Today we will discuss two sections:

Section 10.5 *Euler and Hamilton Paths*

Section 10.6 *Shortest-Paths problems*

10.5 *Euler and Hamilton Paths*

At the previous meeting we posed to questions:

Can we travel long the edges of a graph starting at a vertex and returning to the same vertex by traversing each edge of the graph exactly once?

Can we travel long the edges of a graph starting at a vertex and returning to the same vertex by visiting each vertex in the graph exactly once?

10.5 *Euler and Hamilton Paths*

At the previous meeting we posed to questions:

Can we travel long the edges of a graph starting at a vertex and returning to the same vertex by traversing each edge of the graph exactly once?
- we have answer to this one for both undirected and directed graphs

Can we travel long the edges of a graph starting at a vertex and returning to the same vertex by visiting each vertex in the graph exactly once?
- let's get the answer to this one

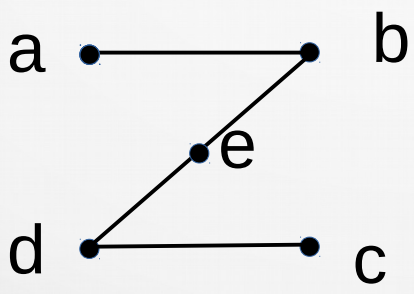
10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

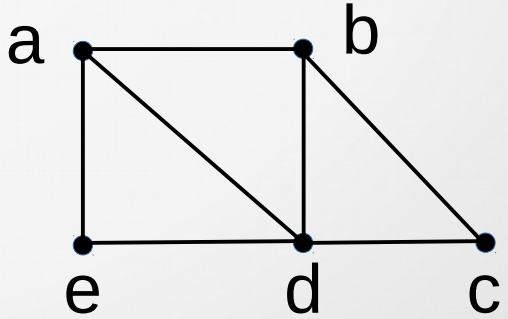
[Def] A *Hamilton circuit* in a graph G is a simple circuit that passes through every vertex in G exactly once.

[Def] A *Hamilton path* in a graph G is a simple path that passes through every vertex in G exactly once.

Examples:



G



H

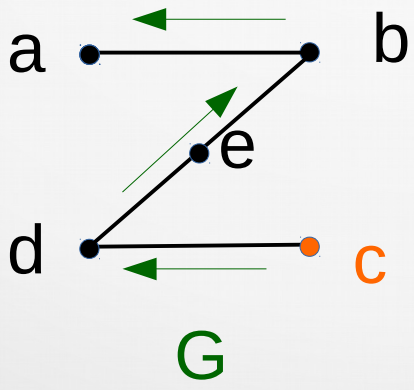
10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

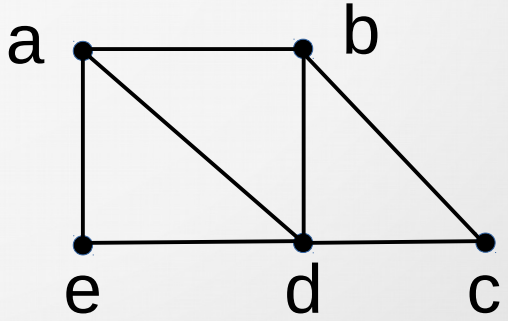
[Def] A *Hamilton circuit* in a graph G is a simple circuit that passes through every vertex in G exactly once.

[Def] A *Hamilton path* in a graph G is a simple path that passes through every vertex in G exactly once.

Examples:



G



H

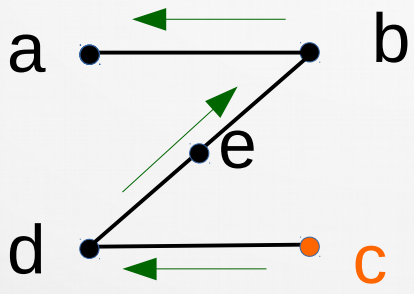
10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

[Def] A *Hamilton circuit* in a graph G is a simple circuit that passes through every vertex in G exactly once.

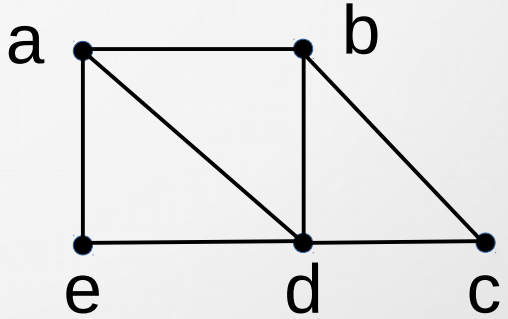
[Def] A *Hamilton path* in a graph G is a simple path that passes through every vertex in G exactly once.

Examples:



G

a *Hamilton path*



H

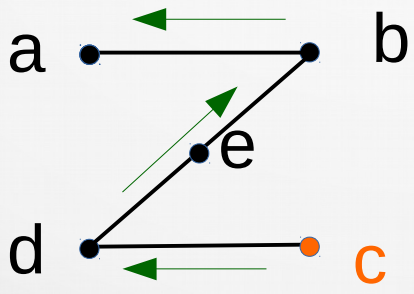
10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

[Def] A *Hamilton circuit* in a graph G is a simple circuit that passes through every vertex in G exactly once.

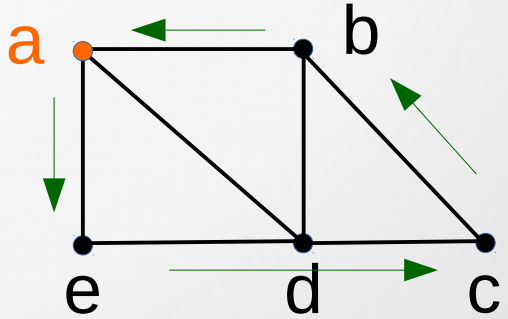
[Def] A *Hamilton path* in a graph G is a simple path that passes through every vertex in G exactly once.

Examples:



G

a *Hamilton path*



H

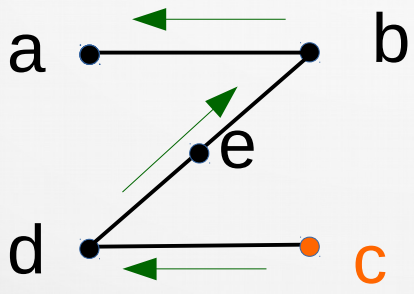
10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

[Def] A *Hamilton circuit* in a graph G is a simple circuit that passes through every vertex in G exactly once.

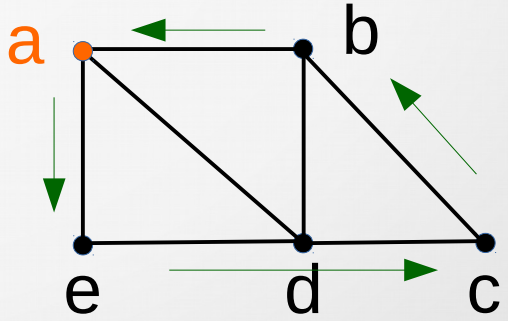
[Def] A *Hamilton path* in a graph G is a simple path that passes through every vertex in G exactly once.

Examples:



G

a *Hamilton path*



H

a *Hamilton circuit*

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.

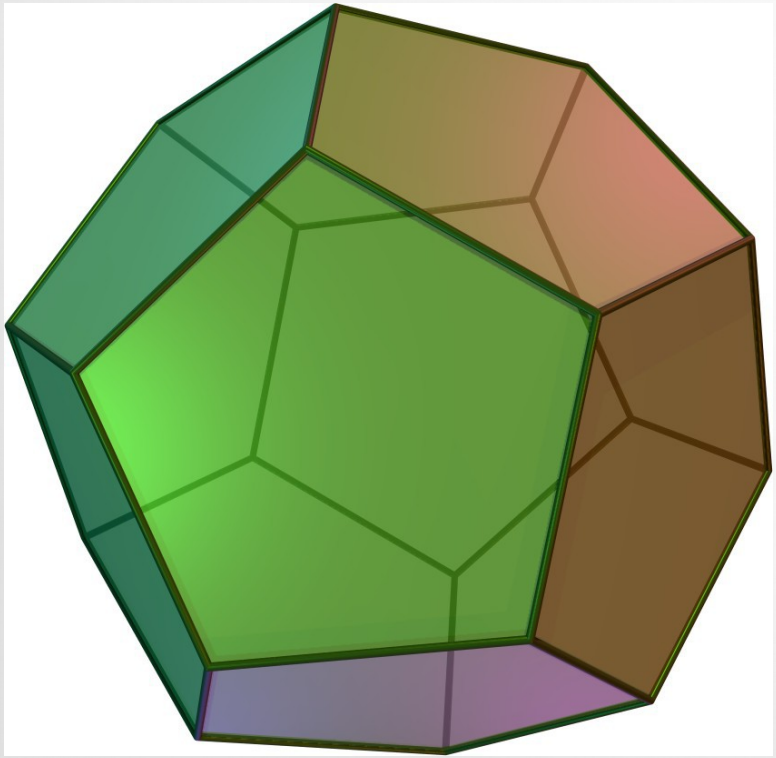
It consisted of a wooden dodecahedron with a peg at each vertex, and a string. The 20 vertices were labeled with 20 cities. The goal was to start in one city and travel along the edges of dodecahedron to visit each of the other 19 cities once, and end back at the first city.

Visit http://puzzlemuseum.com/month/picm02/200207icosian.htm to find more about it.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



A dodecahedron is a polyhedron with 12 regular pentagons as faces.

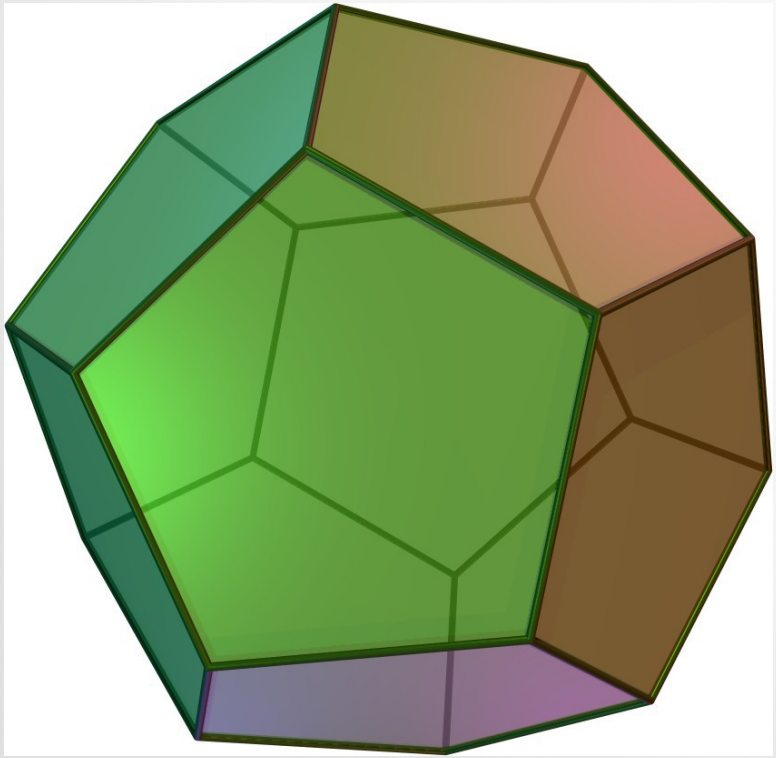
Is there a circuit in the graph that passes each vertex exactly once?

Visit http://puzzlemuseum.com/month/picm02/200207icosian.htm to find more about it.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



A dodecahedron is a polyhedron with 12 regular pentagons as faces.

Is there a circuit in the graph that passes each vertex exactly once?

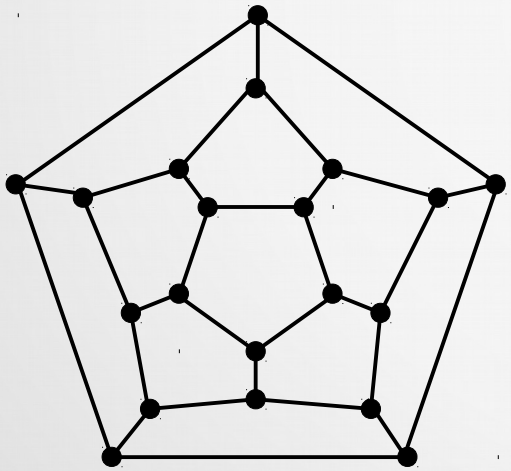
Yes, there is such a circuit.

Visit http://puzzlemuseum.com/month/picm02/200207icosian.htm to find more about it.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



The graph on the left is isomorphic to the graph consisting all the edges and vertices of a dodecahedron.

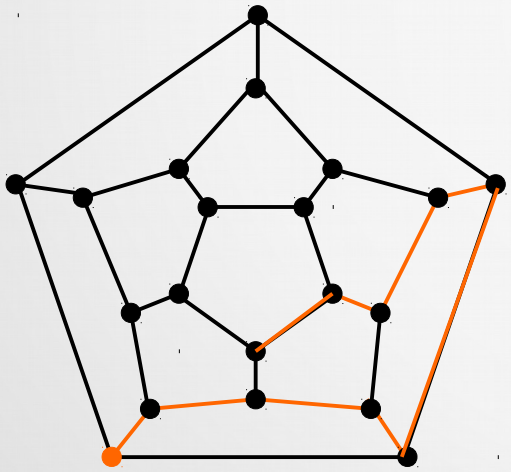
We will show the circuit on it.

Yes, there is such a circuit.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



The graph on the left is isomorphic to the graph consisting all the edges and vertices of a dodecahedron.

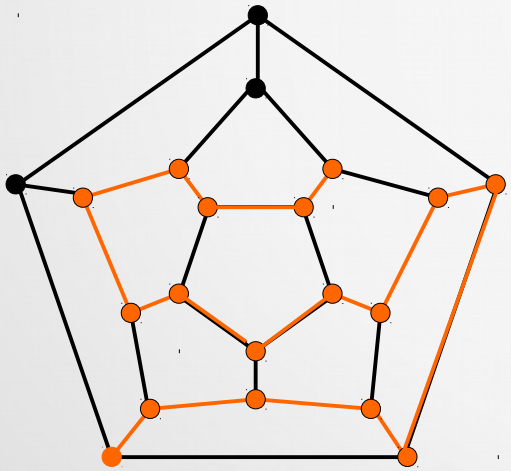
We will show the circuit on it.

Yes, there is such a circuit.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



The graph on the left is isomorphic to the graph consisting all the edges and vertices of a dodecahedron.

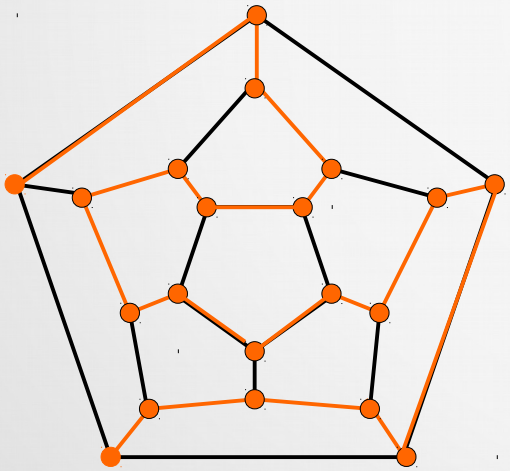
We will show the circuit on it.

Yes, there is such a circuit.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



The graph on the left is isomorphic to the graph consisting all the edges and vertices of a dodecahedron.

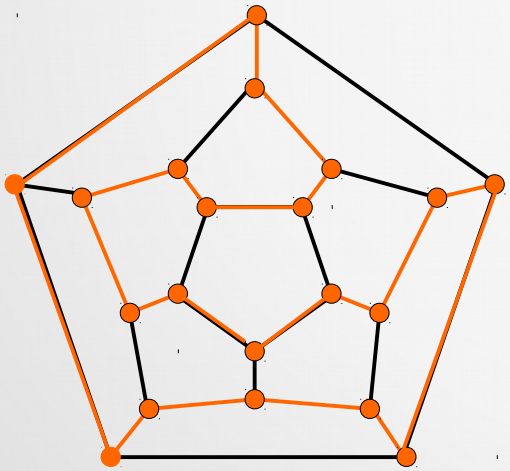
We will show the circuit on it.

Yes, there is such a circuit.

10.5 *Euler and Hamilton Paths*

Hamilton circuits and paths

The terminology comes from a game *Icozian puzzle*, invented in 1875 by Irish mathematician Sir William Rowan Hamilton.



The graph on the left is isomorphic to the graph consisting all the edges and vertices of a dodecahedron.

We will show the circuit on it.

Yes, there is such a circuit.

10.5 *Euler and Hamilton Paths*

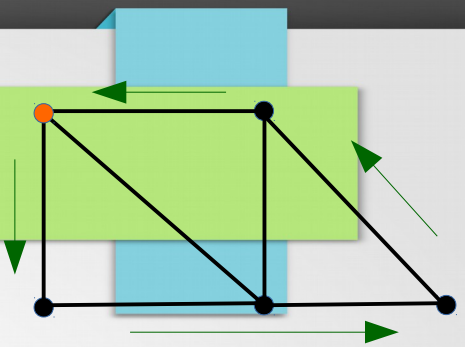
Hamilton circuits

There are no simple *necessary* and *sufficient* criteria for the existence of Hamilton circuits.

However, many theorems are known that give *sufficient* conditions for the existence of Hamilton circuits.

Also, certain properties can be used to show that a graph has no Hamilton circuit.

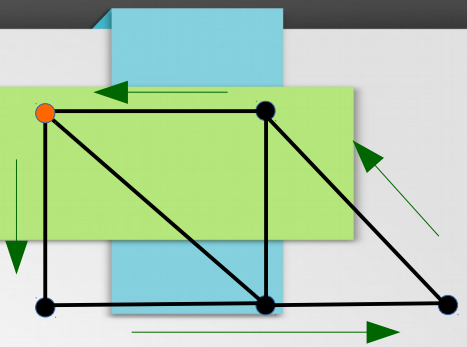
10.5 *Euler and Hamilton Paths*



Hamilton circuits

[Property 1] A graph with a vertex of degree 1 cannot have a Hamilton circuit.
Reason: in Hamilton circuit each vertex is incident with two edges.

10.5 *Euler and Hamilton Paths*

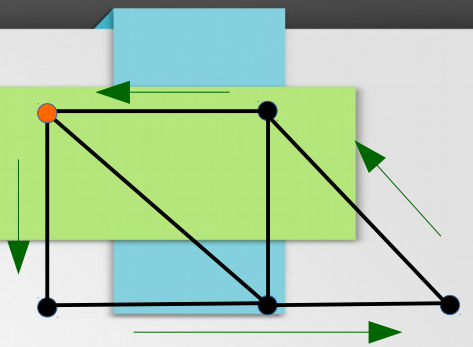


Hamilton circuits

[Property 1] A graph with a vertex of degree 1 cannot have a Hamilton circuit.
Reason: in Hamilton circuit each vertex is incident with two edges.

[Property 2] If a vertex in the graph has degree 2, then both edges that are incident with this vertex must be part of the Hamilton circuit.

10.5 *Euler and Hamilton Paths*



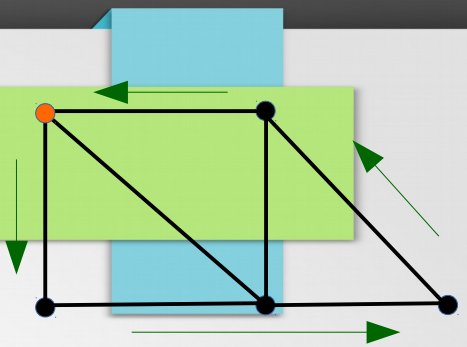
Hamilton circuits

[Property 1] A graph with a vertex of degree 1 cannot have a Hamilton circuit.
Reason: in Hamilton circuit each vertex is incident with two edges.

[Property 2] If a vertex in the graph has degree 2, then both edges that are incident with this vertex must be part of the Hamilton circuit.

[Property 3] When a Hamilton circuit is being constructed and it passed through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed from consideration.

10.5 *Euler and Hamilton Paths*



Hamilton circuits

[Property 4] A Hamilton circuit cannot contain a smaller circuit within it.

10.5 *Euler and Hamilton Paths*

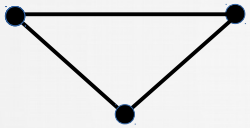
Hamilton circuits

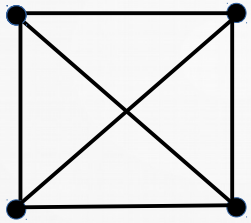
Example: show that K_n has a Hamilton circuit if $n \geq 3$.

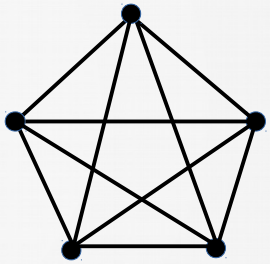
10.5 *Euler and Hamilton Paths*

Hamilton circuits

Example: show that K_n has a Hamilton circuit if $n \geq 3$.



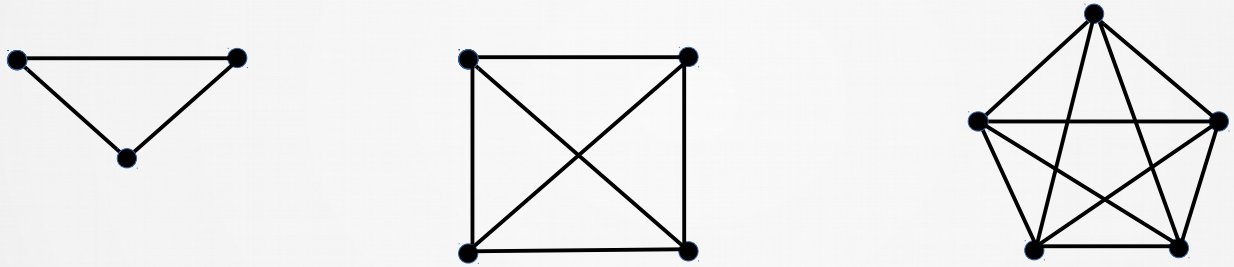




10.5 *Euler and Hamilton Paths*

Hamilton circuits

Example: show that K_n has a Hamilton circuit if $n \geq 3$.



Note that the more edges a graph has, the more likely it has a Hamilton circuit.

10.5 *Euler and Hamilton Paths*

Hamilton circuits

[Theorem] *Dirac's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$, then G has a *Hamilton circuit*.

10.5 *Euler and Hamilton Paths*

Hamilton circuits

[Theorem] *Dirac's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$, then G has a *Hamilton circuit*.

[Theorem] *Ore's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u)+\deg(v) \geq n$ for every pair of non-adjacent vertices u and v in G, then G has a *Hamilton circuit*.

10.5 *Euler and Hamilton Paths*

Hamilton circuits

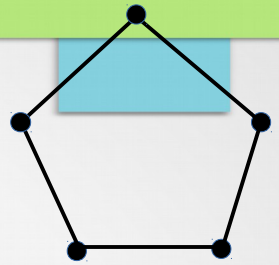
[Theorem] *Dirac's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$, then G has a *Hamilton circuit*.

[Theorem] *Ore's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u)+\deg(v) \geq n$ for every pair of non-adjacent vertices u and v in G, then G has a *Hamilton circuit*.

Both theorems provide sufficient conditions for a connected simple graph to have a Hamilton circuit.

10.5 *Euler and Hamilton Paths*

Hamilton circuits



[Theorem] *Dirac's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$, then G has a *Hamilton circuit*.

[Theorem] *Ore's Theorem*
If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u)+\deg(v) \geq n$ for every pair of non-adjacent vertices u and v in G, then G has a *Hamilton circuit*.

Example: graph C_5 has a Hamilton circuit, but doesn't satisfy the hypotheses of either Ore's theorem or Dirac's theorem.

10.5 *Euler and Hamilton Paths*

Hamilton circuits

The best algorithm known for finding a Hamilton circuit in a graph or determining that no such circuit exists have exponential worst-case time complexity (in the number of vertices of the graph)

10.5 *Euler and Hamilton Paths*

Applications of Hamilton circuits and paths

Traveling salesperson problem (TSP):
Asks for a shortest route a traveling salesperson should take to visit a set of cities.

10.5 *Euler and Hamilton Paths*

Applications of Hamilton circuits and paths

Traveling salesperson problem (TSP):
Asks for a shortest route a traveling salesperson should take to visit a set of cities.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

10.5 *Euler and Hamilton Paths*

Applications of Hamilton circuits and paths

Gray Codes:
Read about in the book
Example 8, page 702

10.6 *Shortest-paths Problems*

Many problems can be modeled using graphs with weights assigned to their edges.

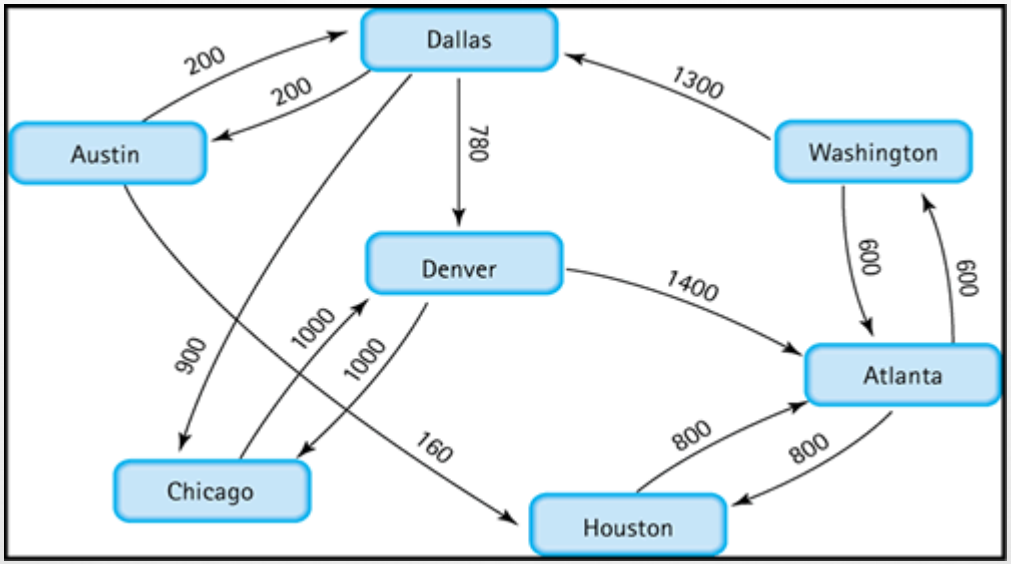
Graphs that have *weights* assigned to their edges are called *weighted graphs*.

10.6 *Shortest-paths Problems*

Many problems can be modeled using graphs with weights assigned to their edges.

Graphs that have *weights* assigned to their edges are called *weighted graphs*.

Example: flight times between some cities.

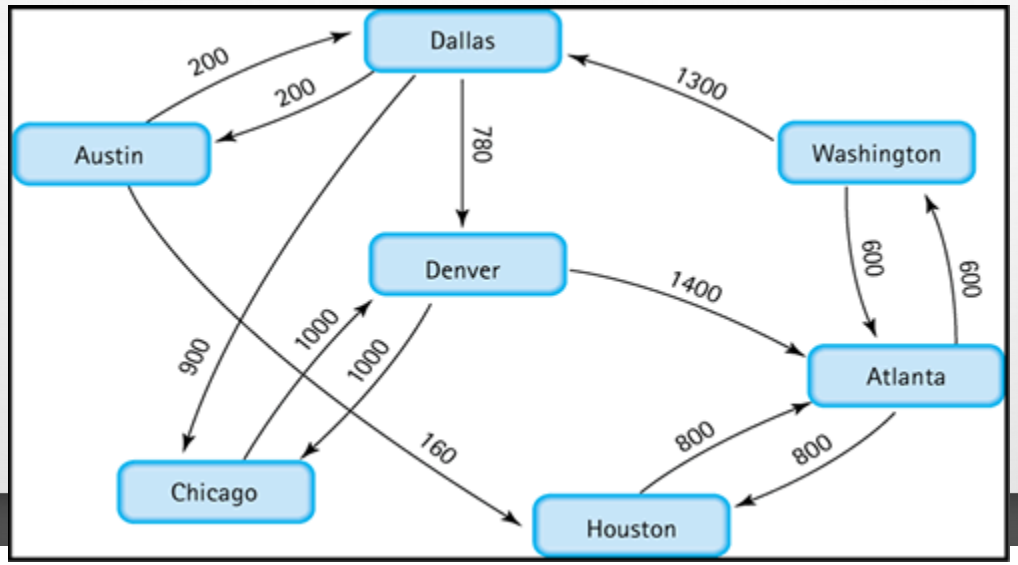


10.6 *Shortest-paths Problems*

Many problems can be modeled using graphs with weights assigned to their edges.

Graphs that have *weights* assigned to their edges are called *weighted graphs*.

The *length of a path in a weighted graph* is the sum of the weights of the edges in this graph.

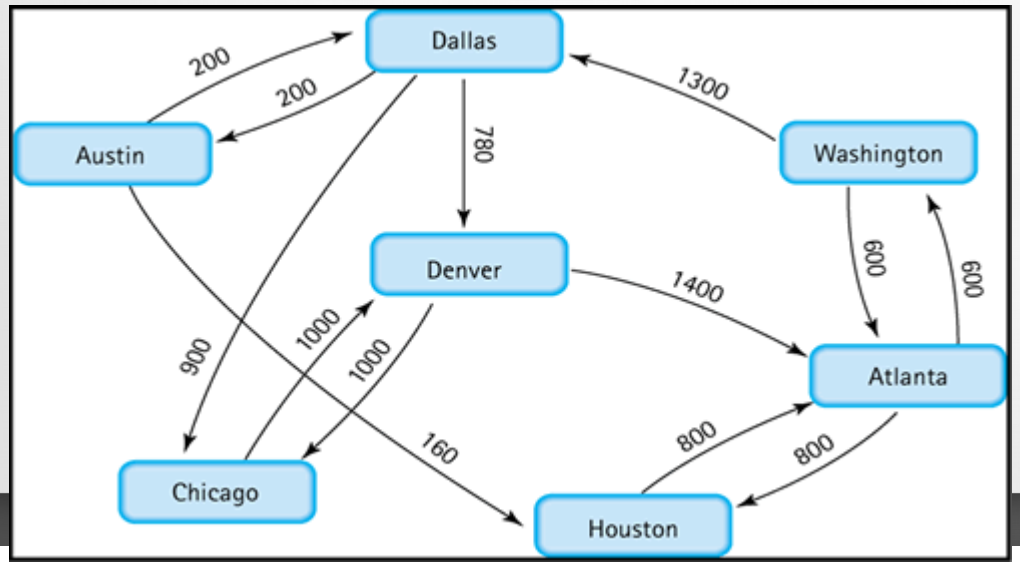


10.6 *Shortest-paths Problems*

Many problems can be modeled using graphs with weights assigned to their edges.

Graphs that have *weights* assigned to their edges are called *weighted graphs*.

The *length of a path in a weighted graph* is the sum of the weights of the edges in this graph.



A question:
What is the shortest path between Austin and Atlanta?

10.6 *Shortest-paths Problems*

There are several different algorithms that find a shortest path between two vertices in a weighted graph.

Dijkstra's algorithm given in the Rosen book allows to find a shortest path between two vertices in an undirected weighted graph.

In CSI 33 you will study Dijkstra's algorithm that works on directed graphs as well.

10.6 *Shortest-paths Problems*

Dijkstra's algorithm

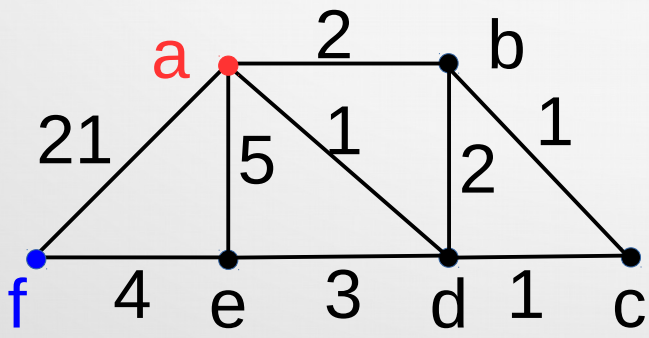
procedure *Dijkstra*(G: weighted connected simple graph, with all weights positive)
{G=(V,E) has vertices a = v_0, …, v_n = z, and weights $w(v_i,v_j)$, where $w(v_i,v_j) = \infty$ if $\{v_i,v_j\} \notin$ E}

for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty

while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

10.6 *Shortest-paths Problems*

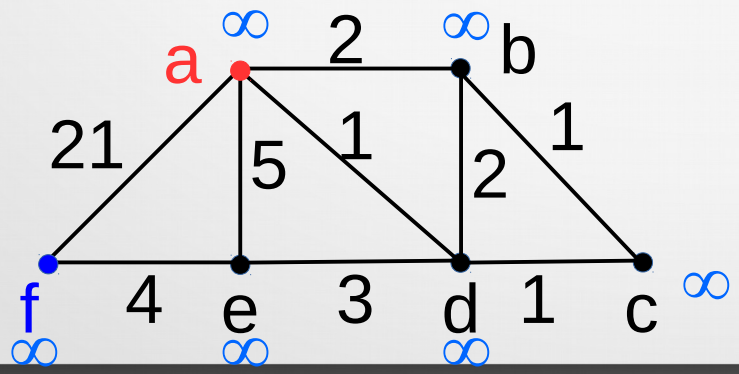
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



Goal: find a shortest path from a to f

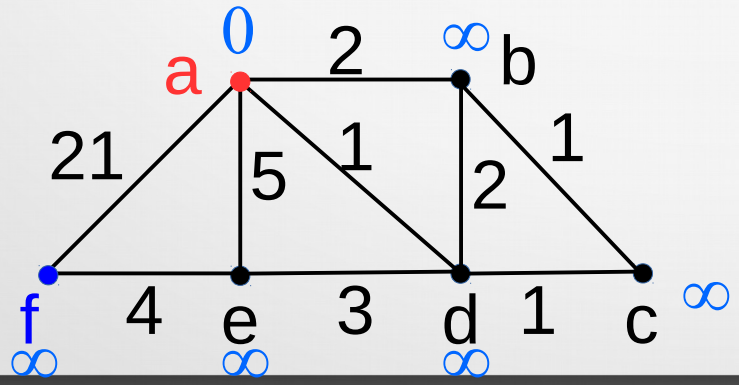
10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



10.6 *Shortest-paths Problems*

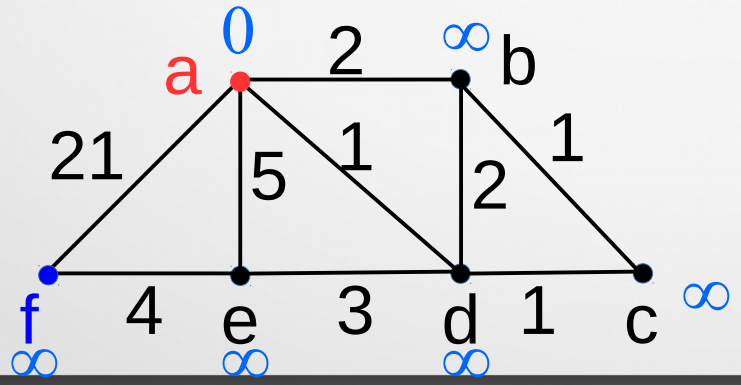
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
$L(a) := 0, S := \varnothing$ // set a's label to 0 and set S is empty infty
while $z \notin S$
    u := a vertex not in S with smallest L(u)
    $S := S \cup \{u\}$ // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := ∅

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

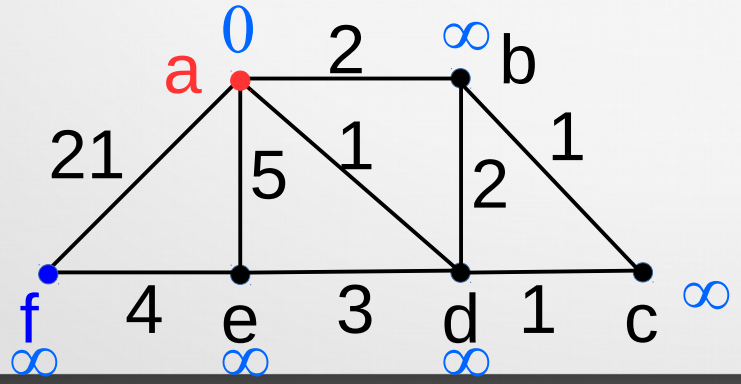


S := ∅

u := a, because L(a) = 0

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

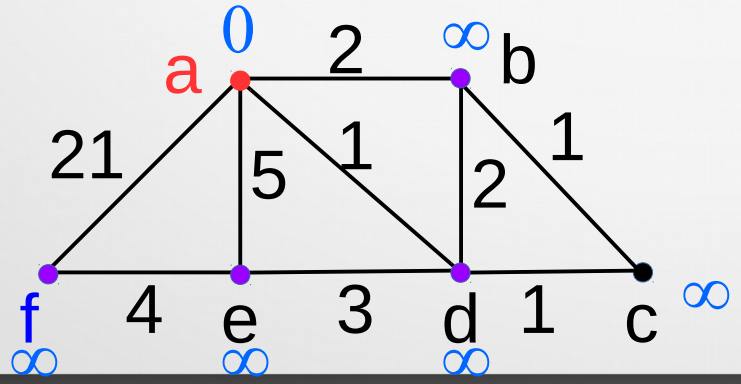


S := {a}

u := a, because L(a) = 0

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

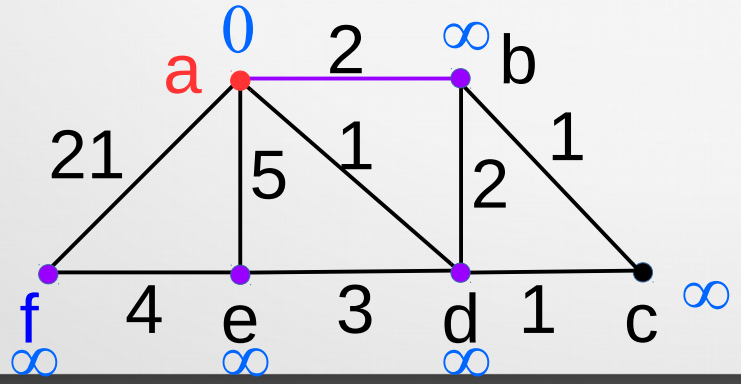


S := {a}

u := a, because L(a) = 0
v's: b,d,e,f (adjacent to a)

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

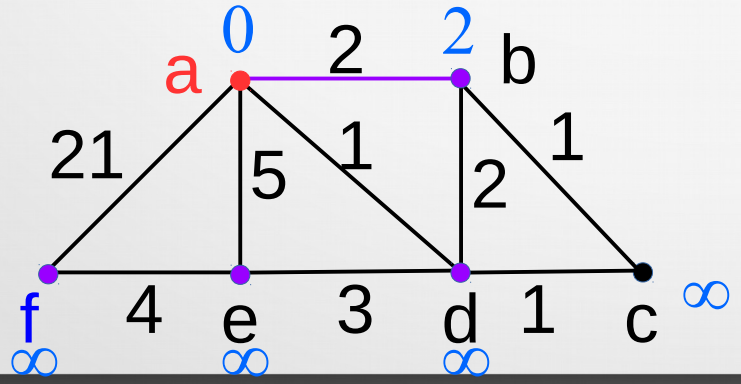
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 2 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

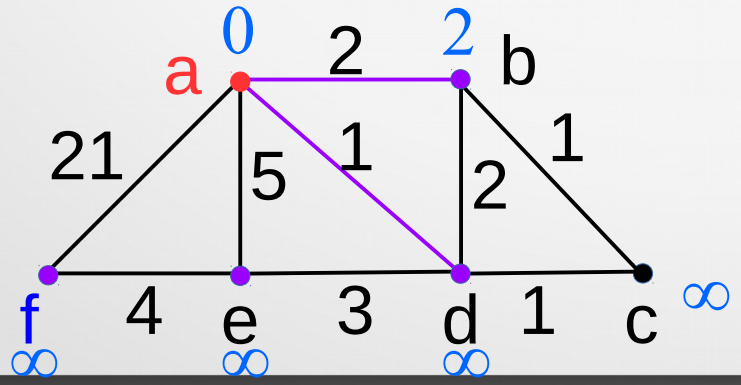
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 2 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

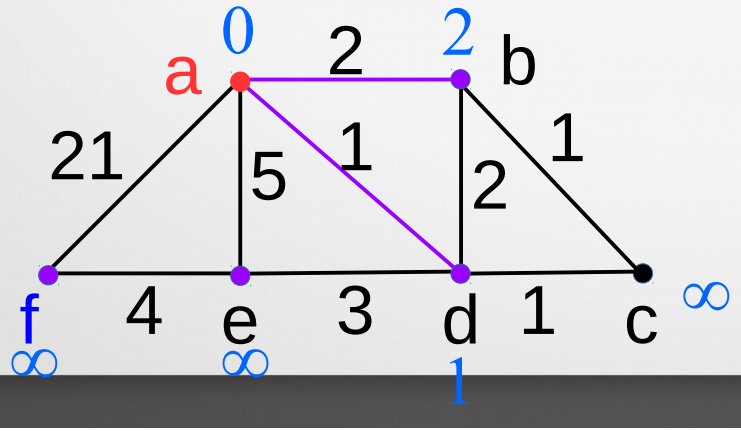
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 1 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

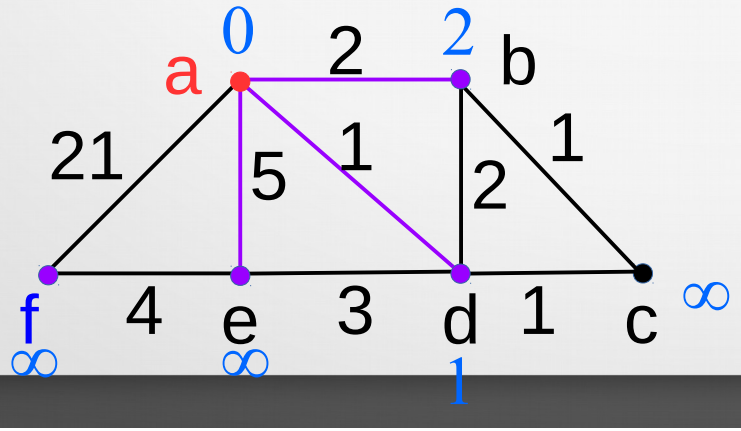
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 1 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

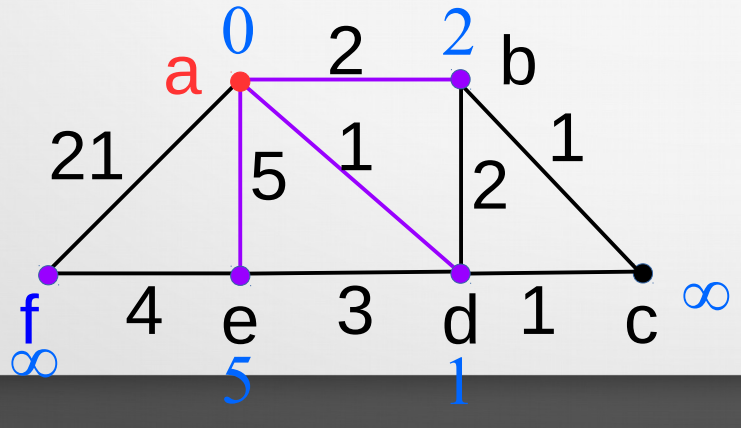
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 5 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

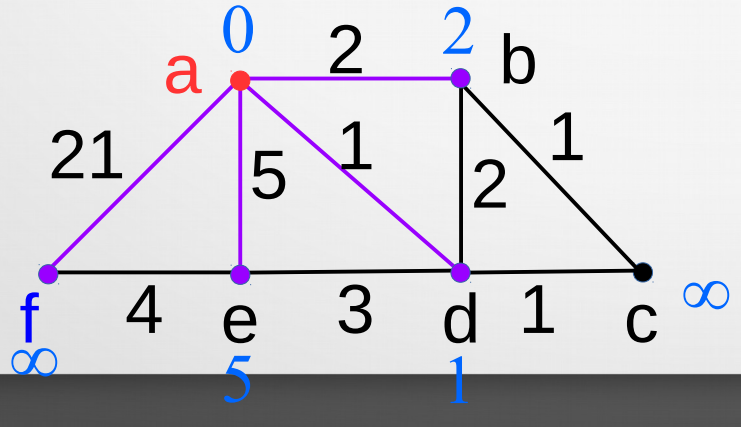
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 5 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

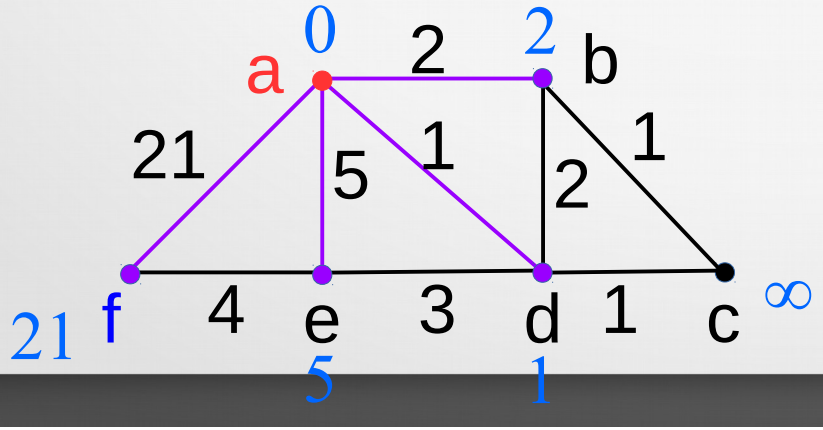
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 21 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

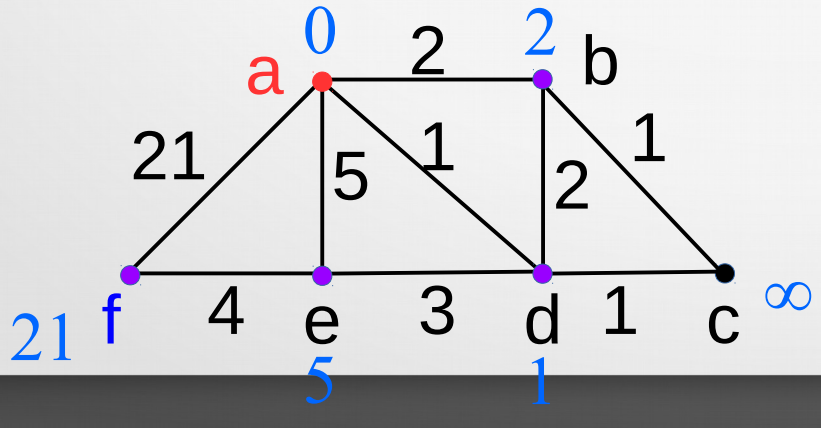
u := a, because L(a) = 0

v's: b,d,e,f (adjacent to a)

0 + 21 < ∞

10.6 *Shortest-paths Problems*

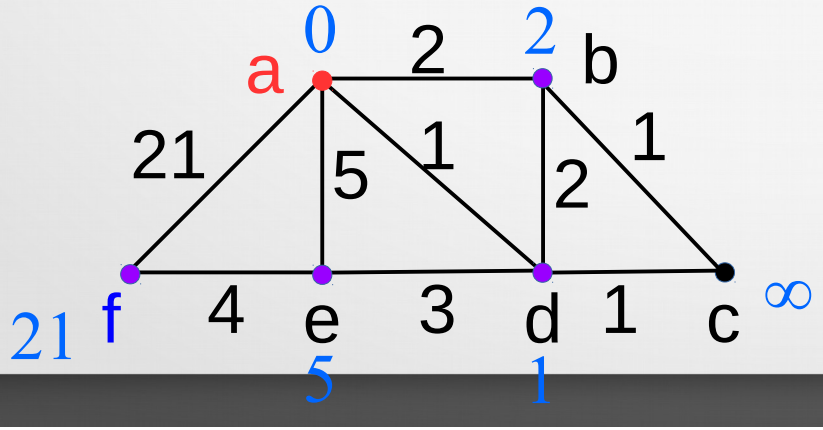
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a}

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

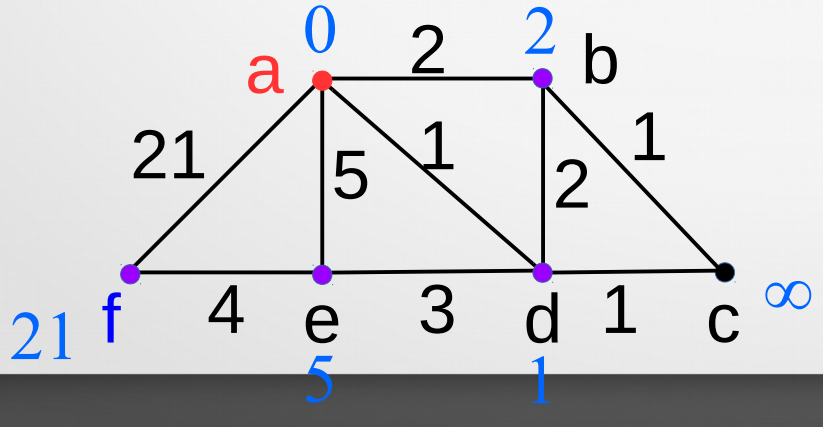


S := {a}

u := d, because L(d) = 1

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

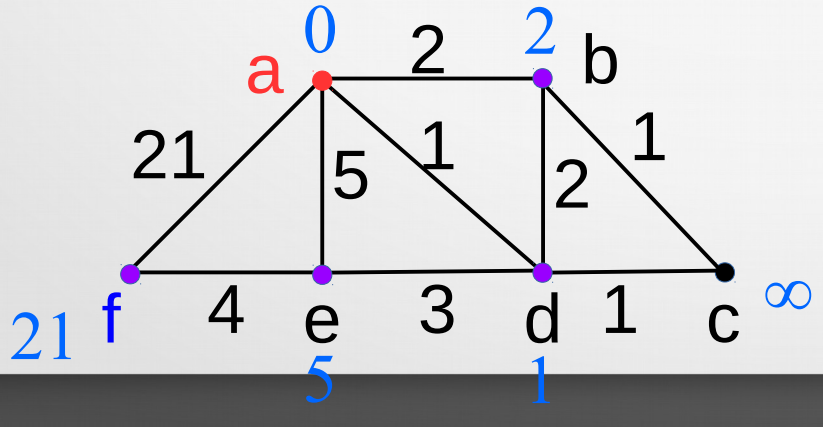


S := {a, d}

u := d, because L(d) = 1

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z

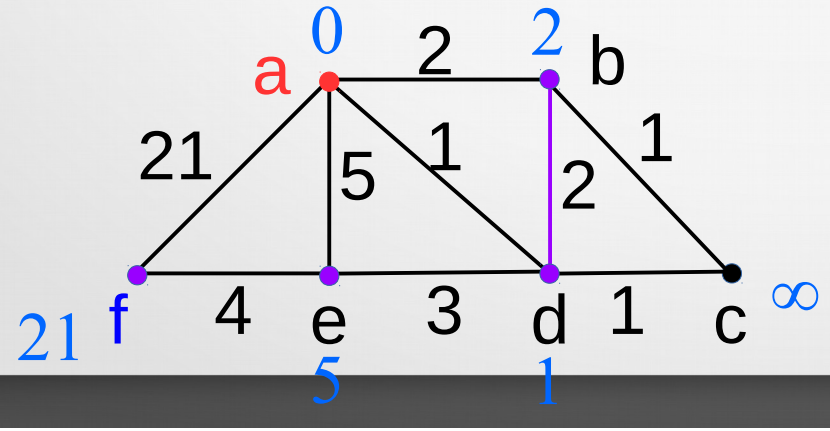


S := {a, d}

u := d, because L(d) = 1
v's: b, c, e

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}

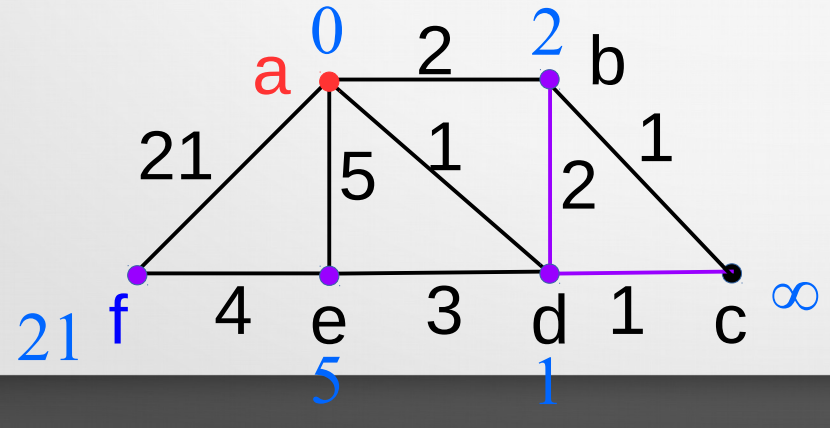
u := d, because L(d) = 1

v's: b, c, e

1+2 < 2 False

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}

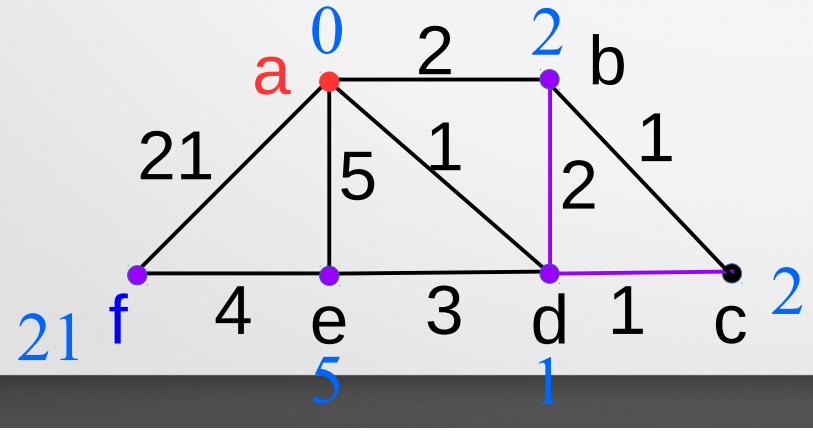
u := d, because L(d) = 1

v's: b, c, e

1+1 < ∞

10.6 *Shortest-paths Problems*

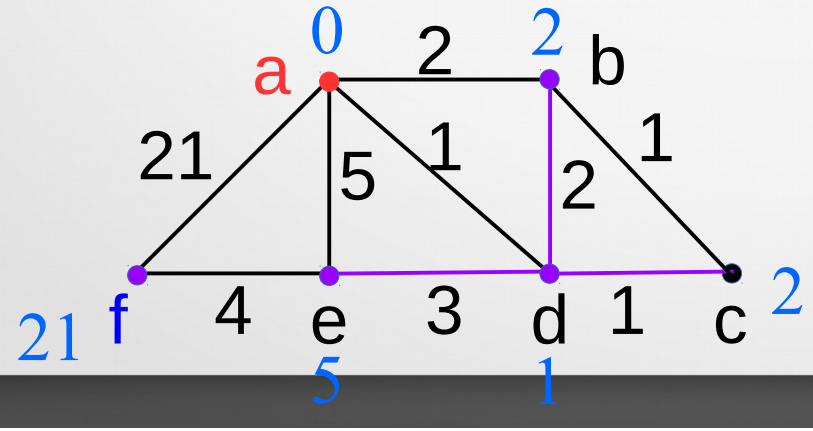
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}
u := d, because L(d) = 1
v's: b, c, e
1+1 < ∞

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}

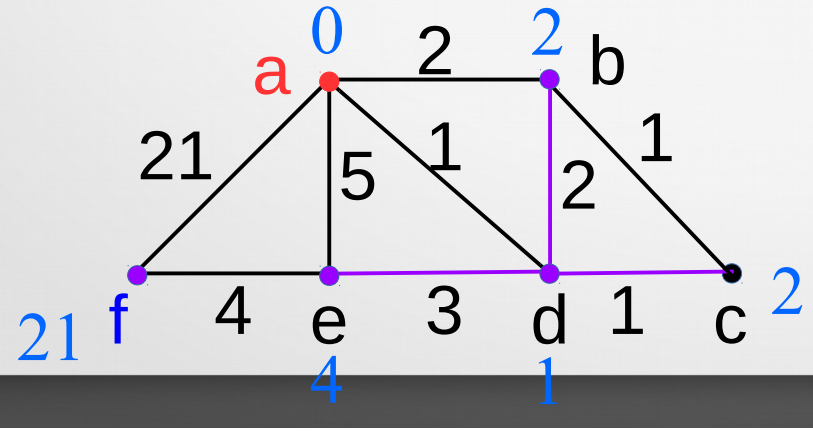
u := d, because L(d) = 1

v's: b, c, e

1+3 < 5

10.6 *Shortest-paths Problems*

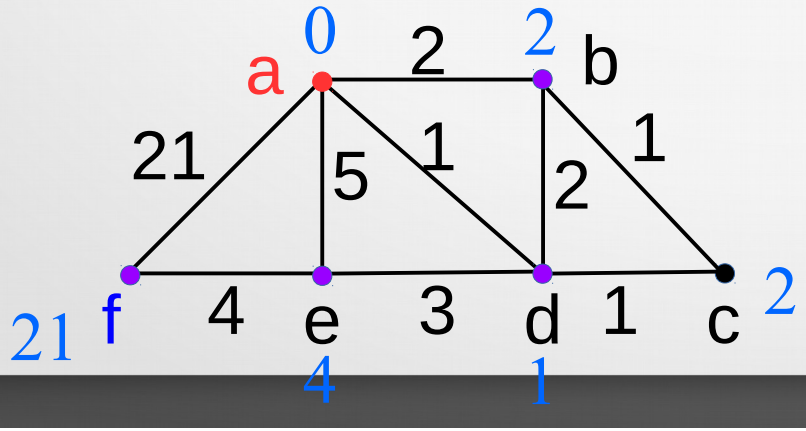
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}
u := d, because L(d) = 1
v's: b, c, e
1+3 < 5

10.6 *Shortest-paths Problems*

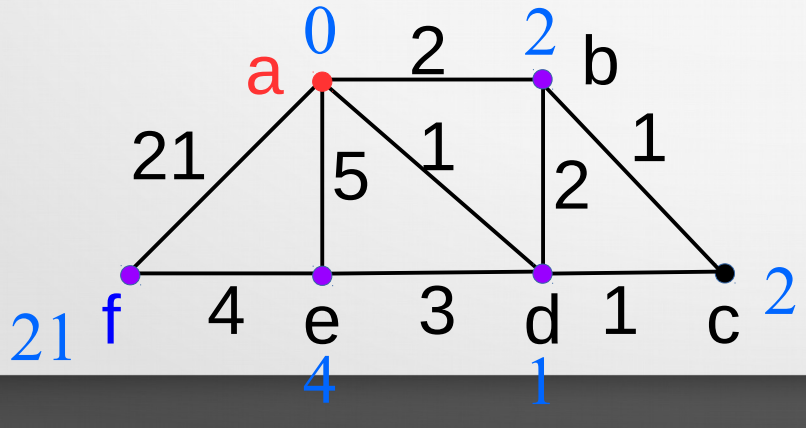
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}

10.6 *Shortest-paths Problems*

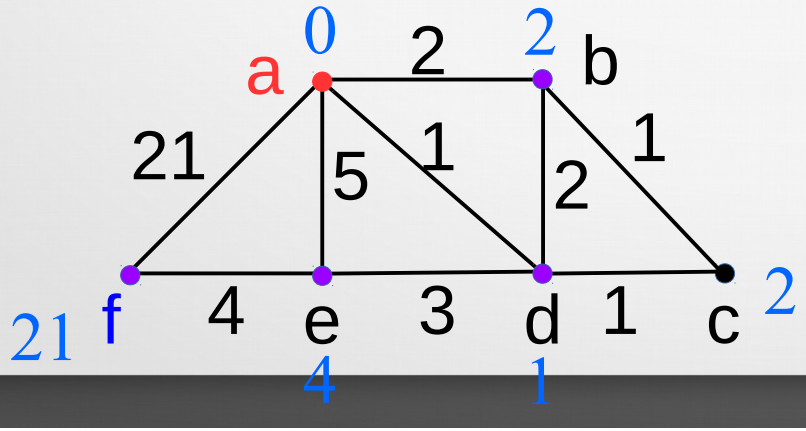
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
→ u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d}
u := b, because L(b) = 2
and b comes before c

10.6 *Shortest-paths Problems*

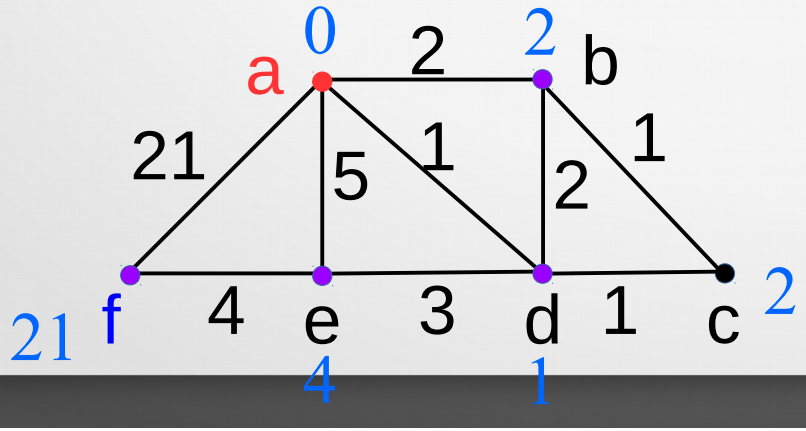
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
→ S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b}
u := b, because L(b) = 2
and b comes before c

10.6 *Shortest-paths Problems*

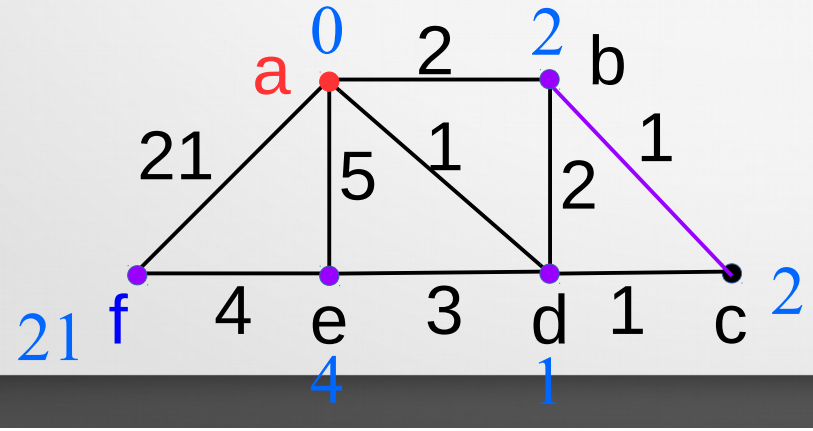
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b}
u := b, because L(b) = 2
and b comes before c
v's: c

10.6 *Shortest-paths Problems*

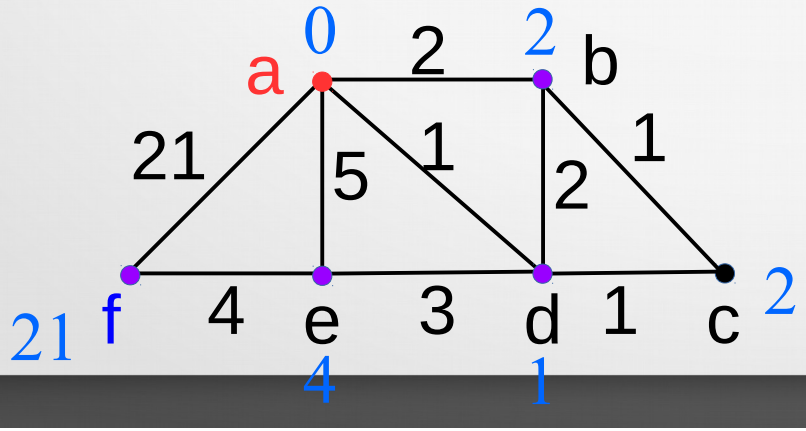
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b}
u := b, because L(b) = 2
and b comes before c
v's: c
2+1 < 2 False

10.6 *Shortest-paths Problems*

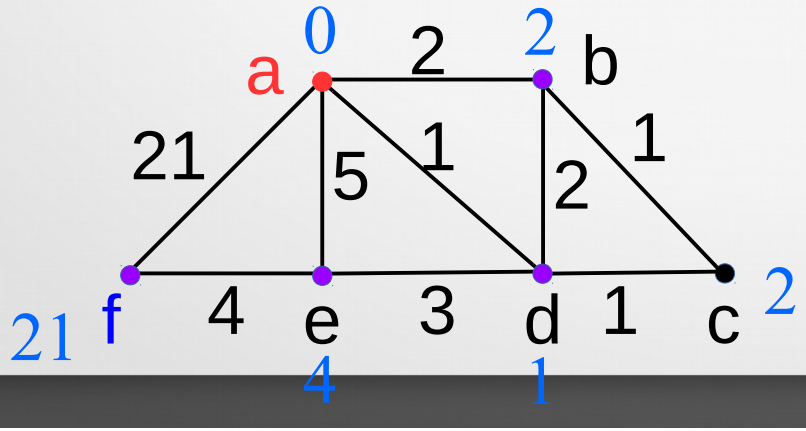
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b}

10.6 *Shortest-paths Problems*

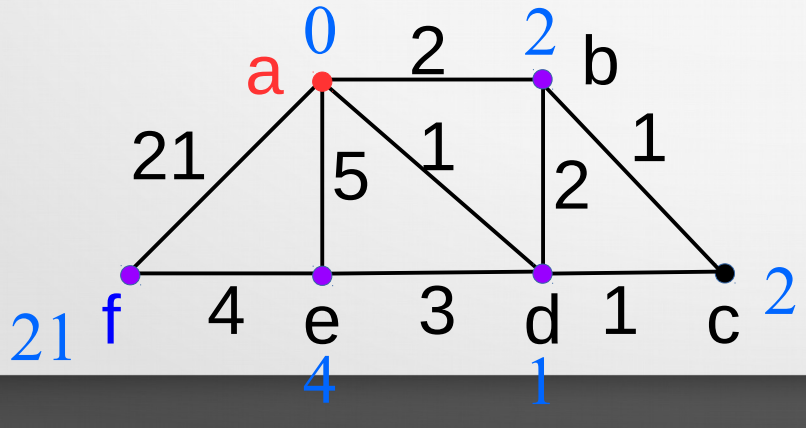
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
→ u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b}
u := c, because L(c) = 2

10.6 *Shortest-paths Problems*

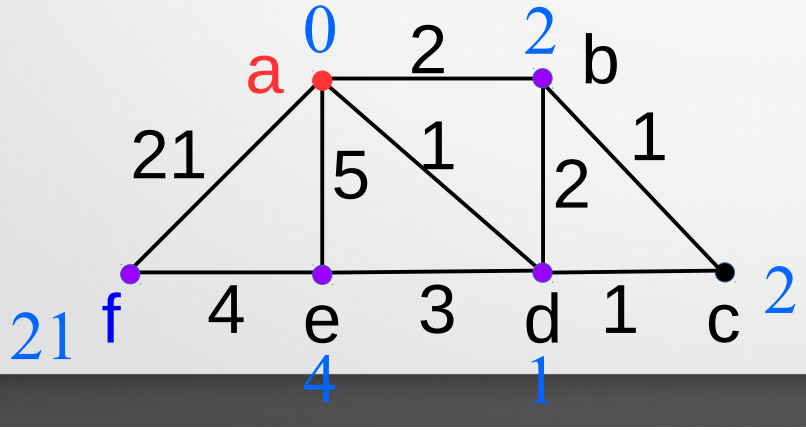
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
→ S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c}
u := c, because L(c) = 2

10.6 *Shortest-paths Problems*

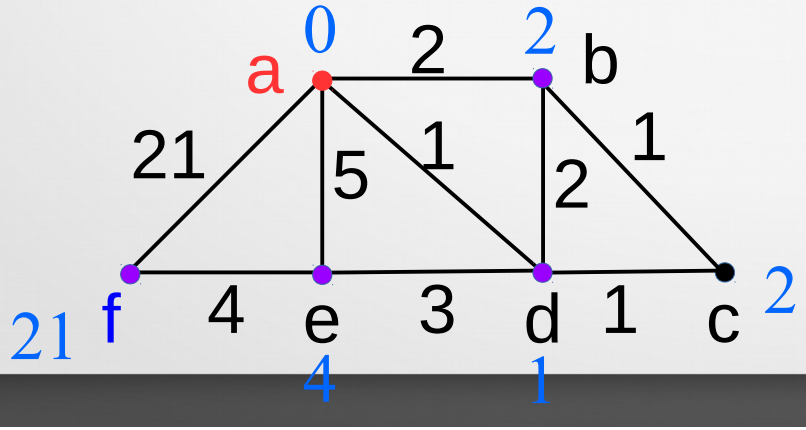
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c}
u := c, because L(c) = 2
v's: ∅

10.6 *Shortest-paths Problems*

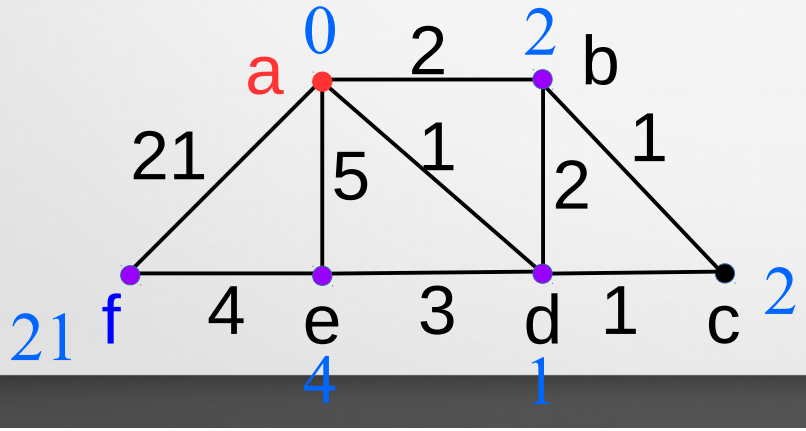
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c}

10.6 *Shortest-paths Problems*

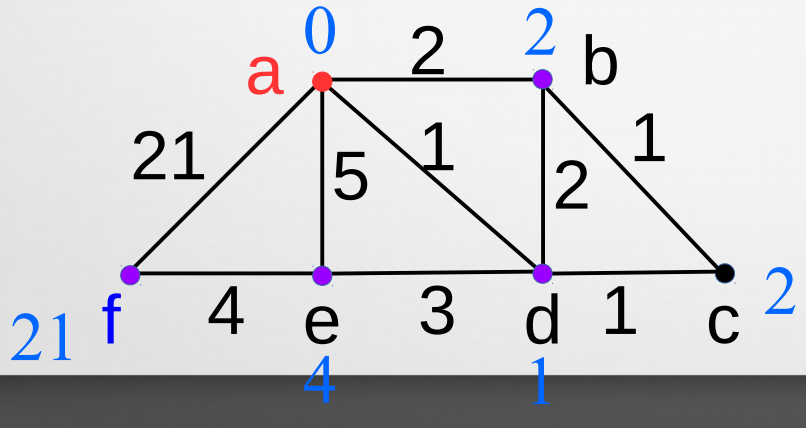
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
→ u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c}
u := e, because L(e) = 4

10.6 *Shortest-paths Problems*

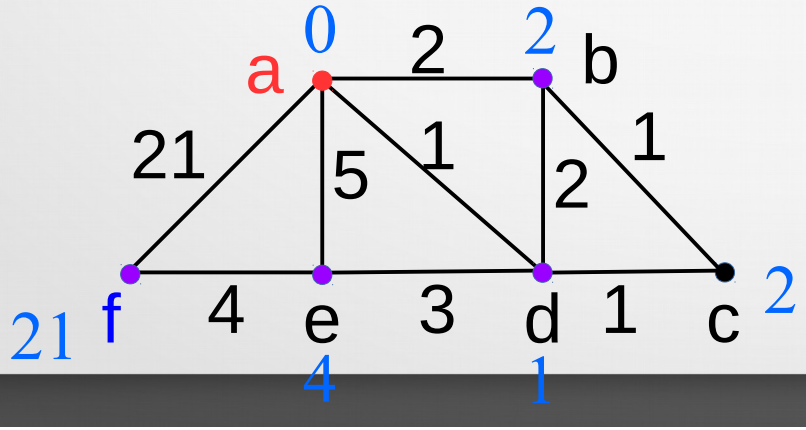
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
→ S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}
u := e, because L(e) = 4

10.6 *Shortest-paths Problems*

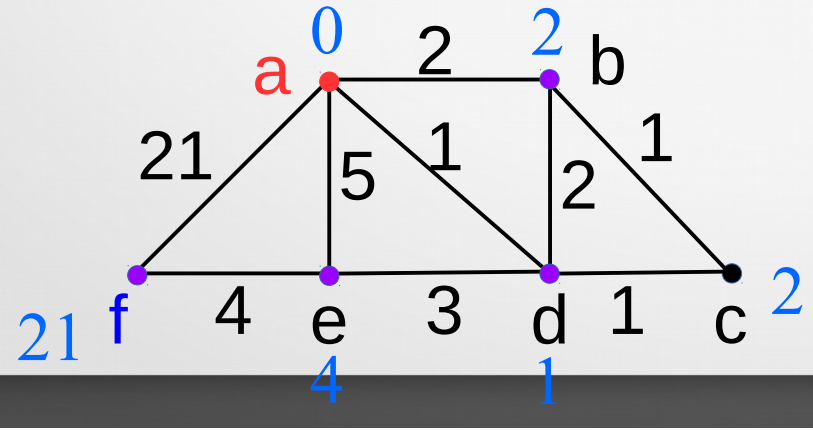
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}
u := e, because L(e) = 4
v's: f

10.6 *Shortest-paths Problems*

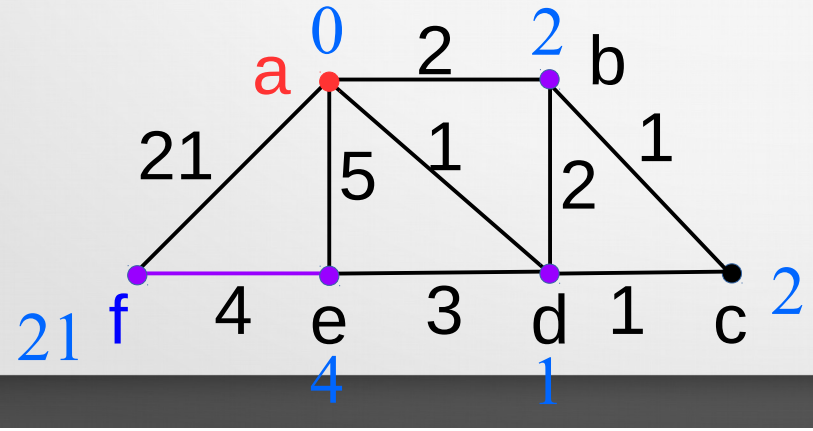
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}
u := e, because L(e) = 4
v's: f

10.6 *Shortest-paths Problems*

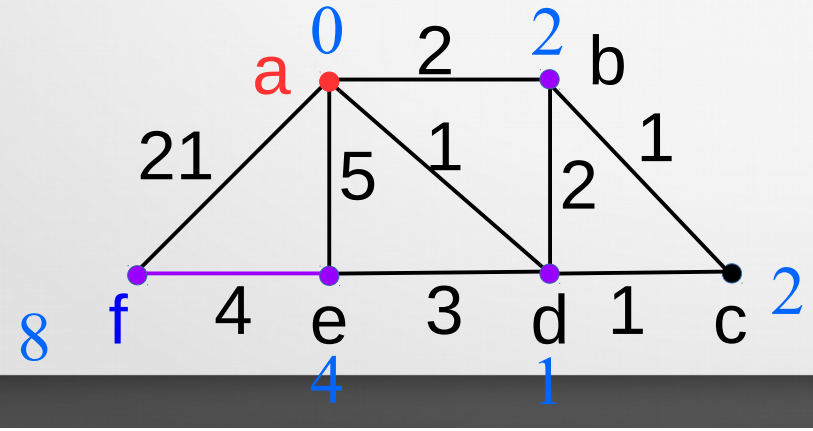
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}
u := e, because L(e) = 4
v's: f
4+4 < 21

10.6 *Shortest-paths Problems*

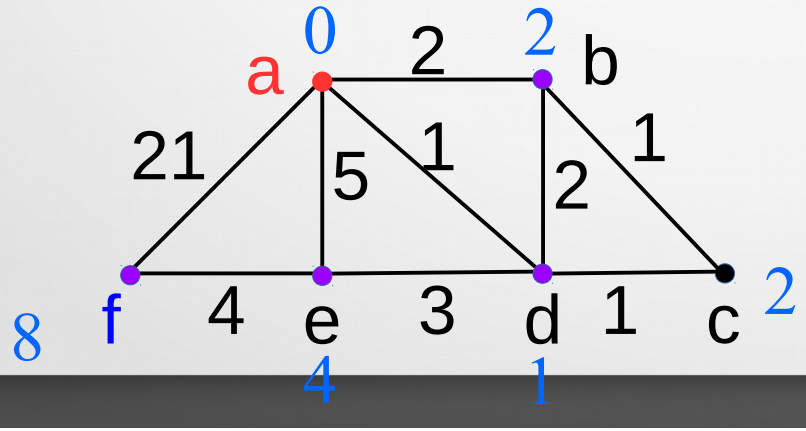
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
    → **if** L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}
u := e, because L(e) = 4
v's: f
4+4 < 21

10.6 *Shortest-paths Problems*

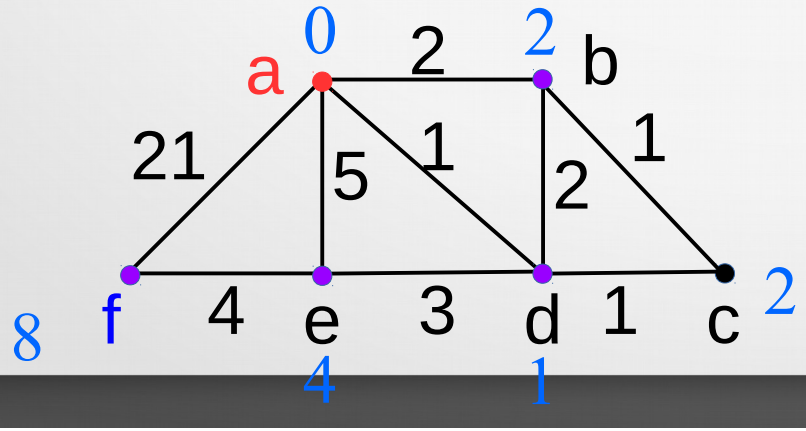
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}

10.6 *Shortest-paths Problems*

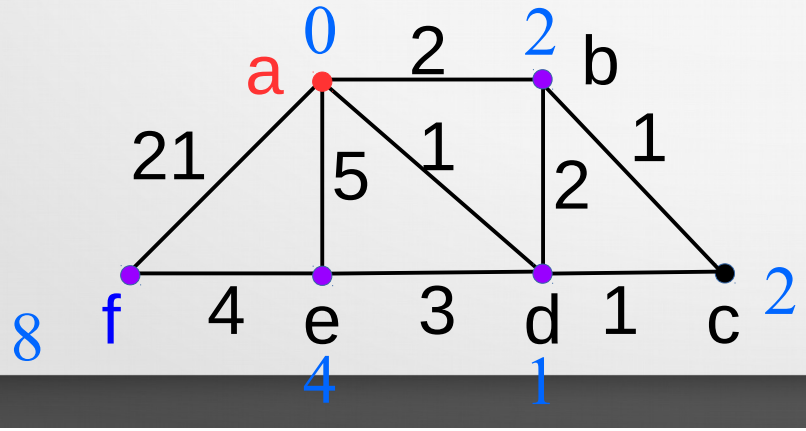
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
→ u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e}
u := f

10.6 *Shortest-paths Problems*

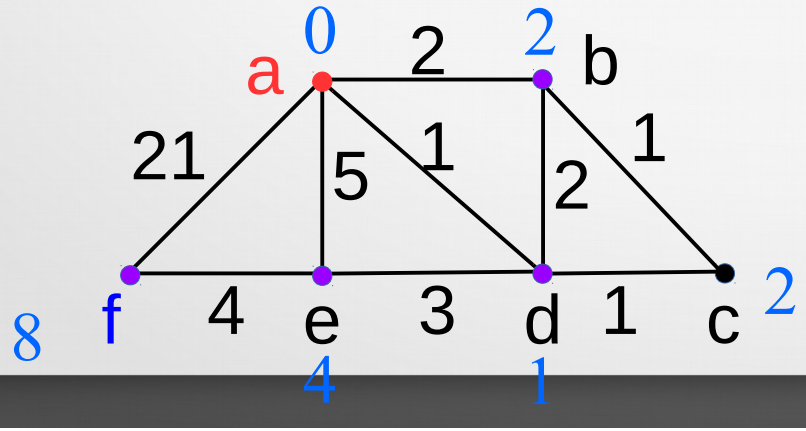
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
→ S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e, f}
u := f

10.6 *Shortest-paths Problems*

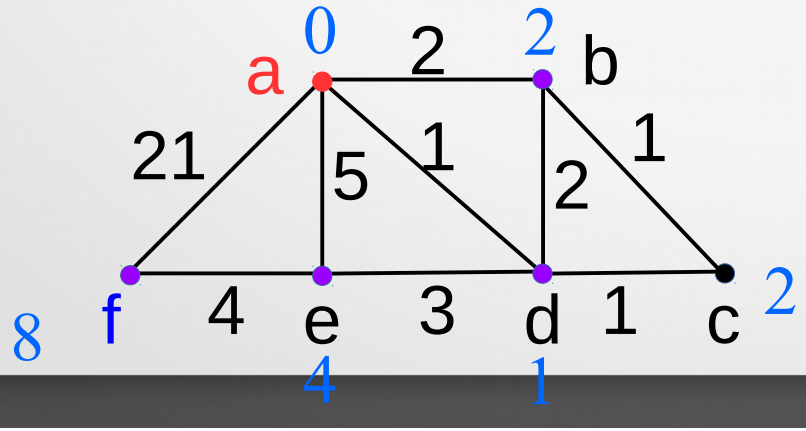
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∉ S
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e, f}
u := f
v's: ∅

10.6 *Shortest-paths Problems*

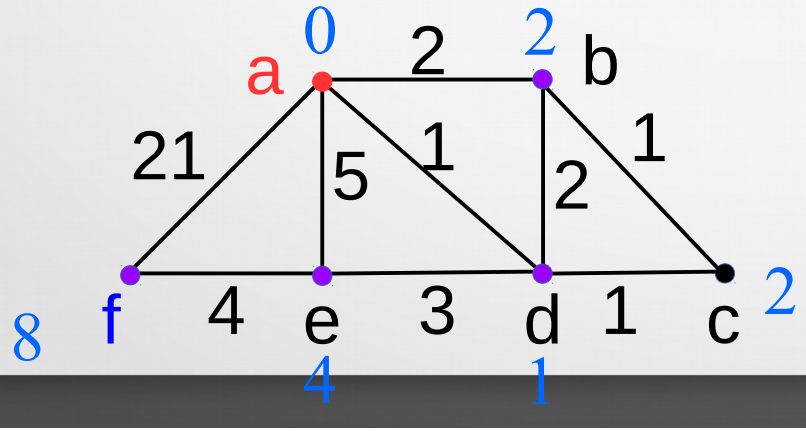
procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∈ S STOP!
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e, f}

10.6 *Shortest-paths Problems*

procedure *Dijkstra*(G)
for i:= 1 **to** n // set all the labels to infty
    $L(v_i) := \infty$
L(a) := 0, S := ∅ // set a's label to 0 and set S is empty infty
while z ∉ S f ∈ S STOP!
    u := a vertex not in S with smallest L(u)
    S := S ∪ {u} // adding a vertex to S
    for all vertices v not in S // update the labels
        if L(u) + w(u,v) < L(v) **then** L(v) := L(u) + w(u,v)
return L(z) // return the length of a shortest path from a to z



S := {a, d, b, c, e, f}

return 8

10.6 *Shortest-paths Problems*

[Theorem] Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.

[Theorem] Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected undirected simple weighted graph with n vertices.

10.5 *Euler and Hamilton Paths*

Traveling salesperson problem (TSP):

Find a shortest route a traveling salesperson should take to visit a set of cities, assuming that the edges have weights.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

10.5 *Euler and Hamilton Paths*

Traveling salesperson problem (TSP):
Find a shortest route a traveling salesperson should take to visit a set of cities, assuming that the edges have weights.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

Straightforward approach: examine all possible Hamilton circuits and select one of minimum length.

10.5 *Euler and Hamilton Paths*

Traveling salesperson problem (TSP):
Find a shortest route a traveling salesperson should take to visit a set of cities, assuming that the edges have weights.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

Straightforward approach: examine all possible Hamilton circuits and select one of minimum length.

It is too expensive!
Once a starting point is chosen, there are *(n-1)!* different Hamilton circuits to examine
(2^{nd} vertex: n-1 choices, 3^{rd} vertex: n-2 choices, ...)

10.5 *Euler and Hamilton Paths*

Traveling salesperson problem (TSP):
Find a shortest route a traveling salesperson should take to visit a set of cities, assuming that the edges have weights.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

Straightforward approach: examine all possible Hamilton circuits and select one of minimum length.

It is too expensive!
We can reduce it to *(n-1)! / 2* different Hamilton circuits to examine to explore because Hamilton circuits can be reversed, but it is still too expensive

10.5 *Euler and Hamilton Paths*

Traveling salesperson problem (TSP):
Find a shortest route a traveling salesperson should take to visit a set of cities, assuming that the edges have weights.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

A great deal of effort was devoted to designing efficient algorithms to solve it, but still no polynomial worst-case time complexity algorithm is found so far.

10.5 *Euler and Hamilton Paths*

Traveling salesperson problem (TSP):
Find a shortest route a traveling salesperson should take to visit a set of cities, assuming that the edges have weights.

This problem can be reduced to finding a Hamilton circuit in a complete graph such that the total weight of its edges as small as possible.

A practical approach is to use *approximation algorithm*. These are algorithms that do not necessary produce the exact solution, but instead are guaranteed to produce a solution close to an exact solution.
i.e. they will produce a circuit with total length/weight W' such that $W \leq W' \leq cW$, c is a constant.