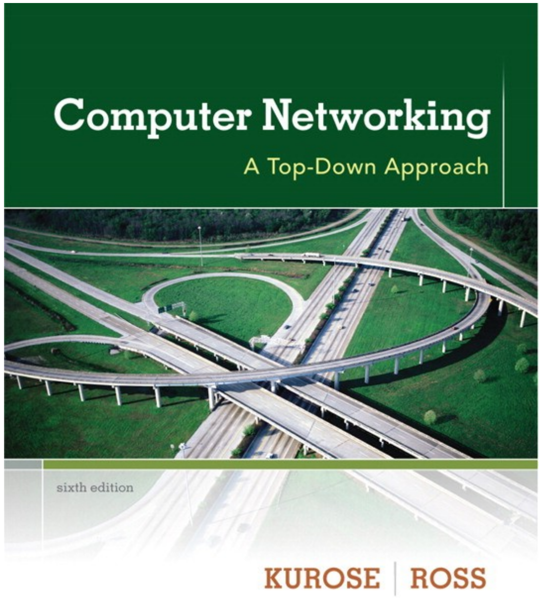
Chapter 4
Network Layer

Part 2 (of 3): Routing (1)



A note on the use of these ppt slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach

6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

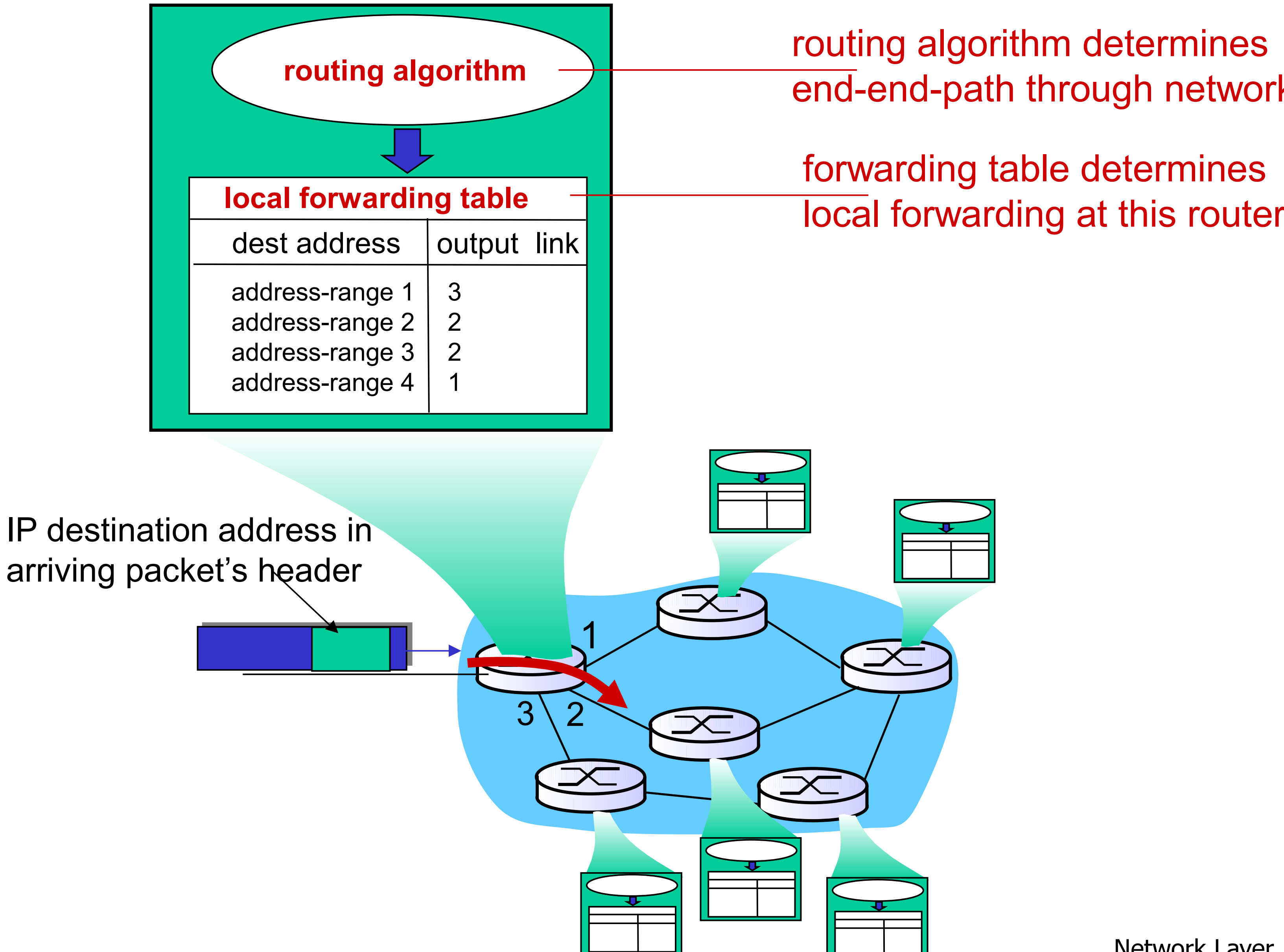
4.4 IP: Internet Protocol
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms
- link state
- distance vector
- hierarchical routing

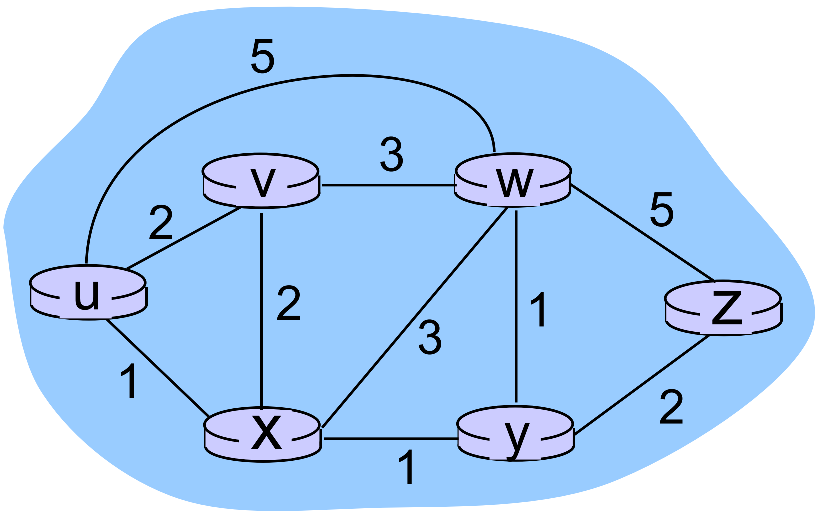
4.6 routing in the Internet
- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

Interplay between routing, forwarding



Graph abstraction



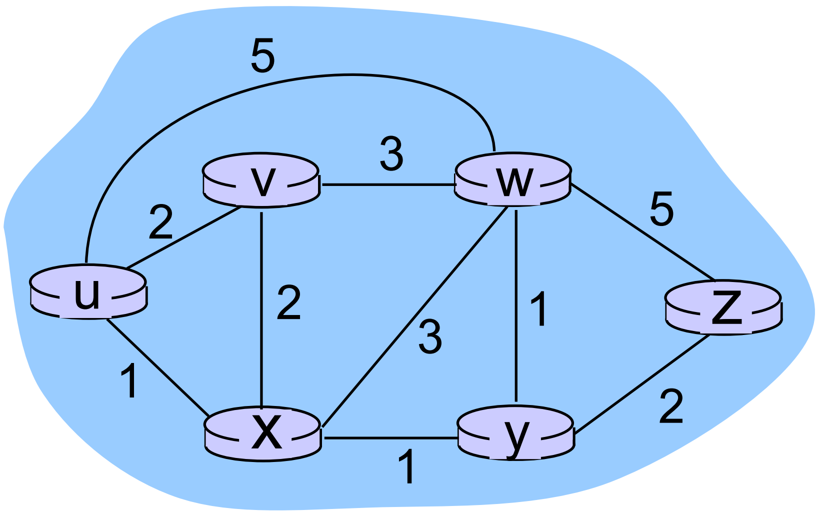
graph: G = (N,E)

N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

aside: graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

Graph abstraction: costs



$c(x,x') = $ cost of link (x,x')

e.g., $c(w,z) = 5$

cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

cost of path $(x_1, x_2, x_3,\ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

key question: what is the least-cost path between u and z ?

routing algorithm: finds that least cost path

Routing algorithm classification

Q: global or decentralized information?

global:

- all routers have complete topology, link cost info
- "**link state**" algorithms

decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "**distance vector**" algorithms

Routing algorithm classification

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

A Link-State Routing Algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.'s

notation:

- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijsktra's Algorithm

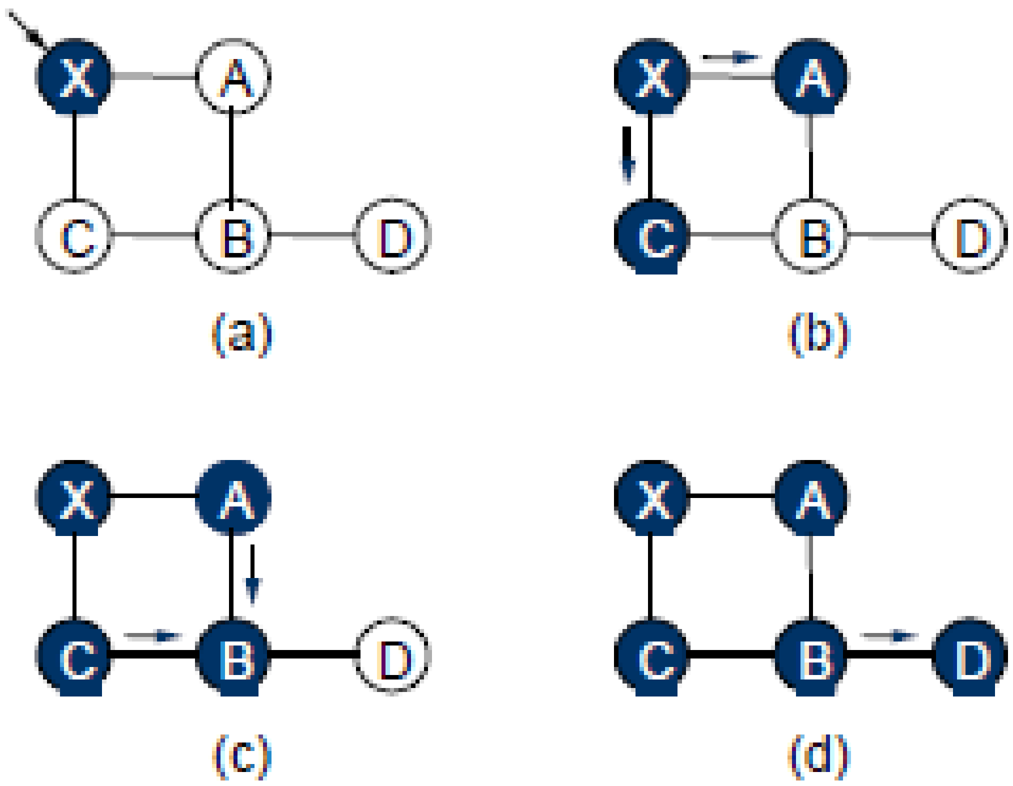
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
     D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Link State Protocol Concept

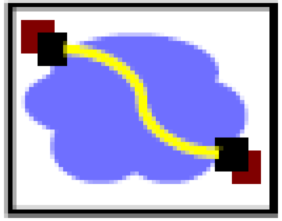
- Every node gets complete copy of graph
  - Every node “floods” network with data about its outgoing links
- Every node computes routes to every other node
  - Using single-source, shortest-path algorithm
- Process performed whenever needed
  - When connections die / reappear

Sending Link States by Flooding

- X Wants to Send Information
  - Sends on all outgoing links
- When Node Y Receives Information from Z
  - Send on all links other than Z

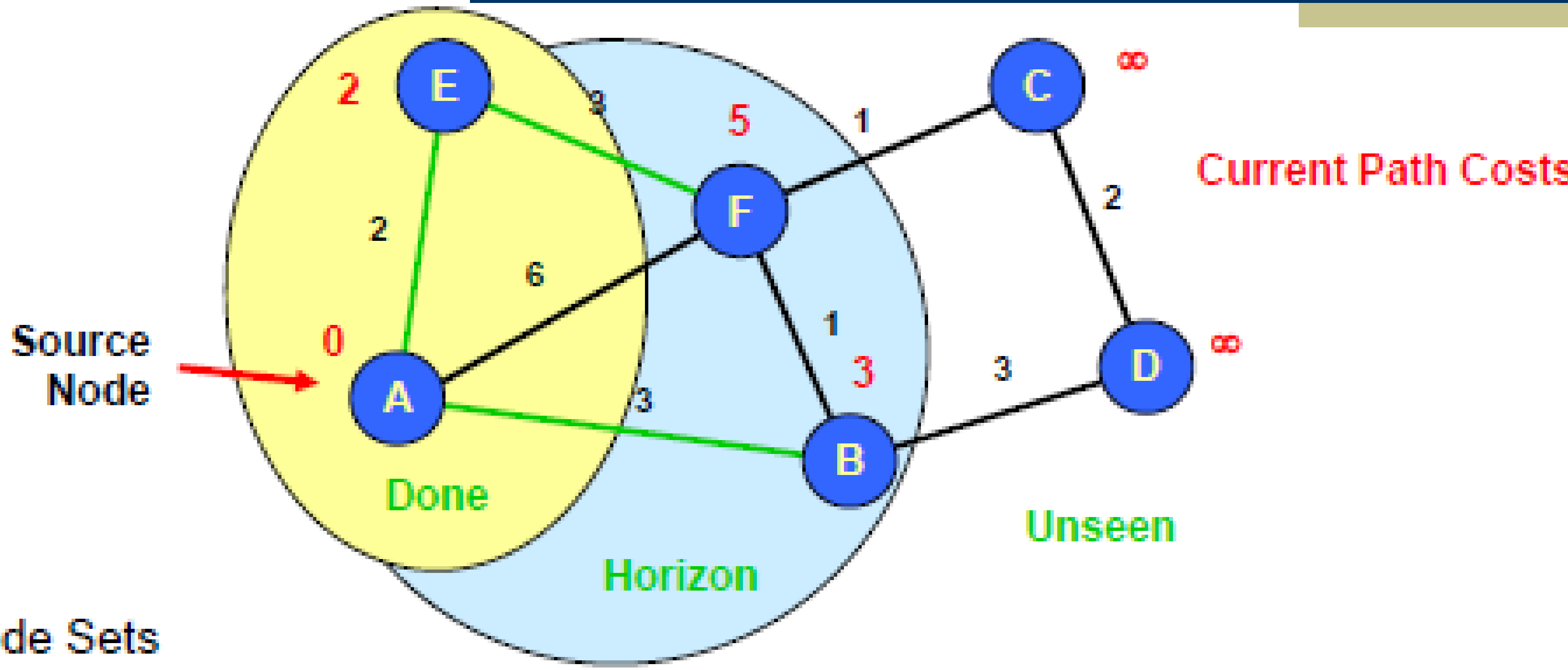


Dijkstra's Algorithm



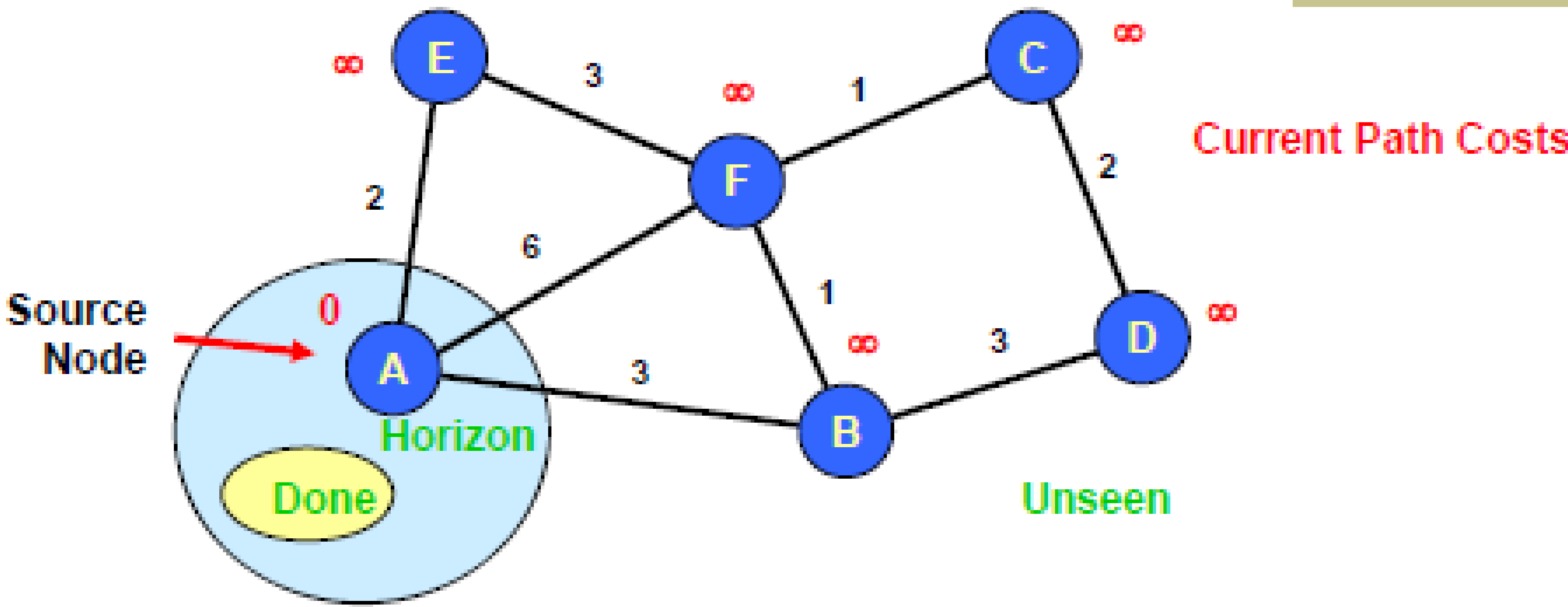
- Given
  - Graph with source node *s* and edge costs *c(u,v)*
  - Determine least cost path from *s* to every node *v*
- Shortest Path First Algorithm
  - Traverse graph in order of least cost from source

Dijkstra's Algorithm: $\boxed{\text{Concept}}$



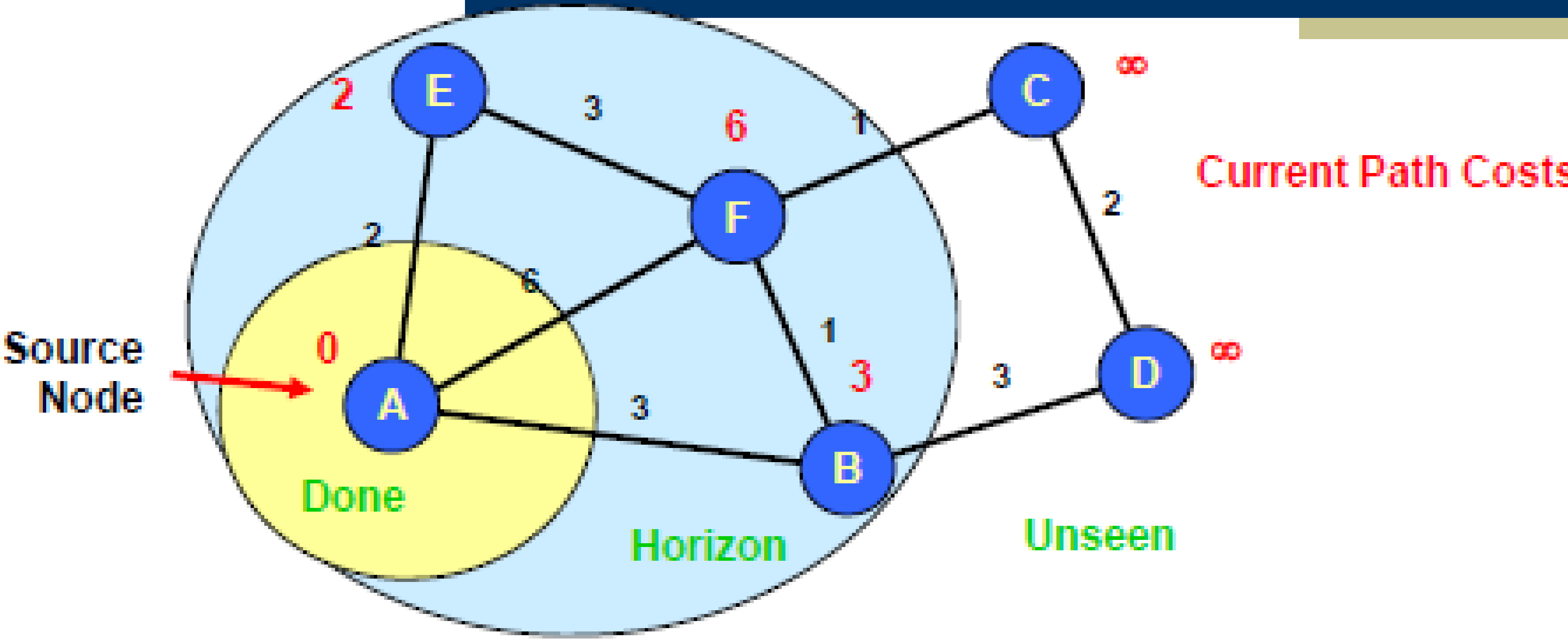
- Node Sets
  - Done
    - Already have least cost path to it
  - Horizon:
    - Reachable in 1 hop from node in Done
  - Unseen:
    - Cannot reach directly from node in Done
- Label
  - *d(v)* = path cost from s to v
- Path
  - Keep track of last link in path

Dijkstra’s Algorithm: Initially



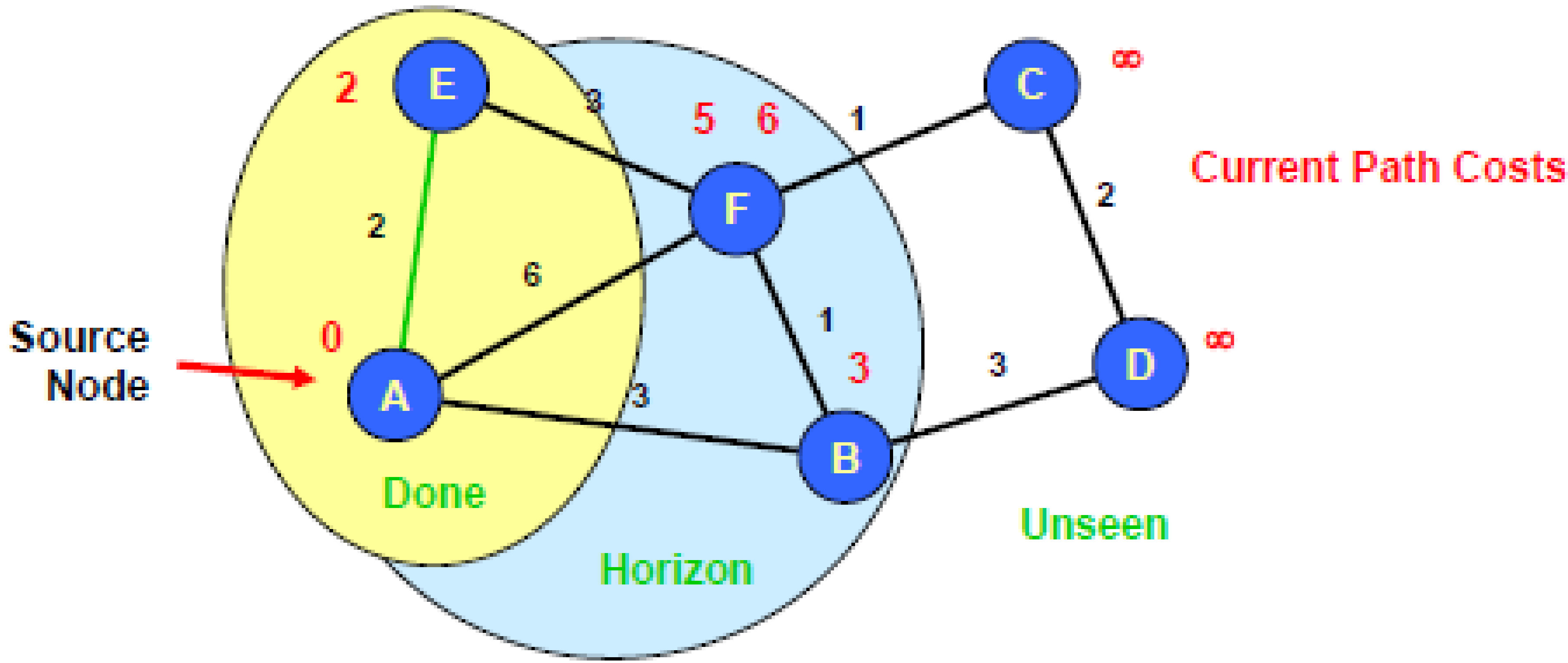
- No nodes done
- Source in horizon

Dijkstra's Algorithm: Initially



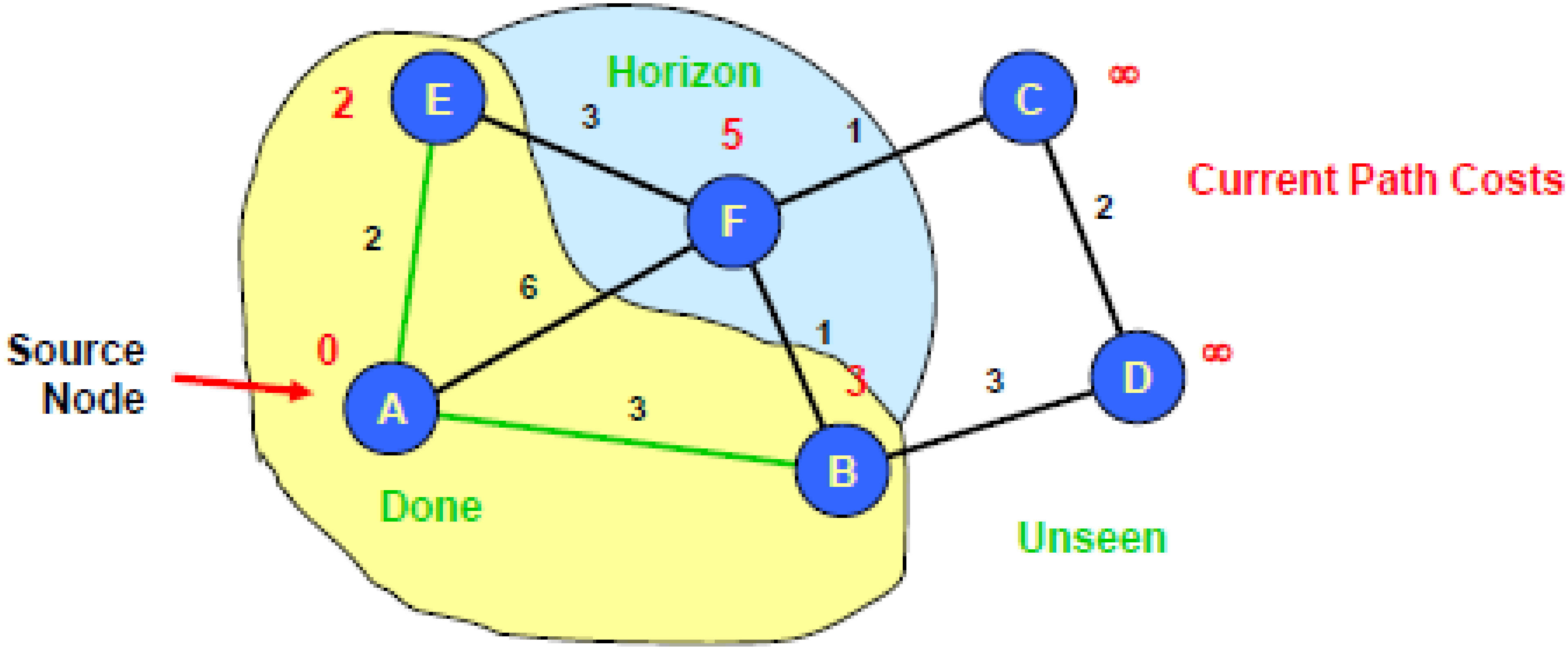
- d(v) to node A shown in red
  - Only consider links from done nodes

Dijkstra's Algorithm



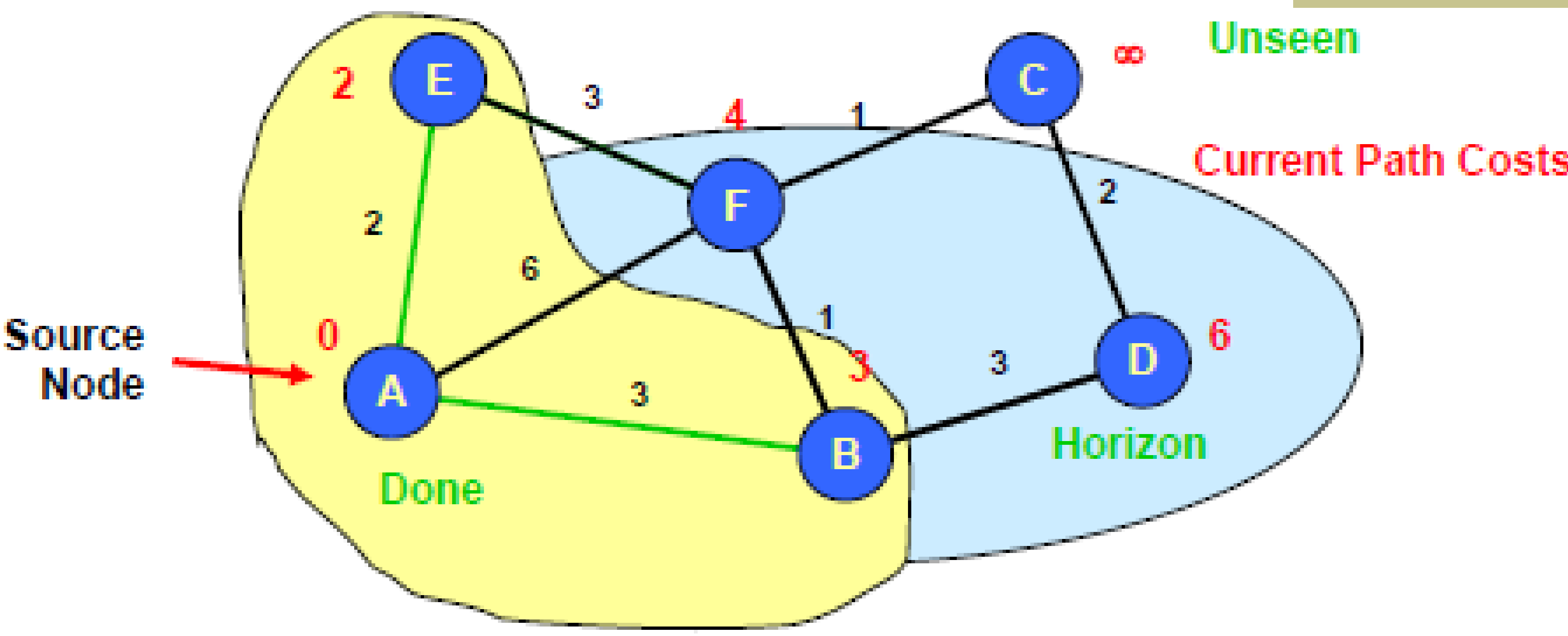
- Select node v in horizon with minimum d(v)
- Add link used to add node to shortest path tree
- Update d(v) information

Dijkstra's Algorithm



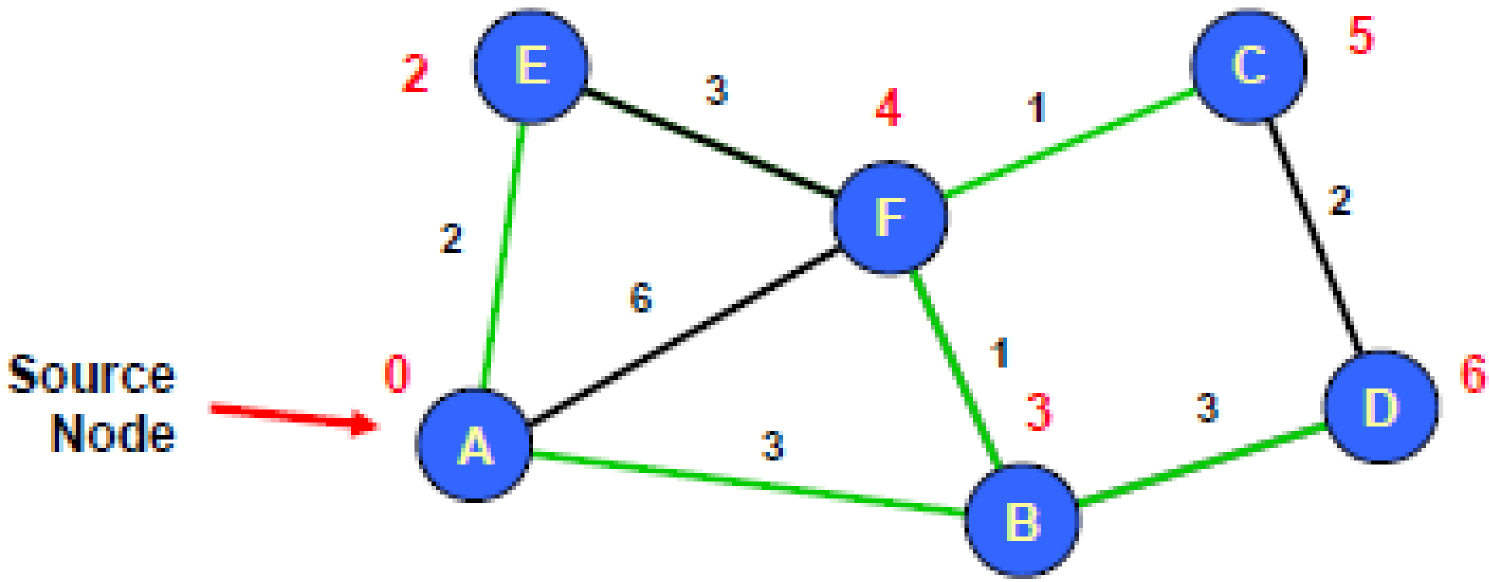
- Repeat…

Dijkstra's Algorithm



- Update d(v) values
  - Can cause addition of new nodes to horizon

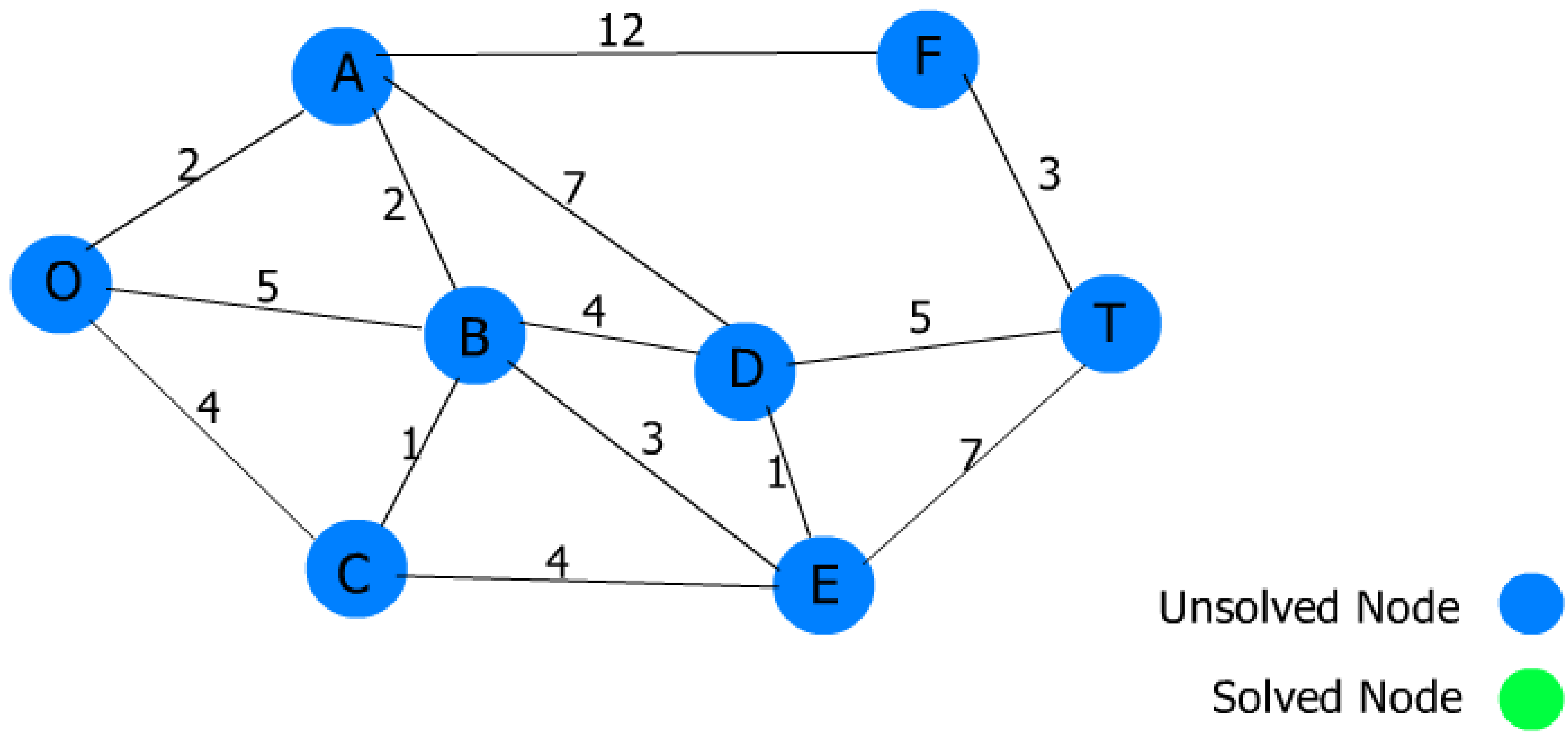
Dijkstra's Algorithm



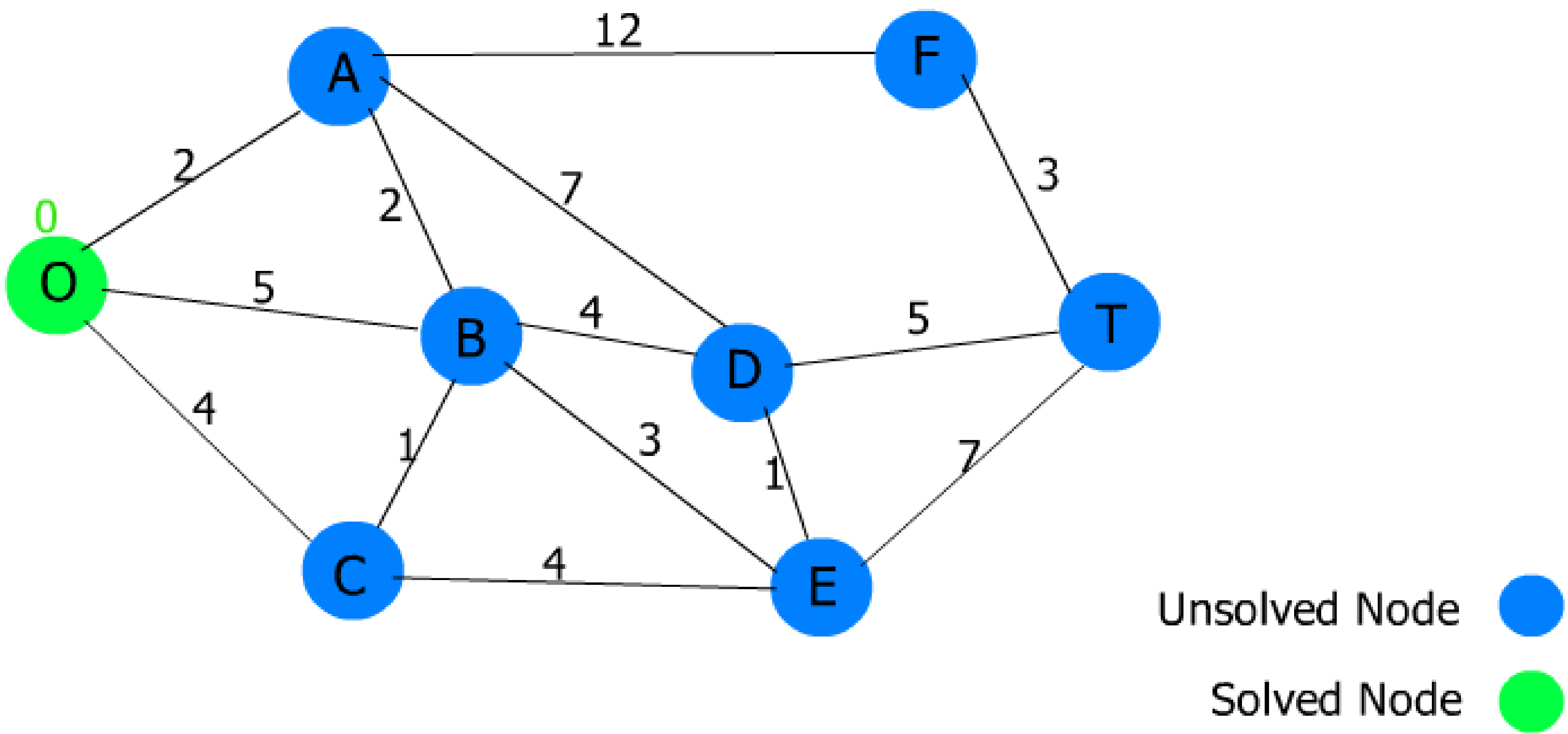
- Final tree shown in green

Another example

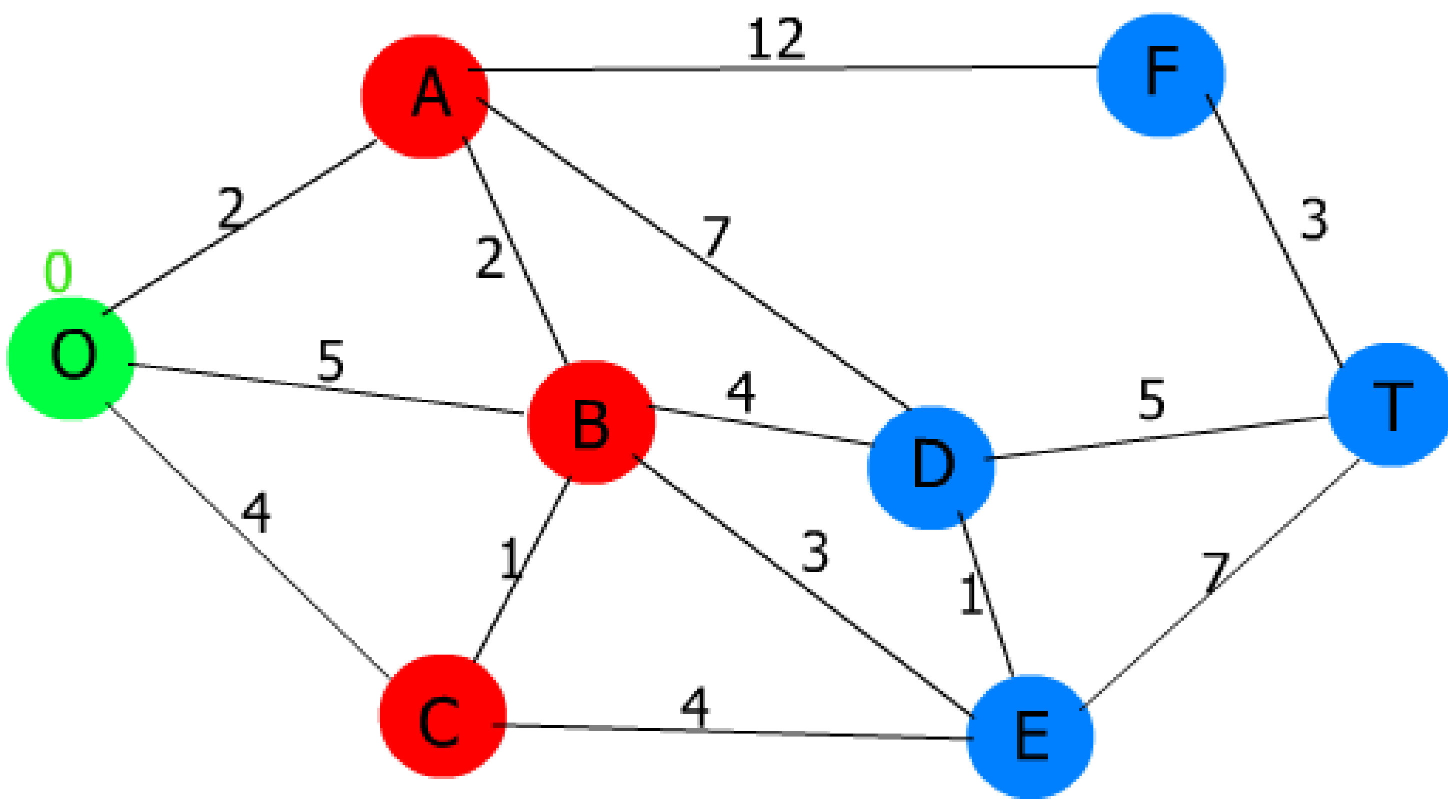
- From O to T

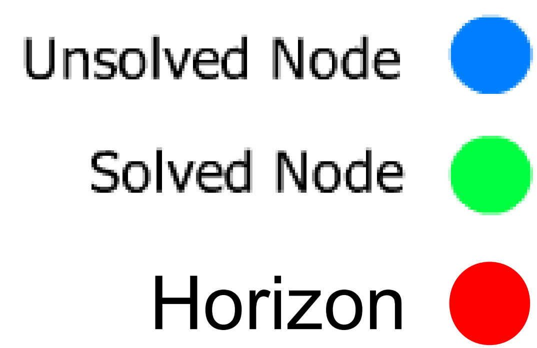


Above is a network whose arcs are labeled with the distance between the two nodes it is connecting.

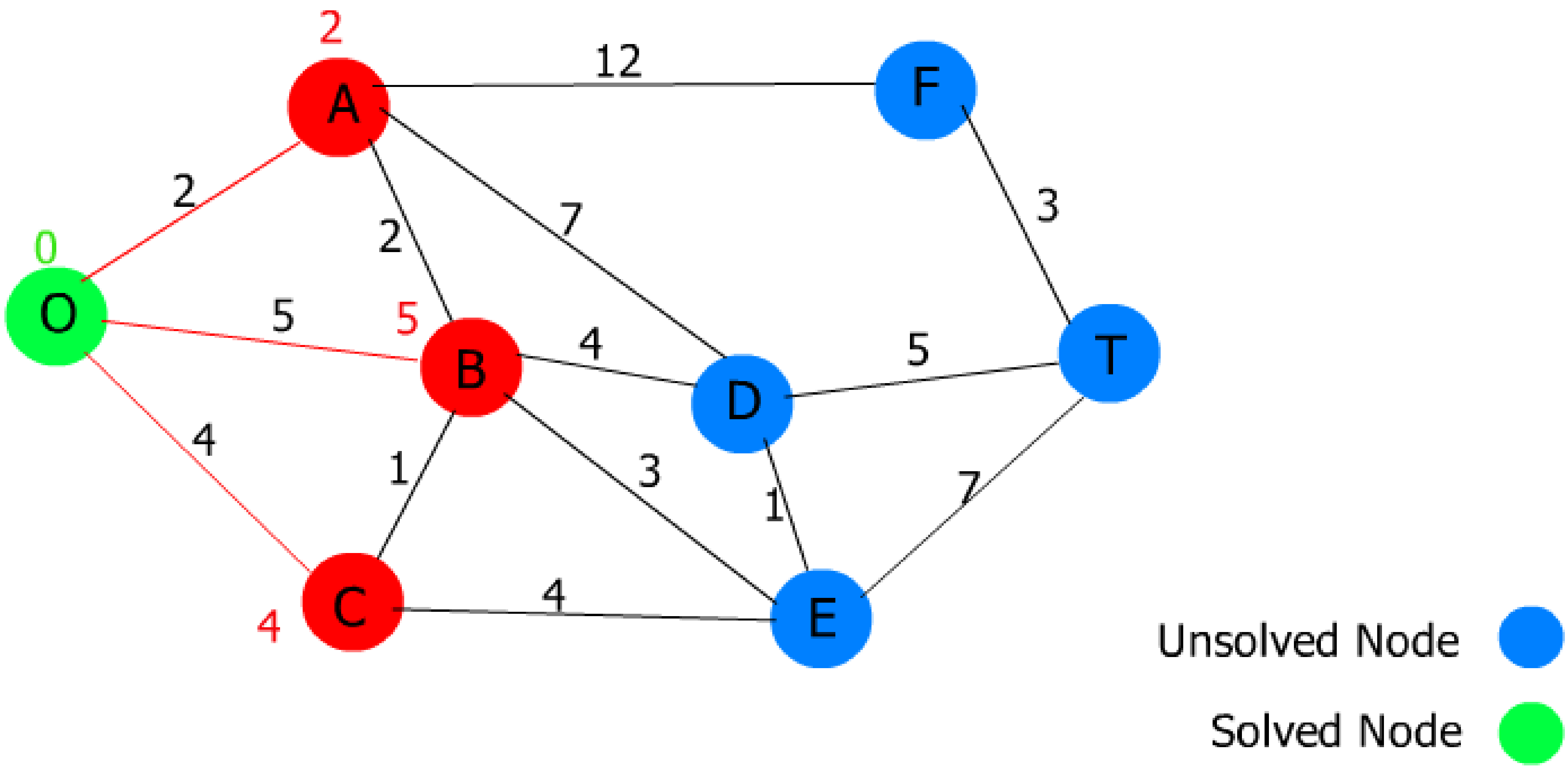


Initialize by displaying the orgin as solved. We will label it with 0, since it is 0 units from the origin.



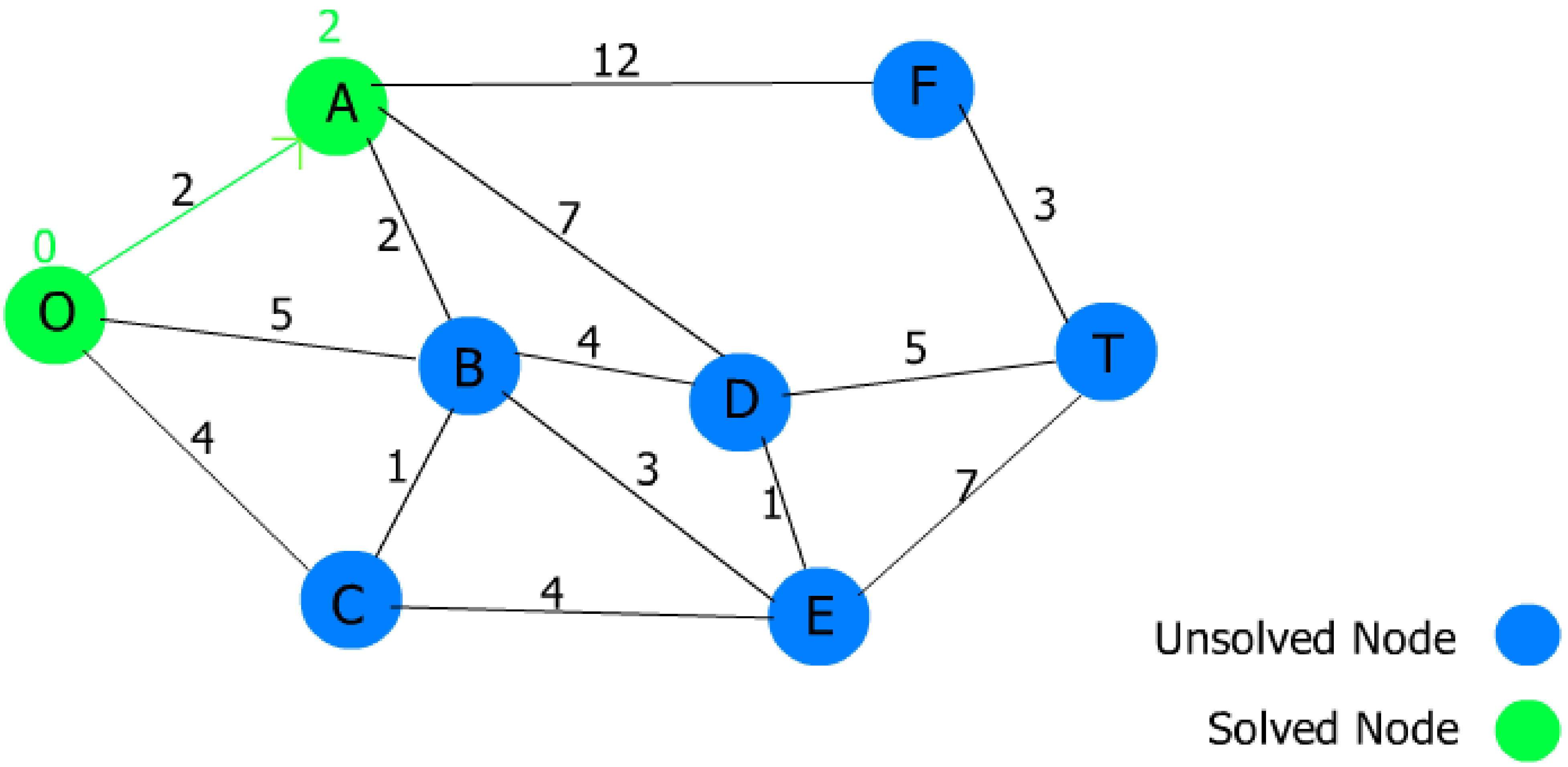


Identify all unsolved nodes connected to any solved node.

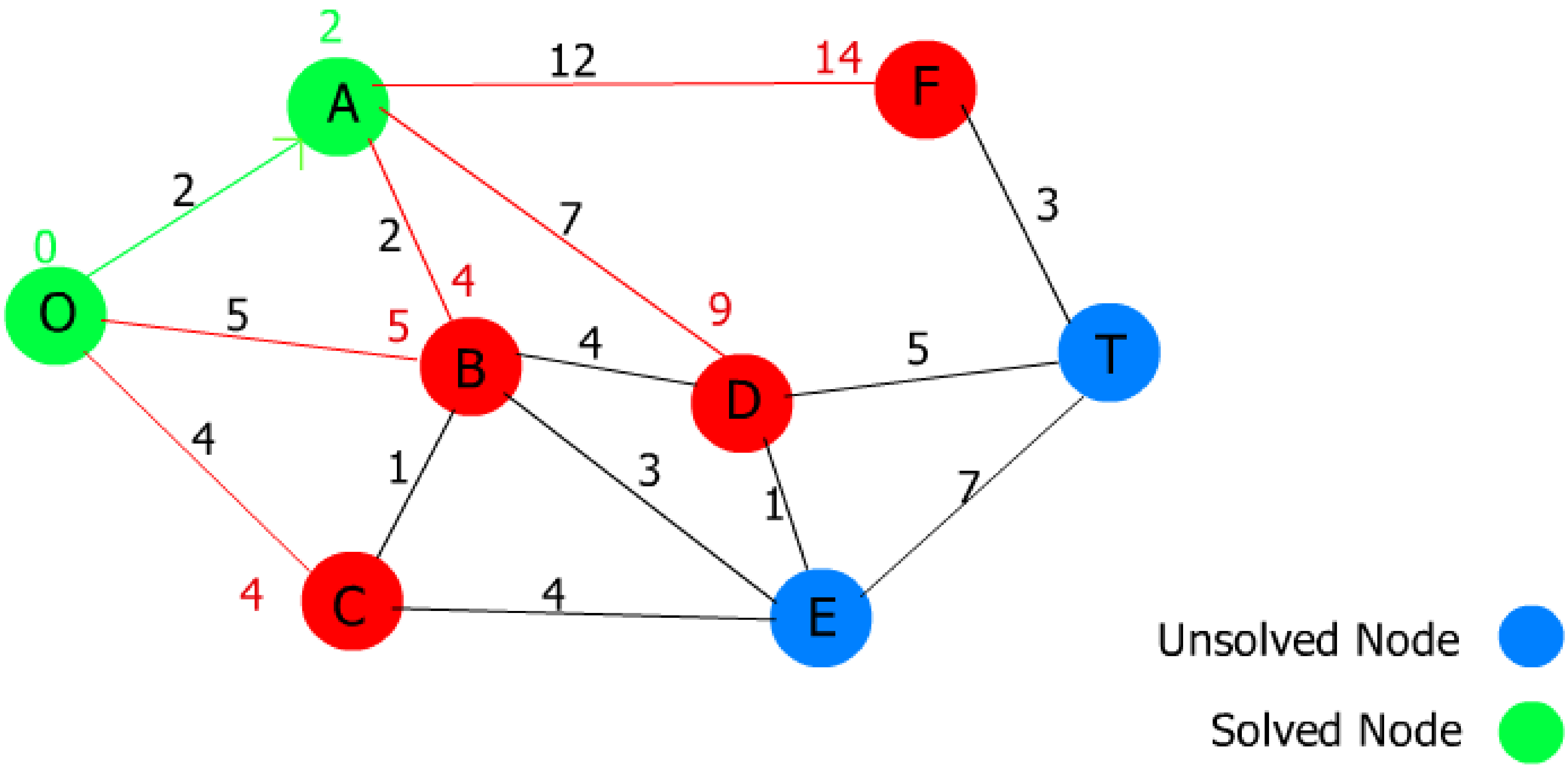


For each arc connecting a solved and unsolved node, calculate the candidate distance.
Candidate distance = distance to the solved node + length of arc

- Choose the min. distance node

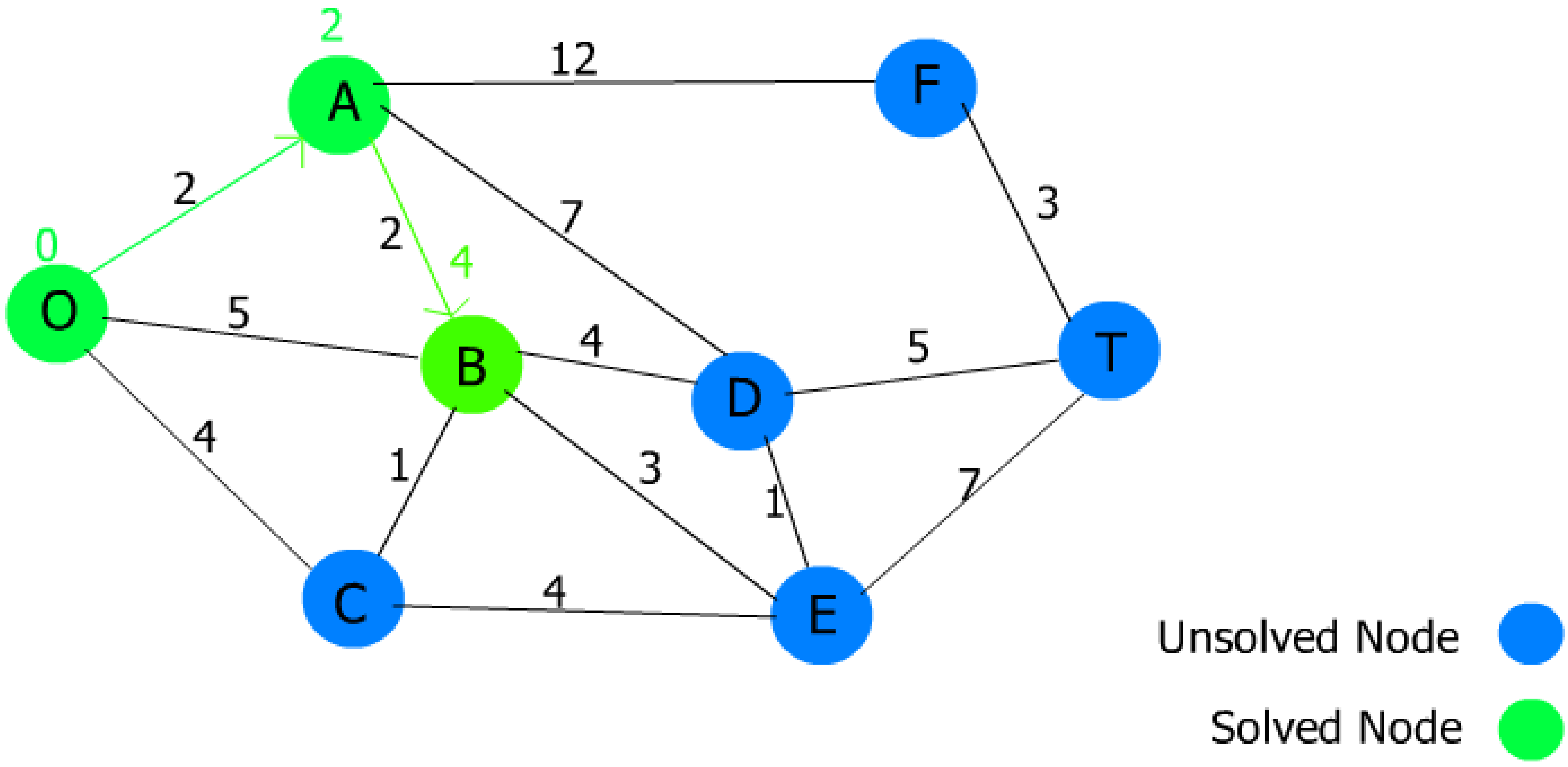


Add the arc to the arc set.

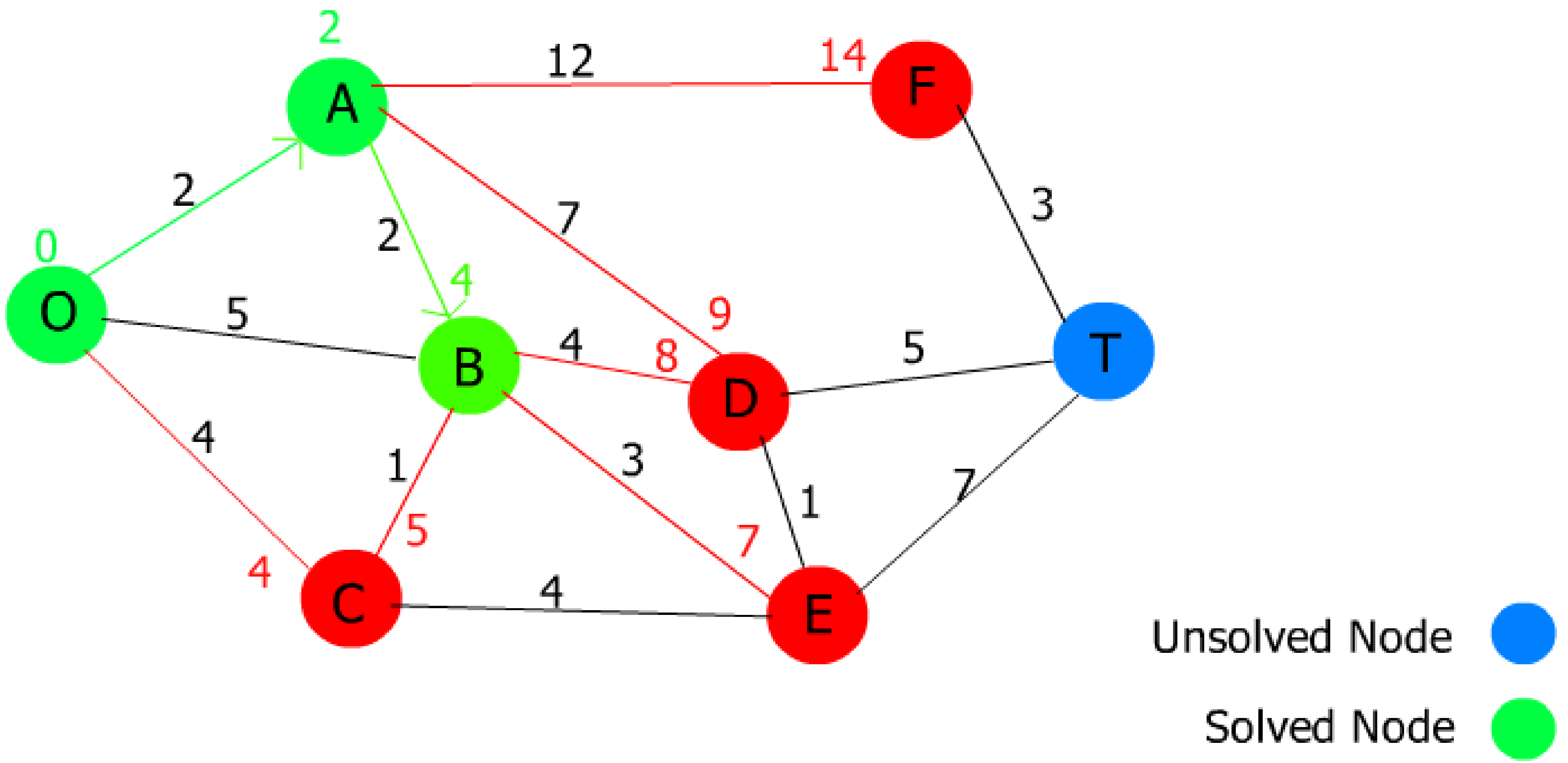


We have a tie for smallest candidate distance.

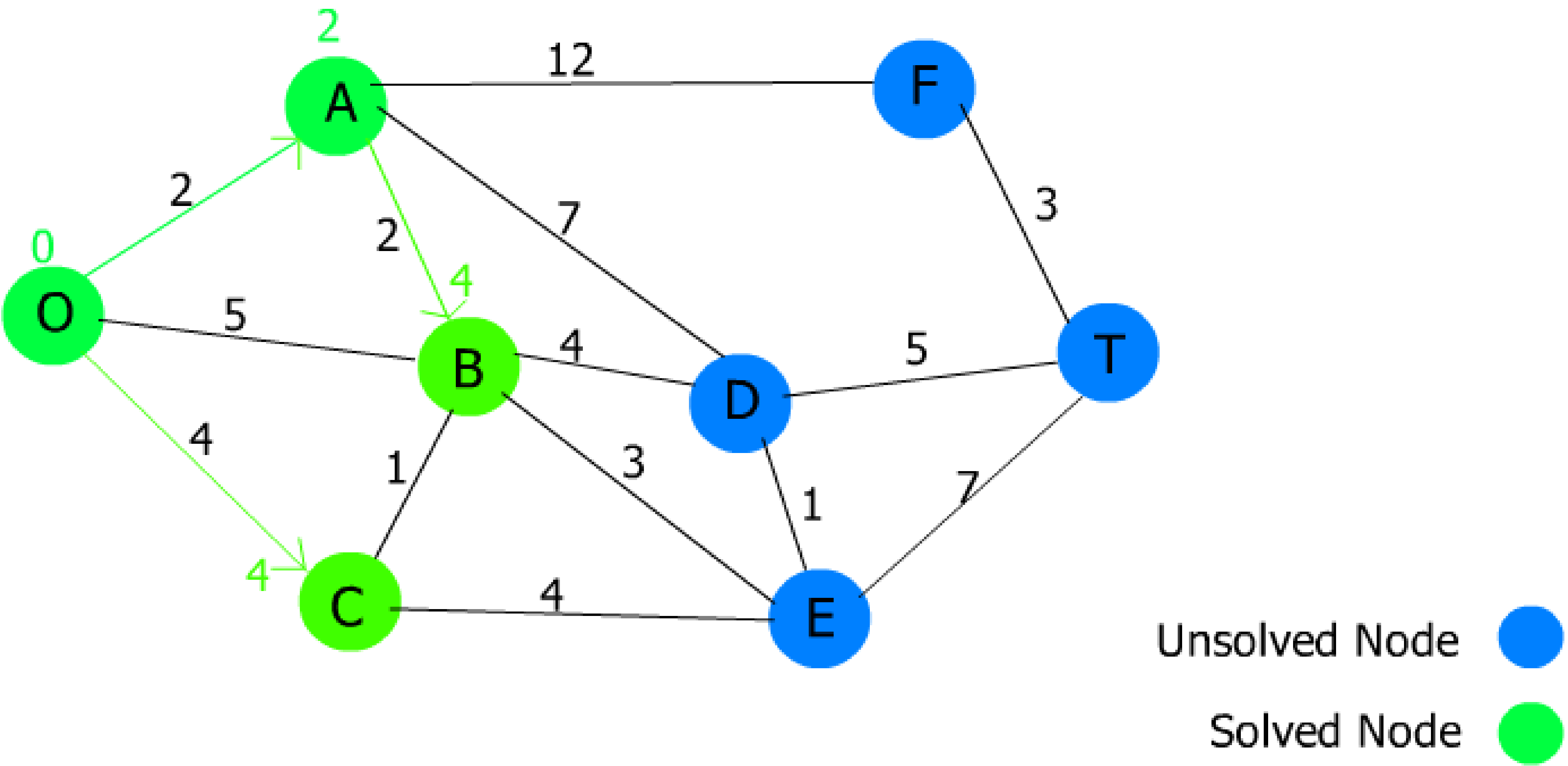
- Choose one arbitrarily



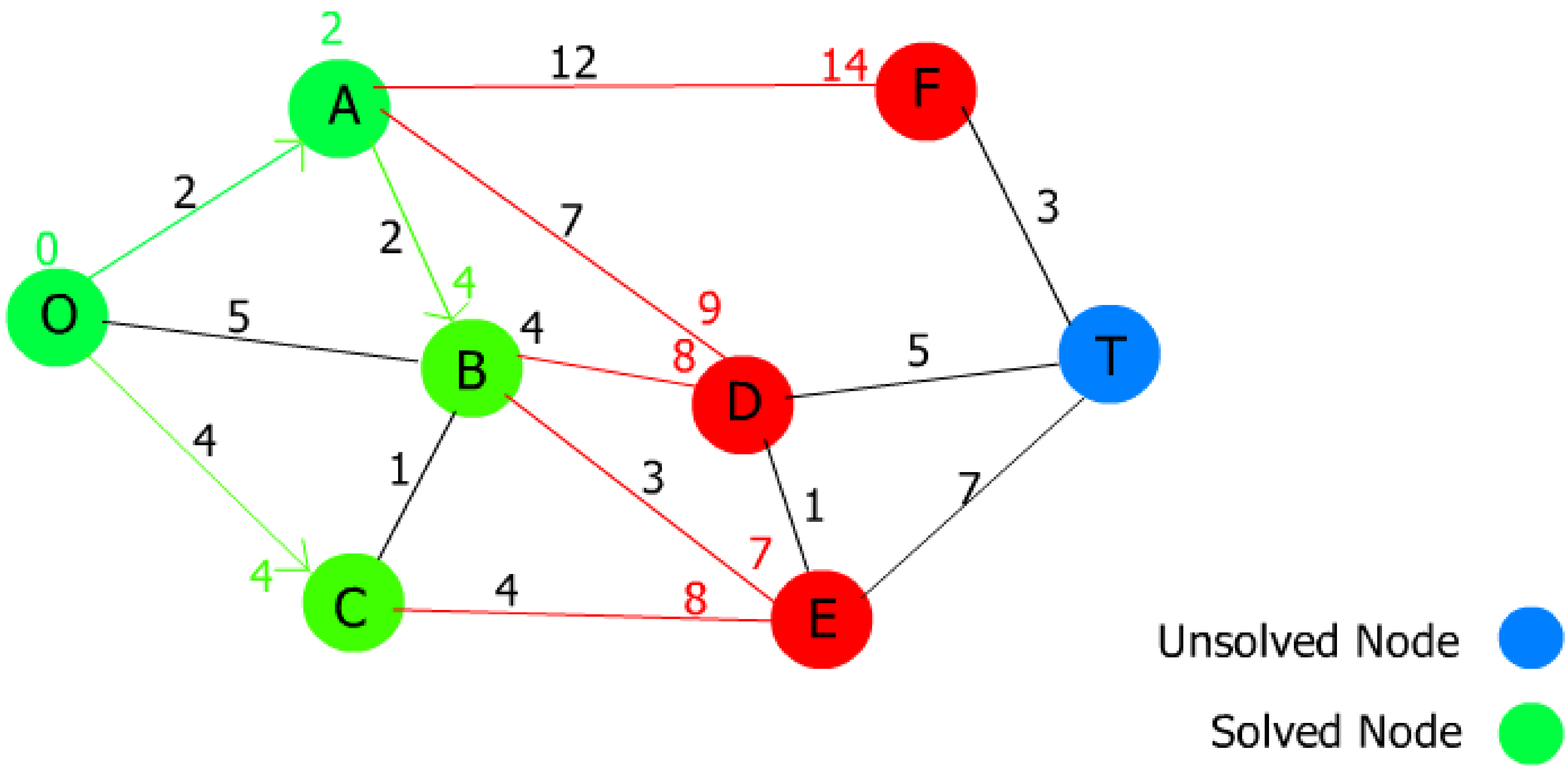
Change node B to solved and label it with the candidate distance.
Add the arc to the arc set.



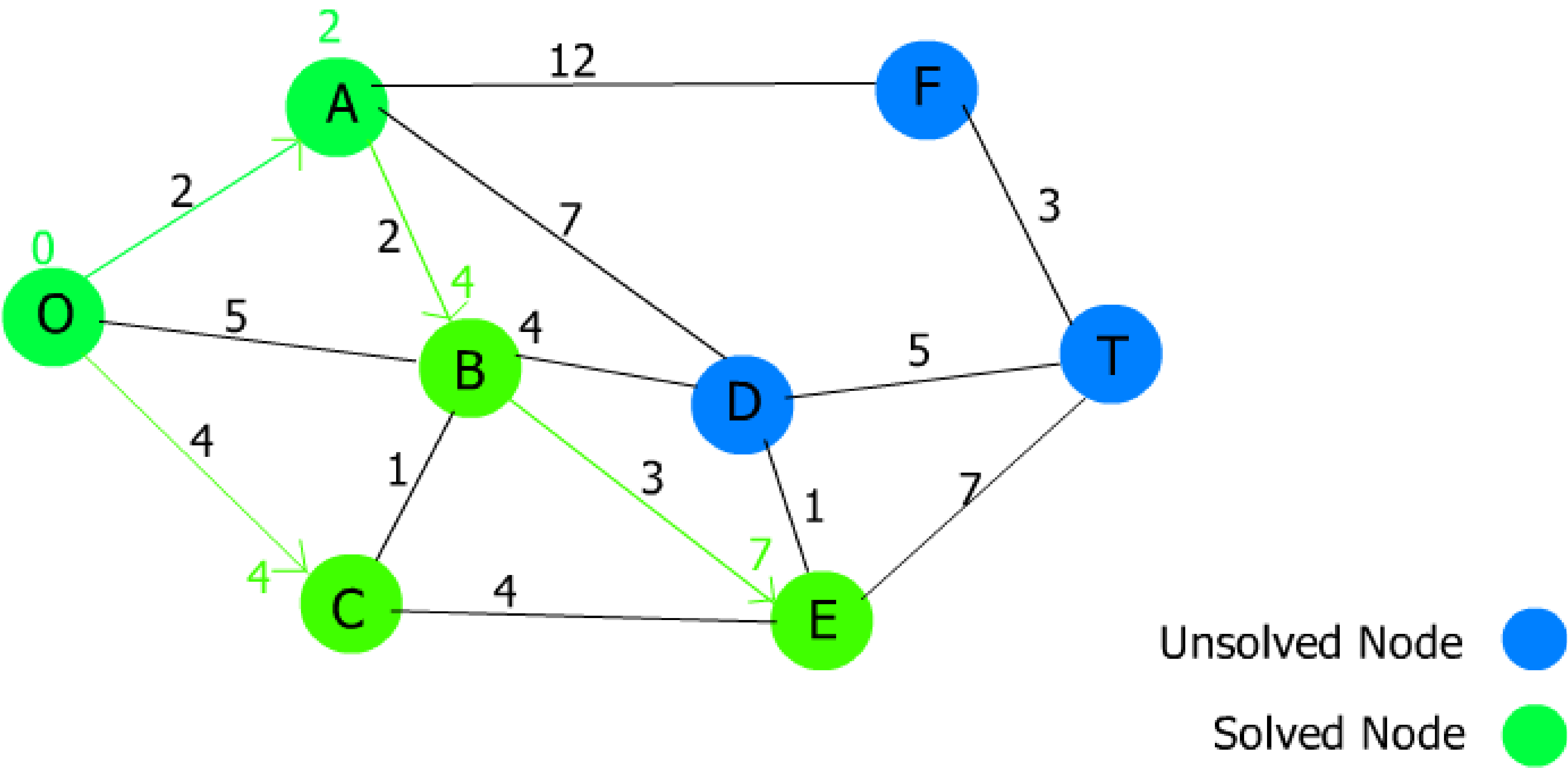
Choose the smallest candidate distance.



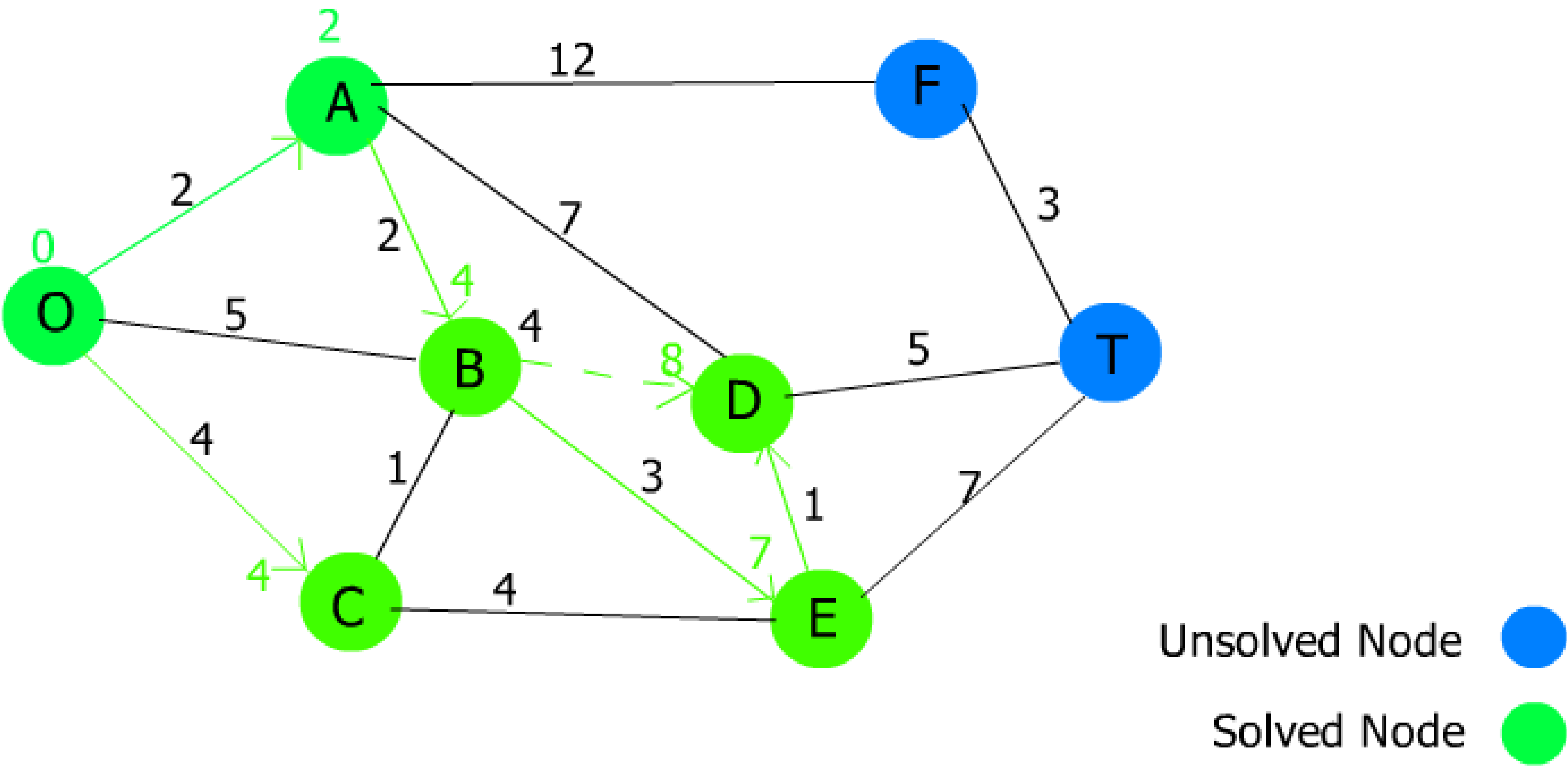
We have not reached our destination node, so we will continue.



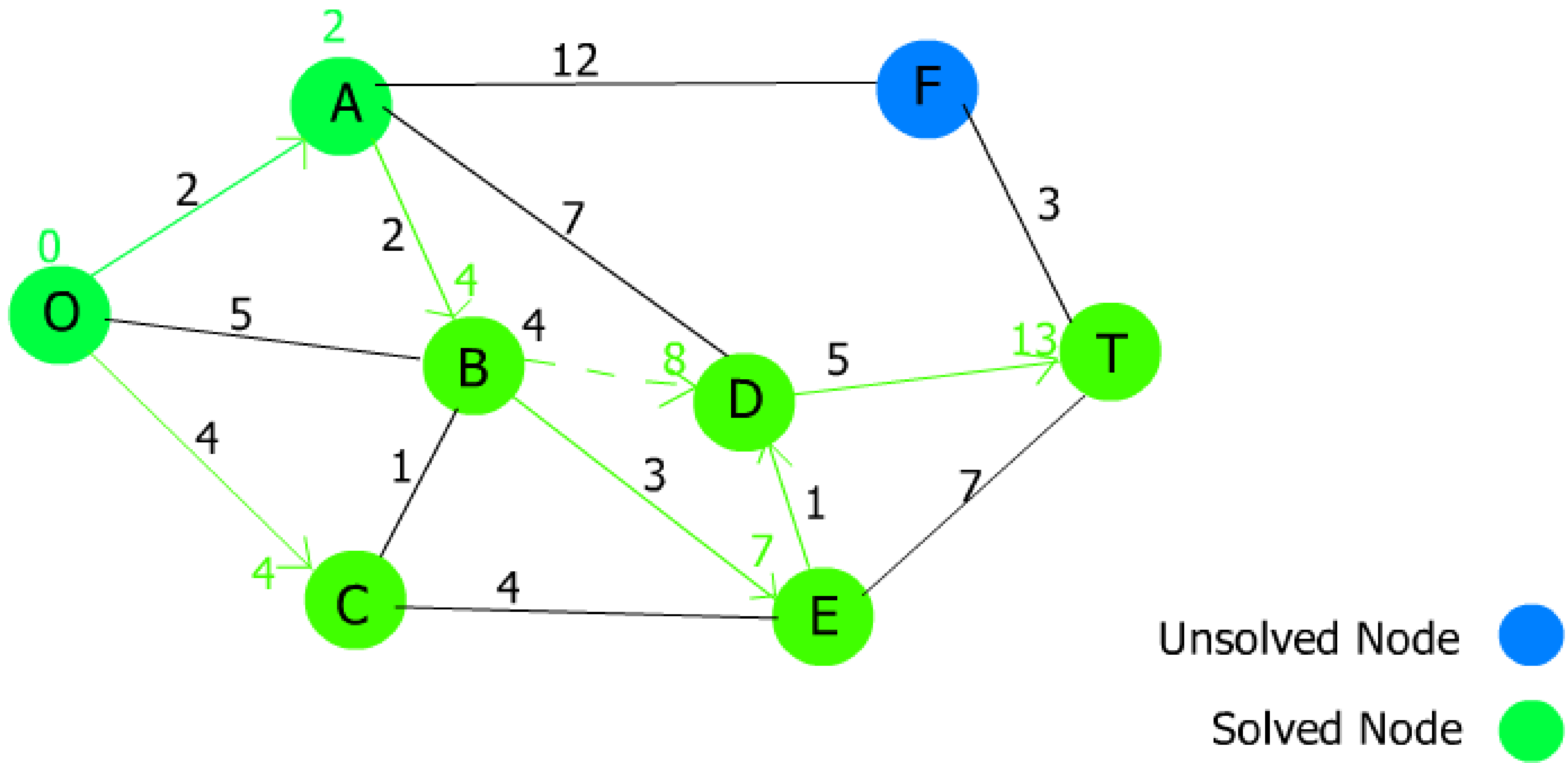
Choose the smallest candidate distance.



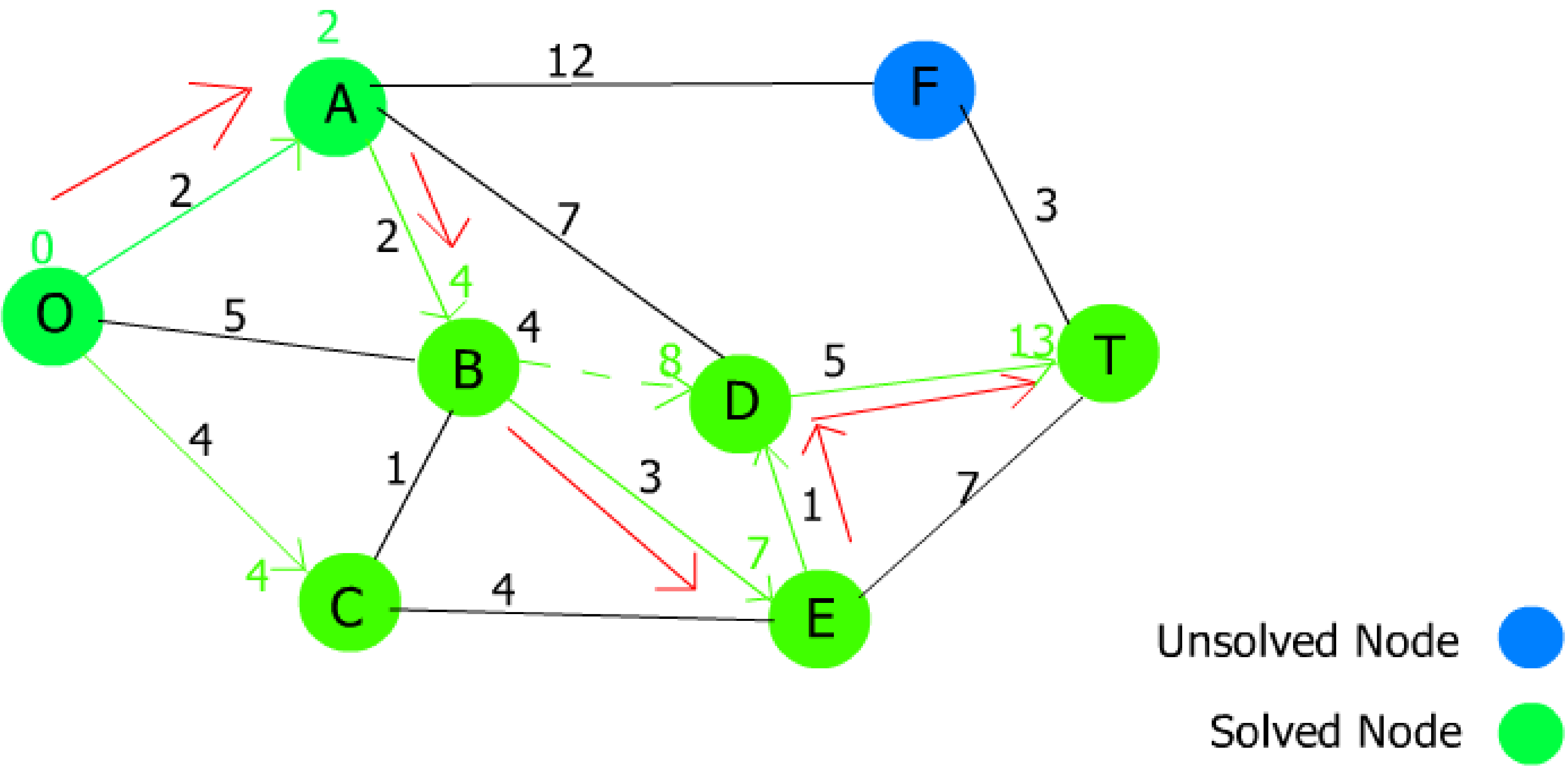
Change node E to solved and label it with the candidate distance.
Add the arc to the arc set.



Change node D to solved and label it with the candidate distance.



Node T is the destination node, therefore we are done.
The shortest route from O to T has a distance of 13.



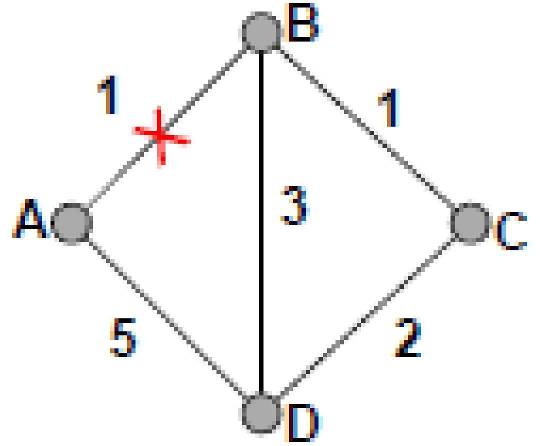
The two shortest routes are:
O - A - B - D - T
and
O - A - B - E - D - T

- Instead of stopping when a specific destination (T here) is done, stop when all nodes are done for a
  - **minimum spanning tree**

Link State Characteristics

- With consistent LSDBs*, all nodes compute consistent loop-free paths
- Can still have transient loops

*Link State Data Base



Packet from C→A may loop around BDC if B knows about failure and C & D do not