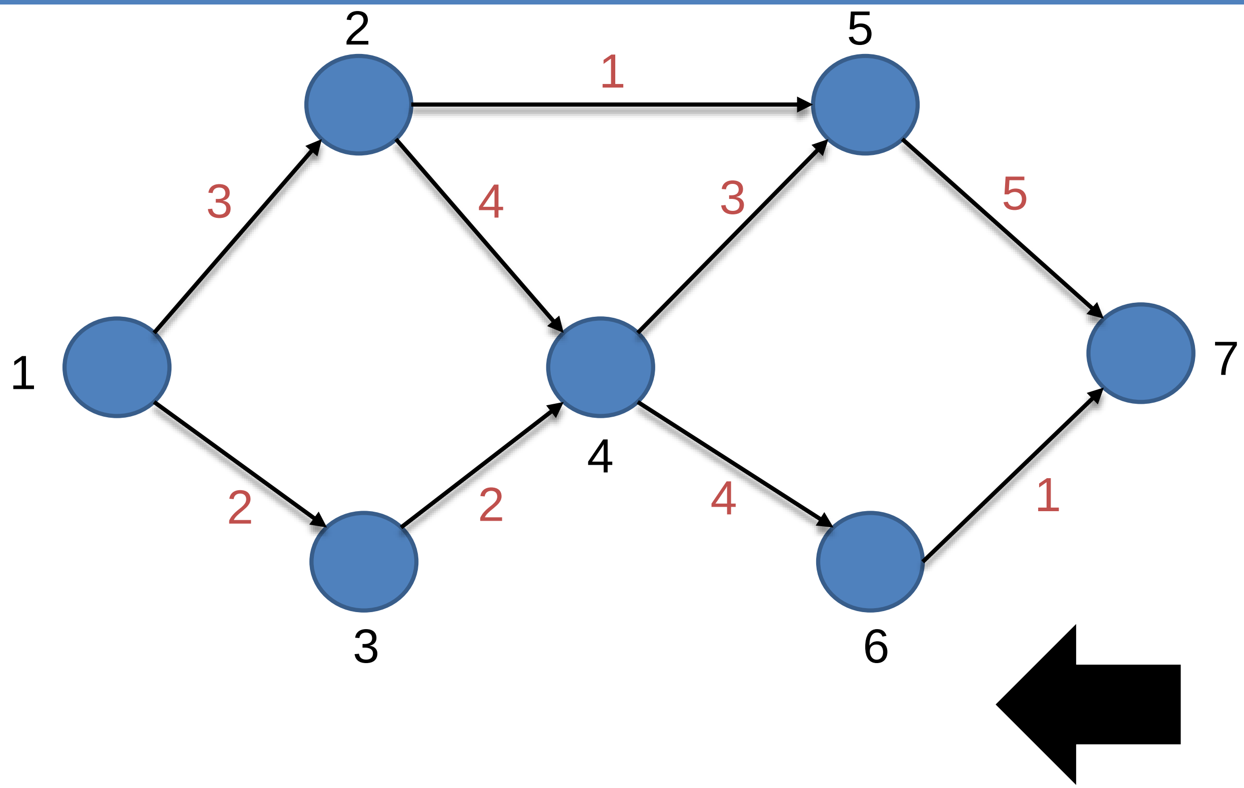
$$f_t(i) = \min_j [c_{ij} + f_{t+1}(j)], \quad i < j$$

Longest Path Example using Backward Recursion

Find Longest Path from Node 1 to 7



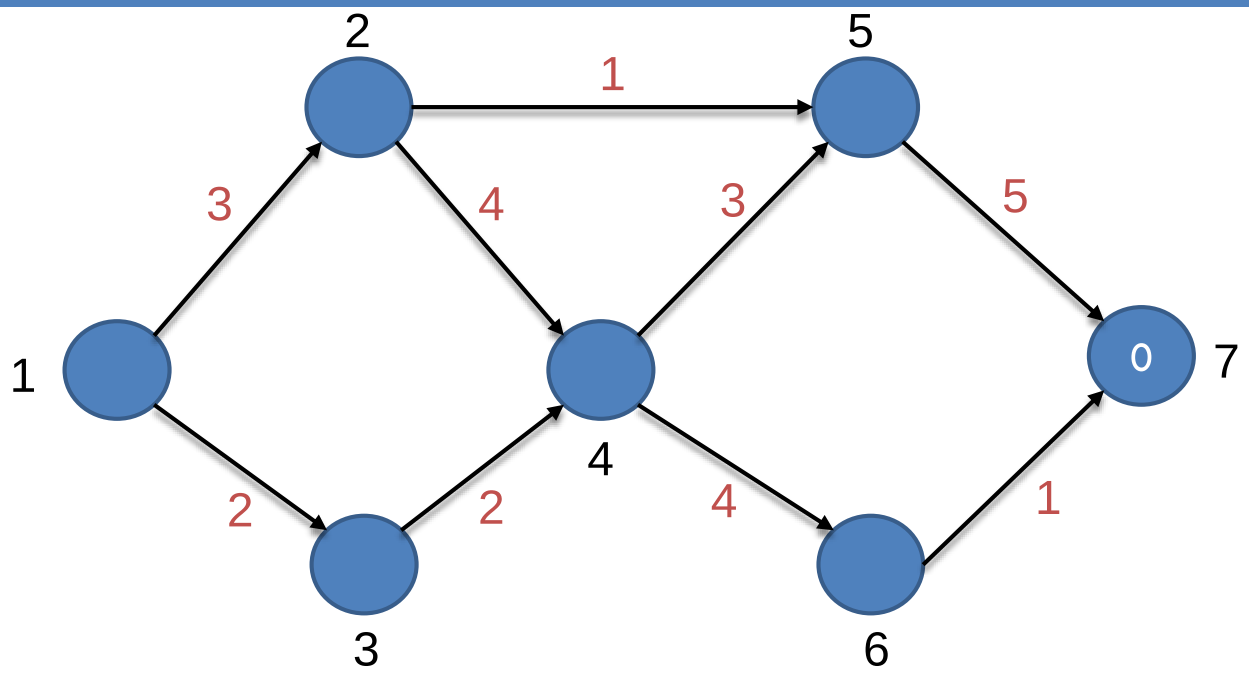
Find Longest Path from Node 1 to 7



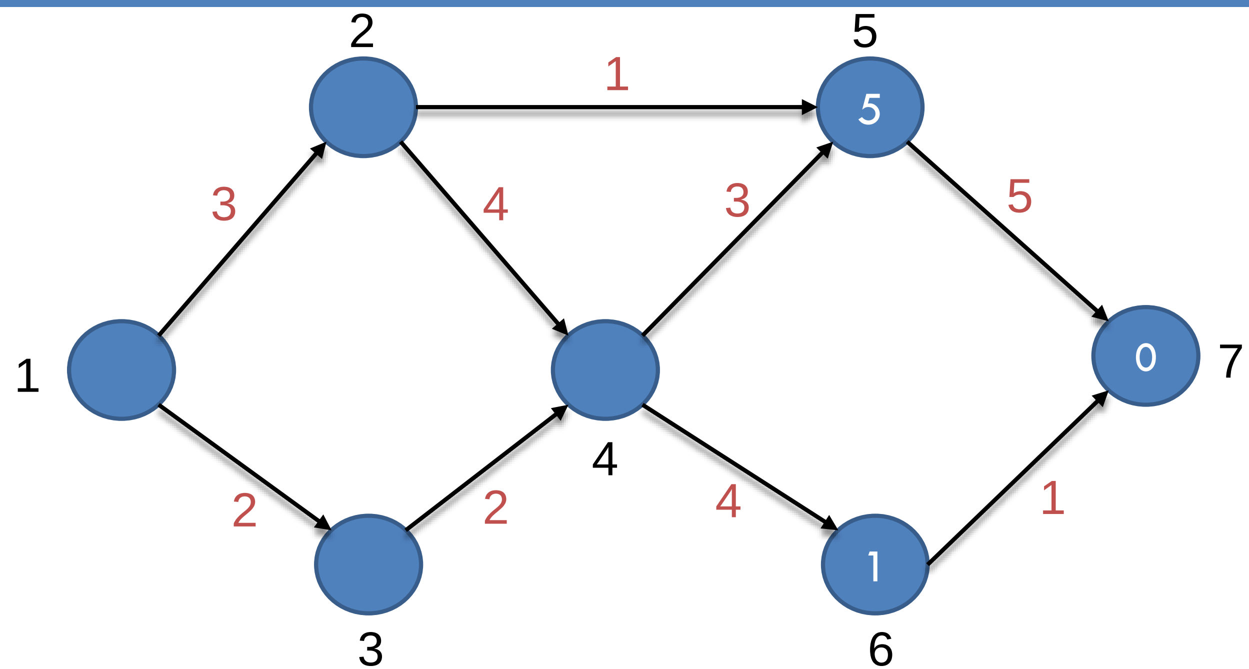
$$f_i = \max_j [c_{ij} + f_j],\ i < j$$

Backward Recursion

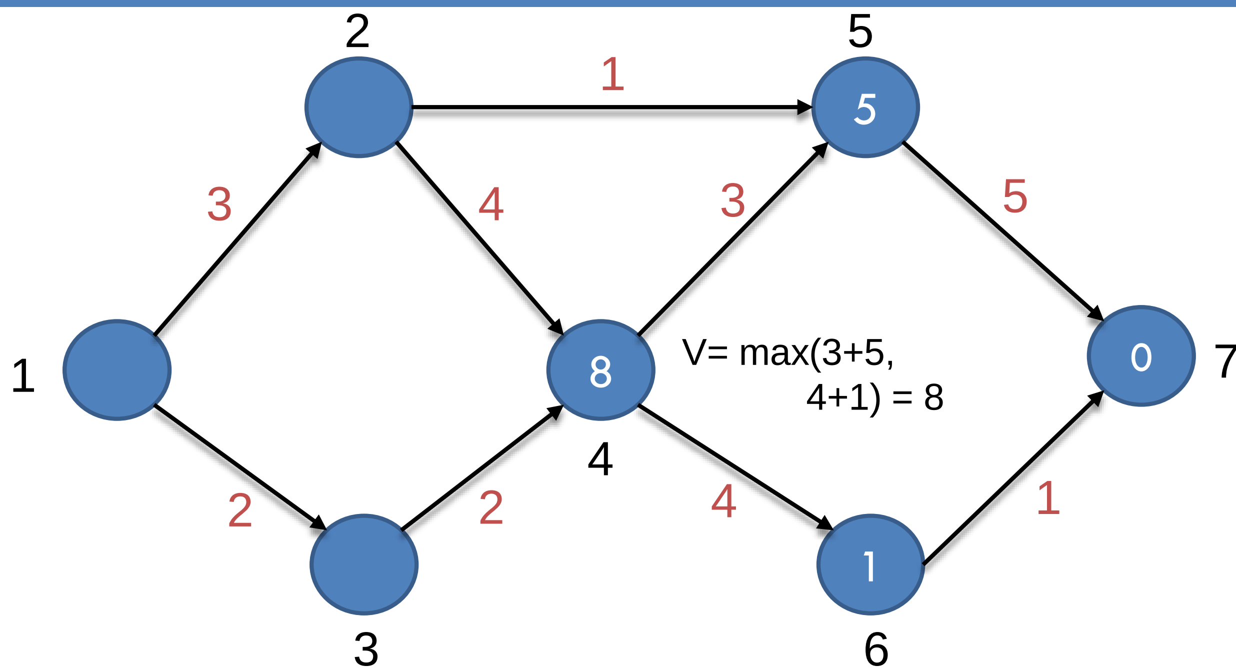
Find Longest Path from Node 1 to 7



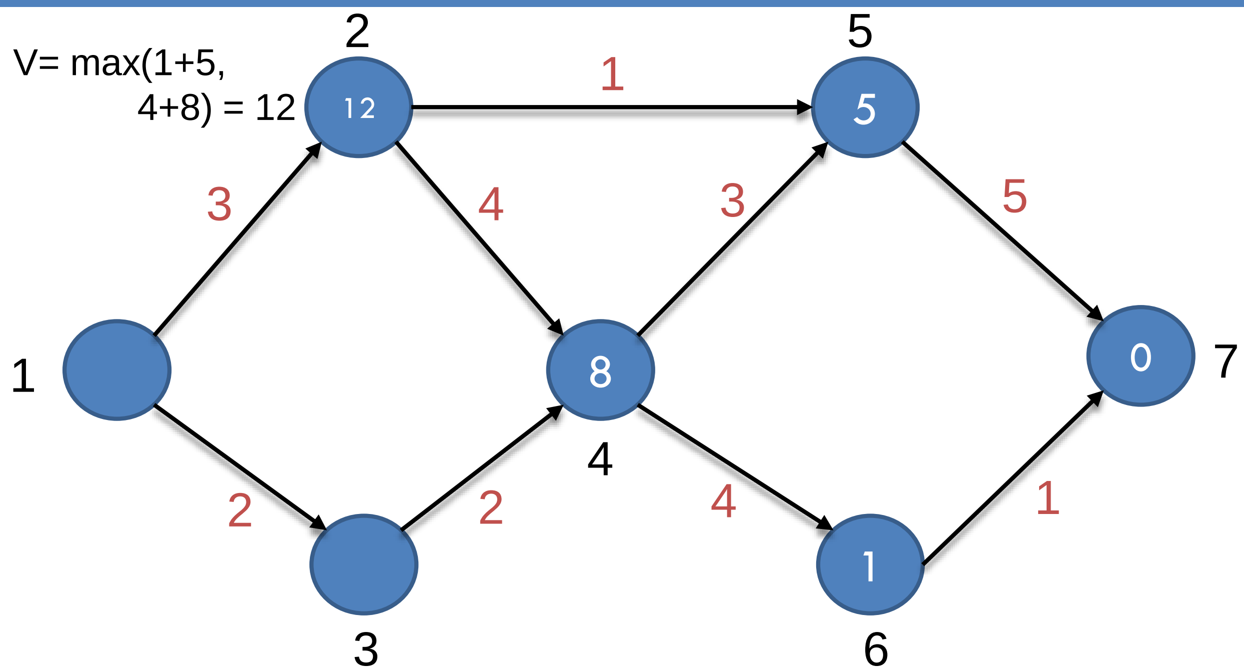
Find Longest Path from Node 1 to 7



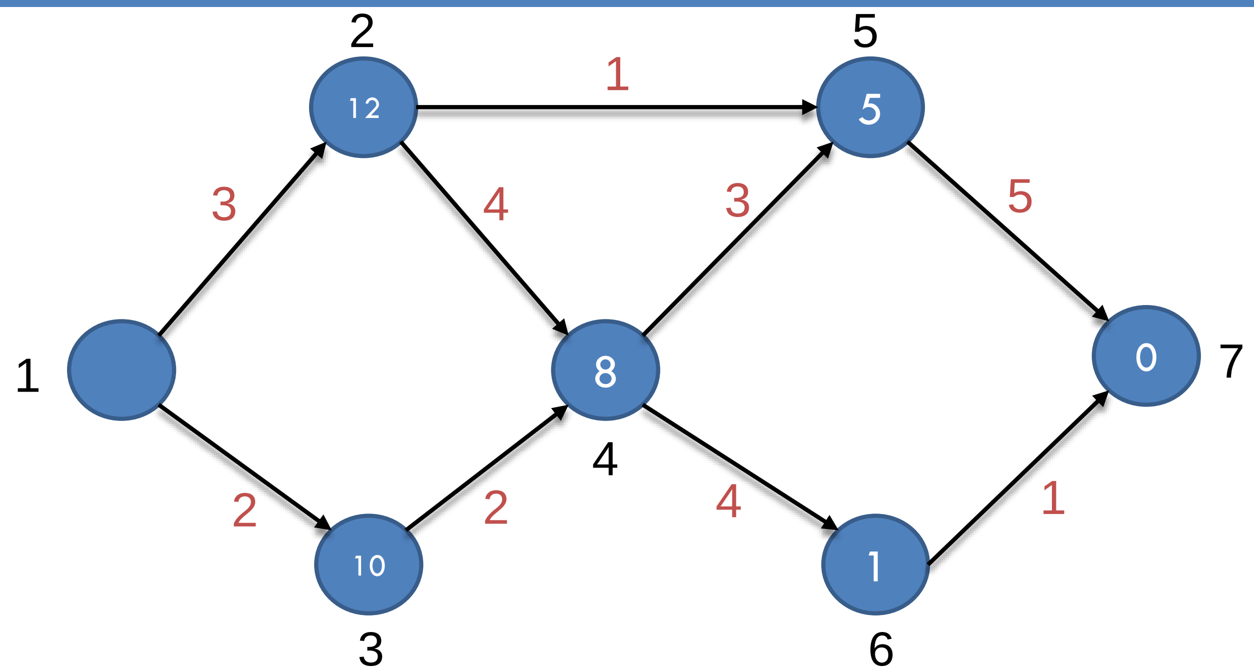
Find Longest Path from Node 1 to 7



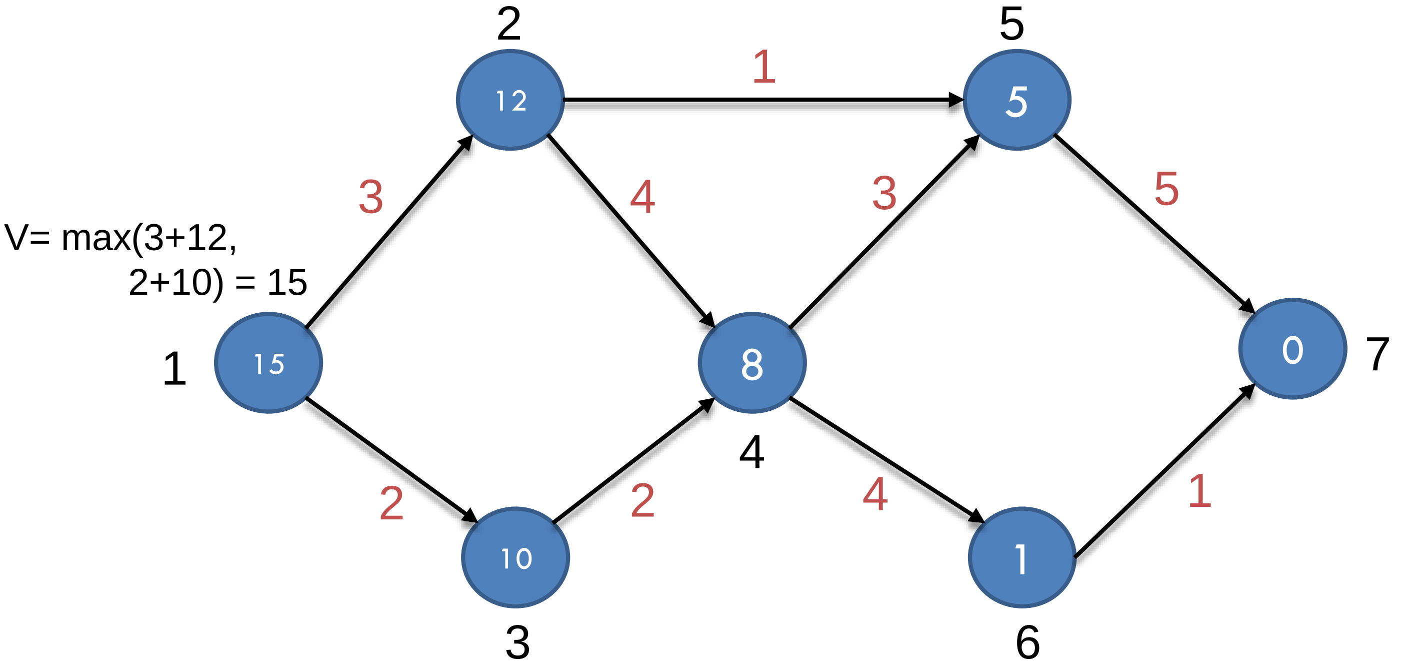
Find Longest Path from Node 1 to 7



Find Longest Path from Node 1 to 7

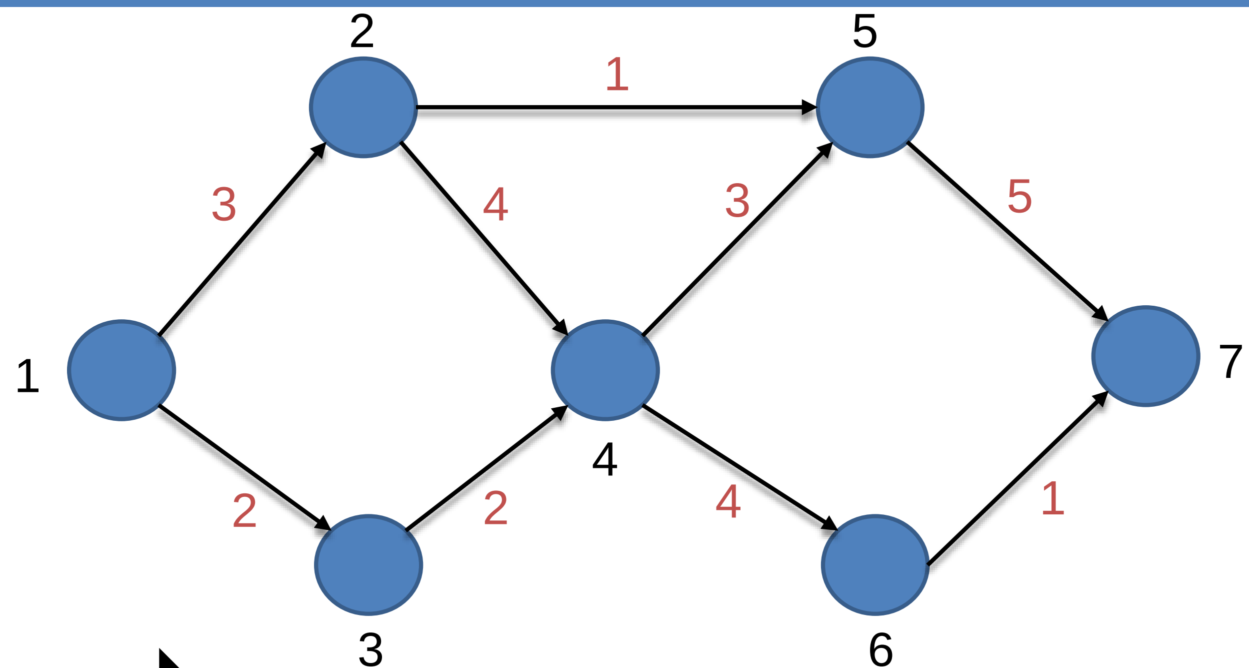


Find Longest Path from Node 1 to 7



Longest Path Example using Forward Reaching

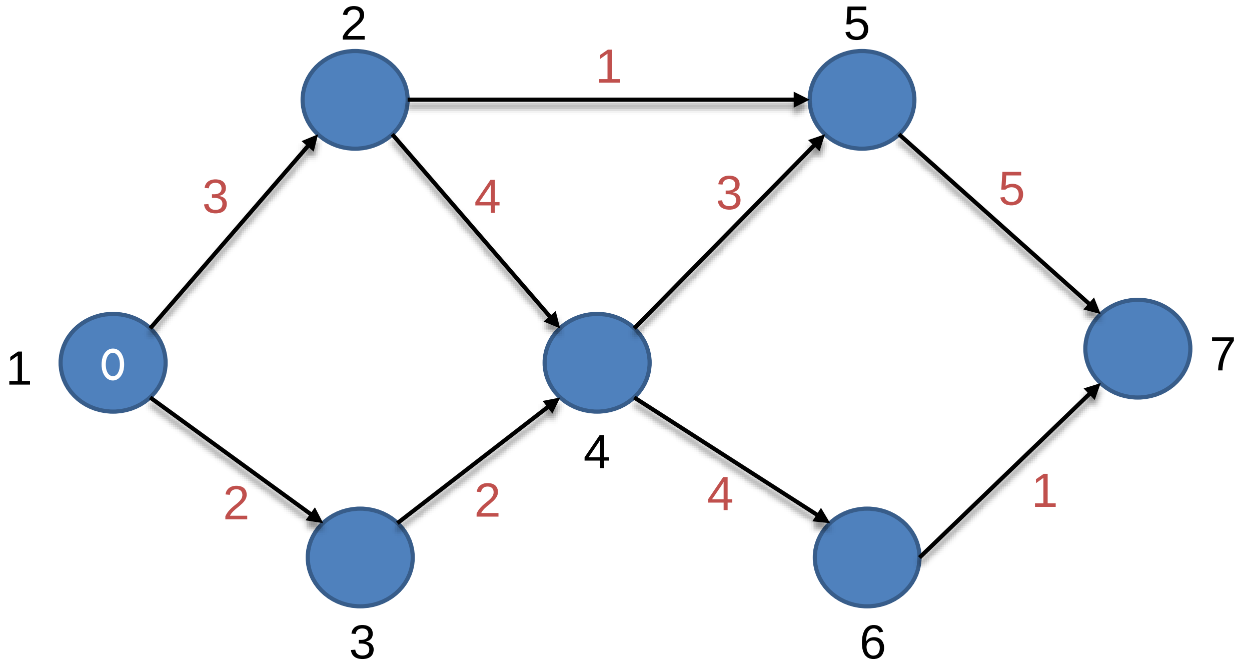
Find Longest Path from Node 1 to 7



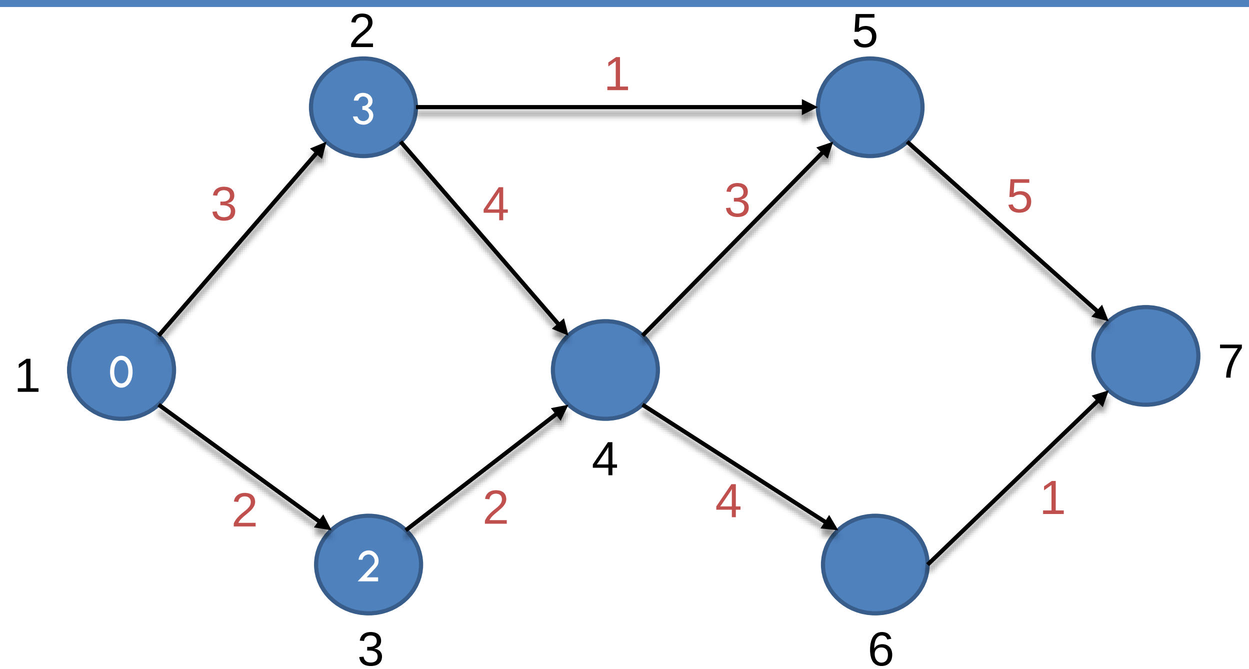
Forward
Reaching

$$f_j = \max_i [c_{ij} + f_i], \; i < j$$

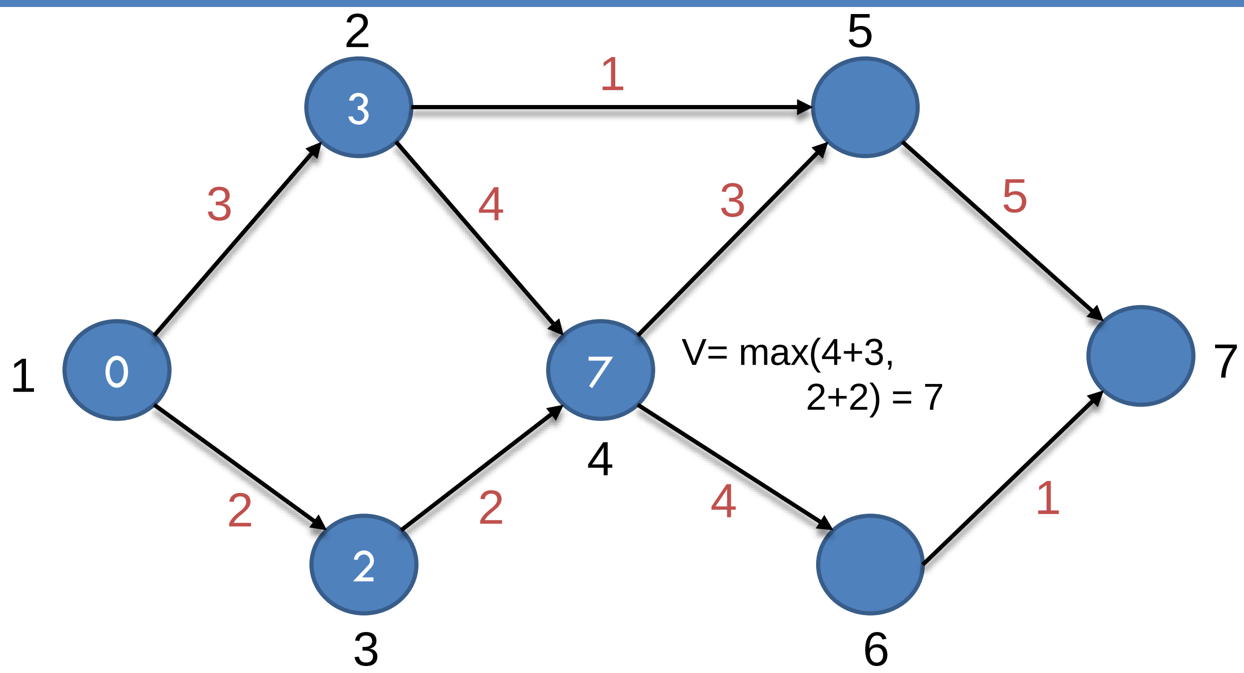
Find Longest Path from Node 1 to 7



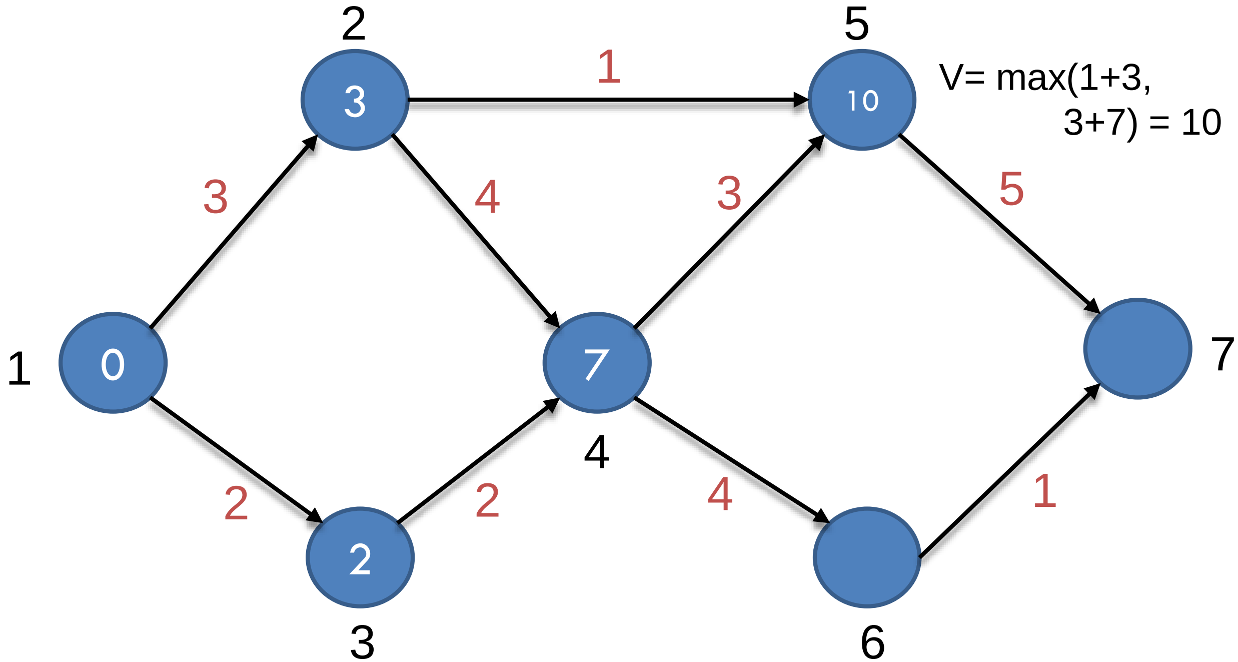
Find Longest Path from Node 1 to 7



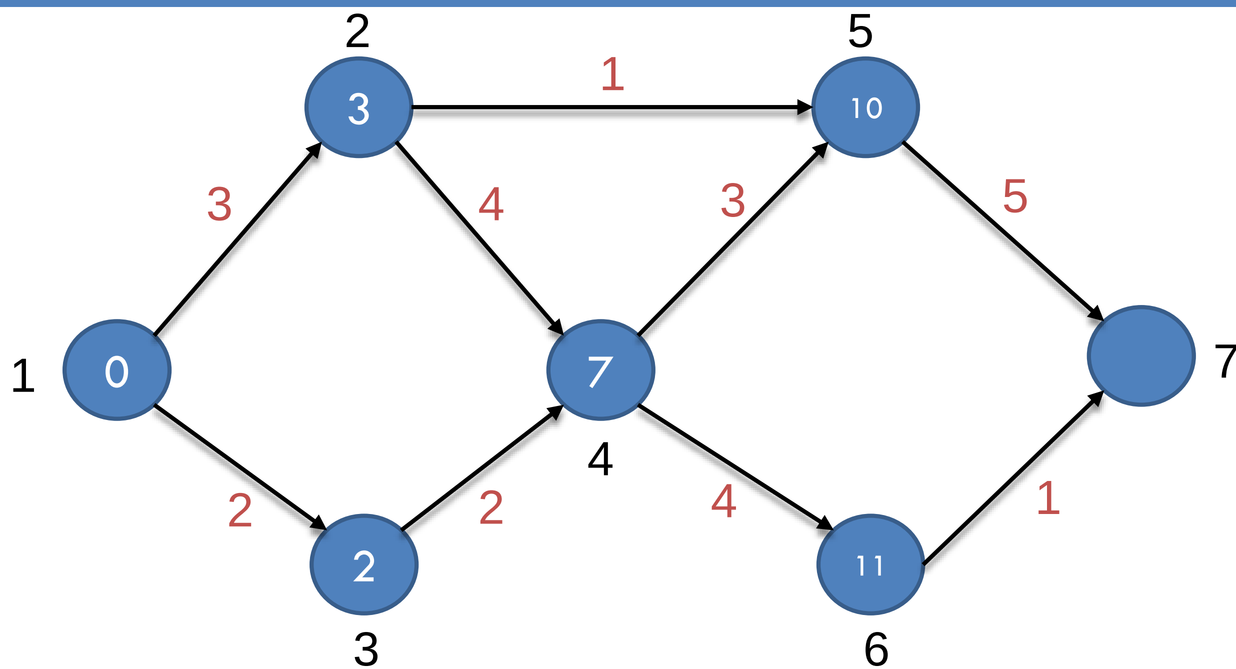
Find Longest Path from Node 1 to 7



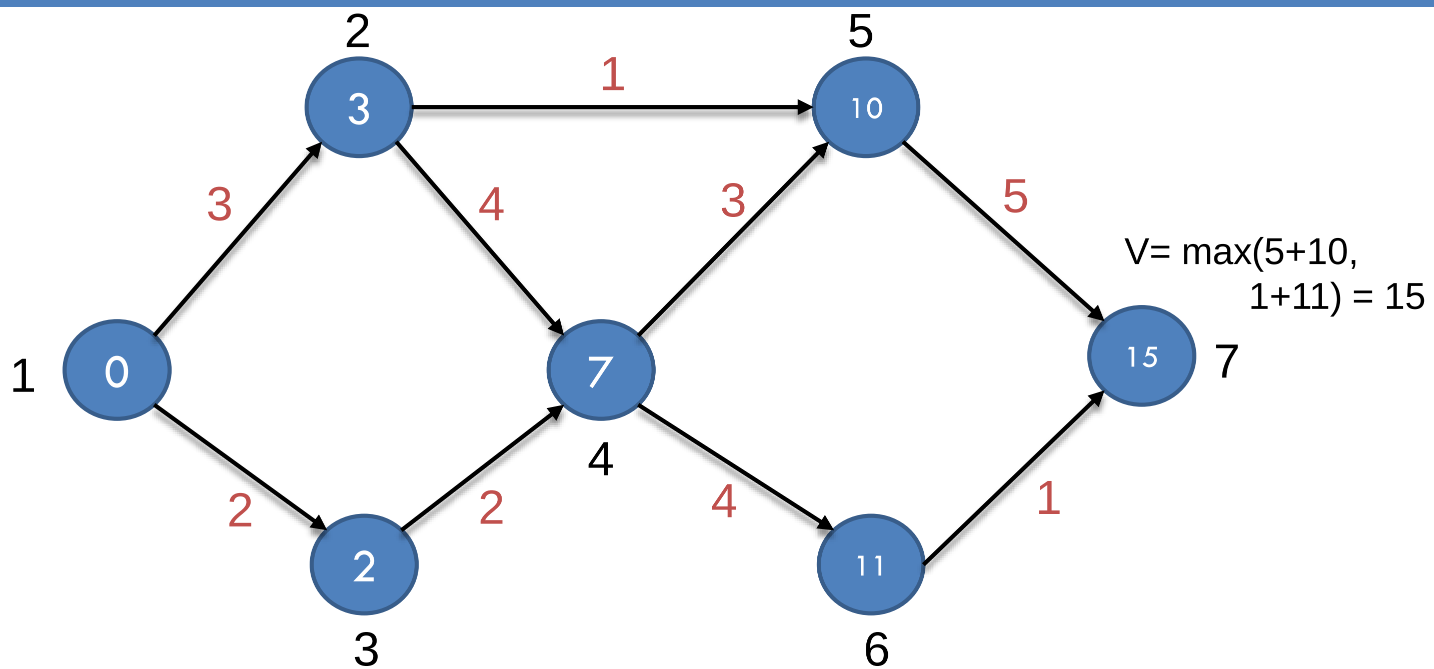
Find Longest Path from Node 1 to 7



Find Longest Path from Node 1 to 7



Find Longest Path from Node 1 to 7



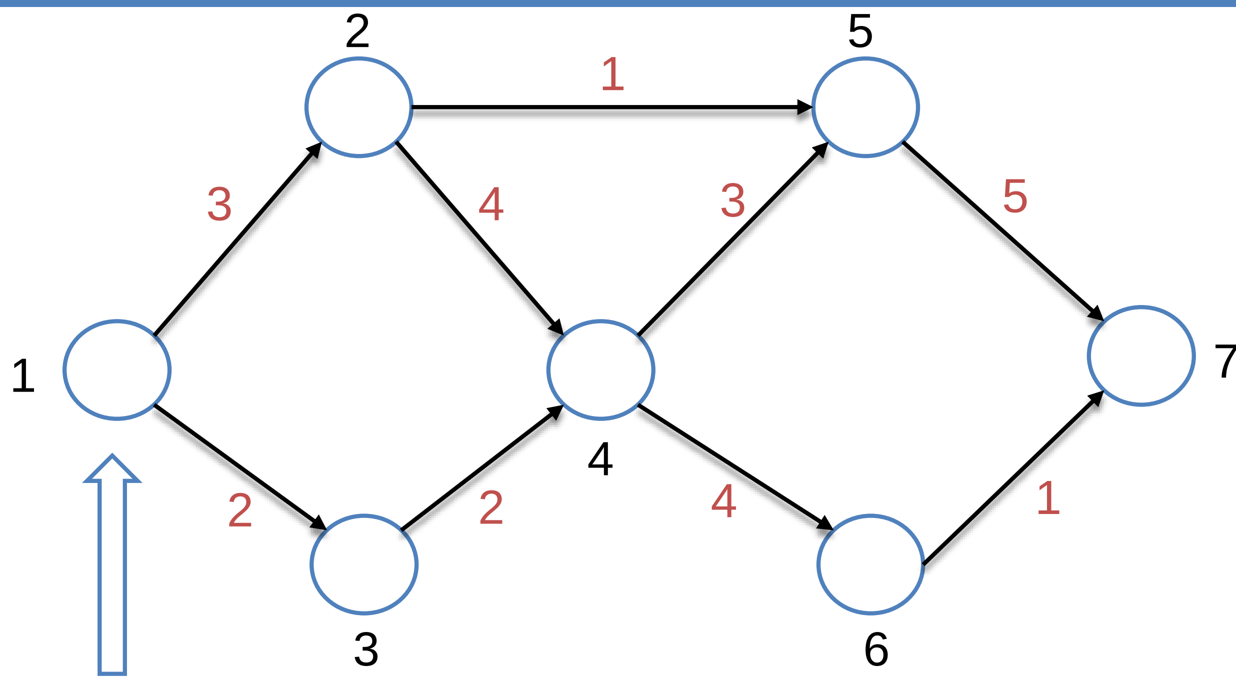
Summary

- A deterministic finite horizon problem can be solved backwards or by going forward.
  - For all problems, we will follow the backward recursion formula.
- Begin solving by setting the last state value in the last stage to zero and work backwards till the first state in the first stage.
- If there is more than one solution to the max or min operator at any state then there are multiple optimal paths.
- At any single iteration, the calculations of the value function is only between the current state t and the future states at $t + 1$.
  - This is a computational advantage as the algorithm performs only a few calculations at t even if the problem has millions of states that may occur at different times.
- The value of the first state in first stage is the solution of the problem.

Excel Demonstration

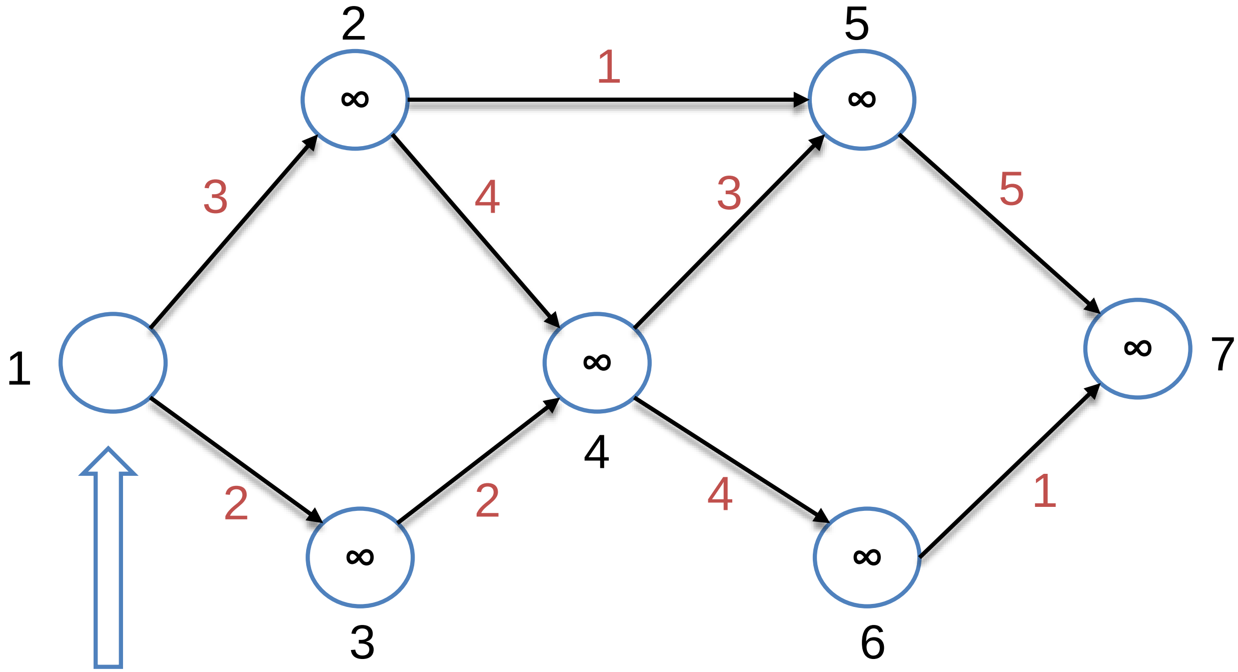
Dijkstra's Algorithm

Find Shortest Path from Node 1 to 7



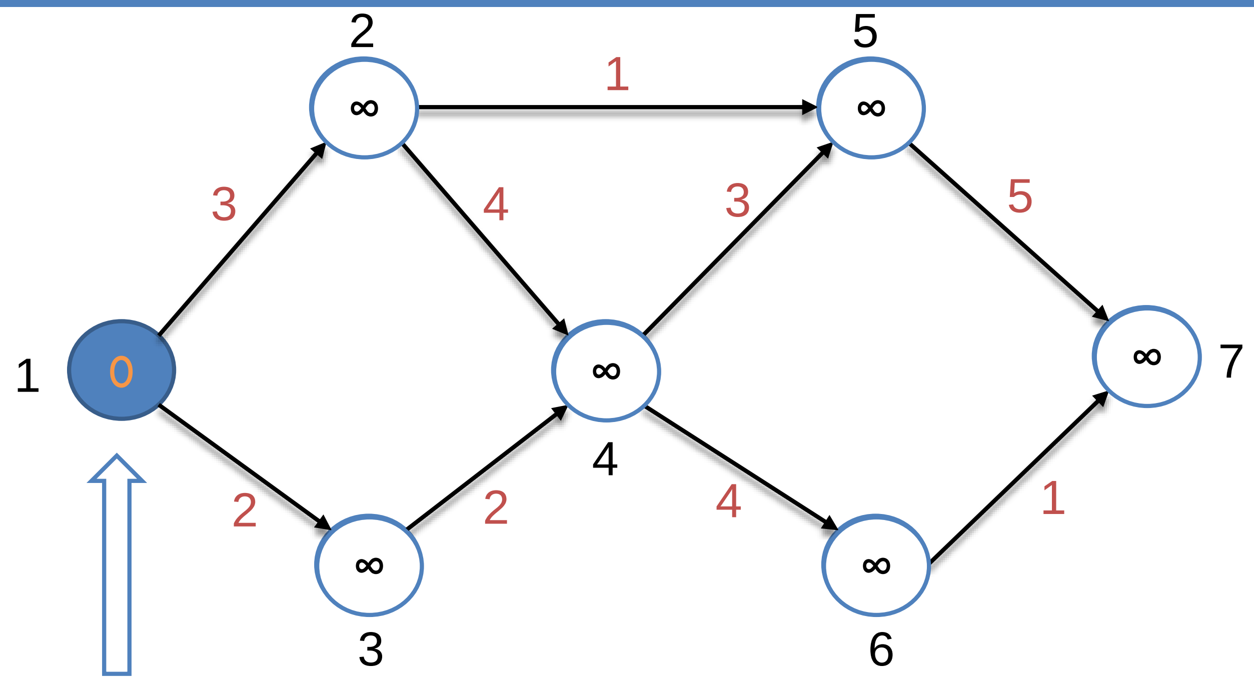
- Start Here

Find Shortest Path from Node 1 to 7



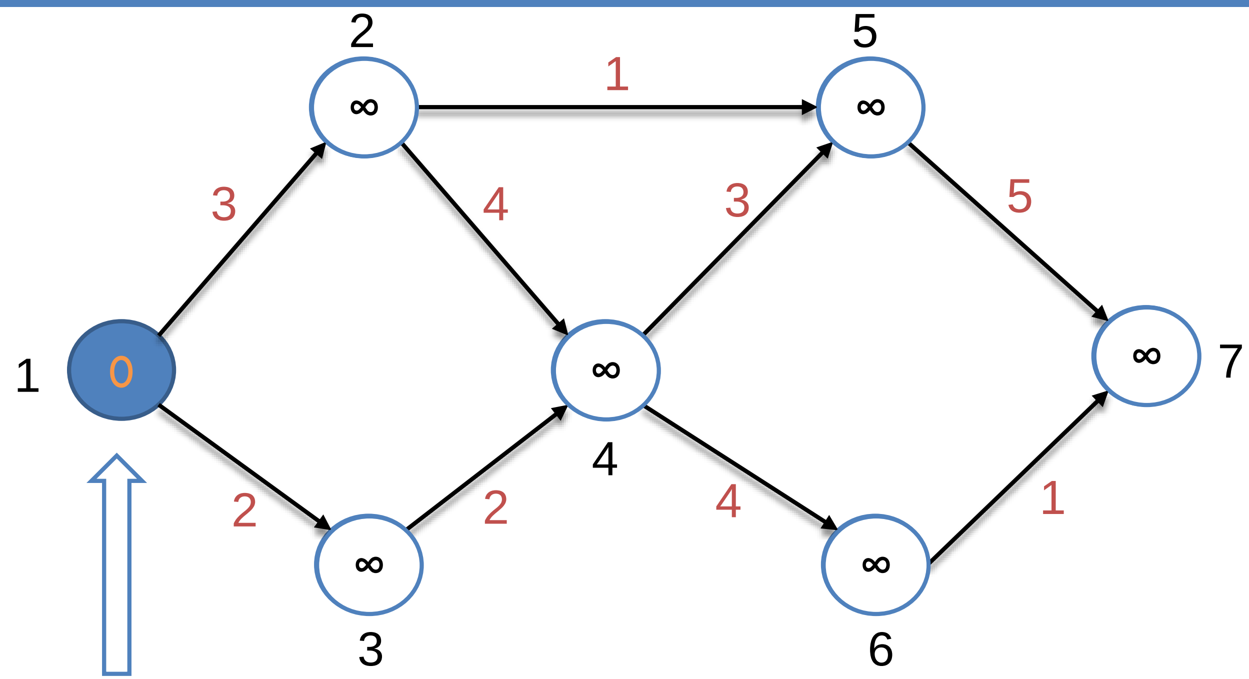
- Start Here

Find Shortest Path from Node 1 to 7



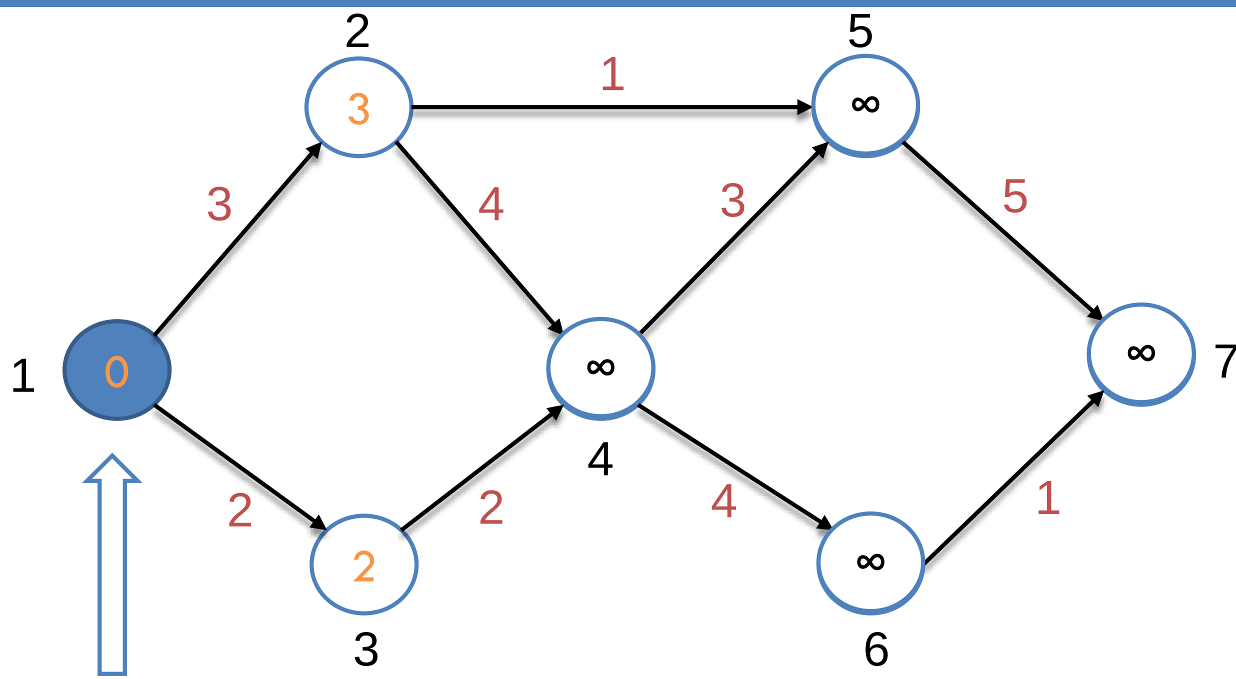
- Start Here
- Add node 1 to the visited node list

Find Shortest Path from Node 1 to 7



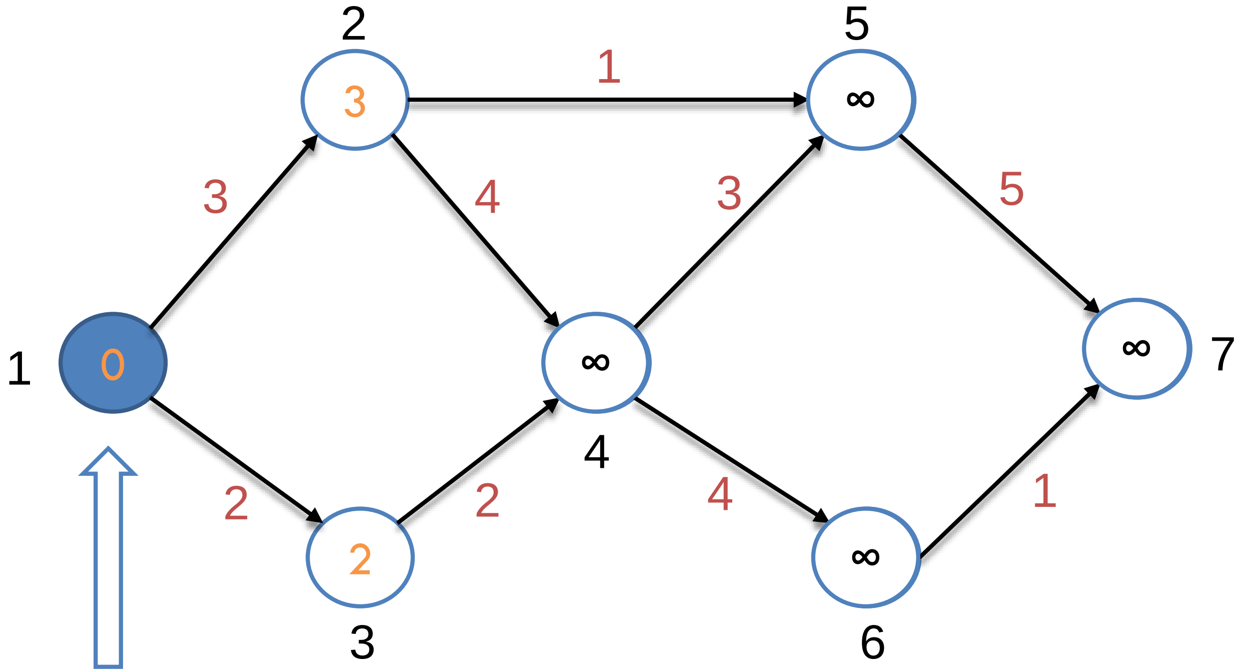
- Start Here
- Add node 1 to the visited node list
- Explore the neighborhood

Find Shortest Path from Node 1 to 7



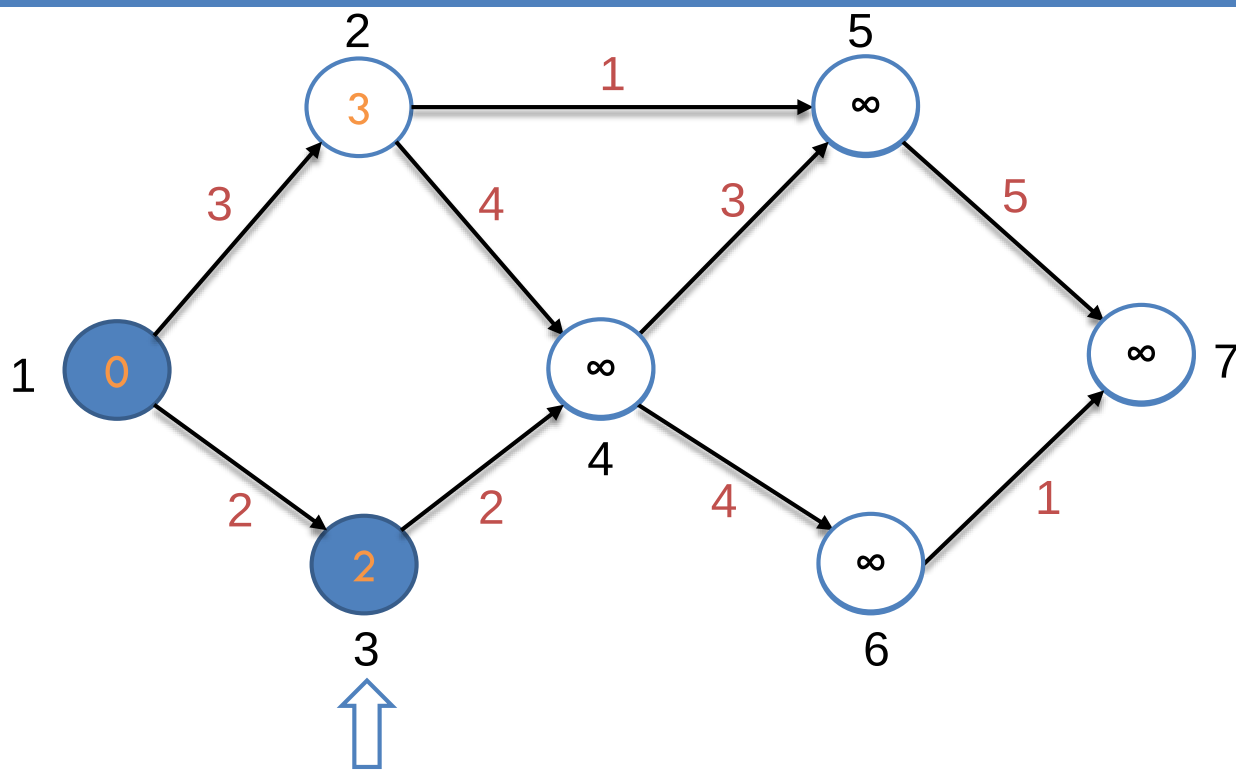
- Start Here
- Add node 1 to the visited node list
- Explore the neighborhood

Find Shortest Path from Node 1 to 7



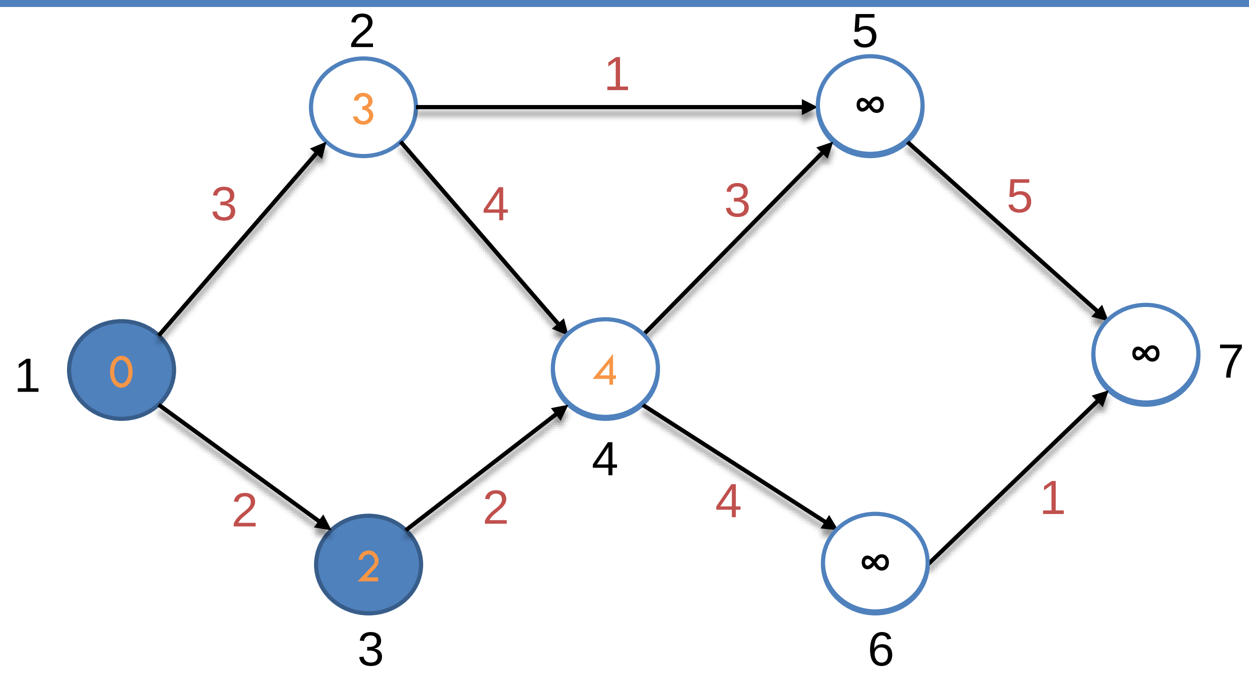
- Start Here
- Add node 1 to the visited node list
- Explore the neighborhood
- Visit the nearest neighbor (node)

Find Shortest Path from Node 1 to 7

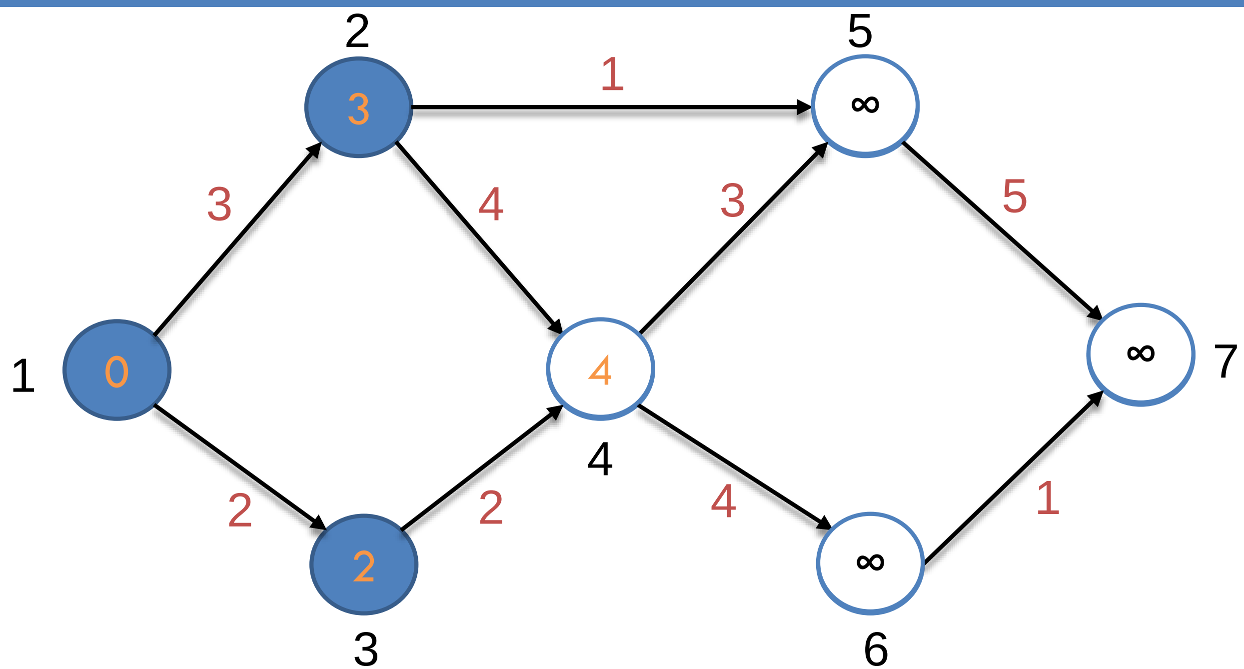


- Add node 3 to the visited node list
- Explore the neighborhood
- Visit the nearest neighbor (node)

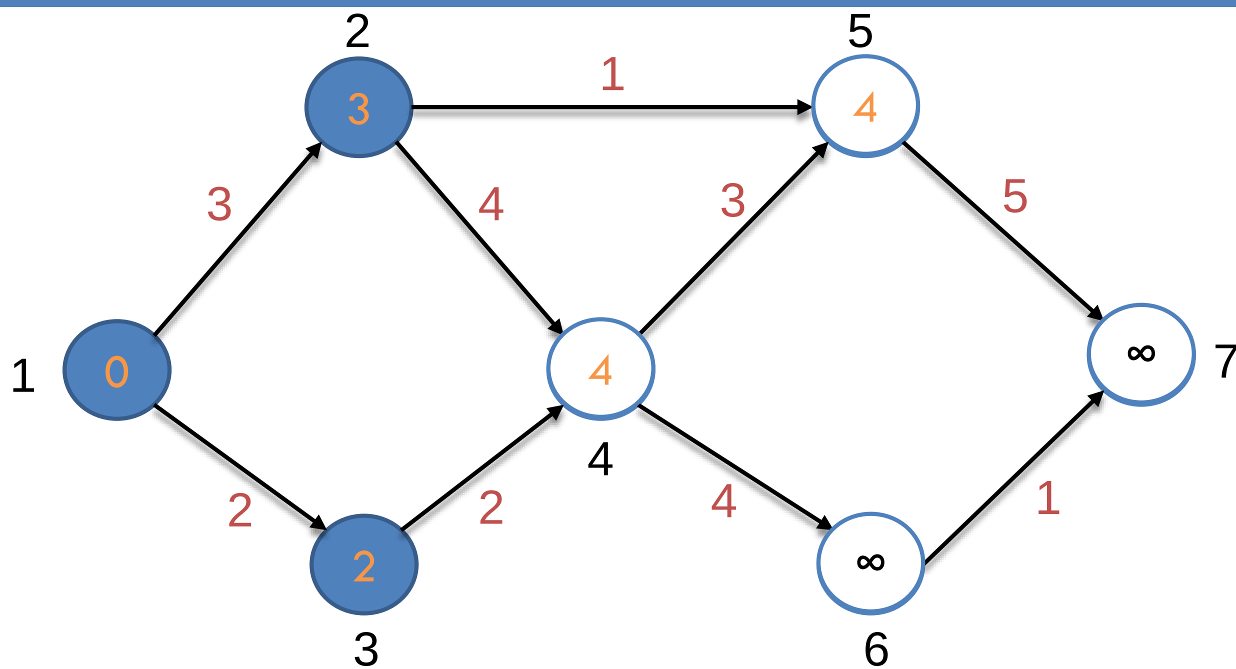
Find Shortest Path from Node 1 to 7



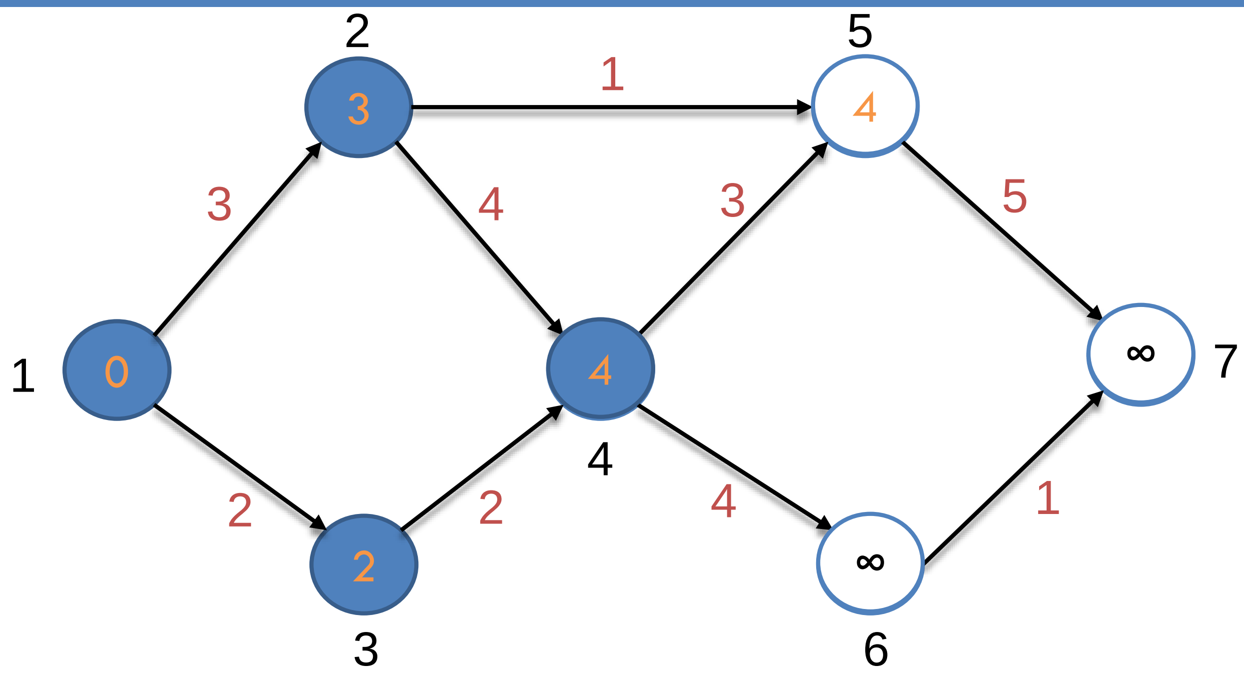
Find Shortest Path from Node 1 to 7



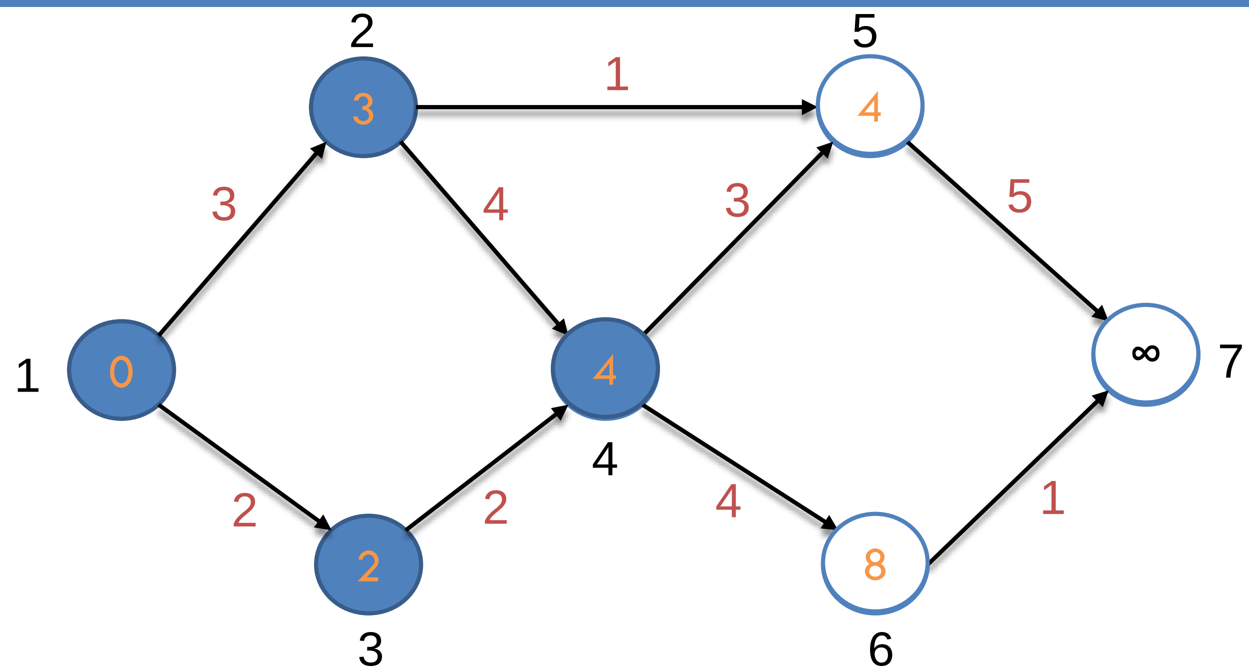
Find Shortest Path from Node 1 to 7



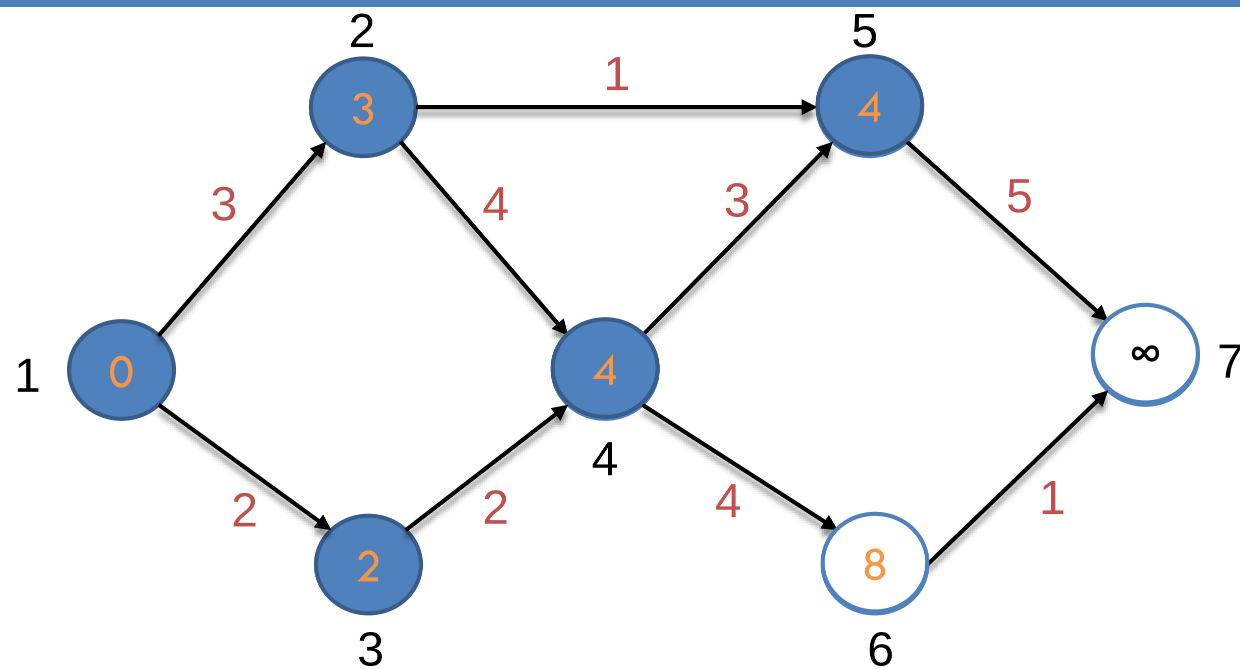
Find Shortest Path from Node 1 to 7



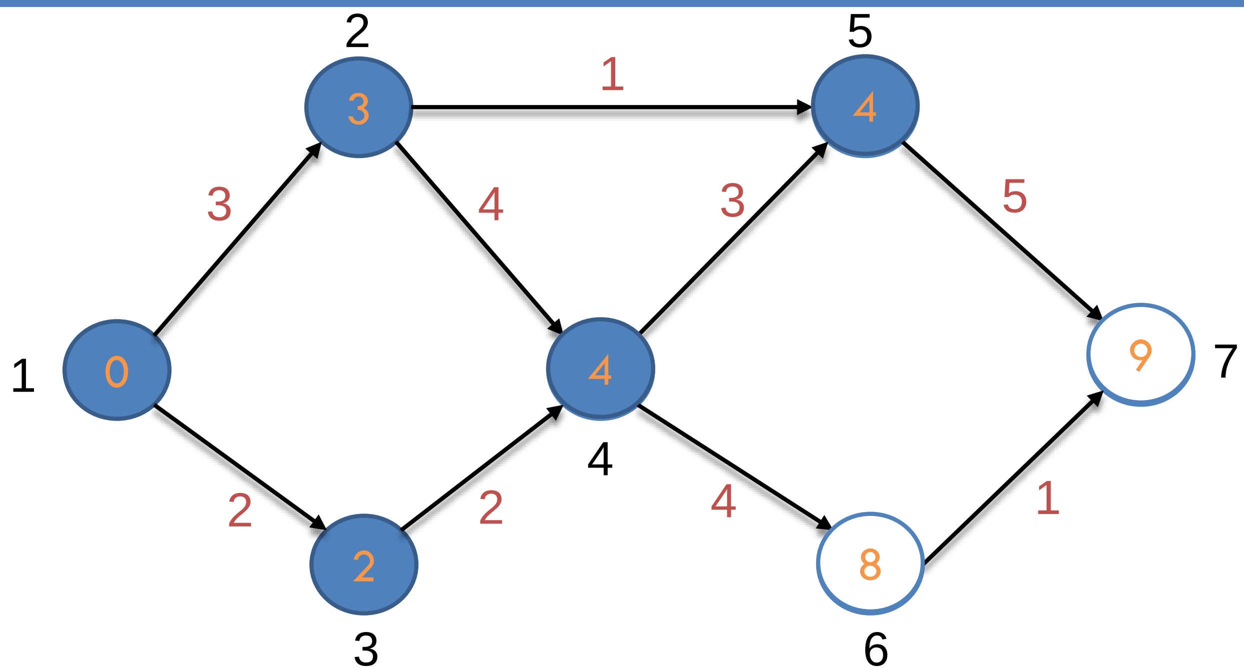
Find Shortest Path from Node 1 to 7



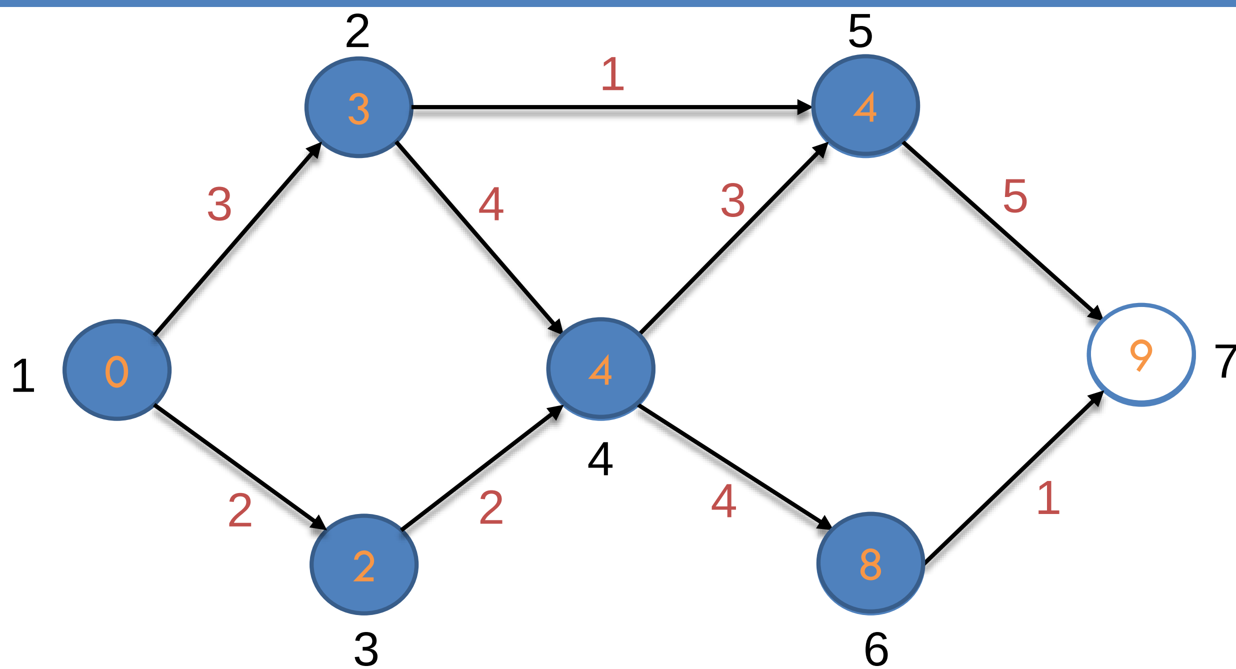
Find Shortest Path from Node 1 to 7



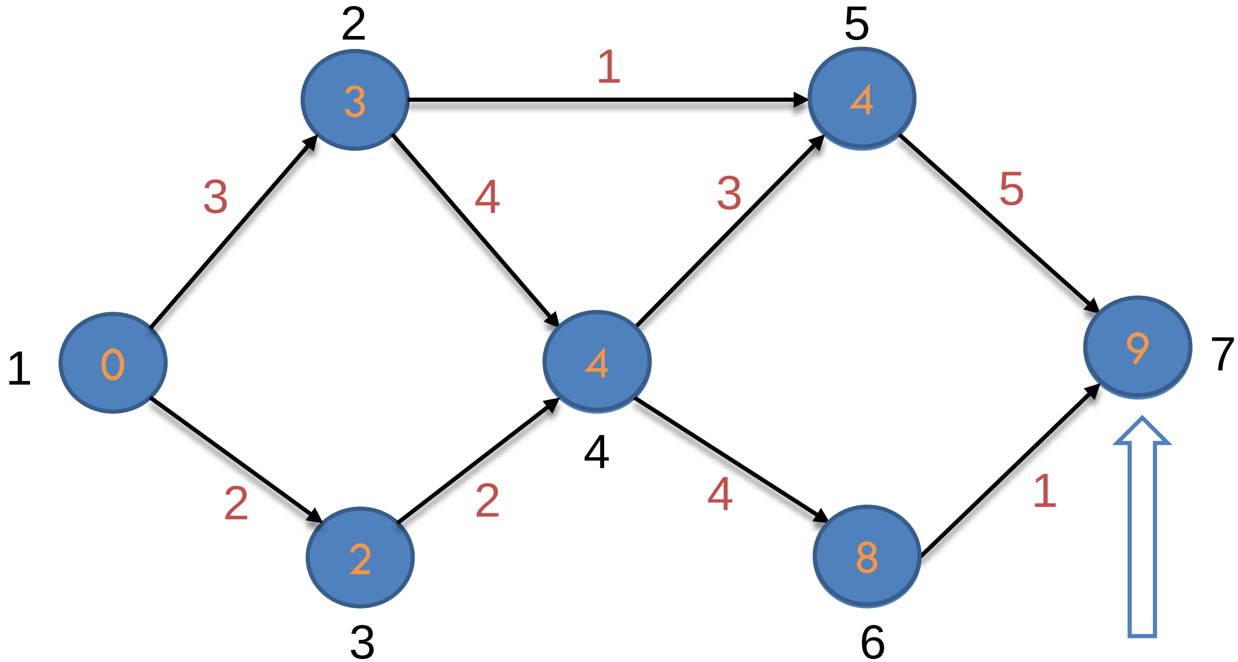
Find Shortest Path from Node 1 to 7



Find Shortest Path from Node 1 to 7



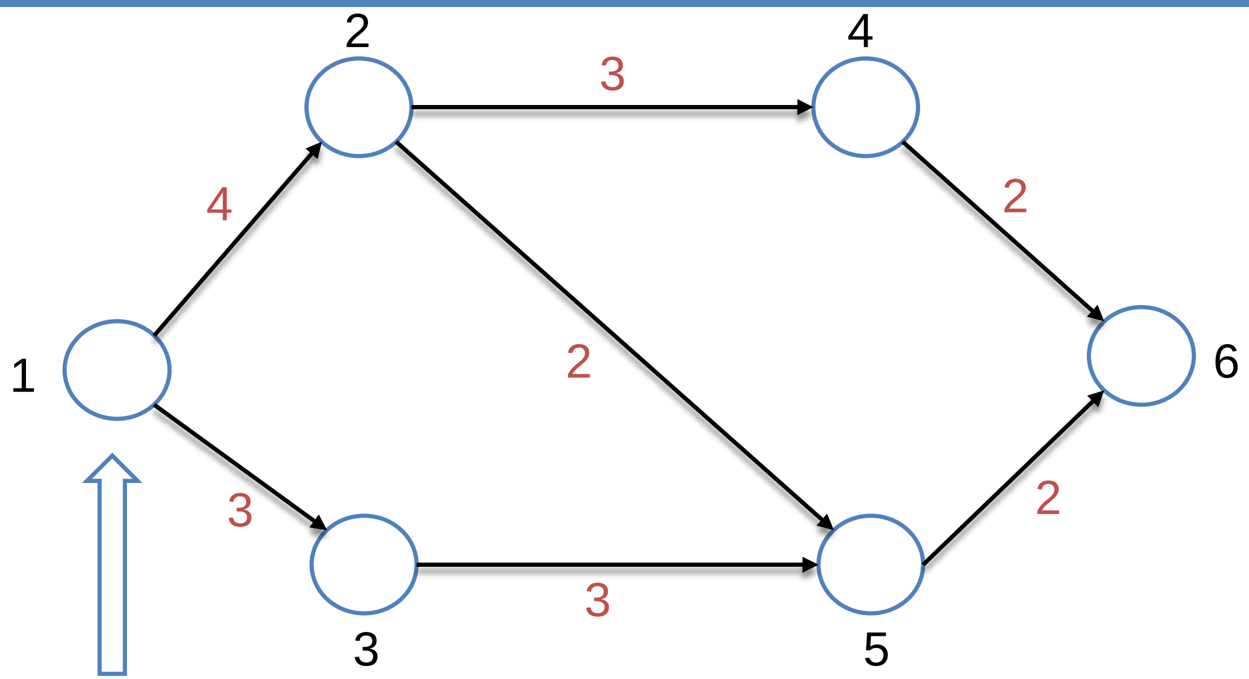
Find Shortest Path from Node 1 to 7



- Stop when all nodes are visited.

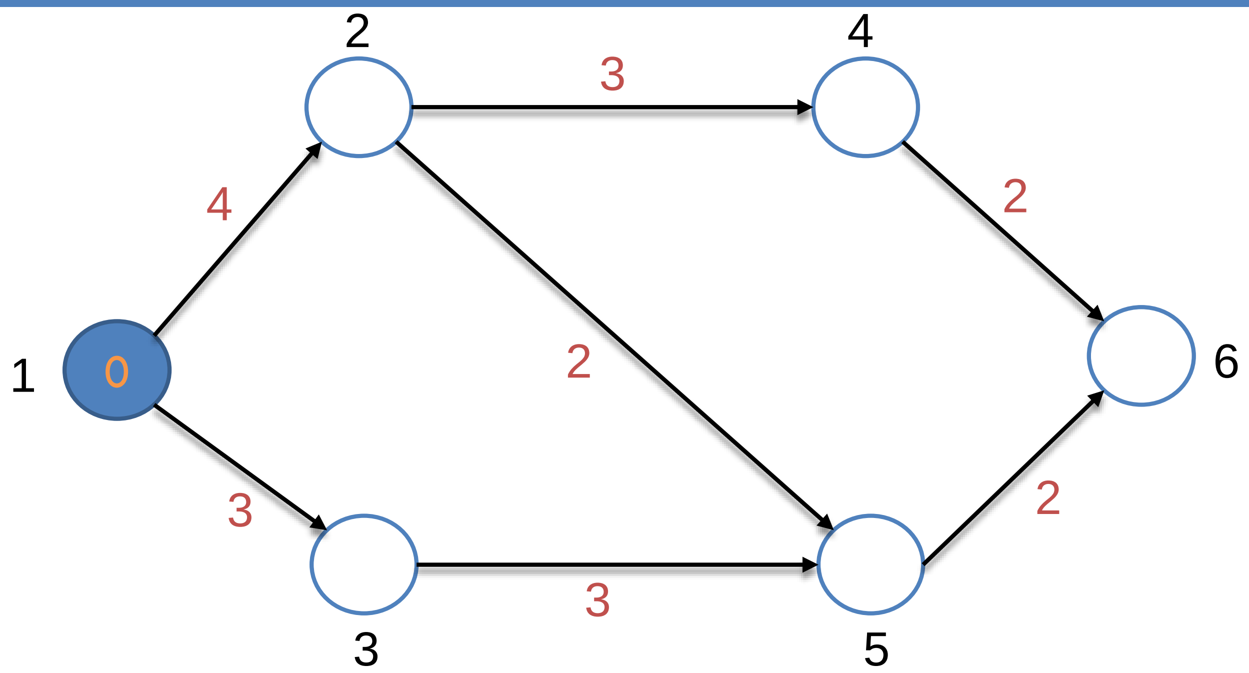
Another Example using Dijkstra's Algorithm

Find Shortest Path from Node 1 to 6

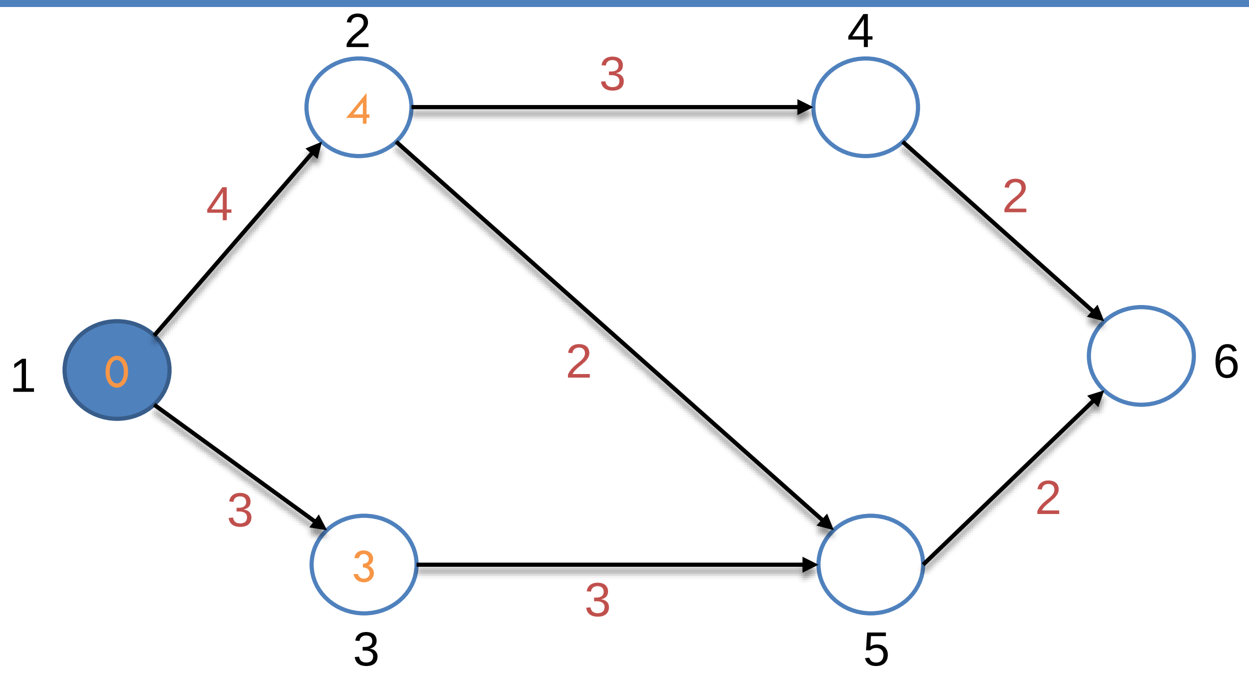


- Start Here

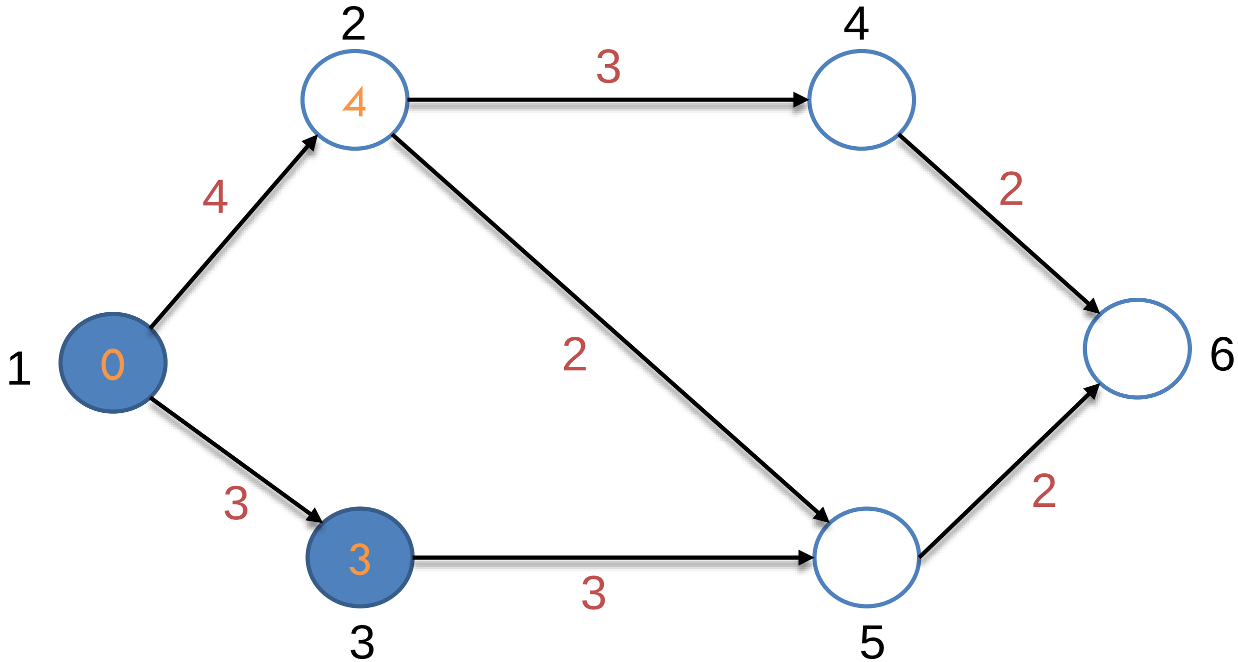
Find Shortest Path from Node 1 to 6



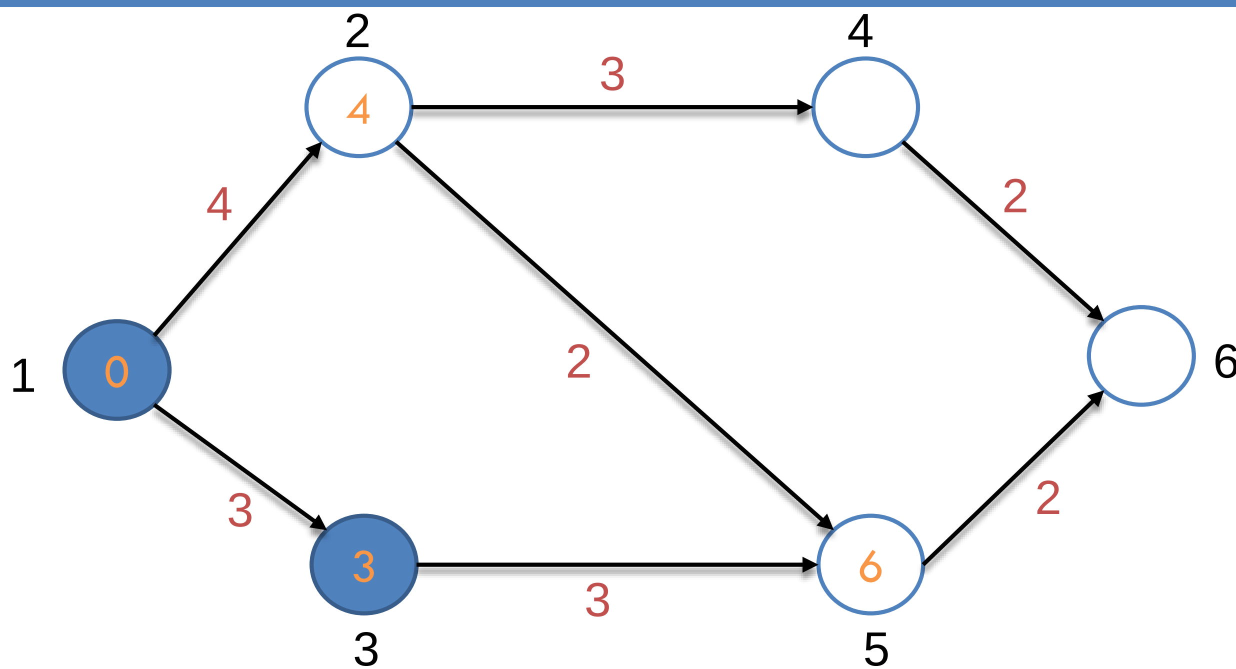
Find Shortest Path from Node 1 to 6



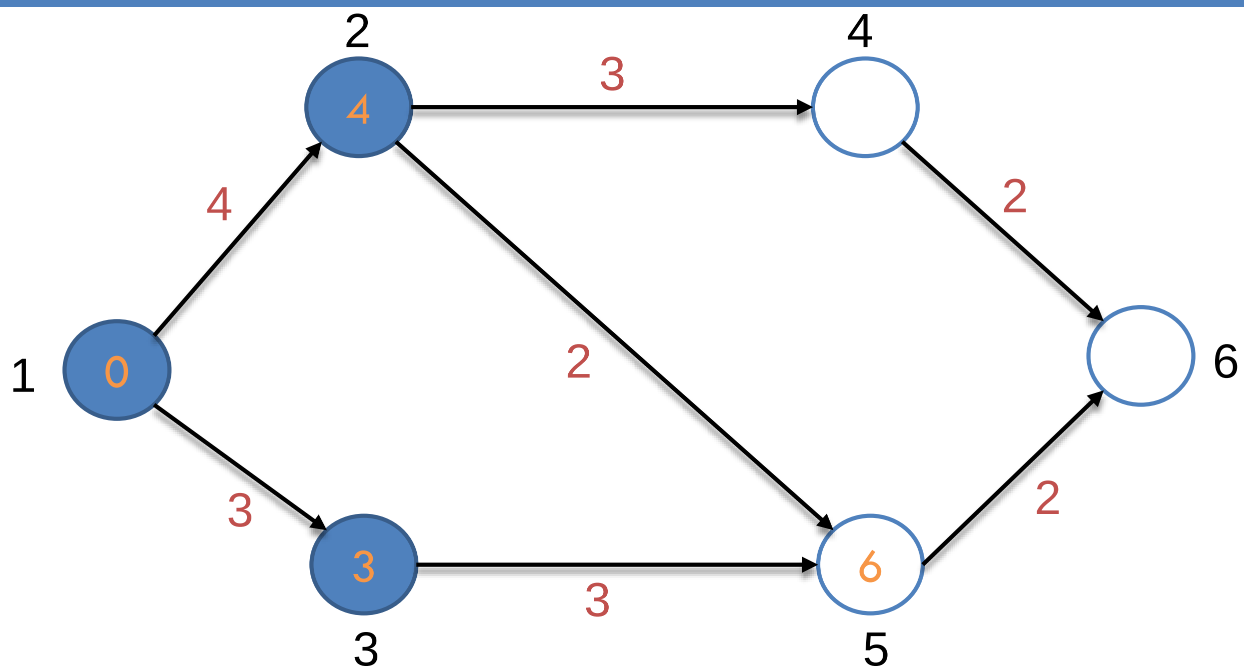
Find Shortest Path from Node 1 to 6



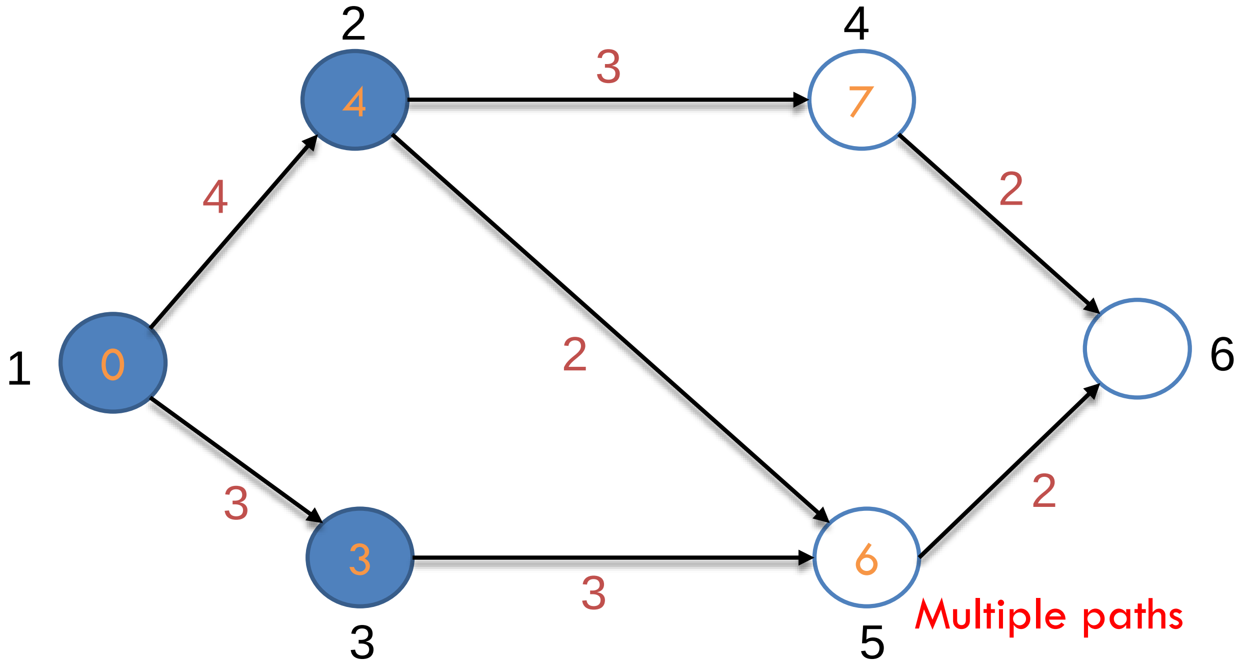
Find Shortest Path from Node 1 to 6



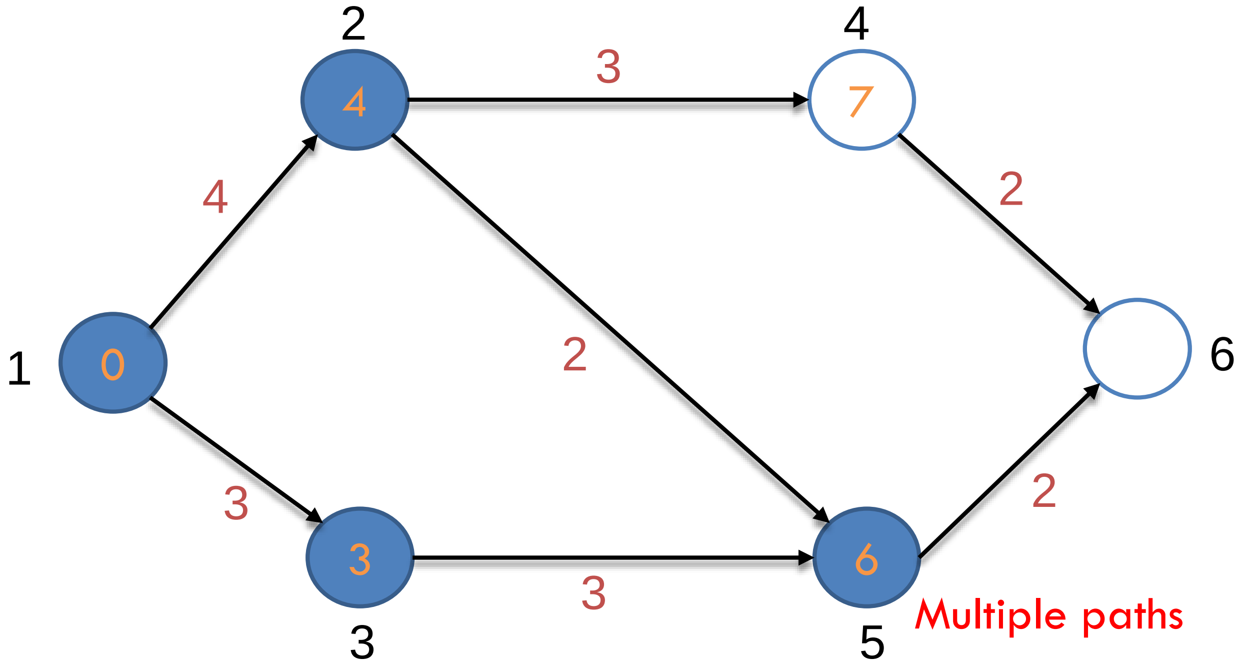
Find Shortest Path from Node 1 to 6



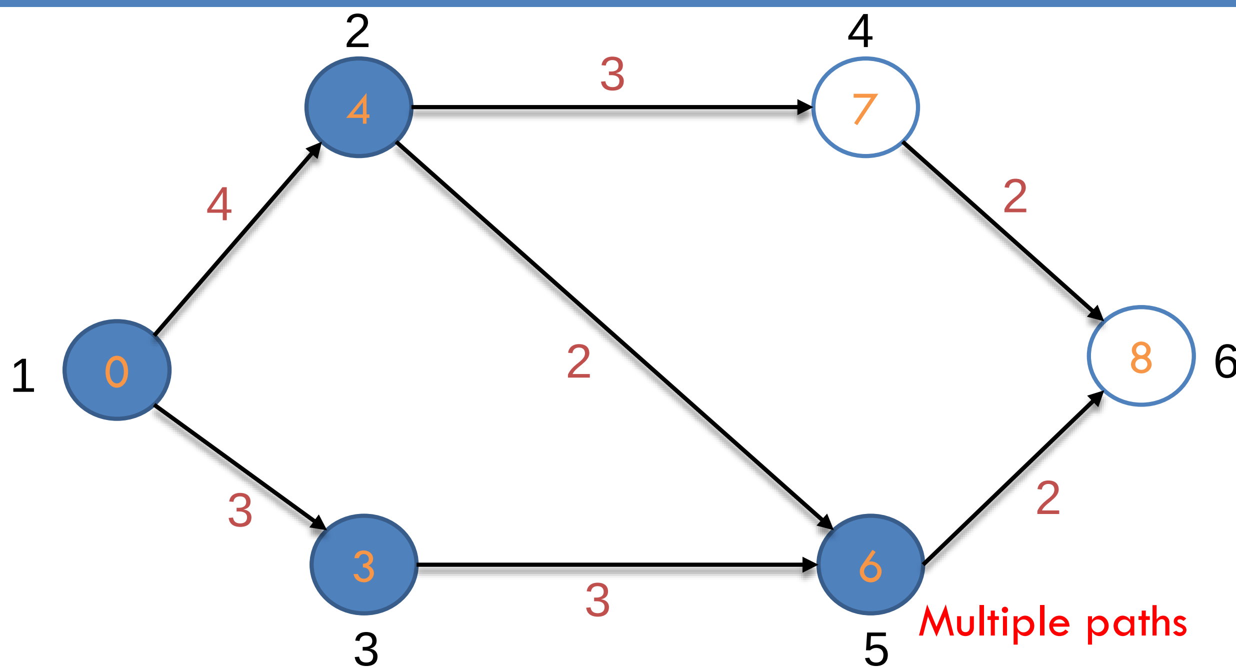
Find Shortest Path from Node 1 to 6



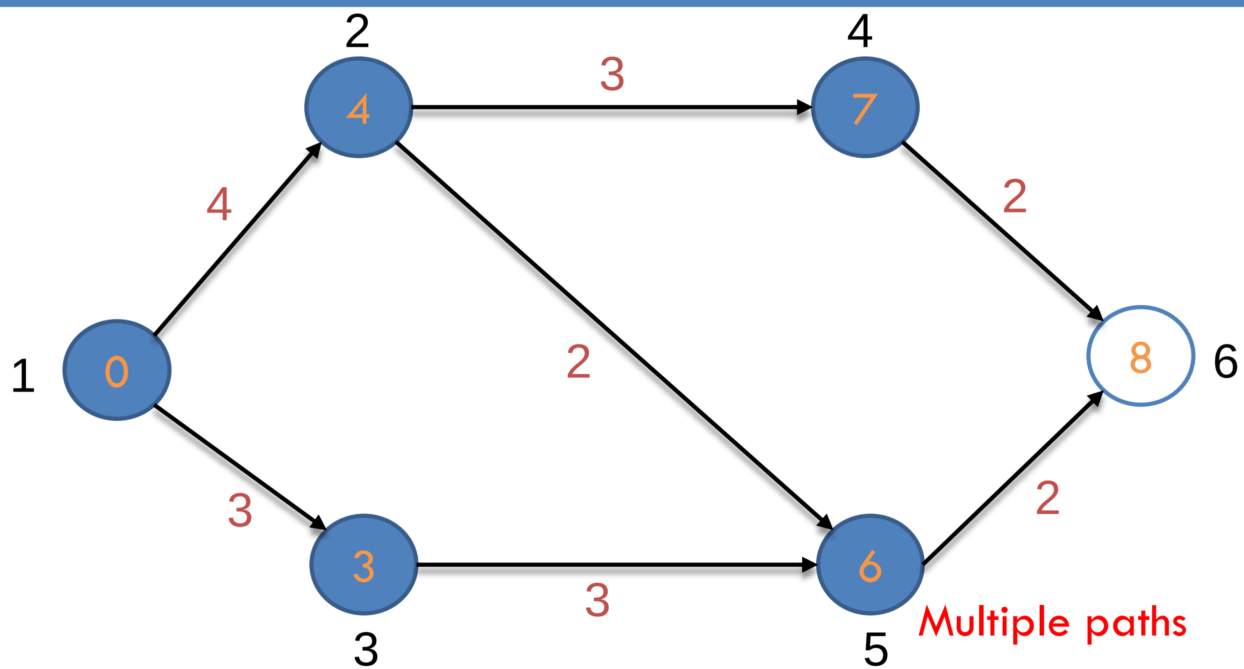
Find Shortest Path from Node 1 to 6



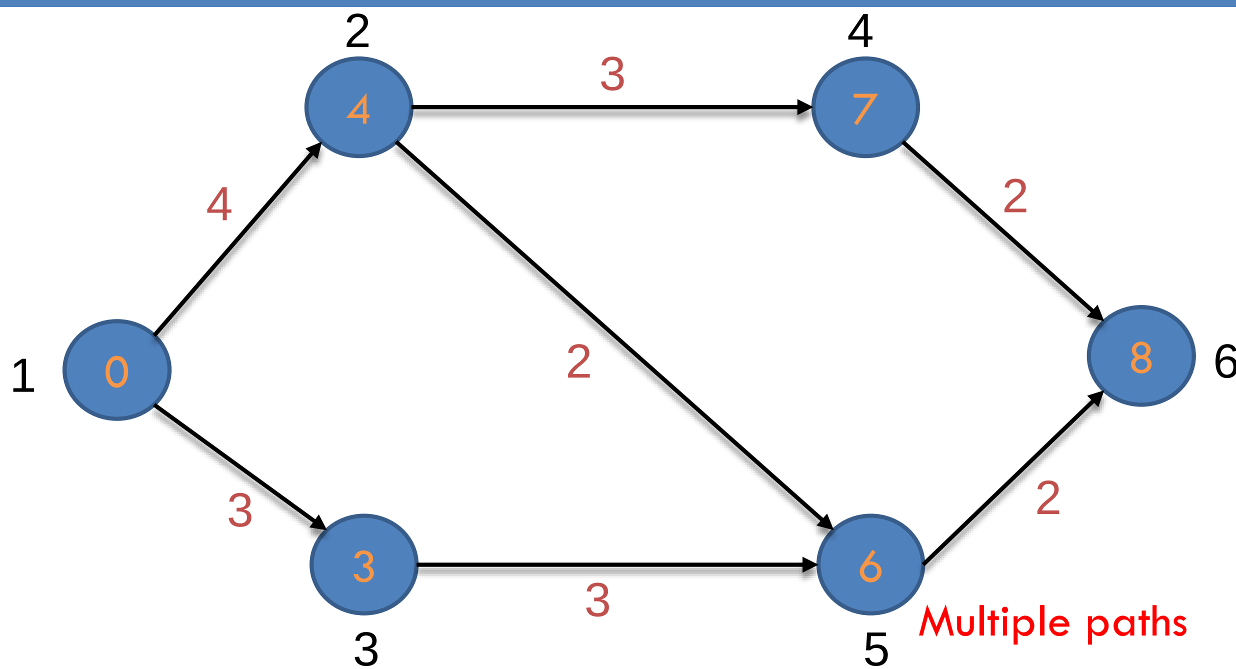
Find Shortest Path from Node 1 to 6



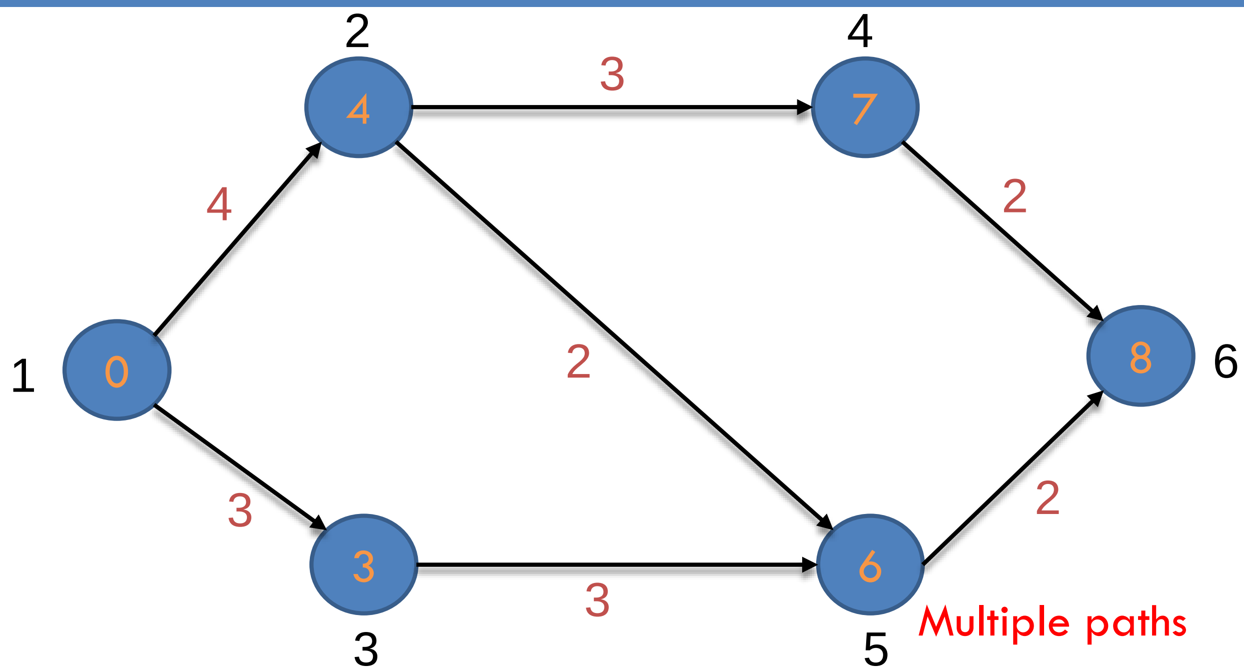
Find Shortest Path from Node 1 to 6



Find Shortest Path from Node 1 to 6



Find Shortest Path from Node 1 to 6



- 2 solutions are: (1-2-5-6) and (1-3-5-6)