



---

---

# IE 680 – Special Topics in Production Systems: Networks, Routing and Logistics\*

**Rakesh Nagi**  
Department of Industrial Engineering  
University at Buffalo (SUNY)

*\*Lecture notes from Network Flows by Ahuja, Magnanti and Wong (1993); Dr. Bruce Golden's  
and Dr. Joy Bhadury's Lecture Notes*



# The shortest path problem

---

---

Suppose that we are given a network  $G$  of  $n$  nodes and  $m$  arcs, and an arc length (or an arc cost  $c_{ij}$ ) associated with every arc  $(i,j)$  in  $G$ .

The Shortest path problem is : Find the shortest (or least costly) path from node 1 to node  $n$  in  $G$ . The length (or cost) of the path is the sum of lengths or (costs) of all the arcs in the path.

The network has a distinguished node called the source and a node called sink.



# The shortest path problem

---

---

- “shortest”, “cheapest” and “most comfortable”
- In economic terms, supply of one or more units at min “cost” or with min “delay”, etc.
- 4 versions
  - ▶ (SP1) Shortest path between specific origin  $s$  to specific destination  $t$
  - ▶ (SP2) Shortest path between specific origin  $s$  to all other nodes
  - ▶ (SP3) Shortest path between all pairs of nodes
  - ▶ (SP4) K-Shortest paths



# The shortest path problem

---

---

LP formulation of the SPP:

$$\text{Min } \sum_{(i,j) \in A} c_{ij} x_{ij}$$

Subject to:

$$\sum_{\{j:(i,j) \in A\}} x_{ij} - \sum_{\{j:(j,i) \in A\}} x_{ji} = \begin{cases} n-1 & \text{for } i = s \\ -1 & \text{for all } i \in N - \{s\} \end{cases}$$

$$x_{ij} \geq 0 \text{ for all } (i,j) \in A$$



# The shortest path problem

---

---

- Assumptions (may or may not be needed for specific algorithms)
- All arc lengths are integers (generally not needed)
- The network contains a directed path from node  $s$  to every other node (otherwise add a “fictitious” arc with suitably large cost)
- The network does not contain a negative cycle, i.e., a directed cycle of negative length (required by most algorithms)
- The network is directed (simple transformation if arc lengths are nonnegative; otherwise it is more complex)



# The shortest path problem

---

---

There are two classes of algorithms to solve the SPPs:

- ☞ Label-setting algorithm.
- ☞ Label-correcting algorithm.

Both the algorithms are iterative and assign tentative distance labels to nodes at each step, which are estimates of the upper bounds on the shortest path distances.

Label setting algorithms designate one label as permanent (optimal) at each iteration.

Label correcting algorithms consider the labels as temporary until the final step when they all become permanent.



# Label Setting Algorithms

---

---

## ★ Label-setting algorithm:

### Dijkstra's labeling algorithm:

Method: This algorithm maintains distance label  $d(i)$  with each node  $i$ , which is the upper bound on the shortest path from node  $i$ . At any intermediate step, the algorithm divides the nodes into two groups: those which it designates as permanently labeled and those as temporarily labeled. The distance label to any permanent node represents the shortest distance from the source to that node.

For any temporary node  $i$ , the distance label is the upper bound on the shortest path distance to that node.

# Label Setting Algorithms

- Steps in the algorithm:

**Step 1: Initialize labels.**

$$U_j = \begin{cases} 0 & \text{if } j \text{ is the source.} \\ \text{weight of the arc from the source to } j & \text{if arc between source and } j \text{ exists} \\ \infty & \text{if arc between source and } j \text{ does not exist.} \end{cases}$$

Let the source node be scanned.



**Step 2: Update labels recursively from unscanned node. Let 'w' be the unscanned node with the smallest label; then let**

$$U_j = \min \left\{ U_j, U_w + C_{wj} \right\} \quad \text{where,}$$

$$C_{wj} = \begin{cases} \text{weight of the arc from } w \text{ to } j & \text{if } w \text{ to } j \text{ has an arc} \\ 0 & \text{if } w = j \\ \infty & \text{otherwise.} \end{cases}$$



**Make w scanned and return to Step 2 as long as there is more than one vertex unscanned.**



# Dijkstra's Algorithm

---

---

- Step 1: Initialization

$$S = \emptyset; R = N$$

$$d(i) = \infty \text{ for each node } i \in N$$

$$d(s) = 0 \text{ and } \text{pred}(s) = 0;$$

- While  $|S| < n$  do

Let  $i \in R$  be a node for which  $d(i) = \min\{d(j): j \in R\}$

$$S = S \cup \{i\}$$

$$R = R - \{i\}$$

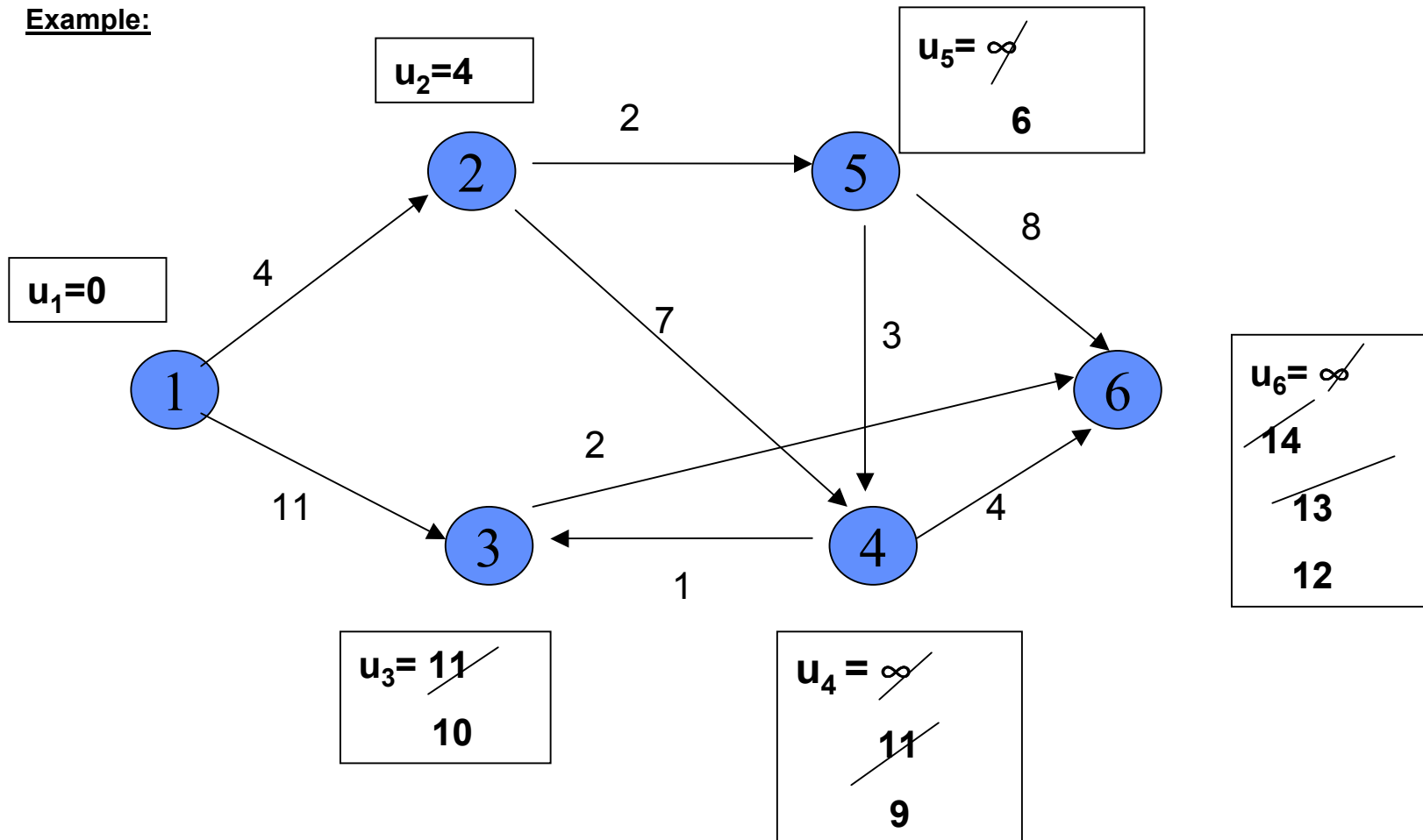
For each  $(i,j) \in A(i)$  do

if  $d(j) > d(i) + c_{ij}$  then  $d(j) = d(i) + c_{ij}$  and  $\text{pred}(j) = i$



# Dijkstra's Algorithm

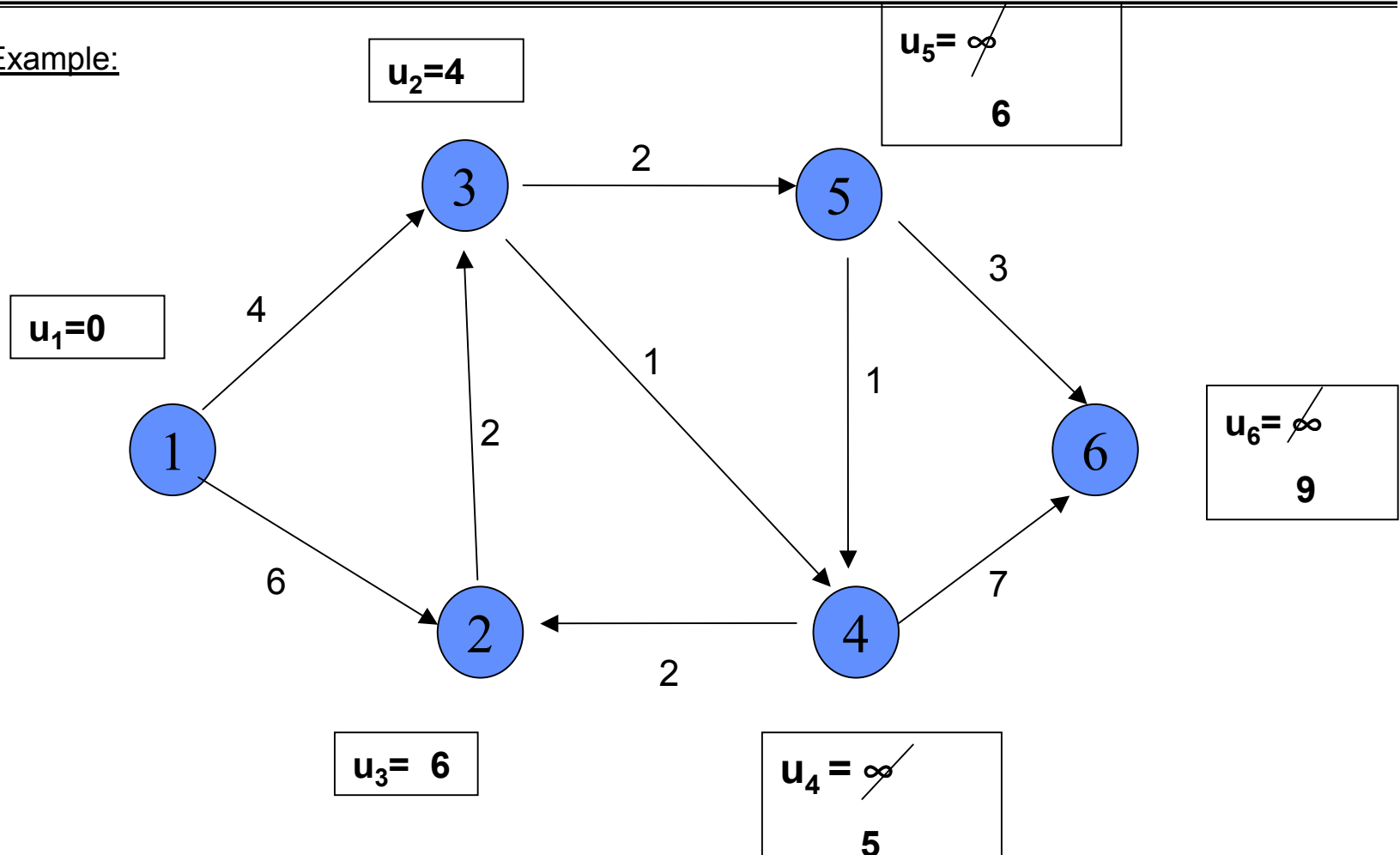
Example:





# Dijkstra's Algorithm

Example:

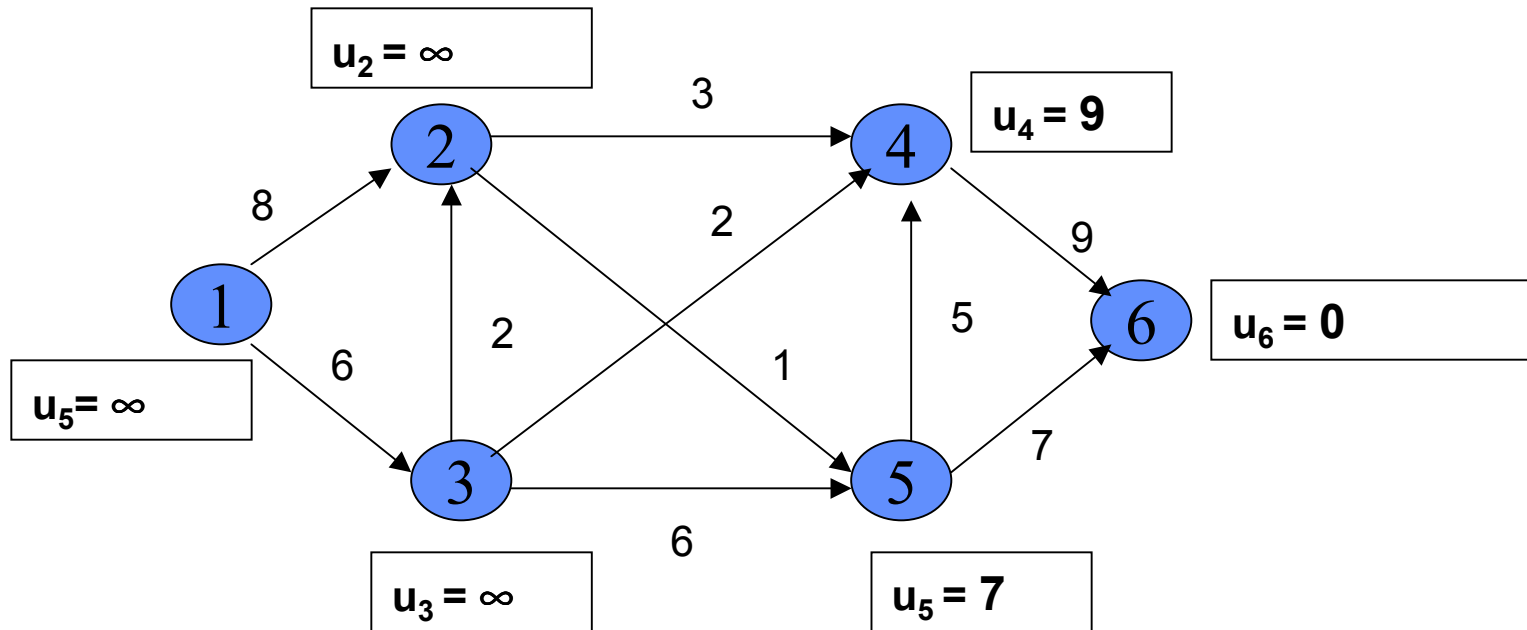




# Dijkstra's Algorithm

Example: Find the shortest path to go from node 6 to node 1.

We modify Dijkstra's algorithm as follows:



Step 1: Take source node **6** and label it as  $u_6=0$ .

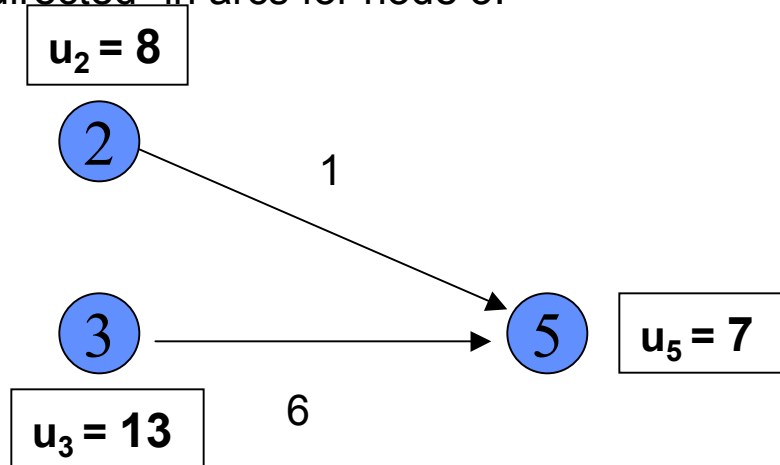
Step 2: Label node **5** as  $u_5=7$  and node **4** as  $u_4=9$ .

Step 3: Label rest of the nodes as  $\infty$ .



# Dijkstra's Algorithm

- Step 4: Choose the minimum of the labels of node 5 and node 4 i.e.  
\_\_\_\_\_  $\text{Min} \{ 7, 9 \} = 7$  for node 5 and scan it.
- Step 5: Choose all the directed -in arcs for node 5.



Node 2:  $\text{Min} \{ \infty, 7+1 \} = \text{Min} \{ \infty, 8 \} = 8$ .

Node 3:  $\text{Min} \{ \infty, 7+6 \} = \text{Min} \{ \infty, 13 \} = 13$ .

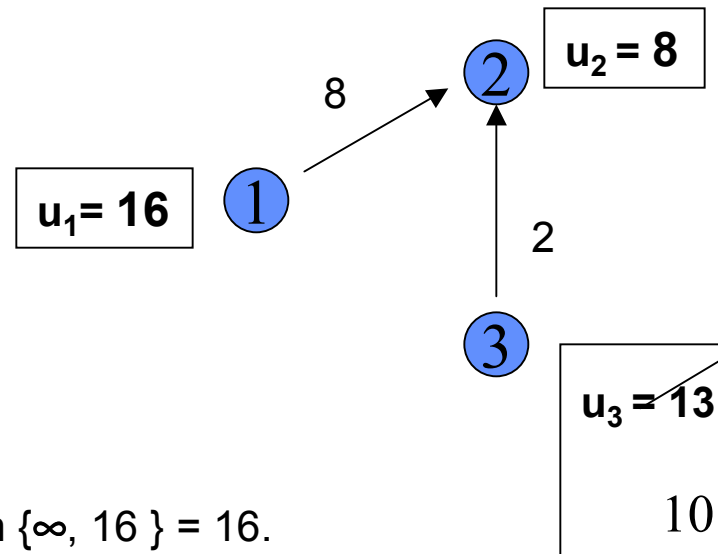
Choose the minimum of the labels of node 2 and node 3 i.e.

\_\_\_\_\_  $\text{Min} \{ 8, 13 \} = 8$  for node 2 and scan it.



# Dijkstra's Algorithm

- Step 6: Choose all the directed -in arcs for node 2.



Node 1:  $\text{Min} \{ \infty, 8+8 \} = \text{Min} \{ \infty, 16 \} = 16$ .

Node 3:  $\text{Min} \{ 13, 8+2 \} = \text{Min} \{ 13, 10 \} = 10$ .

Choose the minimum of the labels of node 1 and node 3 i.e.

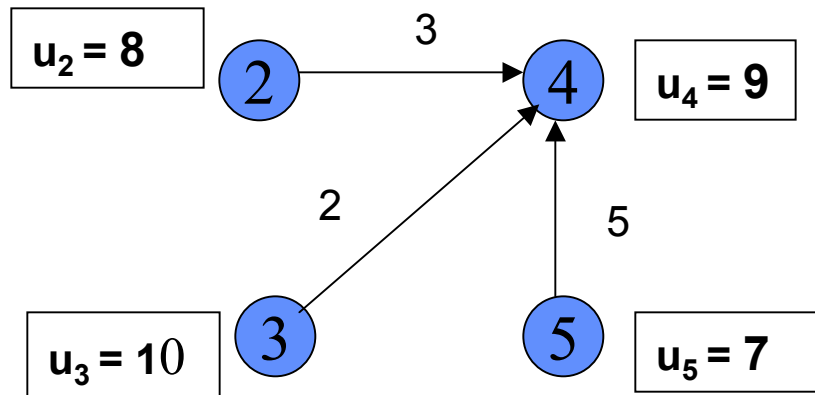
\_\_\_\_\_  $\text{Min} \{ 16, 10 \} = 10$  for node 3 and scan it.

Since node 1 has no directed-in arcs scan node 1.

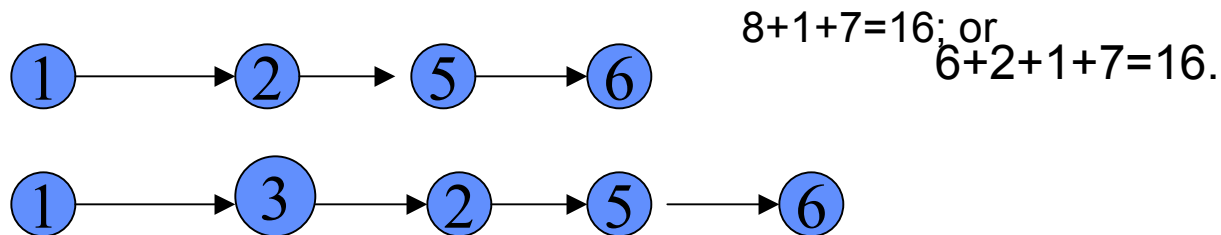


# Dijkstra's Algorithm

- Step 7: Choose all the directed -in arcs for node 4.

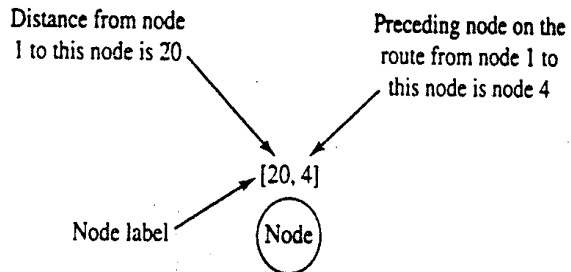
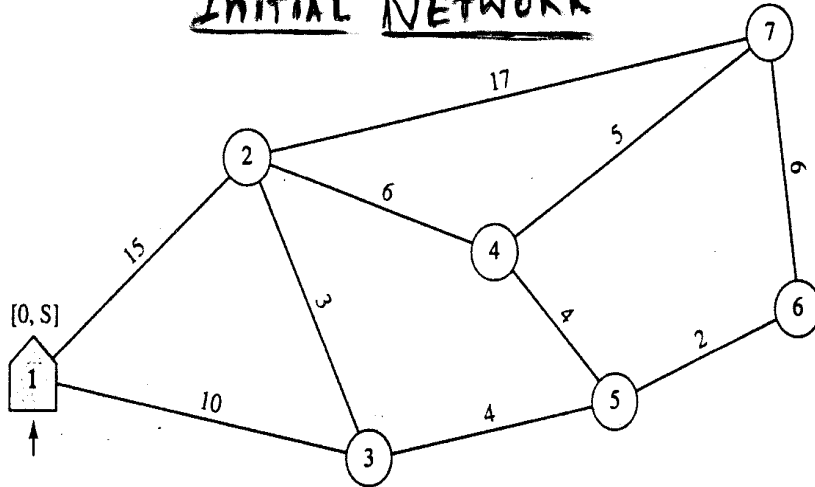


- Therefore the shortest path from node 1 to node 6 is:



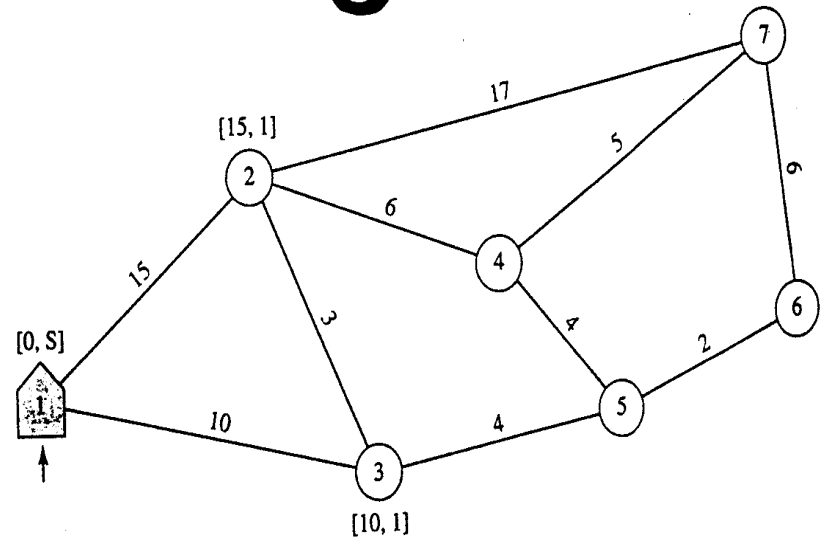
# Shortest Path Problem - Numerical Example

## INITIAL NETWORK



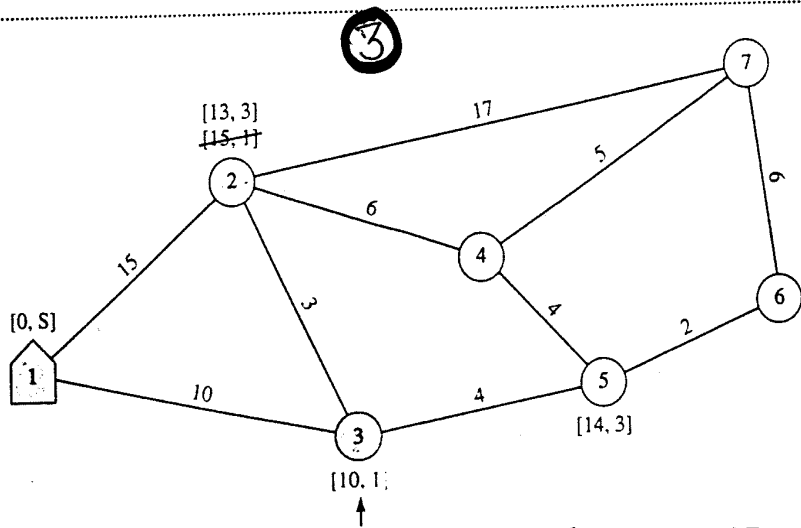
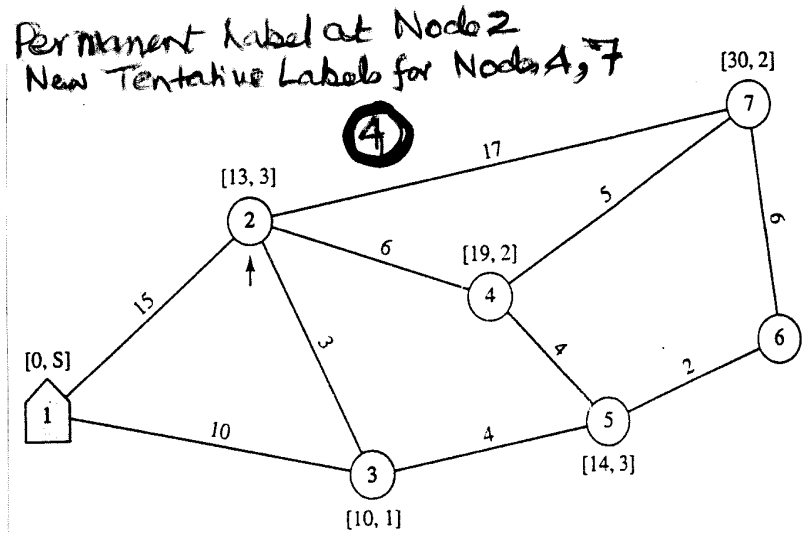
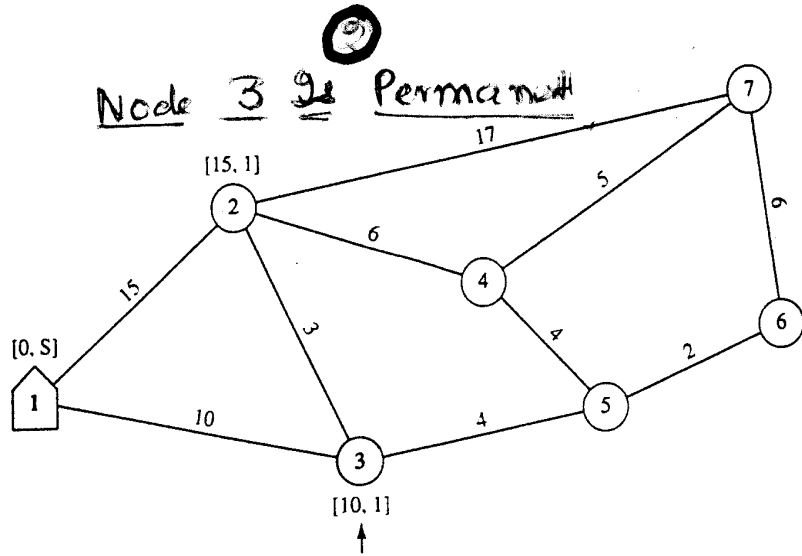
## Labels on Nodes

①

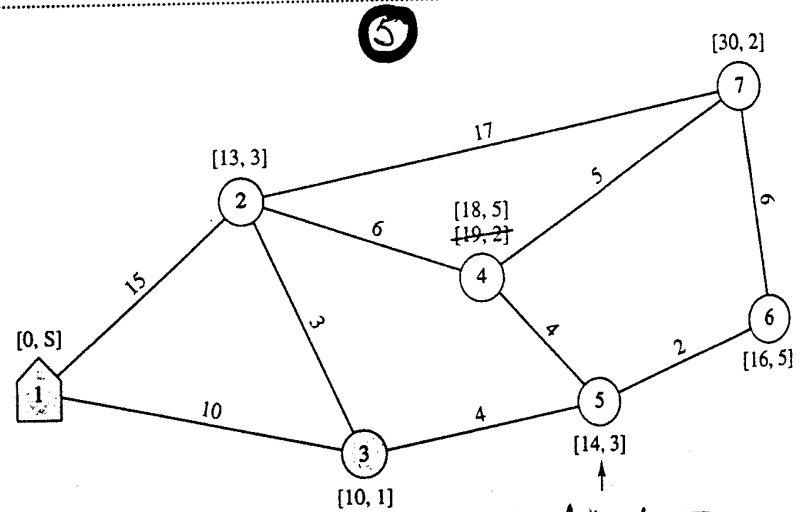


Tentative Labels for Nodes 2 & 3

# Shortest Path Problem - Numerical Example



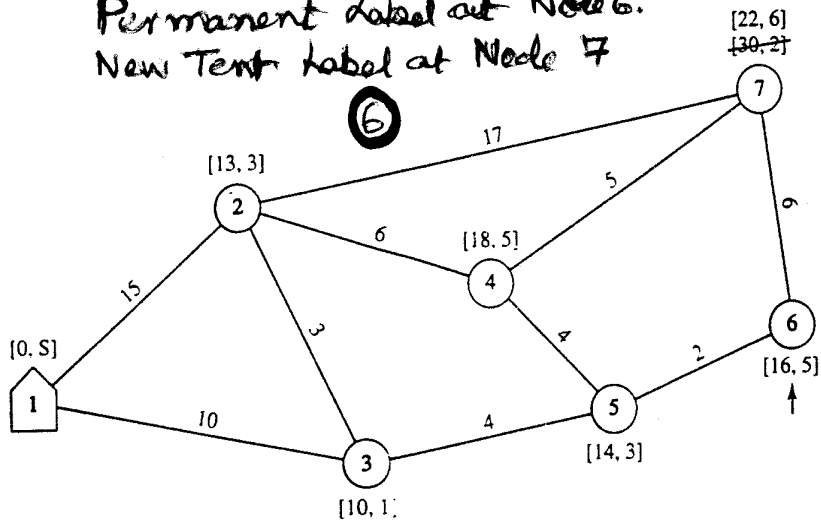
New Tentative Labels for Nodes 3, 5



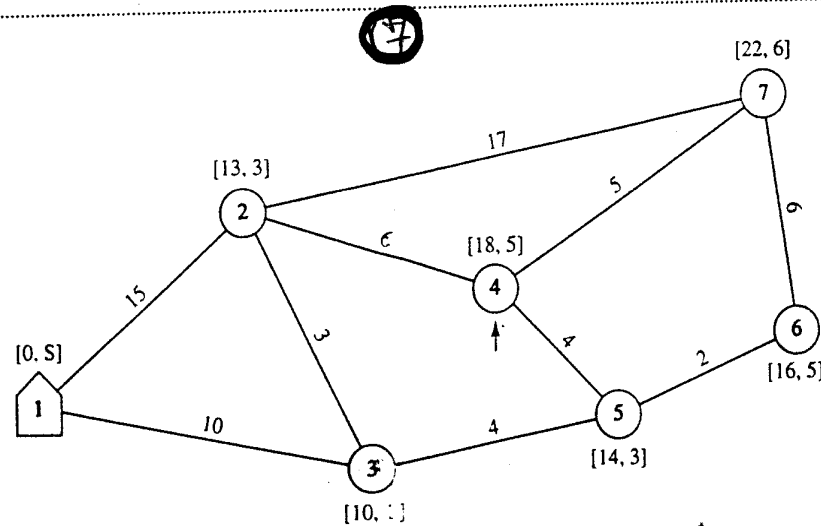
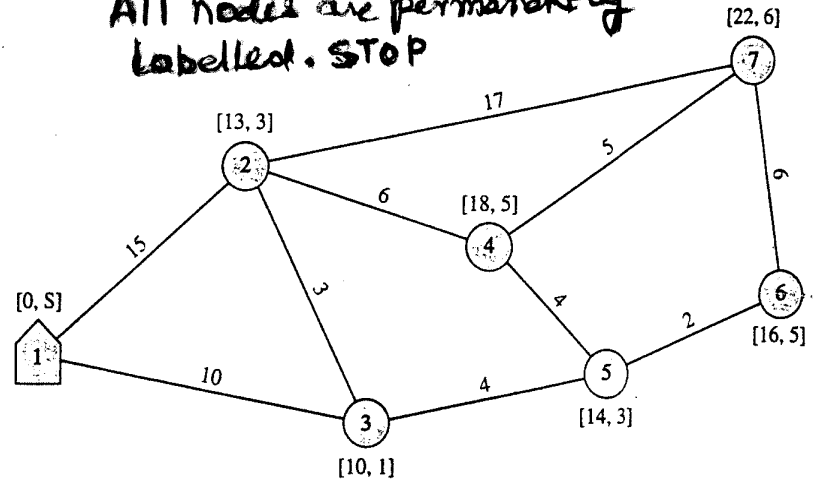
Permanent label at Node 5  
New Tentative labels at Nodes 4, 6, 7

# Shortest Path Problem - Numerical Example

Permanent label at Node 6.  
New Tent label at Node 7



All nodes are permanently  
labelled. STOP



Permanent label at Node 4

| Node | Shortest Route from Node 1 | Distance in Miles |
|------|----------------------------|-------------------|
| 2    | 1-3-2                      | 13                |
| 3    | 1-3                        | 10                |
| 4    | 1-3-5-4                    | 18                |
| 5    | 1-3-5                      | 14                |
| 6    | 1-3-5-6                    | 16                |
| 7    | 1-3-5-6-7                  | 22                |

FINAL RESULTS

## Application of Shortest Path Problem Finding Optimal Replacement Policies for Equipment

Equipment Maintenance and Replacement: As machines grow older, their repair and maintenance costs increase. However, replacing them requires capital investment. An important issue here is how often to buy and replace machines.

Assume that we have a 6 year planning horizon.

Let  $K_1, K_2, K_3, K_4, K_5, K_6$  represent the capital cost of purchasing the machines in years 1 through 6 respectively.

Let  $C_1, C_2, C_3, C_4$  and  $C_5, C_6$  represent the annual repair and maintenance costs on years 1 through 6 of the machine. Obviously, in most cases,  $C_1 \leq C_2 \leq C_3 \leq C_4 \leq C_5 < C_6$ .

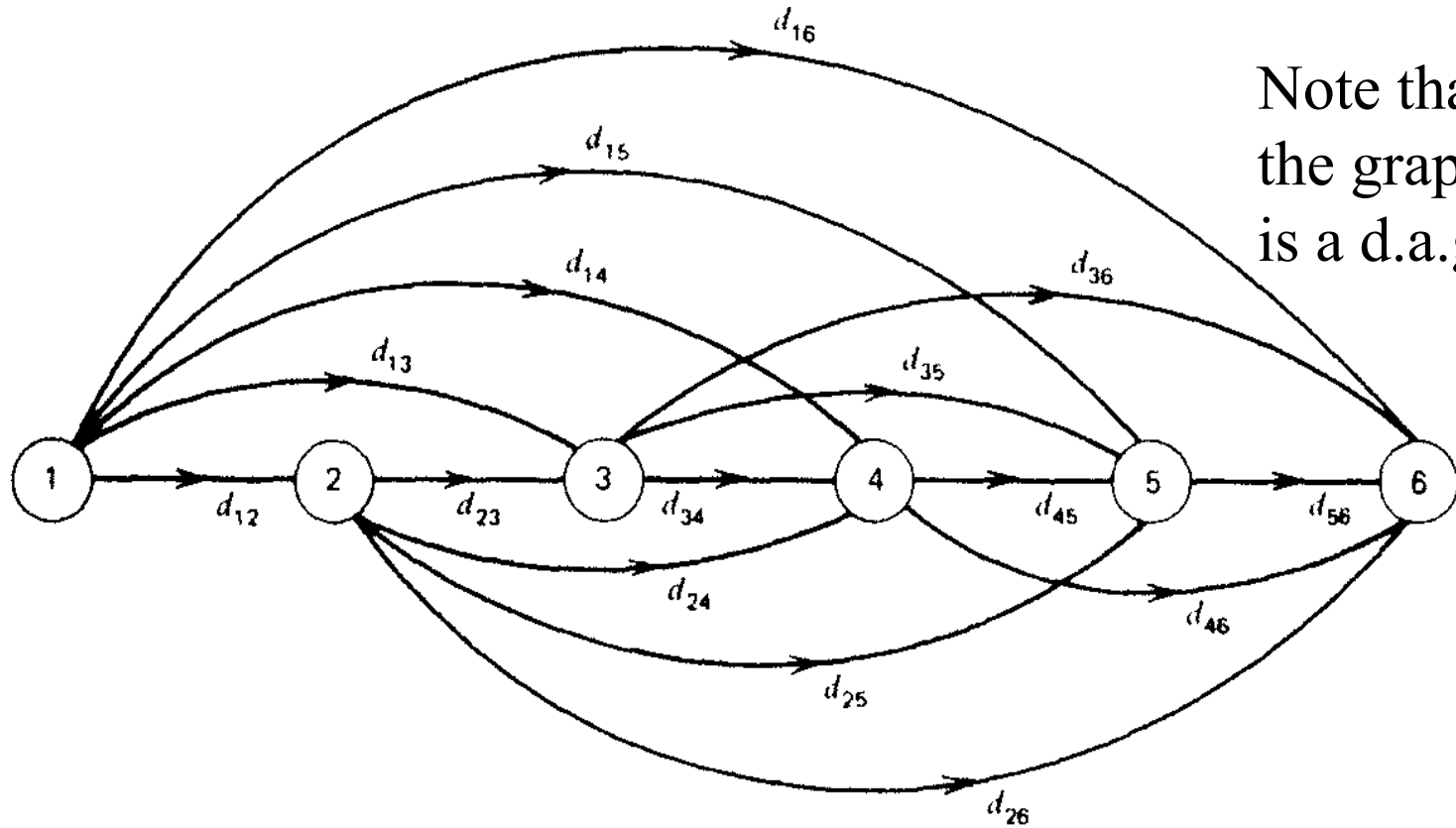
Let  $S_1, S_2, S_3, S_4, S_5, S_6$  represent the salvage value of the machine which has been used for 1, 2, 3, 4, 5 and 6 years respectively.

We can formulate the above problem as a shortest path problem as follows:

Draw a network with as many nodes as there are years in the planning horizon (i.e. 6 in this case - we will assume that node  $i$  represent year  $i$ )

From each node  $i$  draw a directed arc to every node  $j$  with the property that  $j > i$ . In other words, draw an arc from each node to every other node that has a higher number.

**Application of Shortest Path Problem  
Finding Optimal Replacement Policies for Equipment**



**Figure 3.12**  
**An equipment replacement problem as a shortest route problem.**

## Application of Shortest Path Problem Finding Optimal Replacement Policies for Equipment

Arc (i,j) represents the policy of buying a machine in year i and selling it in year j.  $d_{ij}$ , the “length” of this arc (i,j) represents the total costs of such a policy.

Hence,

$$d_{12} = K_1 - S_{(2-1)} + C_1 = K_1 - S_1 + C_1$$

$$d_{13} = K_1 - S_{(3-1)} + (C_1+C_2) = K_1 - S_2 + C_1 + C_2$$

.....

$$d_{24} = K_2 - S_{(4-2)} + (C_1+C_2) = K_2 - S_2 + C_1 + C_2$$

(and so on for each arc in the graph)

Using the idea above, compute the “length” of each arc in the Network above.

Then, find the Shortest Path in the network from Node 1 to Node 6 - this represents the cheapest replacement policy over the planning horizon.

For e.g., if the shortest path is (Node 1) --> (Node2) --> (Node 3) --> (Node 4) --> (Node5) --> (Node 6) , this represents a policy of buying and selling the machine every year.

If the shortest path is (Node 1) --> (Node4) --> (Node 6) , this represents a policy of buying in year 1, selling in year 4 and buying anew one at that time) and then selling it off in year 6.

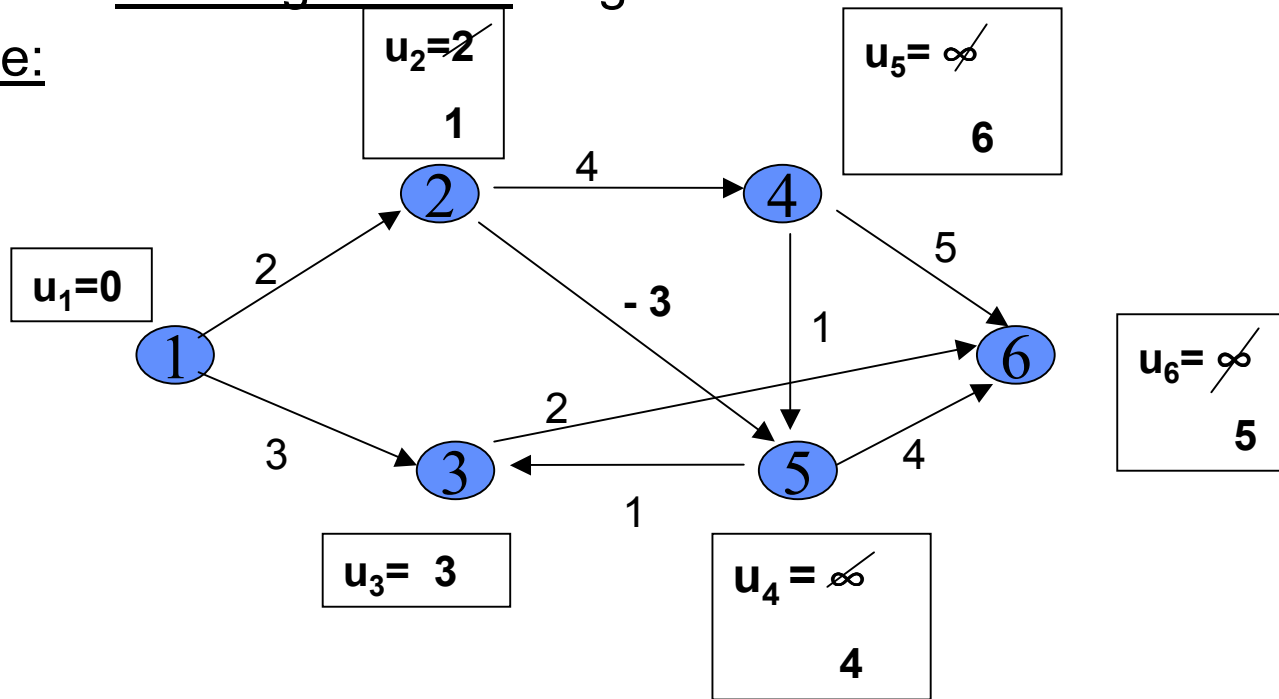
Additional feature: Add present value of money to these cost figures.

# Dijkstra's Algorithm

Limitations of Dijkstra's algorithm:

- 1) SPP defined on acyclic networks with arbitrary arc lengths.
- 2) SPP with non-negative arc lengths.

Example:



Here we find that shortest distance going from 1 to 2 is 2 but the shortest path is actually 1 → 3 → 5 → 2 i.e. 1.



## Dial's Implementation of Dijkstra's Algorithm

---

- In Dijkstra's algorithm, temporarily labeled nodes are treated as a non-ordered list, hence this is a bottleneck as all nodes have to be visited at each iteration to select node with minimum distance label.
- Dial's implementation tries to address this bottleneck by storing nodes with temporary distance labels in some sorted fashion.
- Maintains some "buckets", numbered  $0, 1, 2, \dots, nC$ .
- Bucket 'k' contains all temporarily labeled nodes with distance labels equal to 'k'.
- Nodes in each bucket are represented by "doubly linked list".
- Buckets numbered  $0, 1, 2, \dots, nC$  are checked sequentially until the first non-empty bucket is identified. Each node contained in the first non-empty bucket has the minimum distance label by definition.



## Dial's Implementation.....contd.

---

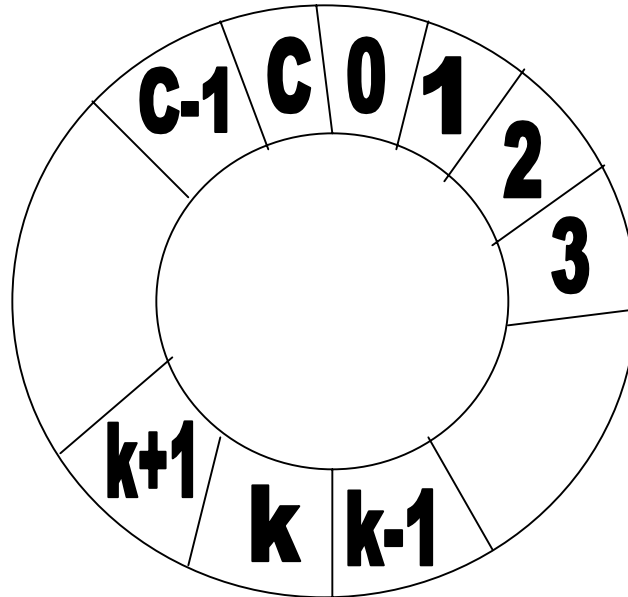
---

- One by one, these nodes with minimum distance label are permanently labeled and deleted from the bucket during the scanning process.
- Thus operations involving nodes include:
  - ✓ Checking if a bucket is empty
  - ✓ Adding a node to a bucket
  - ✓ Deleting a node from a bucket
- The position of a temporarily labeled node in the buckets is updated accordingly when the distance label of a node changes.
- Process repeated until all nodes are permanently labeled.
- Run time :  $O(m+nC)$ , pseudo-polynomial.
- Requires  $(nC+1)$  buckets in the worst case, but it has been proven that for a network with a maximum arc length of  $C$ , only  $(C+1)$  buckets are essential to maintain all temporarily labeled nodes.
- Potential disadvantage compared to  $O(n^2)$  implementation : will require large amount of storage space if 'C' is very large.



# Graphical Representation of Bucket Data Structure

---





## Heap Implementation of Dijkstra's Algorithm

---

- Uses heap data structures.
- A 'heap'  $H$  is a collection of objects with an associated 'key'.
- In this implementation,  $H$  is a collection of nodes with a temporary distance label, distance label being the key.
- Implementation mainly involves the following:
  - Creating an empty heap
  - Inserting a new node with a predefined key
  - Finding and returning a node of minimum key
  - Deleting that node
  - Reducing the key of a node from its current value to a value which is smaller than the key being replaced
- Different types of heaps are used:
  - Binary heap:  $O(m \log n)$ , slower than  $O(n^2)$  for dense networks
  - $d$ -heap:  $O(m \log_d n + n d \log_d n)$ ,  $d$  is a parameter  $\geq 2$
  - Fibonacci heap:  $O(m + n \log n)$



# Radix Heap Implementation

---

---

- Combination of  $O(n^2)$  implementation and Dial's implementation.
- Original implementation considers all temporarily labeled nodes (in one large bucket) and searches for node with minimum distance label.
- Dial's implementation uses buckets.
- Radix heap also uses buckets, which store many more nodes, e.g. bucket 'k' stores nodes with distance labels from  $100k$  to  $(100k+99)$ . All these labels make up the 'range' of the bucket.
- Cardinality of range is called width, e.g. width of above range is 100.
- Has three representative operations:
  - The arc scans in the distance updates, 'm' in number
  - The number of buckets scanned to place the nodes during the distance updates and redistribution of ranges, that are  $n \log(nC)$
  - The number of buckets whose ranges change during redistribution (and there are at most  $n \log(nC)$  such operations, but might vary from instance to instance)



# Radix Heap Implementation

---

---

- Thus has a worst case running time of  $O(m+n\log(nC))$ .
- Overcomes the disadvantage of Dial's implementation of using too many buckets.
- Also ranges of buckets are dynamically altered and nodes with temporary distance labels are reallocated to buckets in such a way that stores the minimum distance label in a bucket of width 1. This overcomes the need to search an entire bucket to find a node with the minimum distance label.



# Large Scale Shortest Paths

## Some References

---

---

- Parallel algorithms for solving aggregated shortest-path problems : *Computers & Operations Research*. v 26 n 10 1999. p 941-953.  
The problem of computing in parallel all pairs of shortest paths in a general large-scale directed network of  $n$  nodes is considered. A hierarchical network decomposition algorithm is provided that yields for an important subclass of problems with  $\log n$  savings in computation time over the traditional parallel implementation of Dijkstra's algorithm.
- Comparison of shortest path algorithms for large scale road networks : *Rairo-Recherche Operationnelle-Operations Research*. v 30 n 4 1996. p 333-357  
This paper reviews the algorithms available for computing shortest paths in large sparse graphs with non-negative weights. The results of a study comparing efficient PC-based implementations, executed on large road-like networks are presented. Some algorithms are up to 218 times faster than Dijkstra's classical algorithm when run on graphs with 15000 nodes.



# Label Correcting Algorithms

---

---

- Label setting are good for special network structures
  - ▶ Acyclic networks
  - ▶ Non-negative arc lengths
- General label-correcting algorithm is finite but requires  $O(n^2C)$  computations, where  $C$  is a bound on the max abs value of arc length.
- Another “balancing” strategy that considers arcs in a sequential wraparound fashion is  $O(nm)$
- Another implementation that gives priority to arcs emanating from nodes whose labels are most recently changed, “dequeue” implementation, has good practical (but poor worst-case) performance



# Label Correcting Algorithms

---

---

- We can also modify label-correcting algorithms to detect negative cycles
  - ▶ A nice feature is that the above does not add to the worst-case complexity of any label-correcting algorithm
- We will also address the all pair shortest path problem (SP3); two approaches:
  - ▶ Repeatedly applied label-setting for each node as source
    - Here a label-correcting procedure is run from an arbitrary node to redefine costs so they are non-negative
    - Then  $n$  single-source label-setting algorithms are applied
  - ▶ Floyd-Warshall  $O(n^3)$ , clever DP label-correcting algorithm



# LCA: Optimality Conditions

---

---

- LCA maintains a distance label  $d(j)$ ,  $\forall j \in N$  as an estimate of upper-bound of shortest path
- Necessary optimality condition is:

$$d(j) \leq d(i) + c_{ij}, \quad \forall (i,j) \in A$$

- It is also sufficient:

This can be shown by recursion over a path

$$s = i_1 - i_2 - \dots - i_k = j$$

$$d(i_k) \leq d(i_{k-1}) + c_{i_{k-1}i_k}$$

$$d(i_{k-1}) \leq d(i_{k-2}) + c_{i_{k-2}i_{k-1}}$$

$$d(i_1) = d(s) = 0$$



# LCA: Reduced Arc Lengths

- Let us define the reduced arc length  $c_{ij}^d$  of an arc  $(i,j)$  with respect to distance labels  $d(\cdot)$  as
- $c_{ij}^d = c_{ij} + d(i) - d(j)$
- The following properties are useful:
  - ▶ For any directed cycle  $W$ ,  $\sum_{(i,j) \in W} c_{ij}^d = \sum_{(i,j) \in W} c_{ij}$
  - ▶ For any directed path  $P$  from node  $k$  to node  $l$ ,  
$$\sum_{(i,j) \in P} c_{ij}^d = \sum_{(i,j) \in P} c_{ij} + d(k) - d(l)$$
  - ▶ If  $d(\cdot)$  represents the shortest path distances,  
$$c_{ij}^d \geq 0, \quad \forall (i,j) \in A$$
- In LP terms, reduced cost of primal variables are non-negative. The presence of a negative cycle implies the unboundedness of the primal problem and hence the infeasibility of the dual problem.



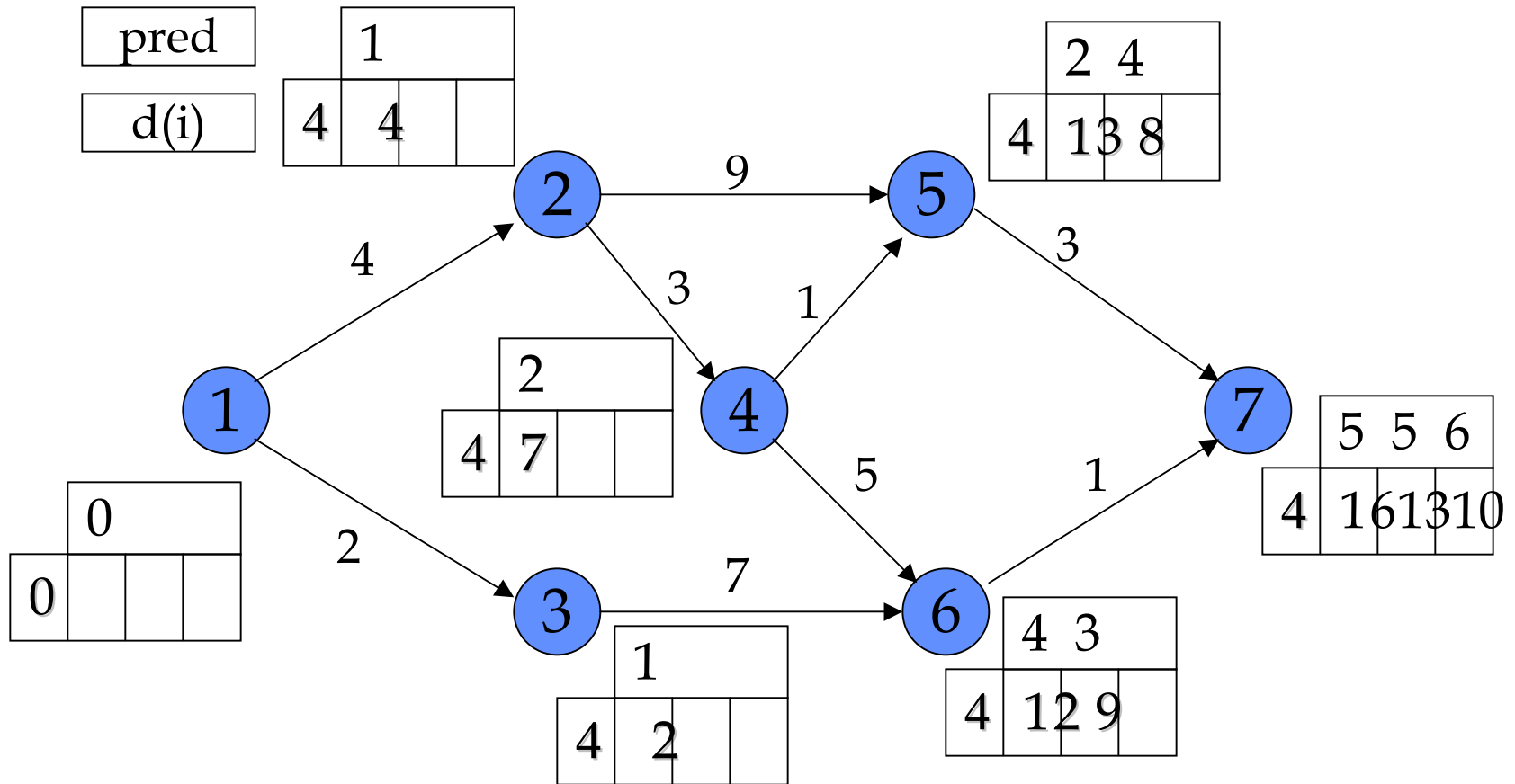
# Generic Label-Correcting Algorithms

---

- Initialization
  - ▶  $d(s)=0; \text{pred}(s)=0;$
  - ▶  $d(j) = \infty$  for each node except  $s$
- Procedure
  - ▶ While some arc  $(i, j)$  satisfies  $d(j) > d(i) + C_{ij}$  do
    - $d(j) = d(i) + C_{ij};$
    - $\text{pred}(j) = i;$



# Example of Generic LCA



Sequence of nodes traveled:-1,2,5,4,6,3,6,5



# Generic Label-Correcting Algorithm: Complexity

---

- Let us assume the data is integral
- Each finite  $d(j)$  is bounded from above by  $nC$  (because a path contains at most  $n-1$  arcs each of at most length  $C$ ) and from below by  $-nC$
- The LCA updates  $d(j)$  at most  $2nC$  times (because each update would be by at least 1)
- Number of distance labels updated is  $2n^2C$
- $O(n^2C)$



# Modified LCA

---

---

- The generic LCA does not specify any method for selecting an arc that violates the optimality condition
- Scanning the arc list is  $O(m)$  per iteration.
- The modified LCA reduces this workload to an average of  $O(m/n)$  per iteration
- The modified LCA maintains a node-list rather than arc-list. Rationale: whenever we update the distance label  $d(j)$  we will be adding all arcs  $A(j)$  to the list of arcs who might violate the optimality condition.
- So, the idea is to maintain a list of nodes with the property that if an arc  $(i,j)$  violates the optimality condition, LIST must contain node  $i$ . It is  $O(nmC)$ .



# Modified LCA

---

- Initialization

- ▶  $d(s)=0; \text{pred}(s)=0;$
- ▶  $d(j) = \infty$  for each node except  $s$
- ▶  $L = \{s\}$

- Procedure

- ▶ While  $L \neq \emptyset$  do
  - Remove  $i$  from  $L$
  - For each  $\text{arc}(i,j) \in A(i)$  do
    - If  $d(j) > d(i) + C_{ij}$  then
      - $d(j) = d(i) + C_{ij}; \text{pred}(j) = i$
      - If  $j$  not in  $L$  then add  $j$  to  $L$





# Special Implementations of MLCA

---

- FIFO label-correcting algorithm is  $O(nm)$ 
  - ▶ Best strongly polynomial implementation to solve SP2 with negative arc lengths.
  - ▶ Order arcs by their tail nodes so that they appear consecutively in the list.
  - ▶ Say in one pass of the algorithm  $d(i)$  is not changed
  - ▶ In the next pass we need not testing  $d(j) \leq d(i) + c_{ij}, \forall (i,j) \in A(i)$



# Special Implementations of MLCA

---

- Dequeue Implementation
  - ▶ Not polynomial (pseudopolynomial) but very efficient in practice
  - ▶ A dequeue always selects nodes from the front of the list, but can add either at front or at the rear.
  - ▶ Rationale: If some node  $i$  has previously appeared in the LIST, some nodes might have  $i$  as their predecessors.
  - ▶ So it is advantageous to examine  $i$  before its successors by adding  $i$  to the front of the list.



# Detecting Negative Cycles

---

- Negative cycle will keep decreasing the labels indefinitely
- The lower bound for a distance label is  $-nC$
- If the label for a node goes below  $-nC$  the algorithm terminates



# All pairs label correcting algorithm

---

- Initialization

- ▶ set  $d[i,j] = \infty$  for all  $[i,j] \in N \times N$
- ▶ set  $d[i,i] = 0$  for all  $i \in N$
- ▶ for each  $(i,j) \in A$  do
  - $d[i,j] = C_{ij}$

- Procedure

- ▶ while the network contains 3 nodes  $i, j$ , and  $k$  satisfying  $d[i,j] > d[i,k] + d[k,j]$  do
  - $d[i,j] = d[i,k] + d[k,j]$



# Floyd - Warshall Algorithm

---

- Initialization

- ▶ set  $d[i,j] = \infty$ ,  $pred[i,j] = 0$  for all  $[i,j] \in N \times N$
- ▶ set  $d[i,j] = 0$  for all  $i \in N$
- ▶ for each  $(i,j) \in A$  do
- ▶  $d[i,j] = C_{ij}$

- Procedure

- ▶ For each  $k = 1$  to  $n$  do
  - For each  $[i,j] \in N \times N$  do
    - If  $d[i,j] > d[i,k] + d[k,j]$  do
      - $d[i,j] = d[i,k] + d[k,j]$
      - $pred[i,j] = pred[k, j]$



# LSA or LCA ?

---

- Similarity
  - ▶ Iterative
  - ▶ Chooses the node with min distance label
- Differences
  - ▶ Distance label updation
  - ▶ Convergence Procedure
  - ▶ Applicability
  - ▶ Efficiency, flexibility and complexity
- Label setting a special case of label correcting