

**15.053**

**March 22, 2007**

## **Introduction to Networks**

**Announcement: no recitations this week**

**Comment on Excel**

## **Quotes for today**

**"A journey of a thousand miles begins  
with a single step."**

**-- Confucius**

**"You cannot travel the path until you  
have become the path itself"**

**-- Buddha**

## **Network Models**

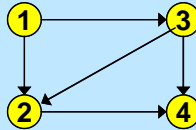
- **Optimization models that exhibit a very special structure.**
- **For special cases, this structure to dramatically reduce computational complexity (running time).**
- **First widespread application of LP to problems of industrial logistics**
- **Addresses huge number of diverse applications**
- **Today's lecture (first of 3): introductory material, Eulerian tours, the Shortest Path Problem**

## Notation and Terminology

**Note:** Network terminology is not (and never will be) standardized. The same concept may be denoted in many different ways.

Called:

- NETWORK
- directed graph
- digraph
- graph



Class Handouts (Ahuja, Magnanti, Orlin)

**Network**  $G = (N, A)$

**Node set**  $N = \{1, 2, 3, 4\}$

**Arc Set**  $\{(1,2), (1,3), (3,2), (3,4), (2,4)\}$

1

Also Seen

**Graph**  $G = (V, E)$

**Vertex set**  $V = \{1, 2, 3, 4\}$

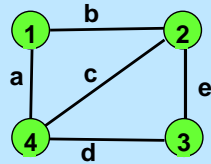
**Edge set:**

$E = \{1-2, 1-3, 3-2, 3-4, 2-4\}$

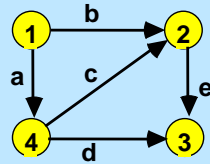
Personally, I find introducing all of the network notation and terminology to be boring. At the same time, it is critical to any further understanding in this area.

I use the notation in the text Network Flows by Ahuja, Magnanti, and Orlin. Tom Magnanti is currently Dean of Engineering at MIT. Ravi Ahuja and I have worked together more than 20 years, and have co-authored around 50 papers together as well as this book.

## Directed and Undirected Networks



An Undirected Graph



A Directed Graph

- **Networks are used to transport commodities**
  - physical goods (products, liquids)
  - communication
  - electricity, etc.
- **The field of Network Optimization concerns optimization problems on networks**

Networks are universal, and are applied to a wide range of situations. Almost any system can be described in large part by a network that describes relationships between parts of a system. This includes communication systems, electrical systems, manufacturing systems, social networks, and much more.

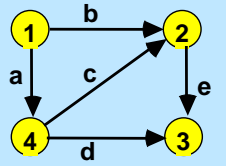
## Networks are Everywhere

- **Physical Networks**
  - Road Networks
  - Railway Networks
  - Airline traffic Networks
  - Electrical networks, e.g., the power grid
- **Abstract networks**
  - organizational charts
  - precedence relationships in projects
- **Others?**

## Overview:

- **Networks and graphs are powerful modeling tools.**
- **Most OR models have networks or graphs as a major aspect**
- **Next five lectures: we will develop models that are efficiently solvable.**
  - **Help form a toolkit for those problems that are harder to solve**
- **Next: representations of networks**
  - **A great insight from computer scientists: how data is represented is important to how it is used**

## The Adjacency Matrix (for directed graphs)



A Directed Graph

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	1	0

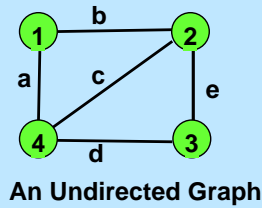
- Have a row for each node
  - Have a column for each node
  - Put a 1 in row i- column j if (i, j) is an arc
- What would happen if (4, 2) became (2, 4)?

8

A really nice description of a network for class work is a pictorial representation. But algorithms rely on other descriptions of networks.

A very simple representation of a network is an adjacency matrix. If a matrix has  $n$  nodes, the adjacency matrix is an  $n \times n$  matrix. There is a 1 in row  $i$  and column  $j$  if  $(i, j)$  is an arc.

## The Adjacency Matrix (for undirected graphs)



	1	2	3	4	degree
1	0	1	0	1	2
2	1	0	1	1	3
3	0	1	0	1	2
4	1	1	1	0	3

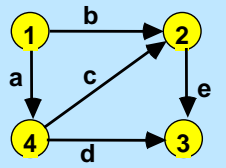
- Have a row for each node
  - Have a column for each node
  - Put a 1 in row i- column j if (i, j) is an arc
- The degree of a node is the number of incident arcs

9

The only difference between the adjacency matrix for directed and undirected networks is that in an undirected network, arc (i, j) is the same as arc (j, i) and so each arc corresponds to two 1's of the adjacency matrix, one in row i and column j, and one in row j and column i.

The degree of a node is the number of incident arcs and is a useful concept. The degree of node j is equal to the row sum of row j of the adjacency matrix.

## Representation of arc lists (for directed graphs)



A Directed Graph

1: (1,2), (1,4)

2: (2,3)

3:  $\emptyset$

4: (4,2), (4,3)

- Create a list arcs for each node

- There are lots of very similar variants of this type of representation

10

The difficulty with adjacency matrices is that they are inefficient for large networks. For example, suppose that a network has a million nodes. (This actually happens in practice.) Then the adjacency matrix has one trillion entries. Just writing the adjacency matrix takes too much time for any algorithm.

Instead, one can write the list of nodes (a million of them in our example), and follow it with a list of arcs (perhaps 10, million arcs). This is manageable, and incredibly more efficient as a representation than the adjacency matrix. To give a sense for how inefficient the adjacency matrix is in this example, note that only 20 million entries out of 1 trillion are non-zero, because there are two entries for each arc. This means that only 1 out of every 50,000 entries is non-zero. But the adjacency matrix would include all entries.

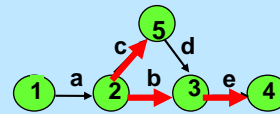
There are different ways of writing the list of arcs that may be more efficient for different applications. But this is too subtle a point to pursue. We just conceptualize this description as a list of nodes and arcs.

## On network representations

- Each representation has its advantages
  - Major purpose of a representation
    - efficiency in algorithms
    - ease of use
- Next: definitions for networks

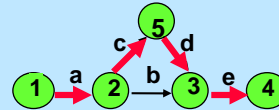
**Path:** Example: 5, 2, 3, 4.  
(or 5, c, 2, b, 3, e, 4)

- No node is repeated.
- Directions are ignored.



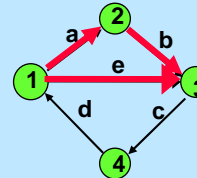
**Directed Path.** Example: 1, 2, 5, 3, 4  
(or 1, a, 2, c, 5, d, 3, e, 4)

- No node is repeated.
- Directions are important.



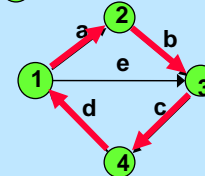
**Cycle (or circuit or loop)**

1, 2, 3, 1. (or 1, a, 2, b, 3, e)  
•A path with 2 or more nodes, except that the first node is the last node.  
•Directions are ignored.



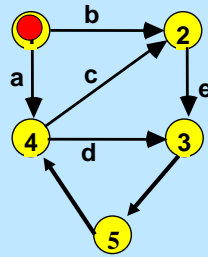
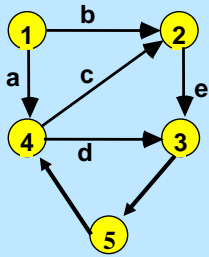
**Directed Cycle:** (1, 2, 3, 4, 1) or  
1, a, 2, b, 3, c, 4, d, 1

- No node is repeated, except that the first node is the last node.
- Directions are important.



If you think of nodes as being intersections of a city and think of arcs as roads, then one may also think about paths and cycles. In our definitions, we do not permit paths to repeat nodes. We also do not permit cycles to repeat nodes, except that the first and last node in the representation of a cycle is the same.

## Walks



**Walks** are paths that can repeat nodes and arcs

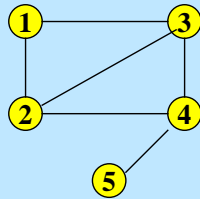
Example of a **directed walk**: 1-2-3-5-4-2-3-5

A walk is **closed** if its first and last nodes are the

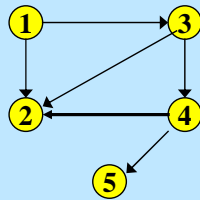
A closed walk is a cycle except that it can repeat nodes and arcs.

A walk is permitted to repeat nodes and arcs. A closed walk starts and ends at the same node.

## More terminology



An undirected network is **connected** if every node can be reached from every other node by a path



A directed network is **connected** if it's undirected version is connected.

This directed graph is connected, even though there is no directed path between 2 and 5.

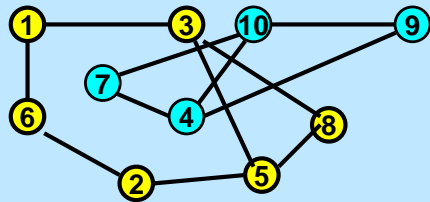
14

Again, if we view the network as a road network, we expect to be able to get from any node to any other node via a path in the network. If there is a path between all pairs of nodes, then the network is connected. Note that for the definition of connected, one is permitted to use directed arcs in the wrong direction, sort of like drivers in Boston go down one way streets the wrong way.

Of course, not all networks are connected. If one restricts attention to road networks, there is no path from Boston, MA to London, England.

## On connectivity

There are simple efficient procedures for determining if a graph is connected.



● Here is a graph with two components, that is maximally connected subgraphs.

We will not describe these algorithms, but will do a more general algorithm later in this lecture

15

If a graph is not connected, then the graph can be described as the union of connected subgraphs, called the *connected components*. In the example above, the blue and the yellow subgraphs are the connected components. Note that there is no arc incident to both a yellow node and a blue node.

Finding connected components is important. We will not describe the algorithm, but will describe a more general algorithm later in this lecture.

## The Bridges of Koenigsberg: Euler 1736

- **“Graph Theory” began in 1736**
- **Leonard Euler**
  - **Visited Koenigsberg**
  - **People wondered whether it is possible to take a walk, end up where you started from, and cross each bridge in Koenigsberg exactly once**
  - **Generally it was believed to be impossible**

16

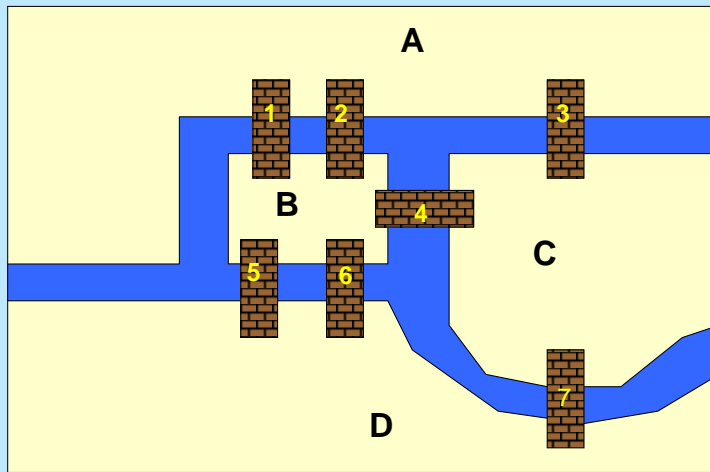
Incidentally, I have a distant connection to Leonard Euler. If you go the math genealogy website starting from me and click on advisors, you will eventually reach the following mathematicians: Lagrange, Euler, Johann Bernoulli, Jacob Bernoulli, and Leibnitz. This means that I am an academic descendent of all of them, through a direct line of advisors. To start go to:

<http://www.genealogy.ams.org/html/id.phtml?id=68357>

Choose Herbert Robbins as the advisor of Cyrus Derman.

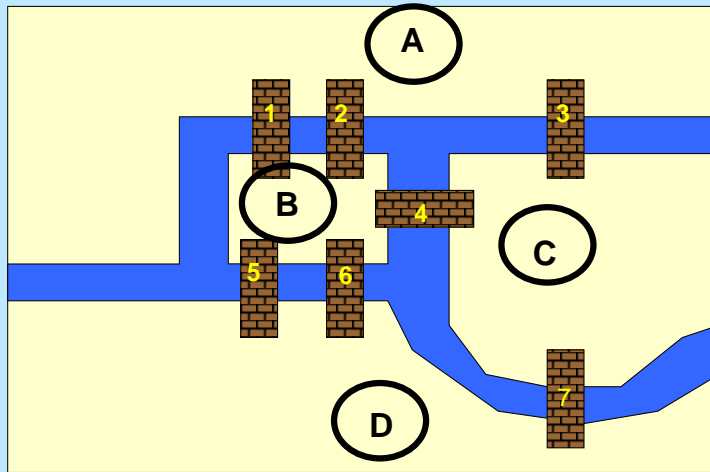
## The town of Koenigsberg

## The Bridges of Koenigsberg: Euler 1736



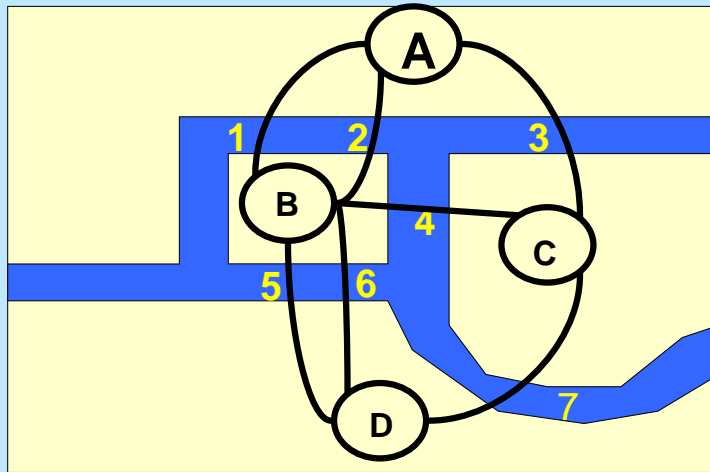
Is it possible to start in A, cross over each bridge exactly once, and end up back in A?

## The Bridges of Koenigsberg: Euler 1736



**Conceptualization: Land masses are nodes**

## The Bridges of Koenigsberg: Euler 1736

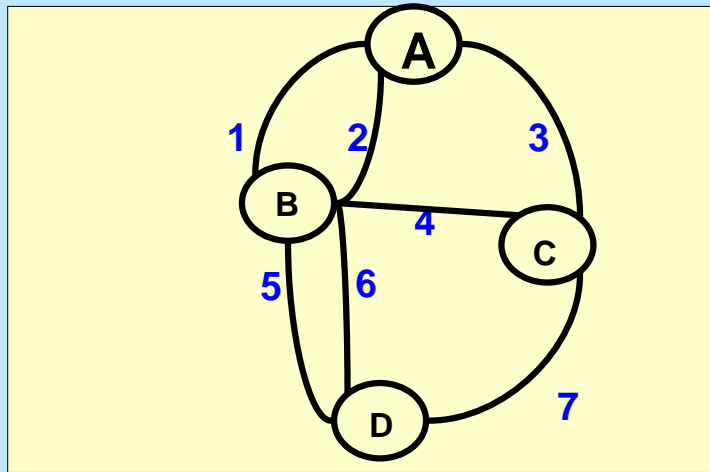


**Conceptualization: Bridges are arcs**

20

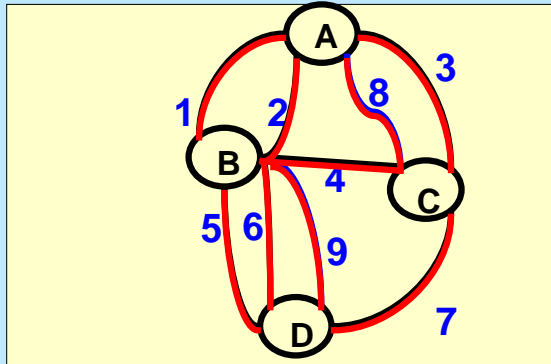
In retrospect, it does not seem like a major conceptual leap to associate a single node with each land mass. Nevertheless, it was very important. Because of this, Euler is credited with the founding of the field of graph theory.

## The Bridges of Koenigsberg: Euler 1736



Translation to graphs or networks: Is there a walk starting at A and ending at A and passing through each arc exactly once? Why isn't there such a walk?

## Adding two bridges creates such a walk



Here is the walk.

A, 1, B, 5, D, 6, B, 4, C, 8, A, 3, C, 7, D, 9, B, 2, A

**Note:** the number of arcs incident to B is twice the number of times that B appears on the walk.

22

We observe that any closed walk that passes through each arc exactly once must enter and leave each node the same number of times. This means that every node has even degree.

**Eulerian cycle: a closed walk that passes through each arc exactly once**

- **Degree of a node = number of arcs incident to the node**
- **Necessary condition: each node has an even degree.**
- **Why necessary? The degree of a node  $j$  is twice the number of times  $j$  appears on the walk (except for the initial and final node of the walk.)**

**Theorem.** *A graph has an eulerian cycle if and only if the graph is connected and every node has even degree.*

23

Another obvious condition for an Eulerian walk is that the graph is connected. Obviously, one cannot find a closed walk through each arc unless the graph is connected, assuming that we ignore nodes with no incident arcs. Nodes of degree 0 do not need to be visited at all in an Eulerian walk.

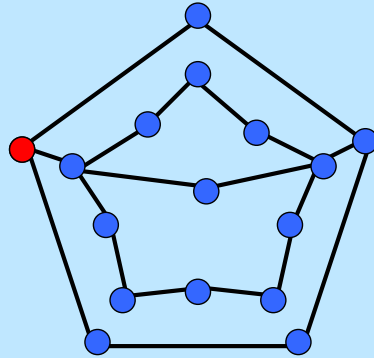
Euler showed that these two necessary conditions are also sufficient. That is, if every node has even degree and if the graph is connected, then there is an Eulerian cycle.

**Eulerian path**: a walk that is not closed and passes through each arc exactly once

**Theorem.** A graph has an eulerian path if and only if exactly two nodes have odd degree and the graph is connected.

An Eulerian path is defined similarly. An Eulerian path is a non-closed walk that passes through every arc exactly once.

**Exercise: Does the graph below have an Eulerian path or cycle? If so, find it.**



## Eulerian cycles

- Eulerian cycles and extensions are used in practice
- Mail Carrier routes:
  - visit each city block at least once
  - minimize travel time
  - other constraints in practice?
- Trash pickup routes
  - visit each city block at least once
  - minimize travel time
  - other constraints in practice?

**The Traveling  
Salesman  
Problem** : find a  
tour that visit all  
cities and  
minimizes the  
total distance  
traveled.

27

Another important type of walk is one that passes through each node exactly once. We call a cycle that passes through every node a Hamiltonian Cycle after the mathematician William Rowan Hamilton. The Traveling Salesman Problem is to find a Hamiltonian Cycle of minimum length. It is a very well studied problem. This picture is taken from a site devoted to the Traveling Salesman Problem. <http://www.tsp.gatech.edu/>

## **Mental Break**

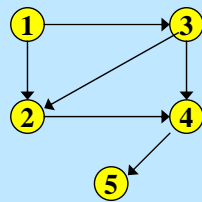
### **Top 10 Reasons Why God Can't Get University Tenure.**

- 10. Researchers can't replicate His results.**
- 9. He failed to get permission from the ethics board to use human subjects.**
- 8. After one of His experiments failed, He drowned the subjects.**
- 7. He had His son teach the class.**
- 6. He didn't show up for lectures, and He told students to "Read the Book."**

## **Mental Break**

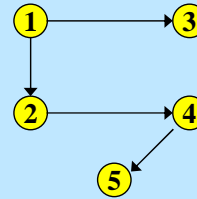
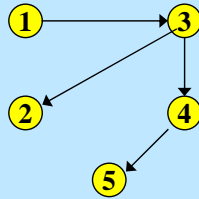
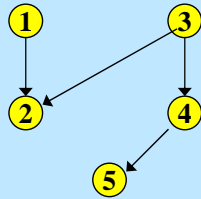
- 5. Although He listed only ten requirements, his exam was far too hard and everyone failed.**
- 4. He expelled his first two students for eating in class.**
- 3. His publications did not cite earlier authorities.**
- 2. He referred to his favorite students as the “Chosen People.”**
- 1. He hasn’t published anything in 2000 years?**

## More Definitions



A network is ***connected*** if every node can be reached from every other node by a path

A ***spanning tree*** is a connected subset of a network including all nodes, but containing no cycles.

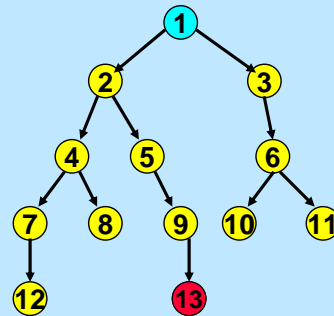
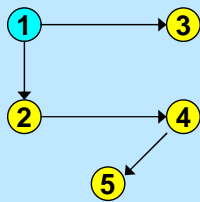


30

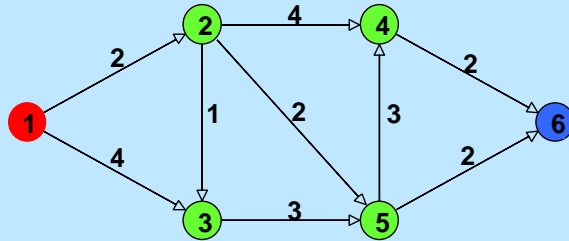
Spanning trees turn out to be surprisingly important. They are used widely as data structures in computer software. In addition, they are closely connected to a variety of network optimization problems.

## More on Trees

- An **out tree** is a spanning tree in which every node has exactly one incoming arc except for the root.
- **Theorem.** In an out tree, there is a directed path from the root to all other nodes. (All paths come out of the root).
- One can find the path by starting at the end and working backwards.



## The Shortest Path Problem



What is the shortest path from a source node (often denoted as  $s$ ) to a sink node, (often denoted as  $t$ )?

What is the shortest path from node 1 to node 6?

**Assumptions for this lecture:**

1. There is a path from the source to all other nodes.
2. All arc lengths are non-negative

32

There are several different algorithms for the shortest path problem. Here we will focus on one due to Dijkstra. For convenience we will assume that the graph is connected.

Also, the algorithm relies on the assumption that all arc lengths are non-negative. This assumption is widely true in practice. But there are applications of the shortest path problem for which this assumption does not hold.

## Shortest Path Problem

- **Where does it arise in practice?**
  - **Common applications**
    - shortest paths in a vehicle (Navigator)
    - shortest paths in internet routing
    - shortest paths around MIT
  - **and less obvious applications, as in the course readings (in documents under lecture notes)**
- **How will we solve the shortest path problem?**
  - **Dijkstra's algorithm**

## A surprising application: Finding optimal paragraph layouts

TeX optimally decomposes paragraphs by selecting the breakpoints for each line optimally. It has a subroutine that computes the attractiveness  $F(i,j)$  of a line that begins at word  $i$  and ends at word  $j-1$ . How can one use  $F(i,j)$  to create a shortest path problem whose solution will solve the paragraph problem?

### Step 1. Assign numbers to the words

TeX optimally decomposes paragraphs by .....

1      2            3            4            5

will solve the paragraph problem. "end"

48 49 50            51            52            53

34

I like this application because it is not at all obvious.

If you ever use LaTeX, it lays out paragraphs optimally using an algorithm of this type.

## On the values of $F( )$

The better the look of the line, the lower the value of  $F$ .  
The values below are illustrative, but are artificial.

TeX	optimally	$F(1, 3) = 50$
1	2	

TeX	optimally	decomposes	$F(1, 4) = 25$
1	2	3	

TeX	optimally	decomposes	paragraphs	$F(1, 5) = 3$
1	2	3	4	

TeX	optimally	decomposes	paragraphs	by	$F(1, 6) = 1$
1	2	3	4	5	

35

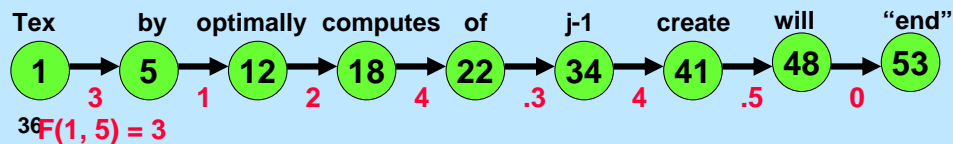
$F(1, j) = 10,000$  for  $j > 6$ .

The key point is that the values of  $F$  are selected carefully so that a “beautiful line” is given a very low number, and an “ugly line” is given a high number. The shortest path problem will select a paragraph so as to minimize the sum of the ugliness of the lines.

## The shortest path formulation

Create a shortest path problem with nodes  $1, 2, \dots, n+1$  where the cost of arc  $(i, j)$  is  $f(i, j)$ .

TeX optimally decomposes paragraphs by selecting the breakpoints for each line optimally. It has a subroutine that computes the attractiveness  $F(i, j)$  of a line that begins at word  $i$  and ends at word  $j-1$ . How can one use  $F(i, j)$  to create a shortest path problem whose solution will solve the paragraph problem?

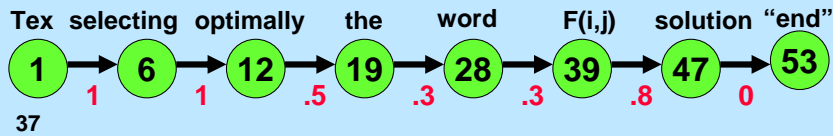


Every path from 1 to 53 corresponds to a laying out of the paragraph. The length of the path is the sum of the  $F$  values and thus is the sum of the ugliness of the lines of the paragraph. The shortest path corresponds to the least ugly paragraph.

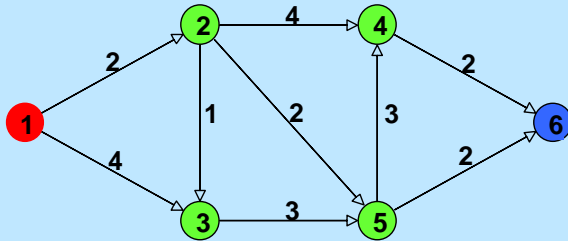
## The shortest path formulation

Each path from 1 to 53 corresponds to a paragraph layout. The most beautiful layout is the shortest path.

TeX optimally decomposes paragraphs by selecting the breakpoints for each line optimally. It has a subroutine that computes the attractiveness  $F(i,j)$  of a line that begins at word  $i$  and ends at word  $j-1$ . How can one use  $F(i,j)$  to create a shortest path problem whose solution will solve the paragraph problem?  
“end”



## Dijkstra's Algorithm for the Shortest Path Problem



Exercise with your partner. Find the shortest paths by inspection.

**Exercise:** find the shortest path from node 1 to all other nodes. Keep track of distances using labels,  $d(i)$  and each node's immediate predecessor,  $\text{pred}(i)$ .

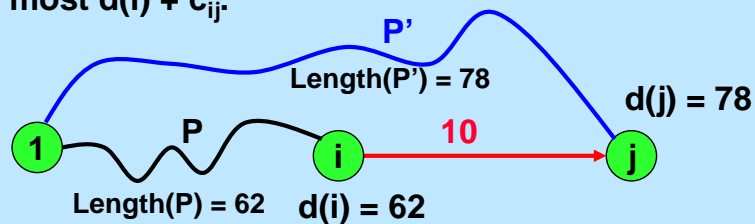
$$d(1) = 0, \text{pred}(1) = 0;$$

$$d(2) = 2, \text{pred}(2) = 1$$

Find the other distances, *in order of increasing distance from node 1*.

## Key observations

- Suppose that  $d(i)$  is the length of some path from node 1 to node  $i$ .
- Suppose that there is an arc  $(i, j)$  of length  $c_{ij}$ .
- Then there is a path from node 1 to node  $j$  of length at most  $d(i) + c_{ij}$ .



**In this case, there is a path from 1 to j of length 72. We can reduce  $d(j)$  to 72.**

39

All shortest path algorithms rely on the following elementary observation. If there is a path from node 1 to node  $i$  of length  $d(i)$  and if there is an arc  $(i, j)$  of length  $c_{ij}$ , then there is a walk from node 1 to node  $j$  of length  $d(i) + c_{ij}$ . Note that we do not assume that it is a path in that it could possibly repeat node  $j$ . (I am assuming that node 1 is the source node).

The algorithm will proceed as follows. It will let  $d(j)$  be the length of some path from node 1 to node  $j$ . Initially,  $d(1) = 0$ , and  $d(j) = \text{infinity}$  for all other nodes.

If it discovers a path of length  $d(i) + c_{ij}$  from node 1 to node  $j$ , and if  $d(i) + c_{ij}$  is less than  $d(j)$ , then the algorithm replaces  $d(j)$  by the value  $d(i) + c_{ij}$ .

There is a little more to the algorithm than this, but this is the essence.

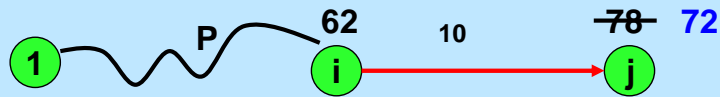
## A Key Procedure in Shortest Path Algorithms

- At each iteration  $d(j)$  is the length of some path from node 1 to node  $j$ . (If no path is known, then  $d(j) = \infty$ )

### Procedure Update(i)

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

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



Up to this point, the best path from 1 to  $j$  had length 78  
But  $P, (i,j)$  is a path from 1 to  $j$  of length 72.

Procedure update is used to decrease  $d(j)$  whenever a shorter path is discovered from node 1 to node  $j$ .

## Dijkstra's Algorithm

**begin**

$d(s) := 0$  and  $\text{pred}(s) := 0$ ;  
 $d(j) := \infty$  for each  $j \in N - \{s\}$ ;  
 $\text{LIST} := \{s\}$ ;

**while**  $\text{LIST} \neq \emptyset$  **do**

**begin**

let  $d(i) := \min \{d(j) : j \in \text{LIST}\}$ ;  
remove node  $i$  from  $\text{LIST}$ ;  
update( $i$ )  
if  $d(j)$  decreases from  $\infty$ ,  
place  $j$  in  $\text{LIST}$

**end**

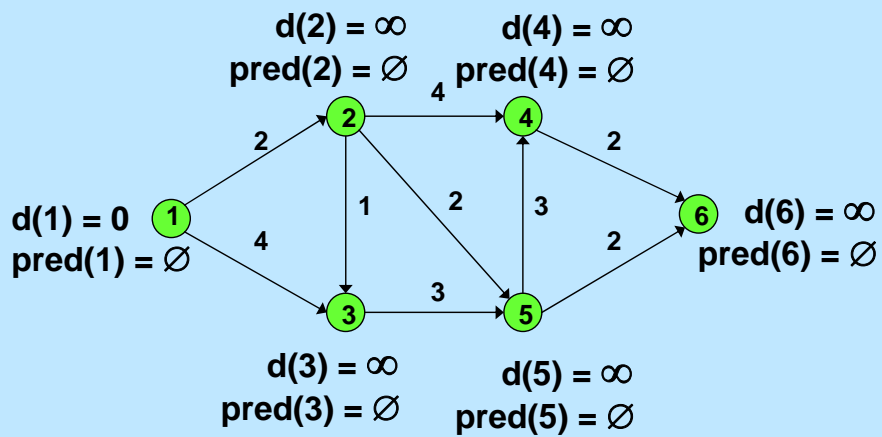
**end**

Initialize distances.

$\text{LIST}$  = set of  
temporary nodes

Select the node  $i$   
on  $\text{LIST}$  with  
minimum distance  
label, and then  
update( $i$ )

## Initialize

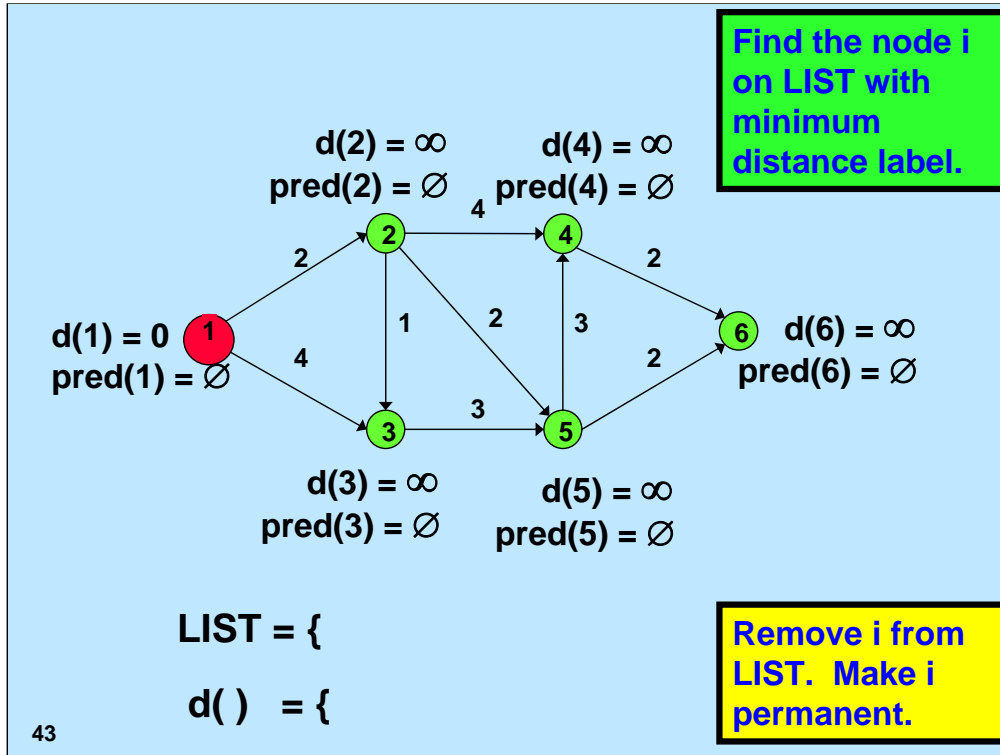


**LIST** = {1,

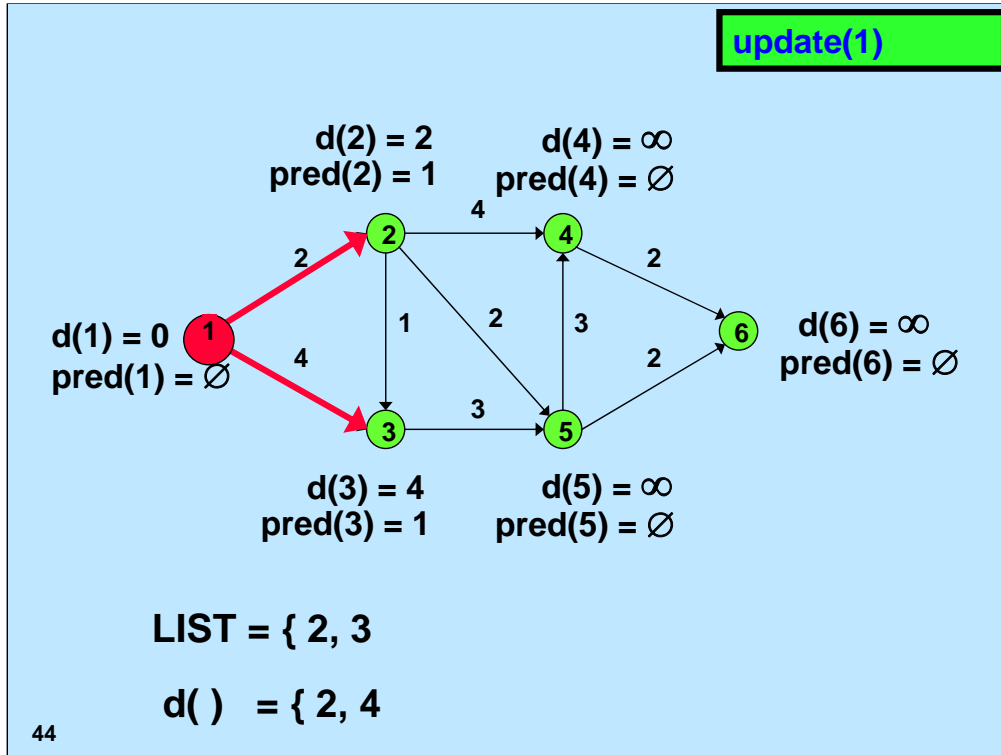
**d( )** = {0,

The algorithm initializes  $d(1)$  as 0 and  $d(j) = \text{infinity}$  otherwise.

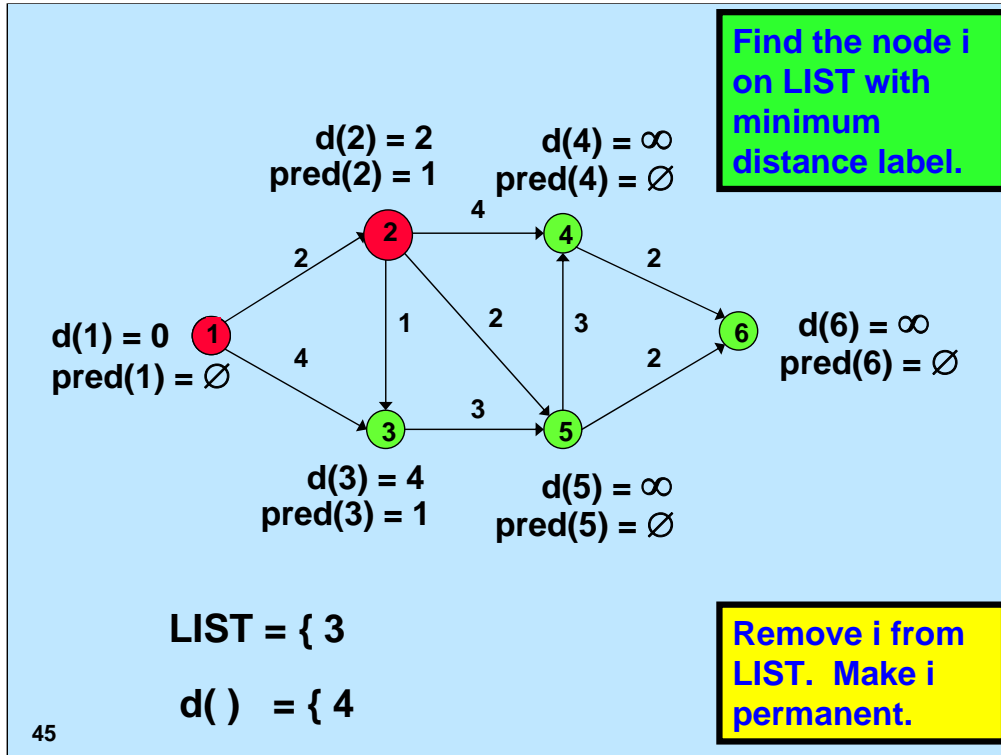
LIST is a list of all nodes for which  $d(j)$  is finite, and for which  $d(j)$  has not been labeled as “permanent”. Once a node  $j$  is labeled as permanent,  $d(j)$  will never again change. So, we are only permitted to label  $j$  as permanent if we know that  $d(j)$  is the length of the shortest path from node 1 to node  $j$ .



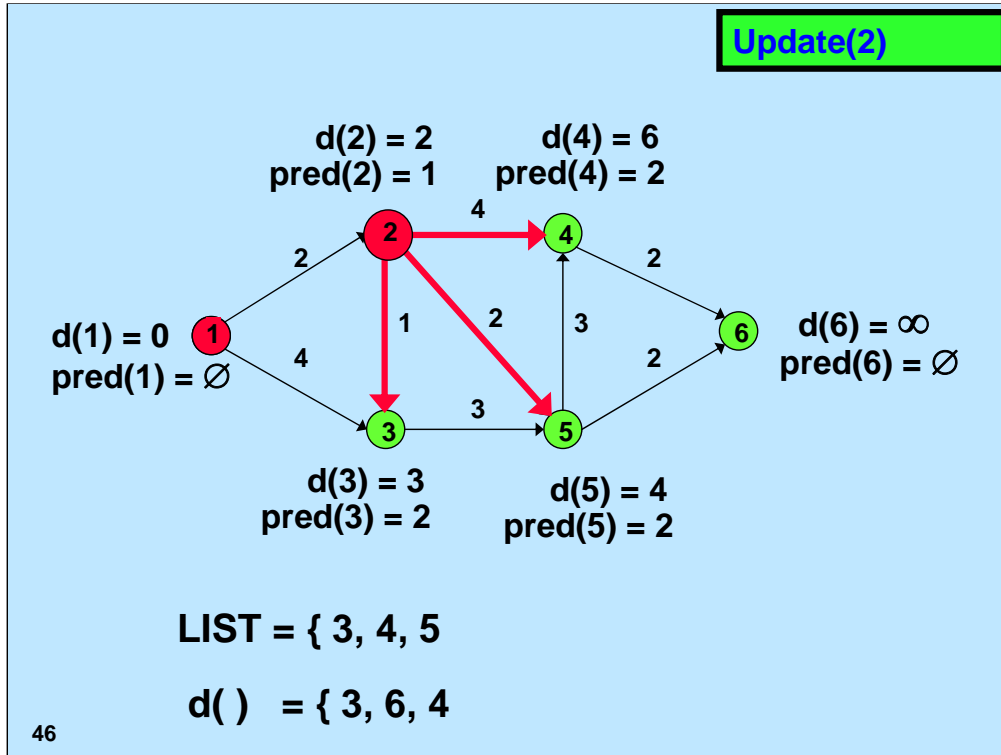
The algorithm finds the node  $j$  on LIST for which  $d( )$  is minimum. It turns out that this node  $j$  can be made permanent. And the algorithm does make it permanent.



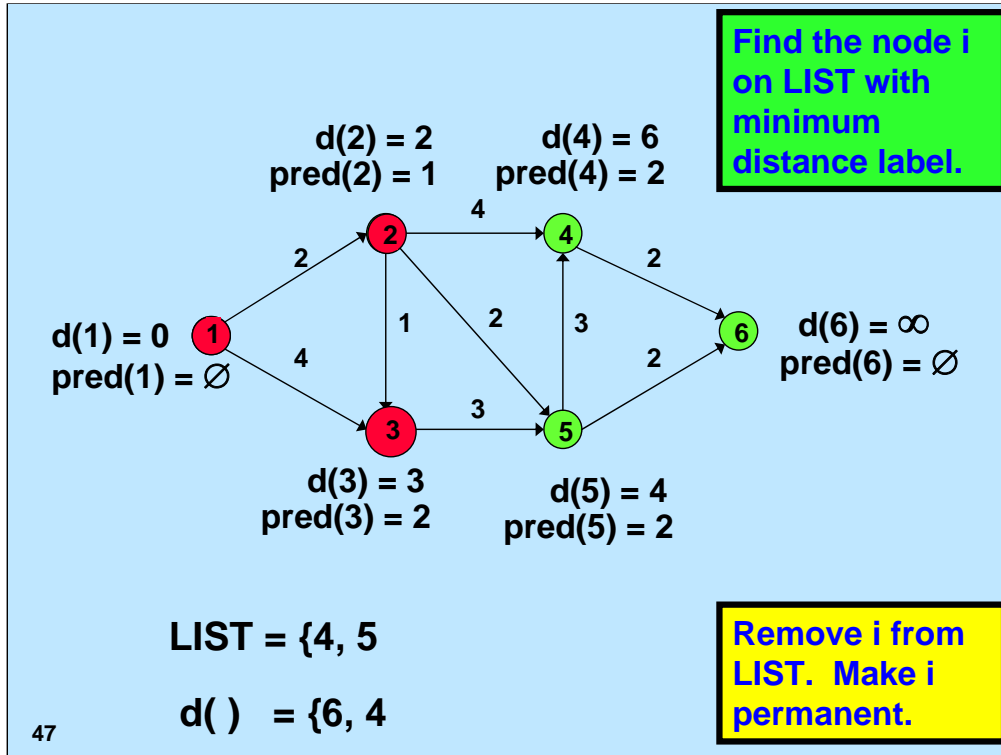
The algorithm then performs an update step on the permanently labeled node by scanning all arcs out of the node and then updating distance labels. Any node not on LIST that has a finite distance label after this scanning is added to LIST.



The find min operations is repeated by scanning nodes of LIST. A new node is made permanent.

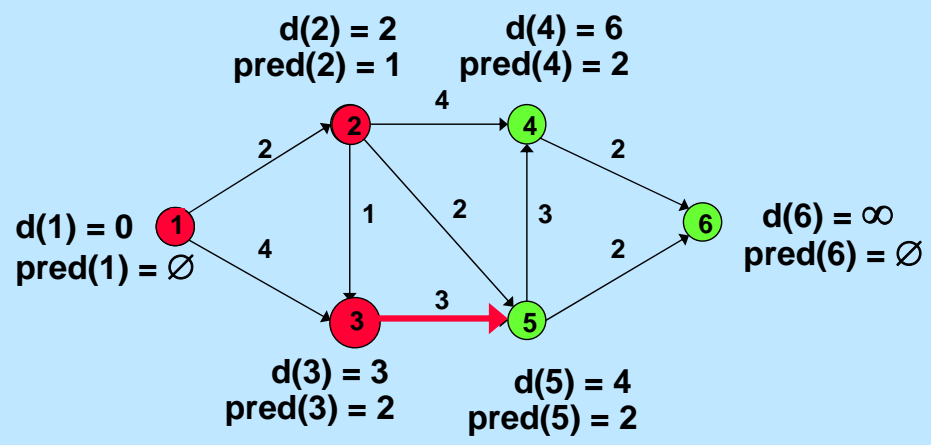


Arcs are scanned from the new permanent node.



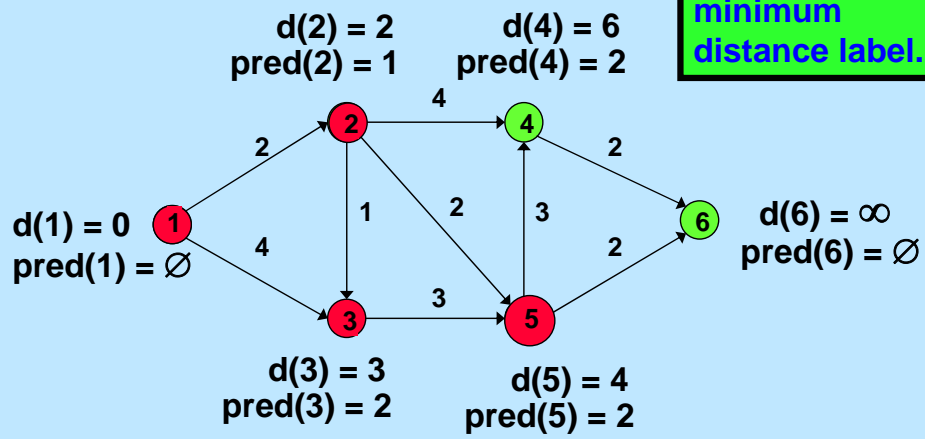
The algorithm alternates between the find min operation and the Update operation. It continues until all nodes are made permanent, at which point it stops with the shortest distances from node 1 to all other nodes.

Update(3)



LIST = {4, 5}

d( ) = {6, 4}

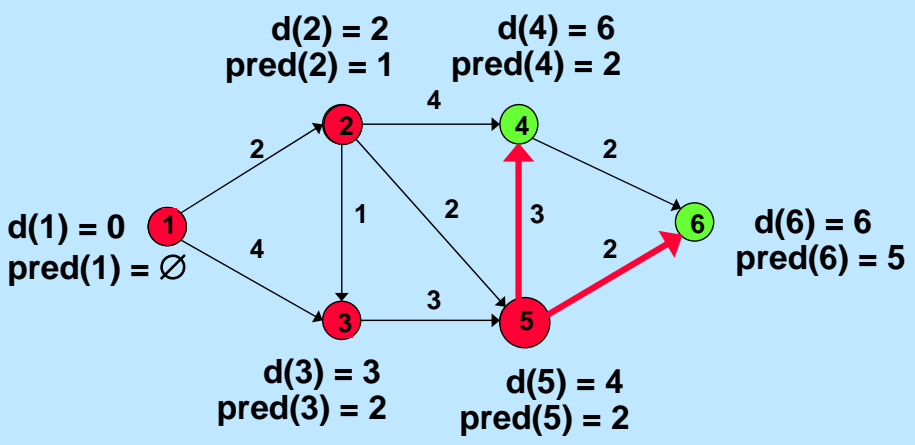


Find the node  $i$  on LIST with minimum distance label.

LIST = {4  
 $d( ) = \{6$

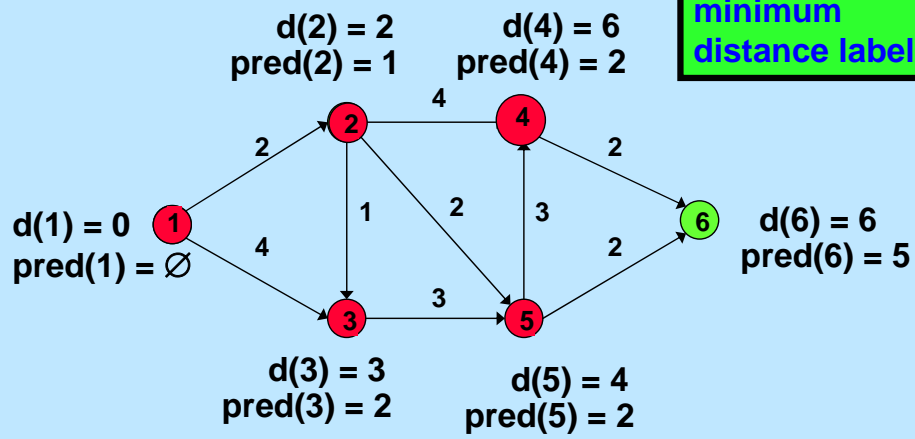
Remove  $i$  from LIST. Make  $i$  permanent.

Update(5)



LIST = {4, 6}

$d( ) = \{6, 6\}$



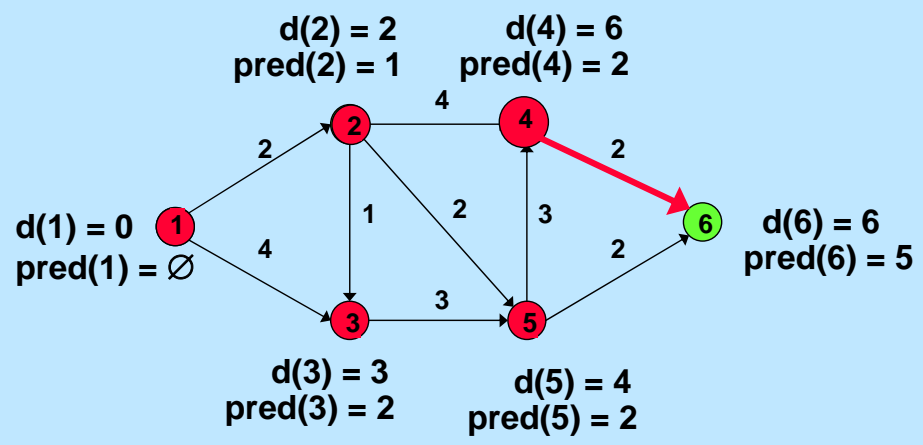
Find the node  $i$  on LIST with minimum distance label.

LIST = {6}

$d(\ ) = \{6$

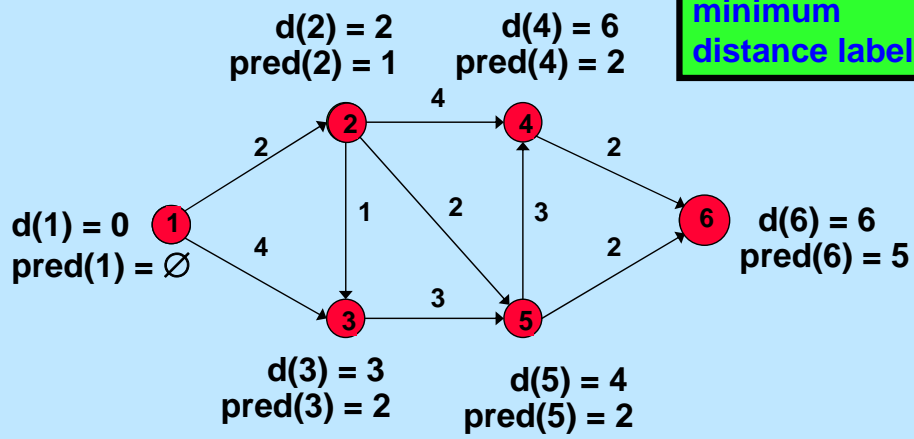
Remove  $i$  from LIST. Make  $i$  permanent.

Update(4)



LIST = {6}

$d() = \{6$



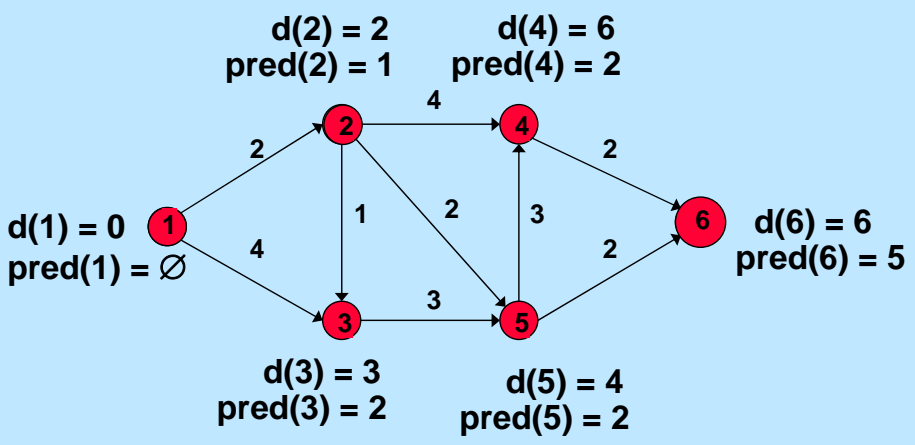
Find the node  $i$  on LIST with minimum distance label.

LIST = {

$d(\ ) = \{$

Remove  $i$  from LIST. Make  $i$  permanent.

Update(6)

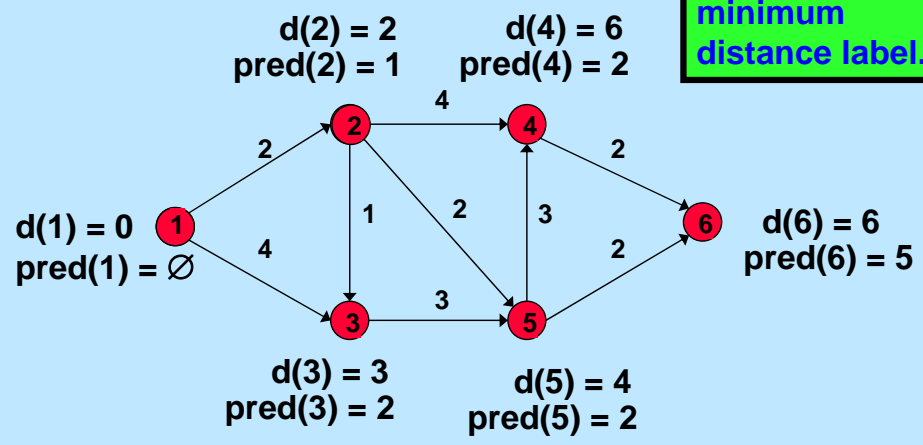


LIST = {

d( ) = {

Node 6 has no outgoing arcs.

Find the node  $i$  on LIST with minimum distance label.

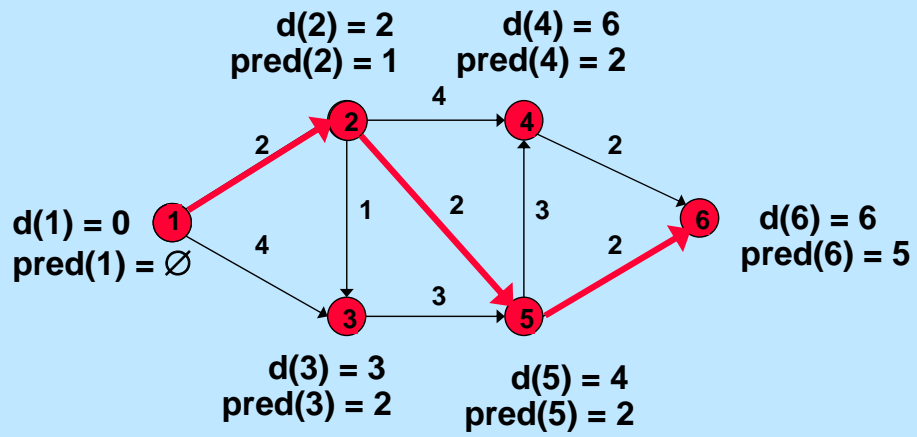


LIST = {

$d( ) = \{$

LIST =  $\emptyset$ . The algorithm ends.

### The shortest path from node 1 to node 6.

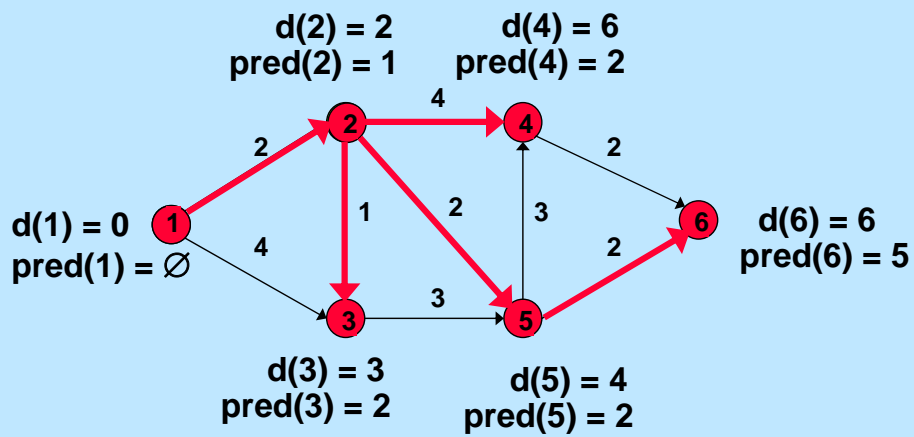


LIST = {

d( ) = {

Trace back the path from node 6 to node 1 using the predecessors.

## The shortest path from node 1 to node 6.



LIST = {

d( ) = {

The "predecessor" arcs form an out-tree rooted at node 1.

The algorithm ends with an optimum path from node 1 to each other node. The paths are stored succinctly as an out tree. The path on the out tree from node 1 to node  $j$  is the shortest path from node 1 to node  $j$ . The arcs of the out tree are the "predecessor arcs" that were kept track of in running Dijkstra's algorithm.

## Comments on Dijkstra's Algorithm

- Dijkstra's algorithm makes nodes permanent in increasing order of distance from the origin node.
- Dijkstra's algorithm is efficient in its current form. The running time grows as  $n^2$ , where  $n$  is the number of nodes
- It can be made much more efficient
- In practice it runs in time linear in the number of arcs (or almost so).

## **Professor Edsger Dijkstra, 1930-2002**

## Summary

- The Eulerian cycle problem
- The shortest path problem
- Dijkstra's algorithm finds the shortest path from node 1 to all other nodes in increasing order of distance from the source node.
- The bottleneck operation is identifying the minimum distance label. One can speed this up, and get an incredibly efficient algorithm