



# The Migration Toward the Optical Internet

## Lesson 4

Luca Valcarenghi



# Directed Graphs (1)

- Def. DIRECTED GRAPH (DIGRAPH)
  - A DIRECTED GRAPH or DIGRAPH  $G(V,A)$  is a finite nonempty set  $V$  of vertices and a collection  $A$  of ORDERED pairs of distinct vertices from  $V$
- Def. ARC
  - Each ordered pair of vertices in  $A$  is called a DIRECT ARC (or simply ARC).
  - For example  $(2,4)$  and  $(4,2)$  are two different arcs
- Given any digraph  $G(V,A)$  there is an associated (undirected) graph  $G'(V',A')$  where  $V'=V$  and  $(i,j) \in A'$  if either  $(i,j) \in A$  or  $(j,i) \in A$  or both
- Def. DIRECTED WALK
  - $(v_1, v_2, \dots, v_l)$  is a DIRECTED WALK in a DIGRAPH  $G$  if  $(v_i, v_{i+1})$  is a DIRECTED ARC in  $G$  for  $1 \leq i \leq l-1$



# Directed Graphs (2)

- Def. DIRECTED PATH
  - A DIRECTED PATH is a directed walk with no repeated vertices
- Def. DIRECTED CYCLE
  - A DIRECTED CYCLE is a directed walk  $(v_1, v_2, \dots, v_l)$  for  $l > 2$  with  $v_1 = v_l$  and no repeated vertices
- Def. STRONG CONNECTEDNESS
  - A digraph is STRONGLY CONNECTED if for each pair of vertices  $i$  and  $j$  there is a directed path  $(i = v_1, v_2, \dots, v_l = j)$  from  $i$  to  $j$
- Def. CONNECTEDNESS
  - A digraph is CONNECTED if the associated undirected graph is connected



# Shortest Path Problem

- Def. PATH LENGTH

- Given a digraph  $G(V,A)$  with  $V$  vertices and  $A$  arcs in which each arc  $(i,j)$  is assigned some real number  $d_{ij}$  as the length or distance of the arc
- Given any directed path  $p=(i, j, k, \dots, l, m)$  the LENGTH of  $p$  is defined as
  - $d_{ij} + d_{jk} + \dots + d_{lm}$
- The length of a directed walk or cycle is defined analogously

- Def. SHORTEST PATH PROBLEM

- Given any two vertices  $i, m$  of the graph the SHORTEST PATH PROBLEM is to find a minimum length (i.e., shortest) directed path from  $i$  to  $m$



# Shortest Path Problem Examples

- IF  $d_{ij}$  = cost of using link  $(i,j) \Rightarrow$  shortest path from  $i$  to  $m$  is the minimum cost route over which to send data
- IF  $d_{ij}$  = average packet delay to cross link  $(i,j) \Rightarrow$  shortest path is the minimum delay route
- IF  $p_{ij}$  = probability that a given arc  $(i,j)$  in a network is usable, and each arc is usable independently of all other arcs  $\Rightarrow$  the shortest path between  $i$  and  $m$  with arc weight  $(-\ln p_{ij})$  is equivalent to finding the most reliable path from  $i$  to  $m$
- NOTE
  - The problem of finding shortest paths from a given origin vertex to all other vertices is equivalent to the problem of finding all shortest paths to a given destination vertex
  - One version of the problem is simply obtained by reversing the direction of each arc while keeping its length unchanged



# Optimal Structure of a Shortest Path

- Shortest path algorithms typically exploit the PROPERTY
  - A SHORTEST PATH between two vertices contains other shortest path within it
- LEMMA (1) SUBPATHS OF SHORTEST PATHS ARE SHORTEST PATHS
  - Given a weighted digraph  $G(V,A)$  with weight function  $w: A \rightarrow \mathcal{R}$  let  $p=(v_1, \dots, v_k)$  be a shortest path from vertex  $v_1$  to vertex  $v_k$  and, for any  $i$  and  $j$  such that  $1 \leq i \leq j \leq k$ , let  $p_{ij}=(v_i, v_{i+1}, \dots, v_j)$  be the subpath of  $p$  from vertex  $v_i$  to vertex  $v_j$ . Then  $p_{ij}$  is a shortest path from  $v_i$  to  $v_j$
  - Proof
    - Decompose  $p$  into  $v_1 \rightarrow (p_{1i}) \rightarrow v_i \rightarrow (p_{ij}) \rightarrow v_j \rightarrow (p_{jk}) \rightarrow v_k$  then  $w(p)=w(p_{1i}) + w(p_{jk})$ . Assume there is a path  $p'_{ij}$  from  $v_i$  to  $v_j$  with weight  $w(p'_{ij}) < w(p_{ij})$ . Then  $v_1 \rightarrow (p_{1i}) \rightarrow v_i \rightarrow (p'_{ij}) \rightarrow v_j \rightarrow (p_{jk}) \rightarrow v_k$  is a path from  $v_1$  to  $v_k$  whose weight  $w(p_{1i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$ . This contradicts the assumption that  $p$  is a shortest path from  $v_1$  to  $v_k$



# Optimal Structure of a Shortest Path (2)

- COROLLARY

- Let  $G(V,A)$  be a weighted digraph with weight function  $w: A \rightarrow \mathcal{R}$  and  $\delta(s,v)$  = weight of the path from  $s$  to  $v$
- Suppose that a shortest path  $p$  from a source  $s$  to a vertex  $v$  can be decomposed into  $s \rightarrow (p') \rightarrow u \rightarrow v$  for some vertex  $u$  and path  $p'$
- Then the weight of a shortest path from  $s$  to  $v$  is  $\delta(s,v) = \delta(s,u) + w(u,v)$
- Proof
  - By LEMMA (1) subpath  $p'$  is a shortest path from source  $s$  to vertex  $u$ . Thus  $\delta(s,v) = w(p) = w(p') + w(u,v) = \delta(s,u) + w(u,v)$



# Optimal Structure of a Shortest Path (3)

- Lemma (2)

- Let  $G(V,A)$  be a weighted digraph with weight function  $w: A \rightarrow \mathcal{R}$  and source vertex  $s$ . Then for all arcs  $(u,v) \in A \Rightarrow \delta(s,v) \leq \delta(s,u) + w(u,v)$

- Proof

- A shortest path from source  $s$  to vertex  $v$  has no more weight than any other path from  $s$  to  $v$ . Specifically, path  $p$  has no more weight than the particular path that takes a shortest path from  $s$  to  $u$  and then takes arc  $(u,v)$



# Bellman-Ford Algorithm

- Assumptions
  - Problem of finding a shortest path from every vertex to vertex  $j$  (e.g.,  $j=1$ )
  - Assume at least one path from every vertex to destination exists
  - $d_{ij} = \infty$  if  $(i,j)$  is not an arc of the graph
  - walks and paths consisting of true network arcs are the only ones with length less than  $\infty$
- Def. SHORTEST ( $\leq h$ ) WALK
  - A shortest walk from a given vertex  $i$  to vertex 1, subject to the constraint that the walk contains at most  $h$  arcs and goes through vertex 1 only once, is referred to as a SHORTEST ( $\leq h$ ) WALK. Its length is denoted by  $D_i^h$



# Bellman-Ford Algorithm (2)

---

## – Note

- such a walk may not be a path because of repeated vertices
- it can be proved that the shortest walk lengths are equal to the shortest path lengths under the additional assumption that all cycles not containing vertex 1 have nonnegative length



# Bellman-Ford Algorithm (3)

- Algorithm

1.  $D_i^h = 0 \quad \forall h, D_i^h = \infty \quad \forall i$

2.  $D_i^h$  is generated by the iteration 3

3.  $D_i^{h+1} = \min_j [d_{ij} + D_j^h] \quad \forall i \neq 1$  (BF1)

4. Termination Condition  $D_i^h = D_i^{h-1} \quad \forall i$

- The Bellman-Ford (BF) algorithms first finds the one-arc shortest walk lengths, then finds the two-arc shortest walk lengths, and so forth



# BF Algorithm (4)

- Bellman-Ford (BF) algorithm proposition
  - Consider the BF algorithm BF1 with the initial conditions  $D_i^0 = \infty \quad \forall i \neq 1$
  - Then
    - a) the scalars  $D_i^h$  generated by the algorithm are equal to the shortest ( $\leq h$ ) walk-lengths from vertex  $i$  to vertex 1
    - b) the algorithm terminates after a finite number of iterations if and only if all cycles not containing vertex 1 have nonnegative length. Furthermore, if the algorithm terminates, it does so after at most  $h \leq N$  iterations. At termination  $D_i^h$  is the shortest path length from 1 to  $i$



# BF Algorithm Proof

- Proof a)

- It is proven by induction from  $D_i^{h+1} = \min_j [d_{ij} + D_j^h] \quad \forall i \neq 1$  and  $D_i^0 = \infty \quad \forall i \neq 1 \Rightarrow$
- $D_i^1 = d_{i1} \quad \forall i \neq 1 \Rightarrow$  shortest ( $\leq 1$ ) walk length from  $i$  to 1
- Suppose  $D_i^k =$  shortest ( $\leq k$ ) walk length from  $i$  to 1  $\forall k \leq h \Rightarrow$  show  $D_i^{h+1}$  is the shortest ( $\leq h+1$ ) walk length from  $i$  to 1
- Shortest ( $\leq h+1$ ) walk length =  $\min\{ D_i^h, \min_{j \neq 1} [d_{ij} + D_j^h] \}^*$
- Indeed a shortest ( $\leq h+1$ ) walk from  $i$  to 1
  - either consists of less than  $h+1$  arcs  $\Rightarrow$  its length is equal to  $D_j^h$
  - or it consists of  $h+1$  arcs with
    - first arc  $(i, j)$  for some  $j \neq 1$
    - followed an  $h$ -arc walk from  $j$  to 1 in which vertex 1 is not repeated (this walk must be a shortest ( $\leq h$ ) walk from  $j$  to 1 (otherwise a shorter walk would exist)  $\Rightarrow$



# BF Algorithm Proof (2)

- $D_j^k \leq D_j^{k-1} \quad \forall k \leq h$  [since the set of ( $\leq k$ ) walks from  $j$  to 1 contains the corresponding set of ( $\leq k-1$ ) walks]  $\Rightarrow$
- $\Rightarrow D_i^{h+1} = \min_j [d_{ij} + D_j^h] \leq \min_j [d_{ij} + D_j^{h-1}] = D_i^h \Rightarrow$
- $\Rightarrow D_i^h \leq D_i^1 = d_{i1} = d_{i1} + D_1^h$
- From eq. \*  $\Rightarrow$  shortest ( $\leq h+1$ ) walk length =  $\min \{D_i^h, \min_j [d_{ij} + D_j^h]\} = \min \{D_i^h, D_i^{h+1}\}$
- by  $D_i^{h+1} \leq D_i^h \Rightarrow$  shortest ( $\leq h+1$ ) walk length =  $D_i^{h+1}$



# BF Algorithm Proof (3)

- Proof b)
  - If BF algorithm terminates after  $h$  iterations  $\Rightarrow$  all cycles not containing vertex 1 have nonnegative length
    - If BF algorithm terminates after  $h$  iterations we must have  $D_i^k = D_i^h \forall i$  and  $\forall k \geq h$  @  $\Rightarrow$
    - $\Rightarrow$  we cannot reduce the lengths of the shortest walks by allowing more and more arcs in these walks  $\Rightarrow$
    - $\Rightarrow$  there cannot exist a negative-length cycle not containing vertex 1 since such a cycle could be repeated an arbitrarily large number of times in walks from some vertices to vertex 1  $\Rightarrow$
    - $\Rightarrow$  make their length arbitrarily small  $\Rightarrow$  contradicts eq. @



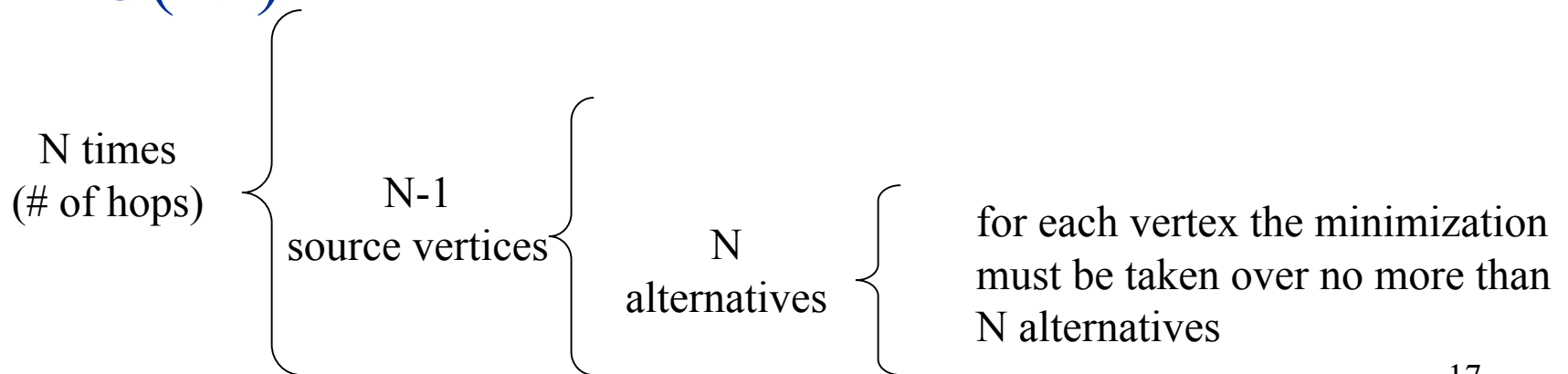
# BF Algorithm Proof (4)

- Proof b)
  - If all cycles not containing vertex 1 have nonnegative length  $\Rightarrow$  BF algorithm terminates after a finite ( $h \leq N$ ) # of iterations
    - By the hypothesis, by deleting all such cycles from shortest ( $\leq h$ ) walks  $\Rightarrow$
    - $\Rightarrow$  obtain paths of less or equal length  $\Rightarrow$
    - $\Rightarrow$  for every  $i$  and  $h \exists$  a path that is a shortest ( $\leq h$ ) walk from  $i$  to 1, and the corresponding shortest path length is equal to  $D_i^h$ .
    - Since paths have no cycles that can contain at most  $N-1$  arcs  $\Rightarrow$   
 $D_i^N = D_i^{N-1} \forall i \Rightarrow$
    - $\Rightarrow$  the algorithm terminates after at most  $N$  iterations



# BF Algorithm Remarks

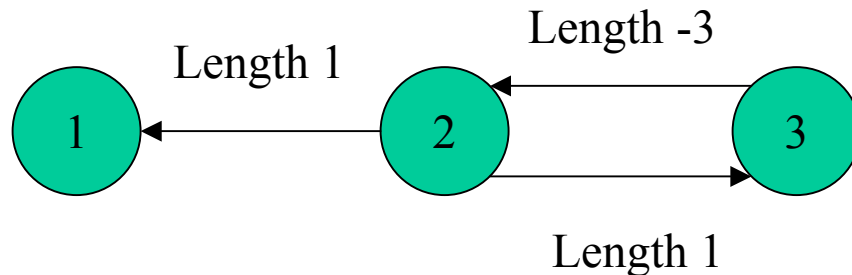
- Proposition 2 valid even if there is no path from source vertices  $i$  to vertex 1 in the original network. Upon termination, for those vertices  $D_i^{h=\infty}$
- Computational complexity (worst case)  $O(N^3)$






# Effect of Negative Cycles

- Not involving vertex 1
- Existing of these cycles can be tested by comparing  $D_i^N$  with  $D_i^{N-1} \forall i$
- By b) of proposition 2 there exists such a negative-length cycle if and only if  $D_i^N < D_i^{N-1}$  for some  $i$
- Example:  $D_2^3 < D_2^2 \Rightarrow$  negative cycle



$$D_2^2 = 1$$
$$D_2^3 = -1$$

Shortest Path  
length from  
2 to 1 is 1



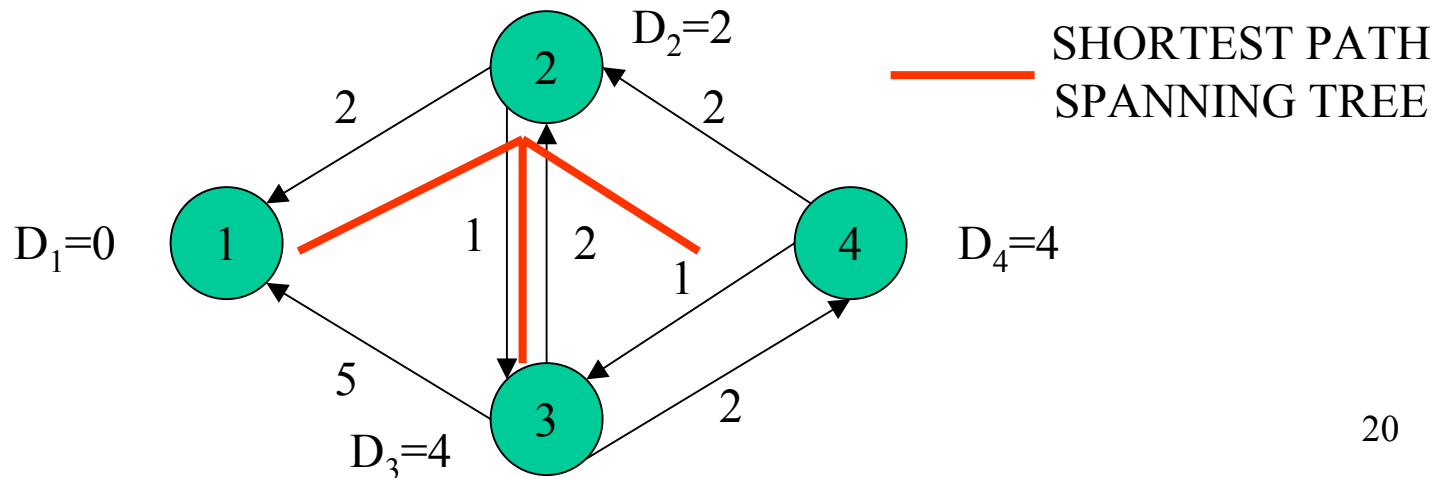
# Bellman's Equation and Shortest Path Construction

---

- Upon termination of the BF algorithm Bellman's equation (BF EQUATION) is given by
  - $D_i = \min_j [d_{ij} + D_j] \quad \forall i \neq 1$
  - $D_1 = 0$
- Bellman's equation expresses that the shortest path length from vertex  $i$  to 1 is the sum of the length of the arc to the vertex following  $i$  on the shortest path plus the shortest path length from that vertex to vertex 1

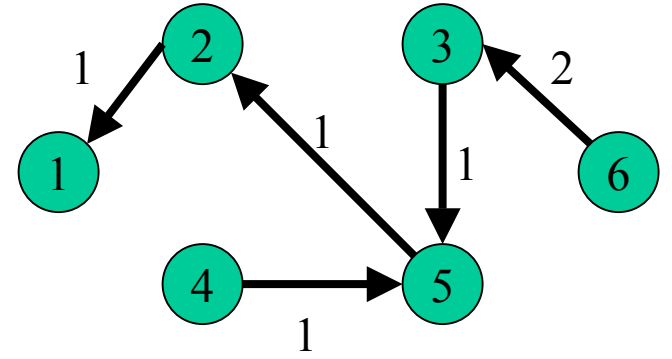
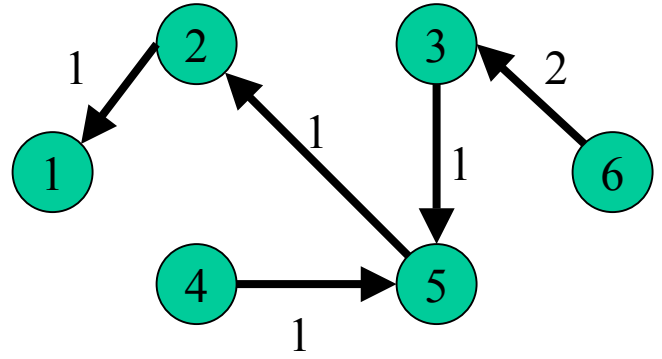
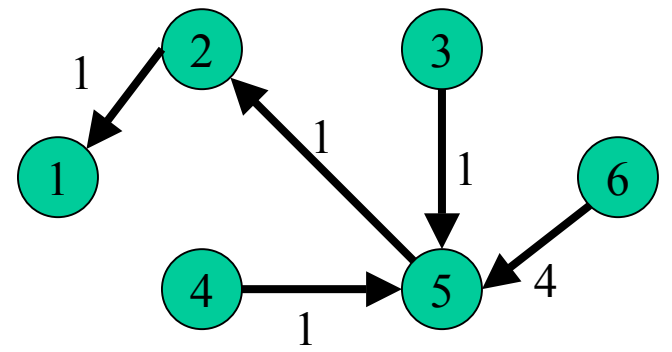
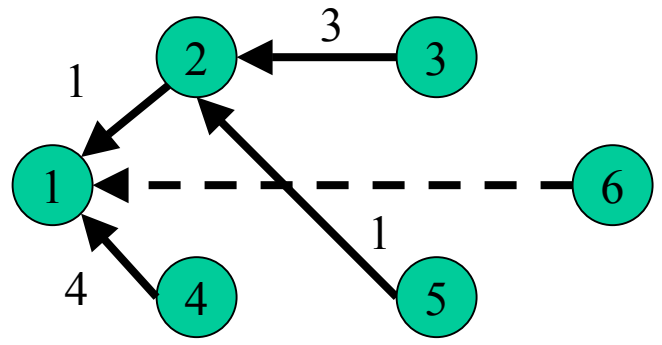
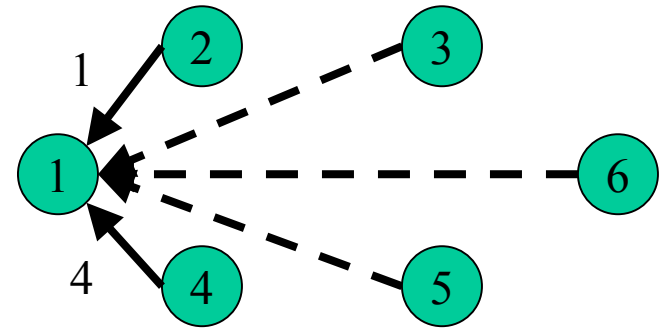
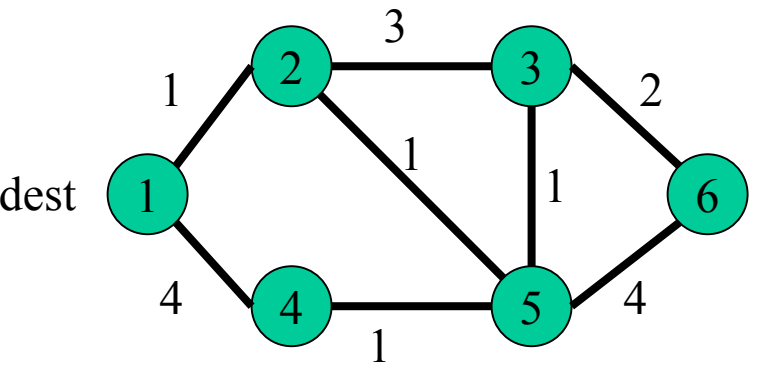
# Shortest Paths Based on BF Equation

- From BF equation it is possible to find the shortest paths
- To find the shortest path
  - Select for each  $i \neq 1$  one edge  $(i,j)$  that attains the minimum  $D_i = \min_j [d_{ij} + D_j]$
  - Consider the subgraph consisting of the  $N-1$  arcs
  - To find the shortest path from any vertex  $i$  start at  $i$  and follow the corresponding arcs of the subgraph until vertex 1 is reached
- Since the subgraph connects every vertex to vertex 1 and has  $N-1$  arcs, it must be a spanning tree  $\Rightarrow$  SHORTEST PATH SPANNING TREE





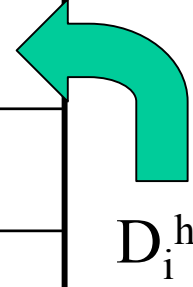
# BF Algorithm Example





# BF Algorithm Example (2)

i/h	1	2	3	4	5	6
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
1	0	1	$\infty$	4	$\infty$	$\infty$
2	0	1	4	4	2	$\infty$
3	0	1	3	3	2	6
4	0	1	3	3	2	5
5	0	1	3	3	2	5



$D_i^h$



# Example of BF Algorithm Iteration

- Graph  $G(V,A)$  from slide 21
- $D_3^3 = \min_j [d_{3j} + D_j^2]$ 
  - $d_{31} + D_1^2 = \infty + 0 = \infty$
  - $d_{32} + D_2^2 = 3 + 1 = 4$
  - $d_{33} + D_3^2 = 0 + 4 = 4$
  - $d_{34} + D_4^2 = \infty + 4 = \infty$
  - $d_{35} + D_5^2 = 1 + 2 = 3$
  - $d_{36} + D_6^2 = 2 + \infty = \infty$
- $D_5^3 = \min_j [d_{5j} + D_j^2]$ 
  - $d_{51} + D_1^2 = \infty + 0 = \infty$
  - $d_{52} + D_2^2 = 1 + 1 = 2$
  - $d_{53} + D_3^2 = 1 + 4 = 5$
  - $d_{54} + D_4^2 = 1 + 4 = 5$
  - $d_{55} + D_5^2 = 0 + 2 = 2$
  - $d_{56} + D_6^2 = 4 + \infty = \infty$



# Dijkstra's Algorithm

- Dijkstra's (D) algorithm requires that all the arc lengths are NONNEGATIVE (case of most data network applications)
- Worst-case computational requirements less than those of BF algorithm
- General idea
  - Base don Lemma (1) (SUBPATHS OF SHORTEST PATHS ARE SHORTEST PATHS)
  - Find the shortest path in order of increasing path length
  - The shortest of the shortest paths to vertex 1 must be the single arc path from the closest neighbor of vertex 1
  - The next shortest of the shortest paths must either be the single-arc path from the closest neighbor of 1 or the shortest two-arc paths through the previously chosen vertex, and so on
  - To formalize the procedure in an algorithm each vertex  $i$  is labeled with an estimate  $D_i$  of the shortest path length to vertex 1
  - When the estimate becomes certain, the vertex is PERMANENTLY LABELED and added to the set P of permanently labeled vertices
  - The vertex added to P at each step will be closest to vertex 1 out of those that are not yet in P



# Dijkstra's Algorithm (2)

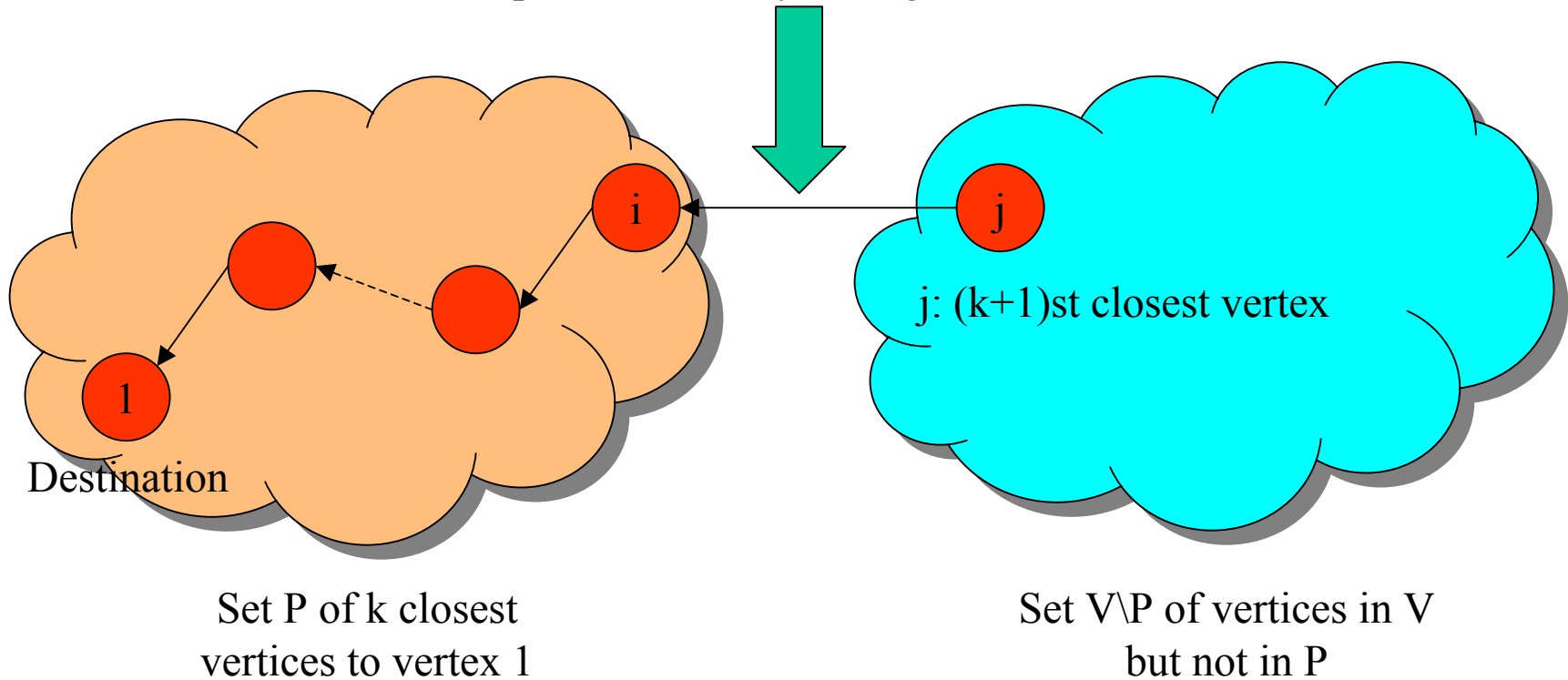
- Assumption
  - find all shortest path from any vertex to vertex 1
- 1. STEP 0
  - $P = \{1\}$ ,  $D_1 = 0$ ,  $D_j = d_{j1}$  for  $j \neq 1$
- 2. STEP 1 (FIND THE NEXT CLOSEST VERTEX)
  - Find  $i \notin P$  such that  $D_i = \min_{j \notin P} D_j$
  - Set  $P := P \cup \{i\}$
  - If  $P = V \Rightarrow \text{STOP}$
- 3. STEP 2 (UPDATING OF LABELS)
  - For all  $j \notin P$  set  $D_j := \min[D_j, d_{ij} + D_i]$
  - GO TO STEP 1



# Dijkstra's Algorithm

## Concept

Shortest path from the  $(k+1)$ st closest vertex must pass exclusively through vertices in  $P$





# Dijkstra's Algorithm Proof (1)

---

- It is shown that
  - a)  $D_i \leq D_j \quad \forall i \notin P, \forall j \in P$
  - b)  $\forall j \in P \quad D_j =$  shortest distance from  $j$  to 1 using paths with all vertices (except possibly  $j$ ) belonging to set  $P$
- a) is satisfied initially because
  - $d_{ij} \geq 0$  and  $D_i = \min_{j \in P} D_j$
- a) is preserved by the formula
  - $D_j := \min[D_j, d_{ji} + D_i] \quad \forall j \notin P$



# Dijkstra's Algorithm Proof (2)

- b) shown by induction
  - b) holds initially
  - Suppose b) holds at the beginning of a generic STEP 1
    - $i$  = vertex added to  $P$  at this STEP 1
    - $D_k$  = label of each vertex  $k$  at the beginning of this STEP 1
      - It holds for  $j=1$
      - It holds  $\forall j \in P$  because of a) and the induction hypothesis
      - for  $j \notin P \cup \{i\}$  consider path from  $j$  to 1 which is the shortest among those with all vertices except  $j$  belonging to  $P \cup \{i\} \Rightarrow D_j$  ' = shortest distance.
      - This path must consist of
        - » arc  $(j, k)$  for some  $k \in P \cup \{i\}$  + shortest path from  $k$  to 1 with vertices in  $P \cup \{i\}$
        - »  $D_k$  = path length  $\Rightarrow$
      - $D_j$  ' =  $\min_{k \in P \cup \{i\}} [d_{jk} + D_k] = \min[\min_{k \in P} [d_{jk} + D_k], d_{ji} + D_i]$



# Dijkstra's Algorithm Proof (3)

---

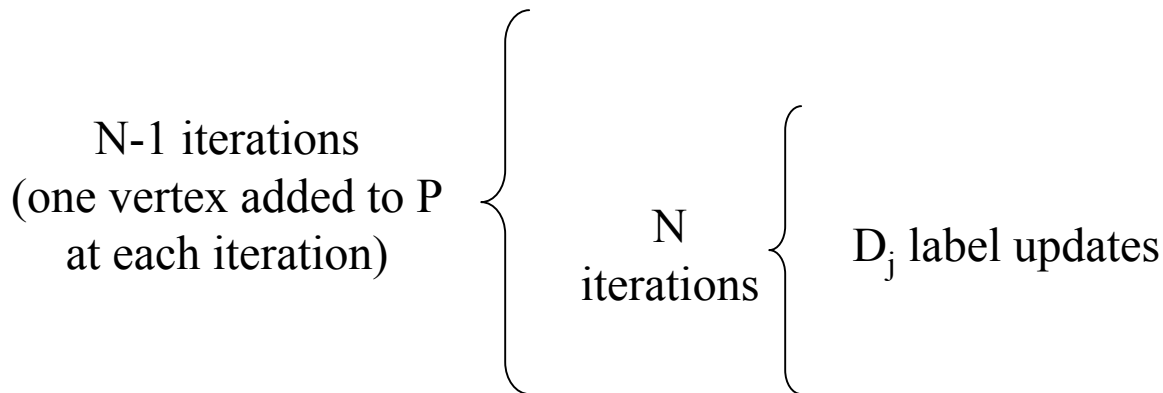
- The induction hypothesis  $\Rightarrow D_j = \min_{k \in P} [d_{jk} + D_k] \Rightarrow D_j' = \min[D_j, d_{ji} + D_i]$
- in STEP 2  $D_j$  set to the shortest distance  $D_j'$  from  $j$  to 1 using paths with all vertices except  $j$  belonging to  $P \cup \{i\}$
- Induction proof completed



# Dijkstra's Algorithm

## Computational Complexity

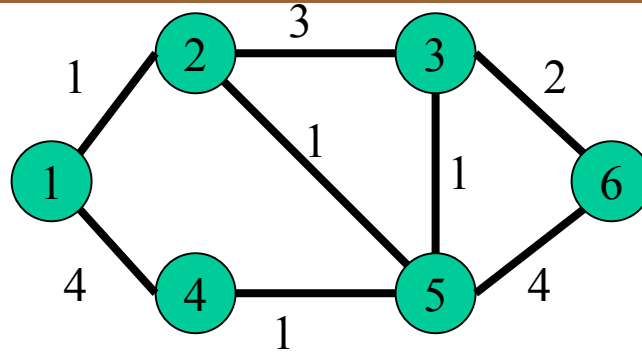
- A new vertex added to  $P$  at each iteration  $\Rightarrow$  Dijkstra's algorithm terminates after  $N-1$  iterations
- Worst case computational complexity  $O(N^2)$
- With particular network graph representation Dijkstra's algorithm complexity  $\Rightarrow O(N \log N)$
- $O(N^2)$  (Dijkstra)  $<$   $O(N^3)$  (BF)





# Dijkstra's Algorithm

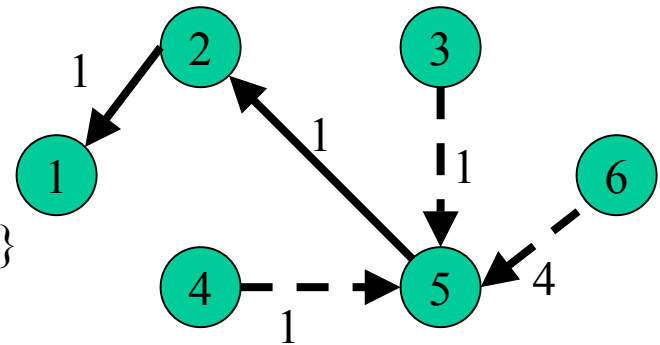
## Example



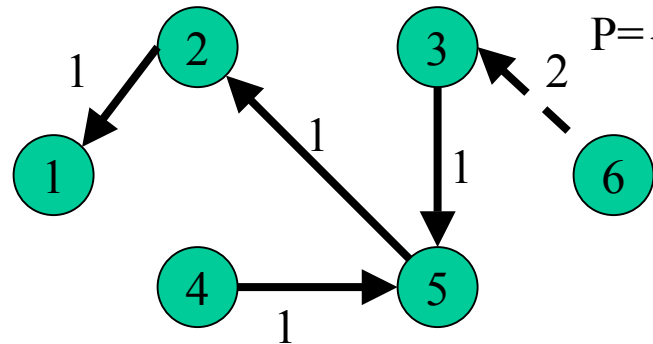
$P = \{1, 2\}$



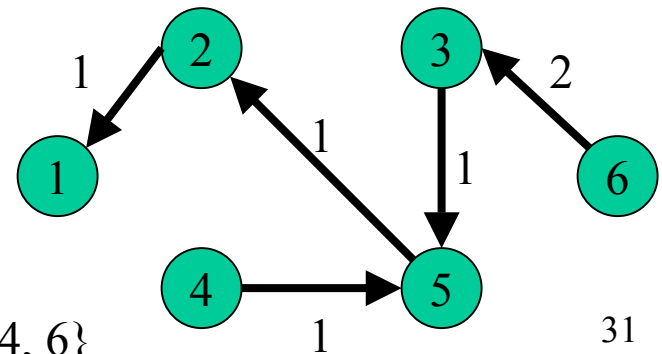
$P = \{1, 2, 5\}$



$P = \{1, 2, 5, 3, 4\}$



$P = \{1, 2, 5, 3, 4, 6\}$





# Dijkstra's Algorithm

## Example (2)

source /step	1	2	3	4	5	6
0	0	1	$\infty$	4	$\infty$	$\infty$
1	0	1	4	4	2	$\infty$
2	0	1	3	3	2	6
3	0	1	3	3	2	5
4	0	1	3	3	2	5



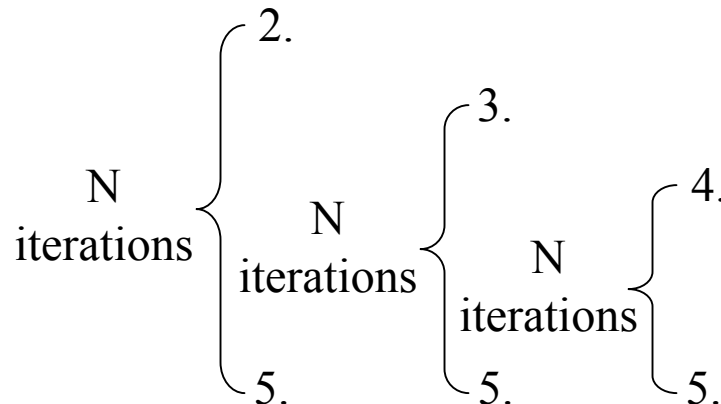
# Floyd-Warshall (FW) Algorithm

- It finds the shortest path between all pairs of vertices together
- The FW algorithm iterates on the set of vertices allowed as intermediate vertices on the paths
- Definitions
  - $D_{ij}^{(n)}$  = shortest path length from vertex  $i$  to vertex  $j$  with constraint that only vertices  $1, 2, \dots, n$  can be used as intermediate vertices on paths  $D_{ij}^{(n)} \in \mathcal{N} \times \mathcal{N}$
  - $n$  = # of network vertices



# FW Algorithm (2)

1.  $D_{ij}^{(0)} = d_{ij} \quad \forall i, j \quad i \neq j$
  2. FOR  $k=1$  TO  $n$
  3.     DO FOR  $i=1$  TO  $n$
  4.         DO FOR  $j=1$  TO  $n$
  5.              $D_{ij}^{(k)} = \min[D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}] \quad \forall i \neq j$
  6. RETURN  $D_{ij}^{(n)}$
- Worst case computational complexity  $O(N^3)$





# Constructing Shortest Path with FW Algorithm

- $\Pi$  predecessor matrix
- A sequence of matrices  $\Pi^{(0)}, \Pi^{(1)}, \dots, \Pi^{(n)}$  is computed where
  - $\Pi = \Pi^{(n)}$
  - $\pi_{ij}^{(k)}$  is defined to be the predecessor of vertex  $j$  on a shortest path from vertex  $i$  with all intermediate vertices in the set  $\{1, 2, \dots, k\}$



# Constructing Shortest Path with FW Algorithm (2)

- Recursive formulation of  $\pi_{ij}^{(k)}$

- for  $k=0$

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ or } w_{ij} < \infty \end{cases}$$

- for  $k \geq 1$

- if the path  $i \rightarrow k \rightarrow j$  is taken the predecessor of  $j$  chosen is the same as the predecessor of  $j$  chosen on a shortest path from  $k$  with all intermediate vertices in the set  $\{1, 2, \dots, k-1\}$
- otherwise the same predecessor of  $j$  chosen on a shortest path from  $i$  with all intermediate vertices in the set  $\{1, 2, \dots, k-1\}$  is chosen

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$



# Print FW Shortest Path

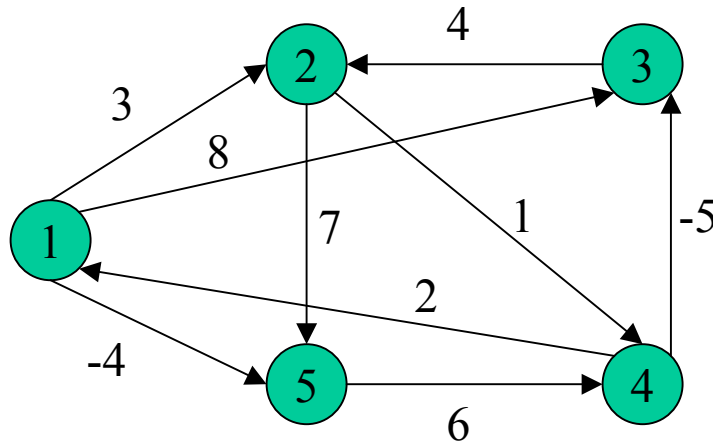
- Given the predecessor matrix  $\Pi$
- PRINT-ALL-PAIRS-SHORTEST-PATH( $\Pi, i, j$ )
  1. IF  $i=j$
  2.       THEN print  $i$
  3. ELSE IF  $\pi_{ij}=\text{NIL}$
  4.       THEN print “no path from”  $i$  “to”  $j$  “exists”
  5. ELSE PRIN-ALL-PAIRS-SHORTEST-PATH
  6.       print  $j$



# FW Algorithm Proof

- By induction
- $k=0 \Rightarrow$  shortest path length with no intermediate vertices
- Suppose for a given  $k$ 
  - $D_{ij}^k \Rightarrow$  shortest path length using vertices 1 to  $k$  as intermediate vertices
    - Then shortest path length from  $i$  to  $j$  allowing vertices 1 to  $k+1$  as intermediate vertices
      1. either contains vertex  $k+1$  on the shortest path
      2. or it does not contain vertex  $k+1$
    - For 1) constrained shortest from  $i$  to  $j \Rightarrow i \rightarrow k+1 \rightarrow j$
    - For 2) constrained shortest path is the same as the one using vertices  $i$  to  $k$  as intermediate vertices

# FW Algorithm Example



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(0)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

# FW Algorithm Example (2)

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & 2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$\Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$



# FW Algorithm Example (3)

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$\Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$