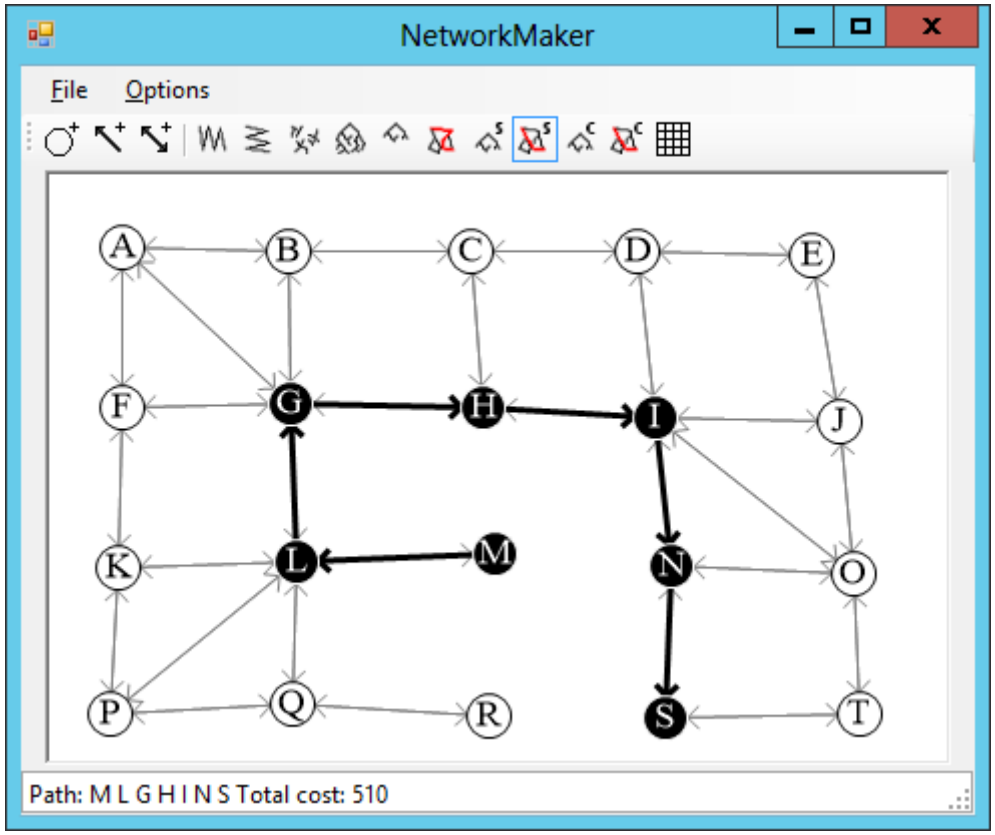
Path-Finding Algorithms



Agenda

- Strongly Connected Components
- Finding Paths
  - Finding Any Path
  - Label-Setting Shortest Paths
  - Label-Correcting Shortest Paths
  - All-Pairs Shortest Paths
- Transitivity
- Shortest Path Modifications
- Summary
- Exercises

Strongly Connected Components

Finding Paths

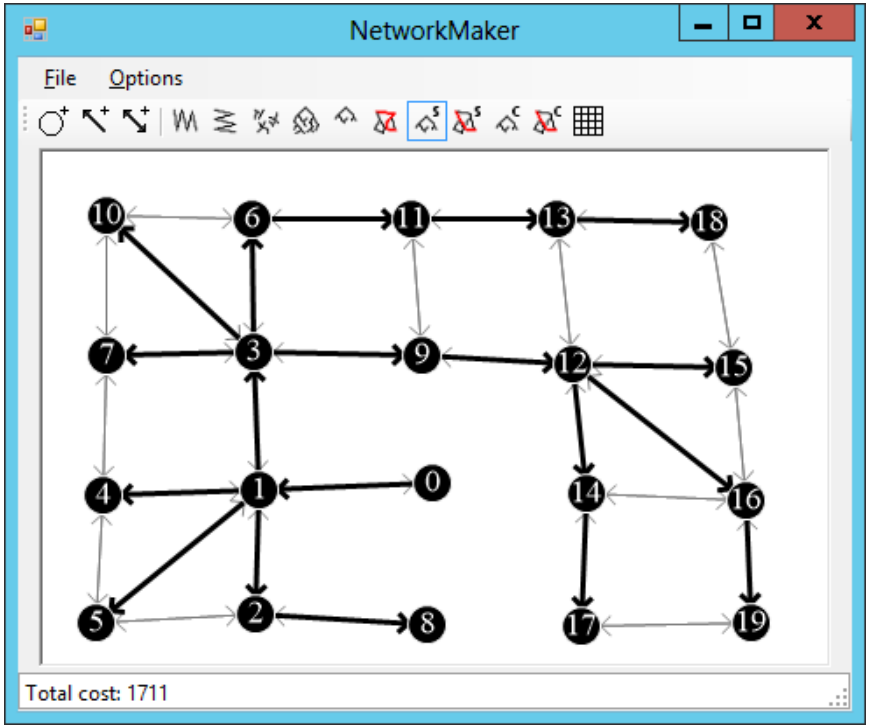
- Finding Any Path
- Label-Setting Shortest Paths
- Label-Correcting Shortest Paths
- All-Pairs Shortest Paths

Finding Any Path

- To find a path from node A to node B:
  1. Find a spanning tree rooted at node A.
  2. Follow the reversed links in the spanning tree from node B to node A.
  3. Reverse the order in which the links were followed.

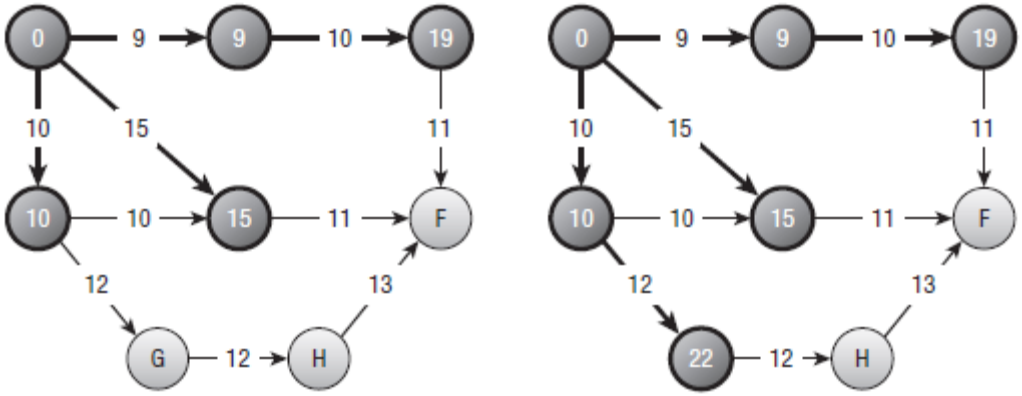
Shortest Path Trees

- A shortest-path tree gives the shortest paths from the root node to any node in the network



Label-Setting Shortest Paths

- Once set, a node's distance is never changed
- At each step, you add the link that gives the least total distance from the root to a node that is not in the tree



Label-Setting Algorithm

1. Set the start node's distance to 0 and mark it as in the tree.
2. Add the start node's links to a *candidate list* of links.
3. While the candidate list is not empty, loop through the list.
   a. If a link leads to a node in the tree, remove the link from the candidate list.
   b. Suppose link L leads $N_1 \rightarrow N_2$ and N_2 is not yet in the tree. If D_1 is the distance to node N_1 in the tree and C_L is the cost of the link, let $D_2 = N_1 + C_L$ be the possible distance for node N_1 that uses this link. As you loop over the links in the candidate list, keep track of the link L_{best} and node N_{best} that give the smallest possible distance D_{best}.
   c. Set the distance for N_{best} to D_{best} and mark N_{best} as part of the shortest path tree.
   d. For all links L leaving node N_{best}, if L leads to a node that is not yet in the tree, add L to the candidate list.

Label-Correcting Shortest Paths

- A node's distance may be updated later if a better path is found

Label-Correcting Algorithm

1. Set the start node's distance to 0 and mark it as in the tree.
2. Add the start node's links to a candidate list of links.
3. While the candidate list is not empty:
   a. Consider the first link in the candidate list.
   b. Calculate the distance to the link's destination node: <distance> = <source node distance> + <link cost>.
   c. If the new distance is better than the destination node's current distance:
      i. Update the destination node's distance.
      ii. Add all the destination node's links to the candidate list.

All-Pairs Shortest Paths, Part 1

- Floyd–Warshall algorithm
- Two-dimensional array named Distance
- Distance[start_node, end_node] is the shortest distance between nodes start_node and end_node

All-Pairs Shortest Paths, Part 2

- Node via can improve a path if:
  - start_node → end_node
- Becomes:
  - start_node → via_node → end_node

All-Pairs Shortest Paths Algorithm

- Initialize the Distance array
- For via_node = 0, 1, 2, ..., N – 1:
  - For every pair of nodes A → B:
    - See if you can use via_node to improve the path to:

      A → via_node → B

All-Pairs Shortest Paths Code, Part 1

1. Initialize the Distance array:
   a. Set Distance[i, j] = infinity for all entries.
   b. Set Distance[i, i] = 0 for all i = 1 to N – 1.
   c. If nodes i and j are connected by a link i → j, set Distance[i, j] to the cost of that link.

All-Pairs Shortest Paths Code, Part 2

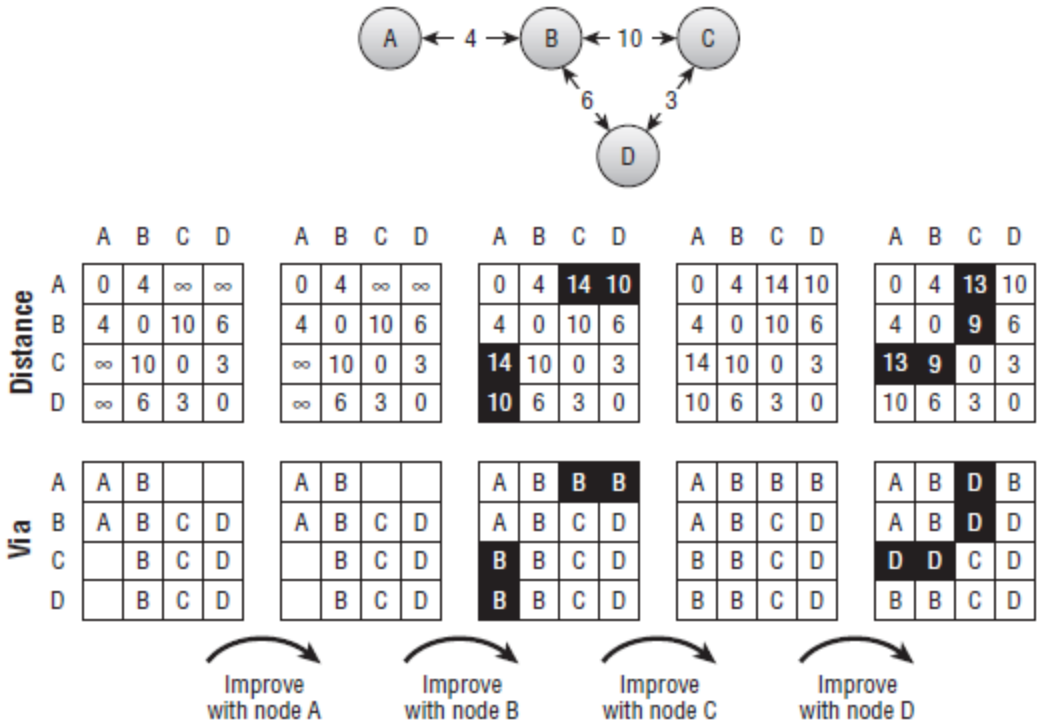
2. Initialize the Via array:
   a. For all i and j:
      i. If Distance[i, j] < infi nity, set Via[i, j] to j to indicate that the path from i to j goes via node j.
      ii. Otherwise, set Via[i, j] to –1 to indicate that there is no path from node i to node j.

All-Pairs Shortest Paths Code, Part 3

2. Execute the following nested loops to find improvements:

```
For via_node = 0 To N - 1
    For from_node = 0 To N - 1
        For to_node = 0 To N - 1
            Integer: new_dist =
              Distance[from_node, via_node] + Distance[via_node, to_node]
                If (new_dist < Distance[from_node, to_node]) Then
                    // This is an improved path. Update it.
                    Distance[from_node, to_node] = new_dist
                    Via[from_node, to_node] = via_node
                End If
        Next to_node
    Next from_node
Next via_node
```

All-Pairs Shortest Paths Example



Transitivity

- Transitive Reduction
- Transitive Closure

Shortest Path Modifications

- Shape Points
- Early Stopping
- Bidirectional Search
- Best-First Search
- Turn Penalties and Prohibitions

Summary

- Finding Paths
  - Finding Any Path
  - Label-Setting Shortest Paths
  - Label-Correcting Shortest Paths
  - All-Pairs Shortest Paths
- Transitivity
- Shortest Path Modifications

Exercises

- Chapter 13 Exercises 8 – 21, 23 – 26.
- Read *Essential Algorithms, 2e* Chapter 14 pages 351 – 470. (Stop before the section "Network Cloning.")