Graph Algorithms

Maximum Flow

Sara Charmchi
2023

Today

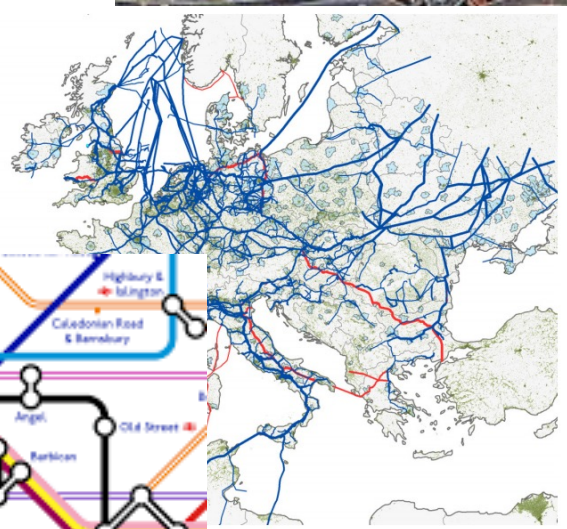
Maximum Flow

- Agenda:
  - Flow networks
  - Ford-Fulkerson algorithm

Flow networks

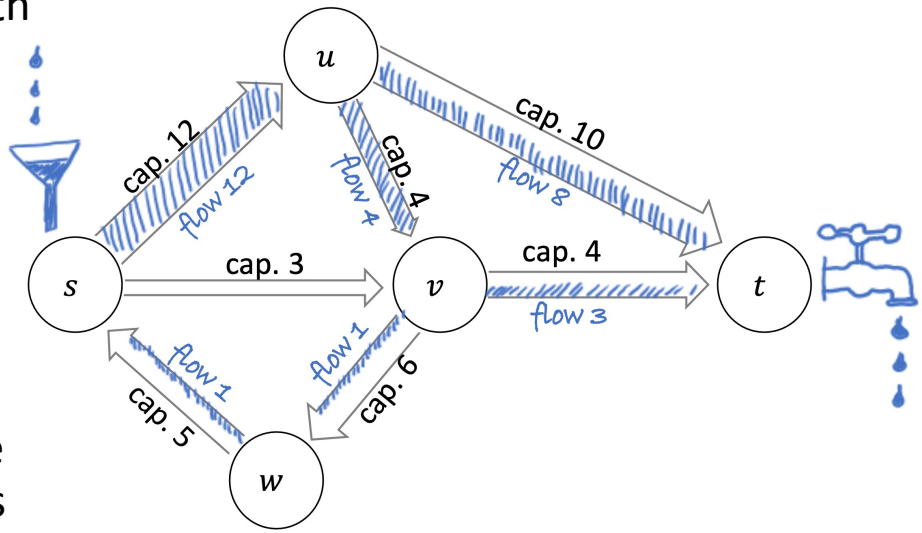
- What is the maximum amount of stuff that can be carried between a given pair of vertices?

- Oil → pipeline network
- Car → road network
- Trips → tube network
- …



Flow networks

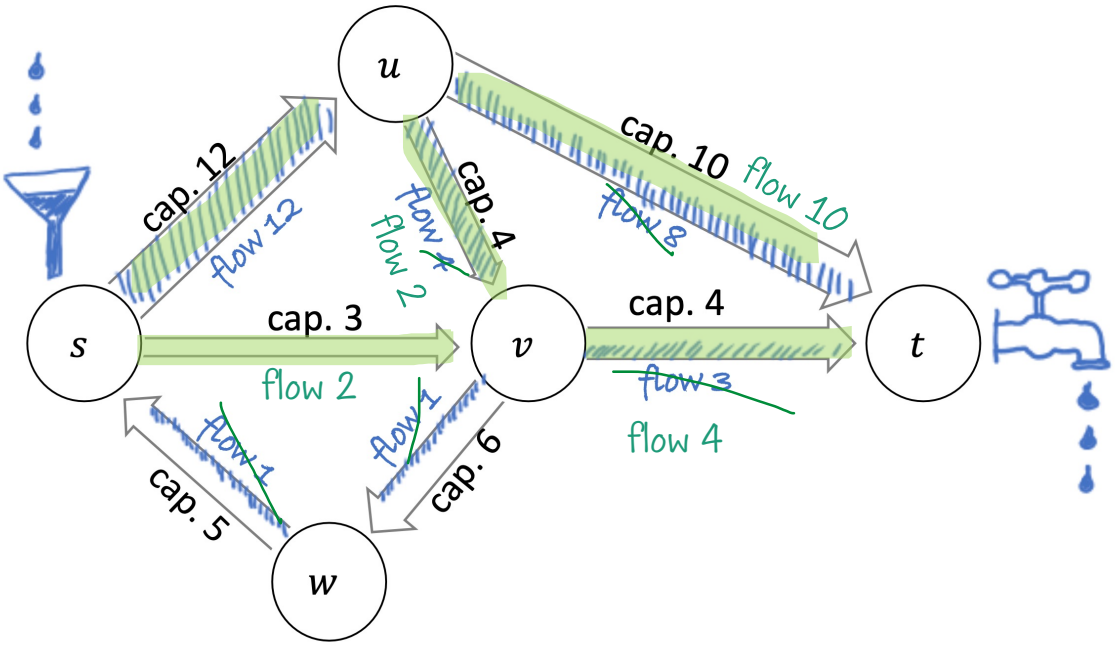
- Source vertex s where flow originates
- Sink vertex t where flow is consumed
- The edges are directed, and labelled with their capacities.



- The flow value is the net flow out of the source vertex, and it's 12 − 1 = 11 in this picture. This is equal to the net flow into the sink vertex, of course.

Flow network

What's the maximum possible flow value, over all possible flows?

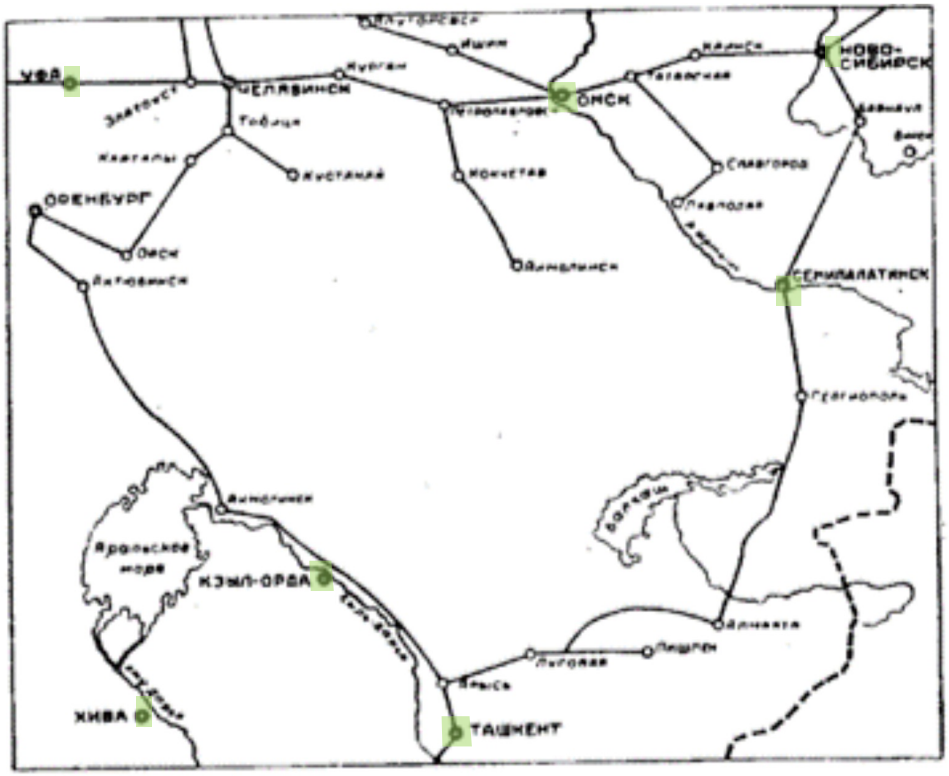


The total capacity of the edges going into the sink is 14, so it's impossible to have a flow of value > 14. Therefore the maximum possible flow value is 14.

Two transportation problems

Soviet 1930

- “Given a graph with edge capacities, and a list of source vertices and their supply capacities, and a list of destination vertices and their demands, find a flow that meets the demands.”



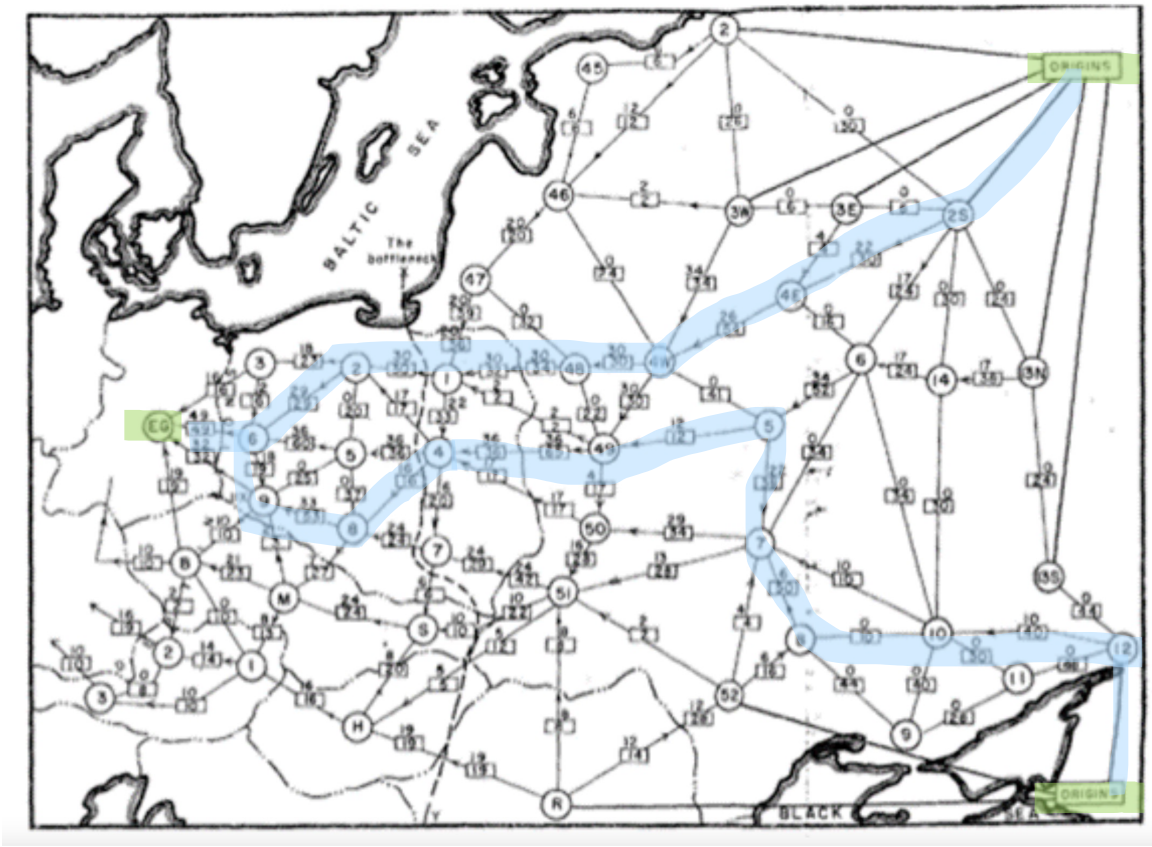
From Methods of find- ing the minimum total kilometrage in cargo- transportation planning in space, A.N.Tolstoy, 1930.

- In this illustration, the circles mark sources and sinks for cargo, from Omsk in the north to Tashkent in the south.

Two transportation problems

Soviet-USA 1955

- What is the max flow from ORIGINS to EG?
- if the US Air Force wants to strike and degrade one of the links, which link should it target in order to reduce the max flow?



From Fundamentals of a method for evaluating rail net capacities, T.E. Harris and F.S. Ross, 1955, a report by the RAND Corporation for the US Air Force (declassified by the Pentagon in 1999).

Flow network

Problem statement

- Given a directed graph with a **source** vertex s and a **sink** vertex t, where each edge $u \to v$ has a **capacity** $c(u \to v) > 0$,
- A **flow** $\boldsymbol{f}$ is a set of edge labels $f(u \to v)$ such that :
  - $0 < f(u \to v) < c(u \to v)$ on every edge
  - Total flow in = total flow out, at all vertices other that s and t.

  And the value of the flow is :
  - $value(f) = net\ flow\ out\ of\ s = net\ flow\ into\ t$

Flow network

Problem statement

- Given a directed graph with a **source** vertex s and a **sink** vertex t, where each edge $u \to v$ has a **capacity** $c(u \to v) > 0$,
- A **flow** f is a set of edge labels $f(u \to v)$ such that :
  - $0 < f(u \to v) < c(u \to v)$ on every edge
  - Total flow in = total flow out, at all vertices other that s and t.

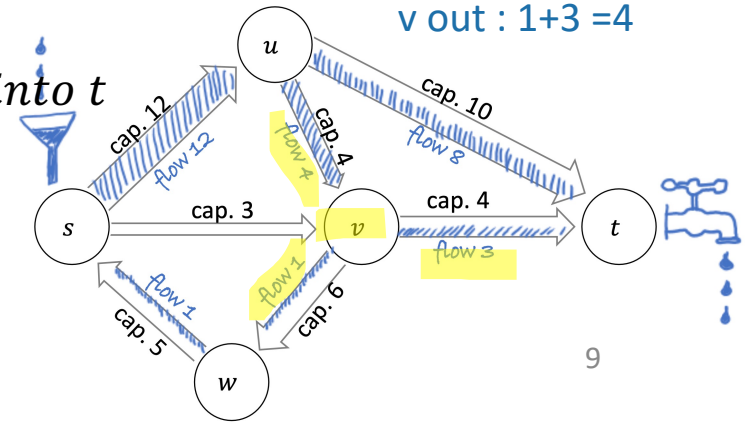
    Flow conservation

$$\sum_{u:\ u \to v} f(u \to v) = \sum_{w:\ v \to w} f(v \to w) \quad \text{at all vertices } v \in V \setminus \{s, t\}.$$

And the value of the flow is :

- $value(f) = net\ flow\ out\ of\ s = net\ flow\ into\ t$

v in : 4
v out : 1+3 =4



Flow network

Problem statement

- Given a directed graph with a **source** vertex s and a **sink** vertex t, where each edge $u \to v$ has a **capacity** $c(u \to v) > 0$,
- A **flow** $\boldsymbol{f}$ is a set of edge labels $f(u \to v)$ such that :
  - $0 < f(u \to v) < c(u \to v)$ on every edge
  - Total flow in = total flow out, at all vertices other that s and t.

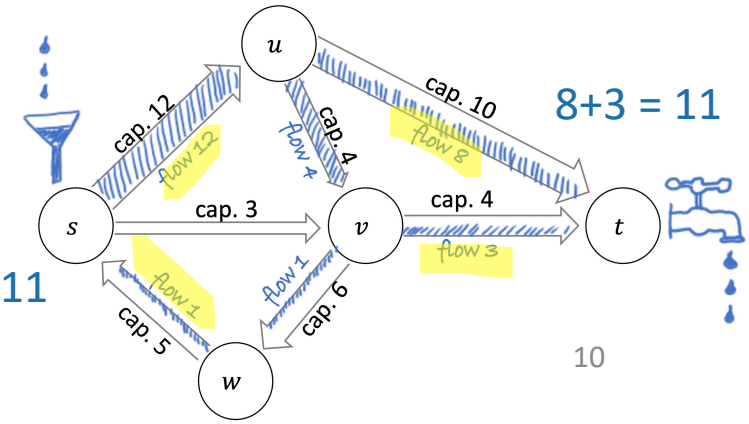
    Flow conservation

$$\sum_{u:\ u \to v} f(u \to v) = \sum_{w:\ v \to w} f(v \to w) \quad \text{at all vertices } v \in V \setminus \{s, t\}.$$

And the value of the flow is :

- $value(f)$ =

  $net\ flow\ out\ of\ s = net\ flow\ into\ t$



Flow network

Problem statement

- Given a directed graph with a **source** vertex s and a **sink** vertex t, where each edge $u \to v$ has a **capacity** $c(u \to v) > 0$,
- A **flow** f is a set of edge labels $f(u \to v)$ such that :
  - $0 < f(u \to v) < c(u \to v)$ on every edge
  - Total flow in = total flow out, at all vertices other that s and t.

    Flow conservation

$$\sum_{u:\, u \to v} f(u \to v) = \sum_{w:\, v \to w} f(v \to w) \quad \text{at all vertices } v \in V \setminus \{s, t\}.$$

And the value of the flow is :

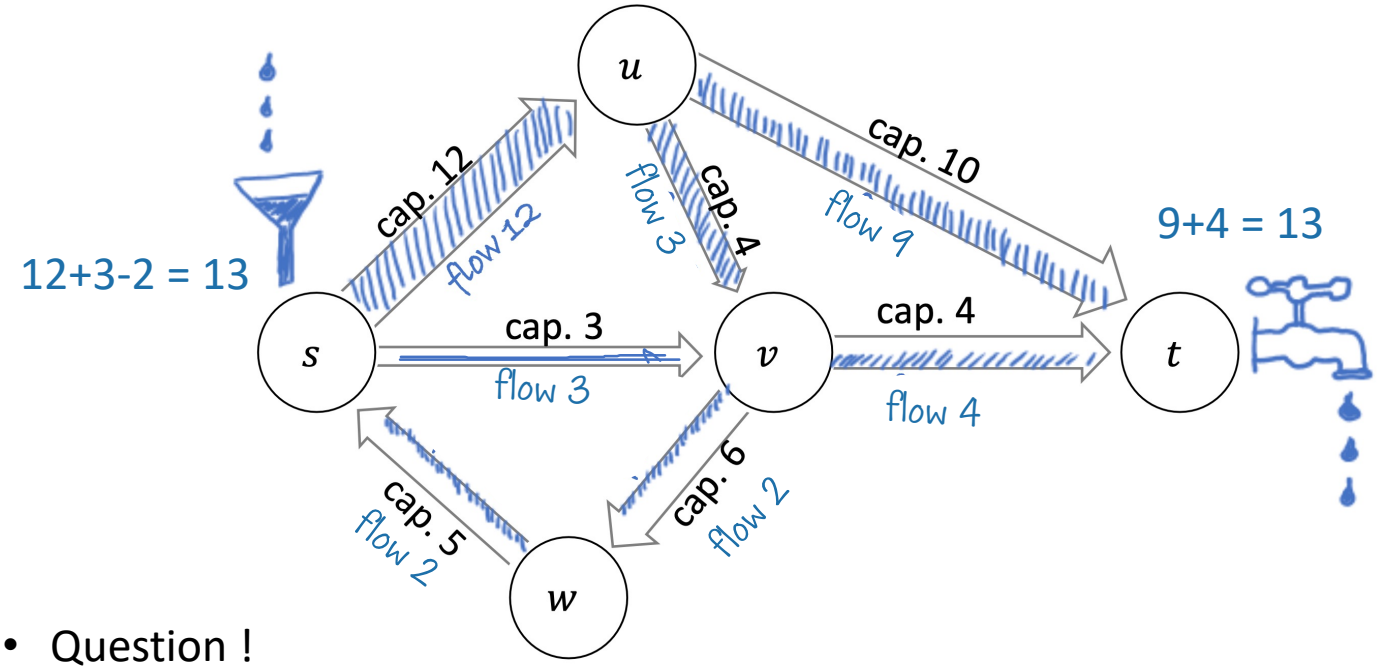
- $value(f)$ = $net\ flow\ out\ of\ s$=$net\ flow\ into\ t$

$$\text{value}(f) = \sum_{u:\, s \to u} f(s \to u) - \sum_{u:\, u \to s} f(u \to s).$$

Ford-Fulkerson algorithm

Problem statement

- Find a flow with maximum possible value (called a maximum flow)



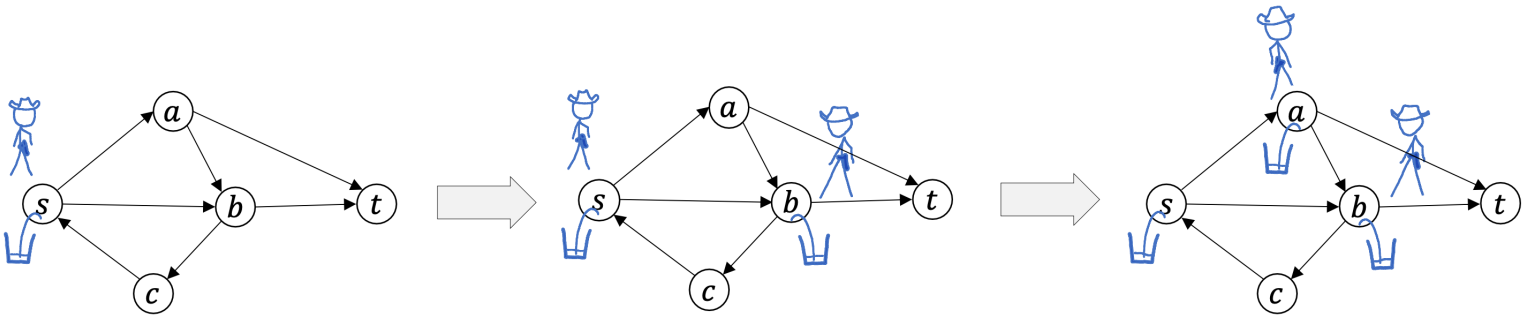
- Question !
  - Can you find a way to increase the value of the flow shown here?

Ford-Fulkerson algorithm

General idea

General idea : “look for vertices to which we could increase flow”

- Imagine that the source and all the other vertices apart from the sink are in bandit country, and the bandits want to siphon off flow from intermediate vertices.
- They can siphon off, redirect existing flows & increase flow at source



Turn up the flow at s, and siphon it off

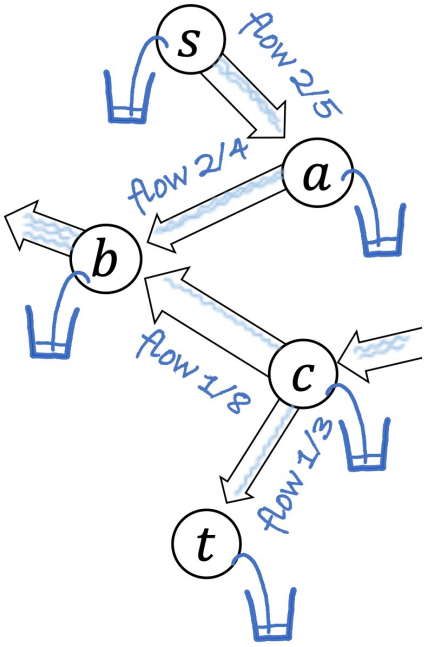
Instead of siphoning off all the excess at s, increase the flow s→b and siphon it off at b

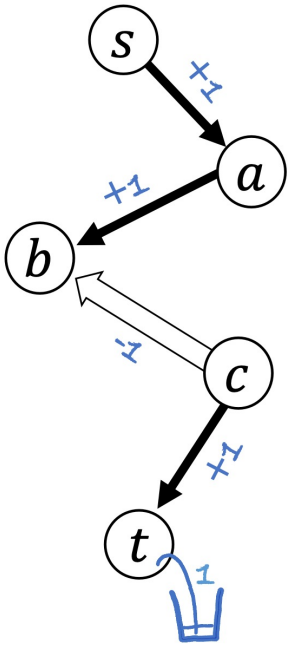
Instead of siphoning off all the excess at b, send some of it along b → t and reduce the a → b flow to match, giving an excess at a that can be siphoned off

Ford-Fulkerson algorithm

General idea

- How much could they siphon off at each of these locations?
  - They could siphon off 3 at a by increasing s → a
  - Or siphon off 2 at b by increasing s → a → b (limiting factor: spare capacity on a → b)
  - Or siphon off 1 at c by increasing s → a → b and decreasing c → b, leaving the other outflow at b undisturbed (limiting factor: existing flow on c → b)
  - Or siphon off 1 at t by increasing s → a → b and decreasing c → b and increasing c → t, leaving the inflow at c undisturbed (limiting factor: existing flow on c → b)

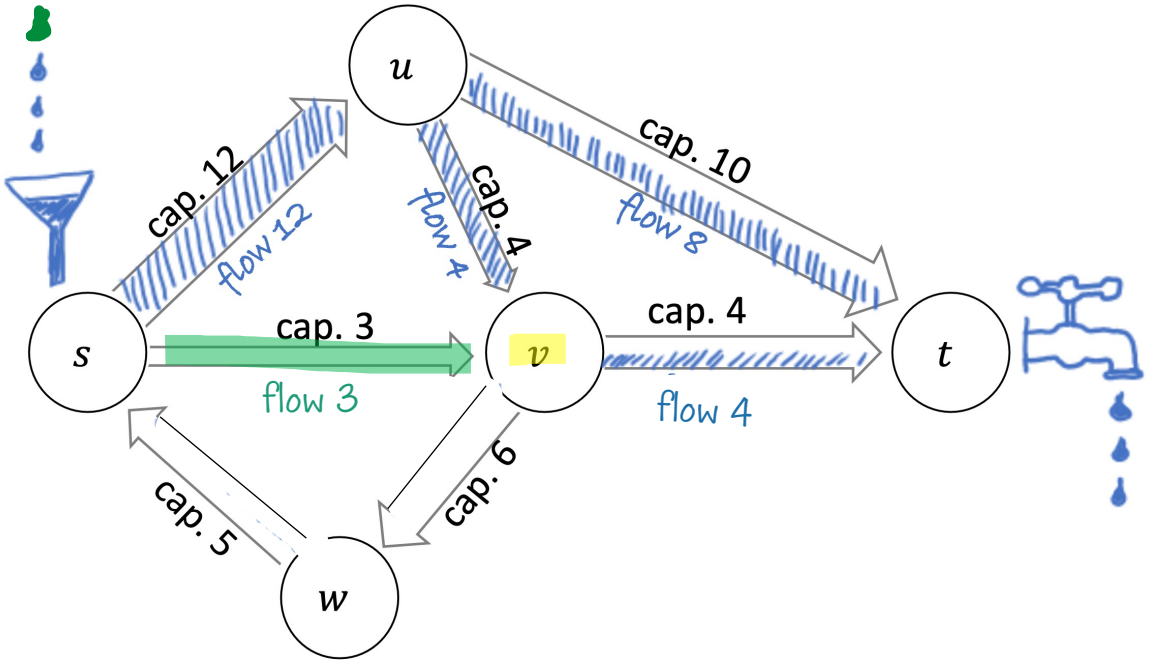




Ford-Fulkerson algorithm

General idea

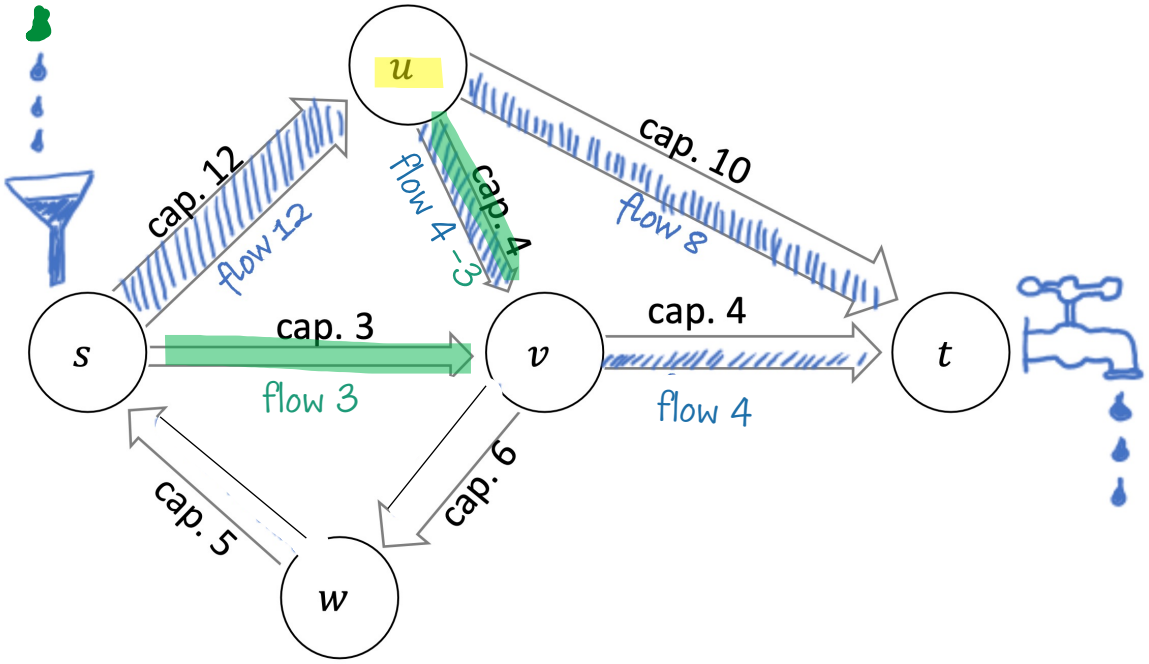
- The network operator only wants to get flow to sink. So it chooses a flow adjustment that gets as much as possible to t, with no excess at any of the other vertices along the path.



Ford-Fulkerson algorithm

General idea

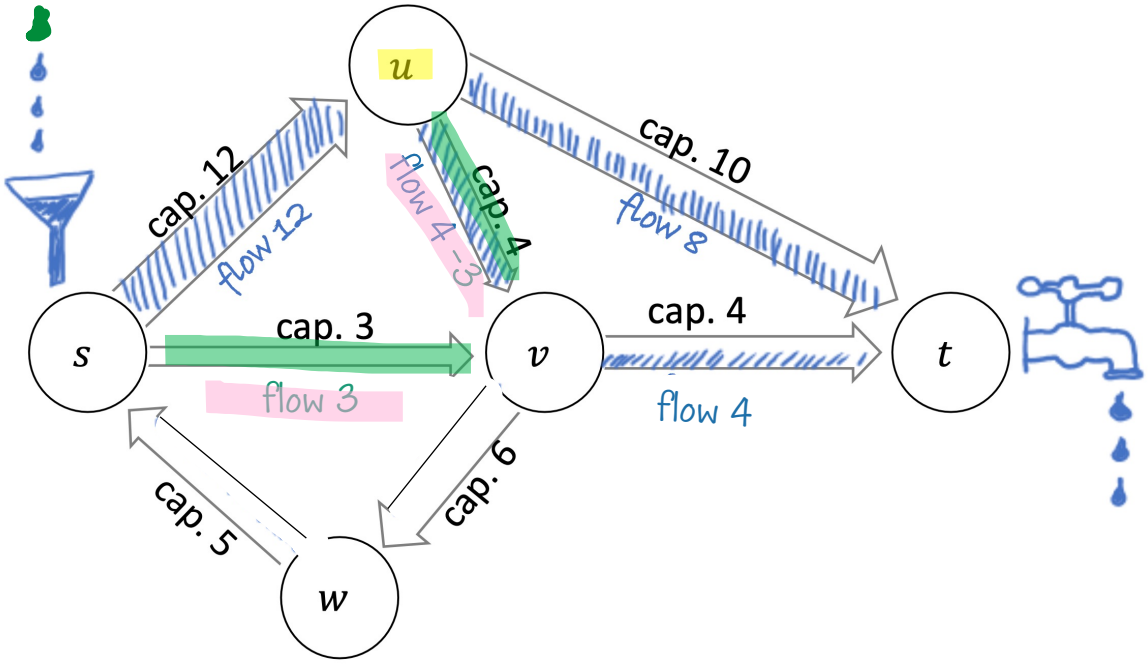
- The network operator only wants to get flow to sink. So it chooses a flow adjustment that gets as much as possible to t, with no excess at any of the other vertices along the path.



Ford-Fulkerson algorithm

General idea

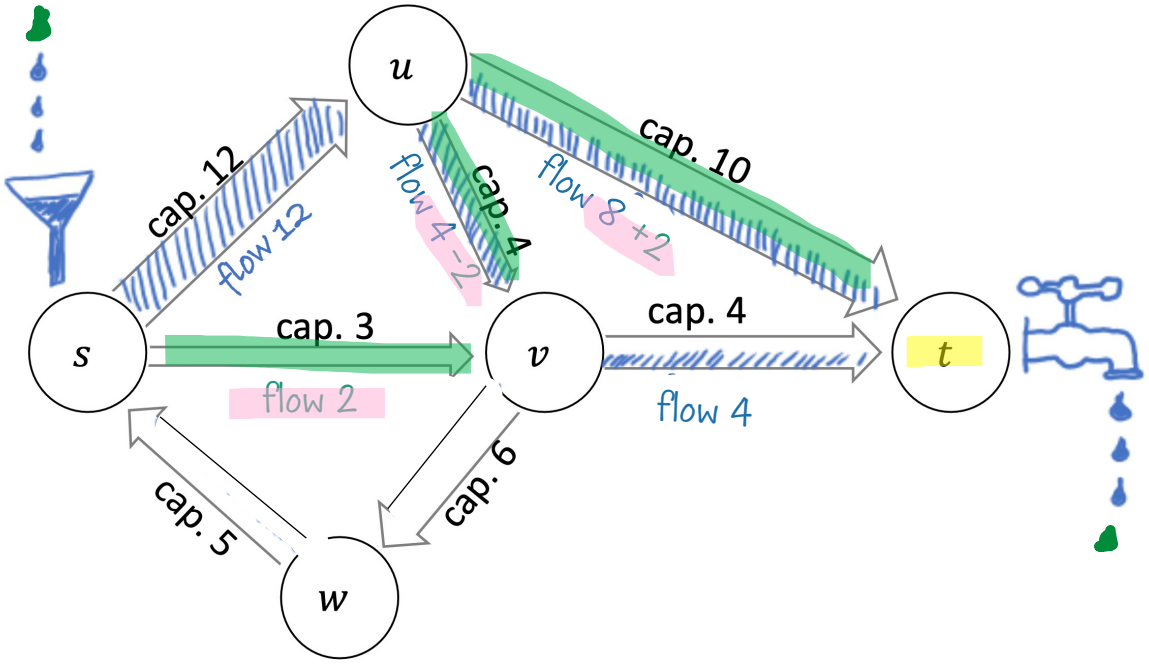
- The network operator only wants to get flow to sink. So it chooses a flow adjustment that gets as much as possible to t, with no excess at any of the other vertices along the path.



Ford-Fulkerson algorithm

General idea

- The network operator only wants to get flow to sink. So it chooses a flow adjustment that gets as much as possible to t, with no excess at any of the other vertices along the path.

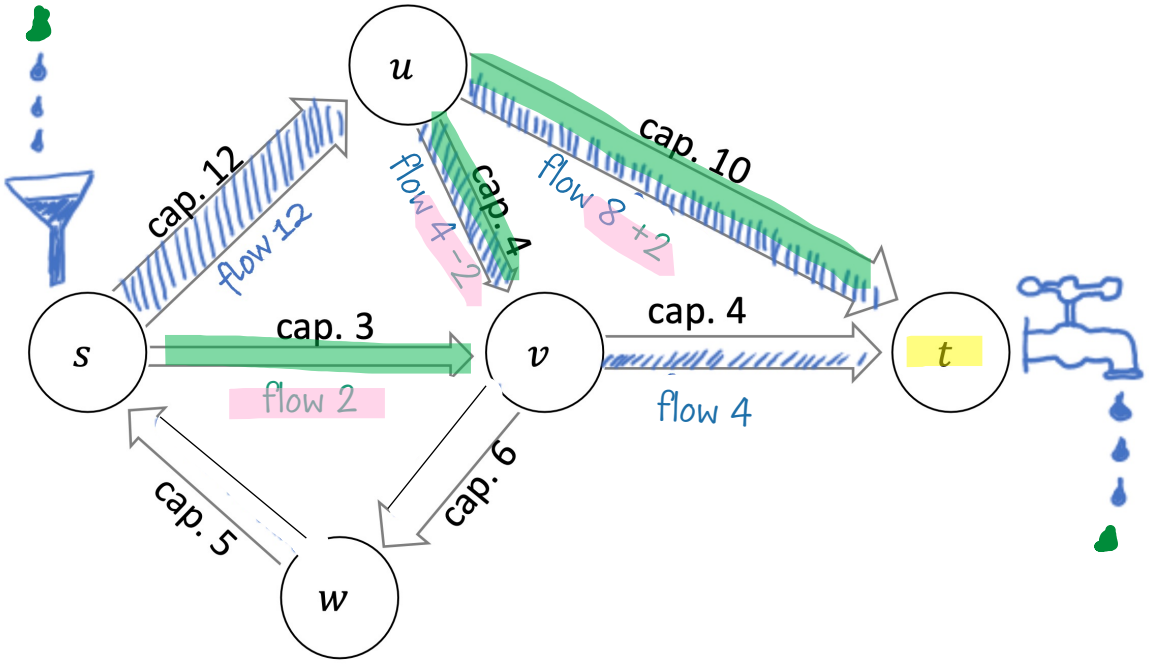


Ford-Fulkerson algorithm

General idea

Let's think about this change; how is the flow conservation here?

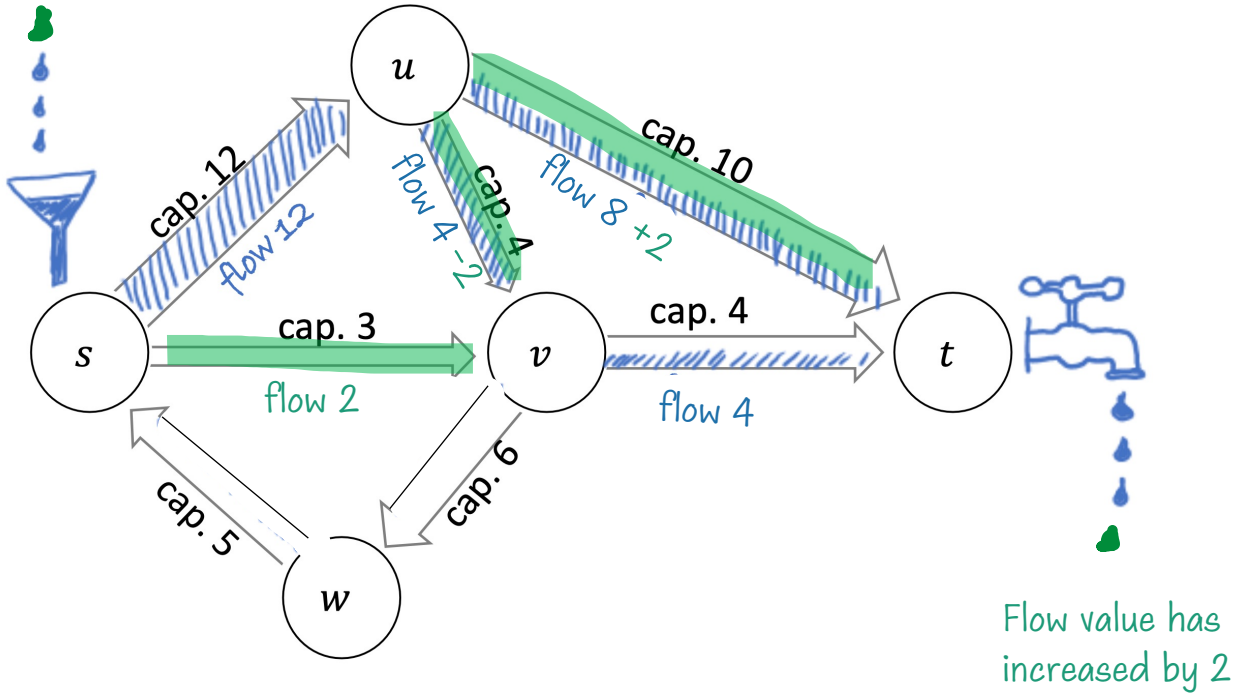
- Take vertex v : flow in : 4-2 +2 = 4 , flow out : 4 → total flow in == total flow out



Ford-Fulkerson algorithm

General idea

Let's think about this change; what is the value of the flow?



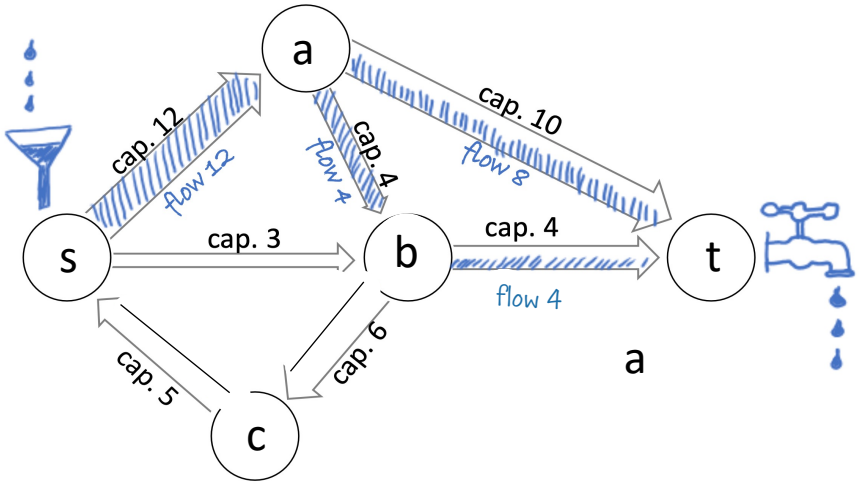
Ford-Fulkerson algorithm

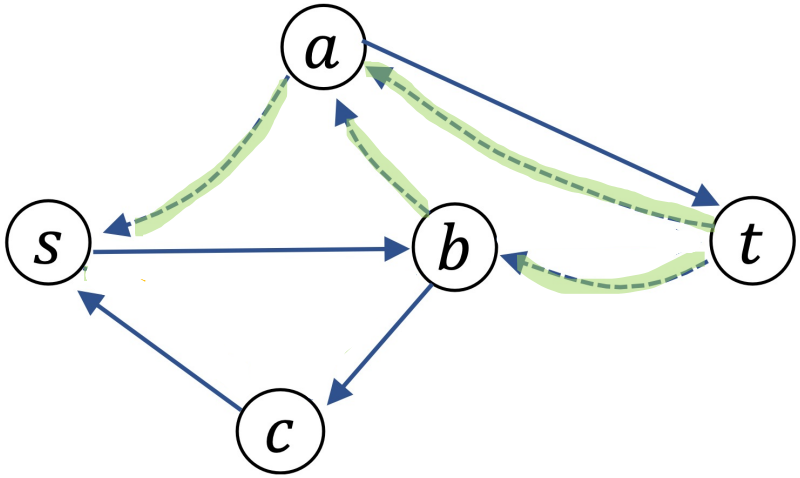
1. Start with zero flow
2. Run bandit search to discover if the flow to t can be increased and if so find an approprate sequence of edges
3. If t can be reached : update the flow along those edges, then go back to step 2.
4. If t can’t be reached : terminate

Ford-Fulkerson algorithm

Step 2a. Build the **residual graph**, which has the same vertices as the flow network, and

- If $f(u \rightarrow v) < c(u \rightarrow v)$ in the flow network, let the residual graph have an edge $u \rightarrow v$ with the label "increase flow $u \rightarrow v$".
- If $0 < f(u \rightarrow v)$ in the flow network, let the residual graph have an edge $v \rightarrow u$ (i.e. in the opposite direction) with the label "decrease flow $u \rightarrow v$".

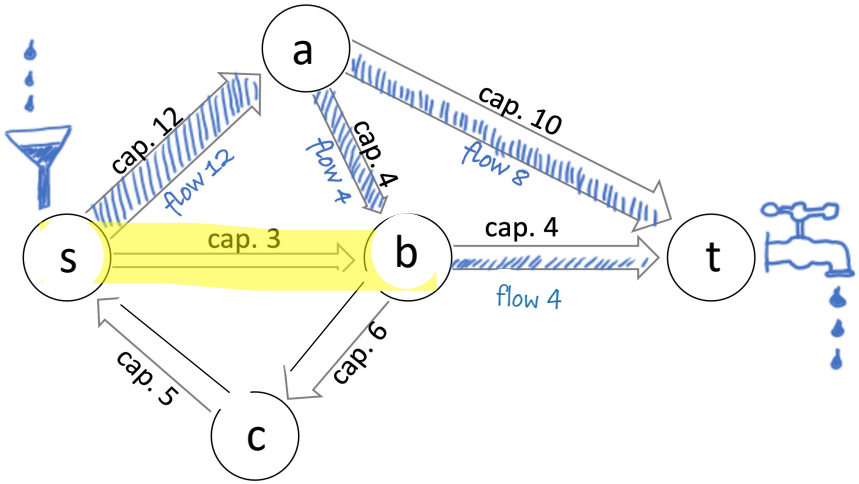


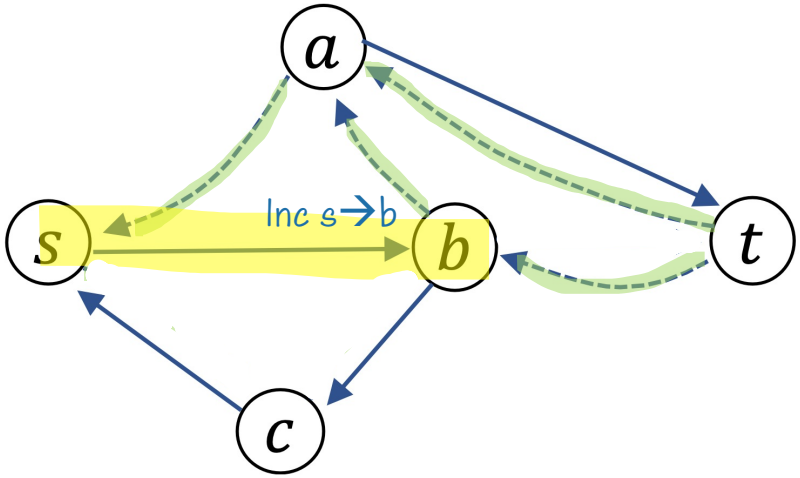


Ford-Fulkerson algorithm

Step 2a. Build the **residual graph**, which has the same vertices as the flow network, and

- If $f(u \rightarrow v) < c(u \rightarrow v)$ in the flow network, let the residual graph have an edge $u \rightarrow v$ with the label "increase flow $u \rightarrow v$".
- If $0 < f(u \rightarrow v)$ in the flow network, let the residual graph have an edge $v \rightarrow u$ (i.e. in the opposite direction) with the label "decrease flow $u \rightarrow v$".

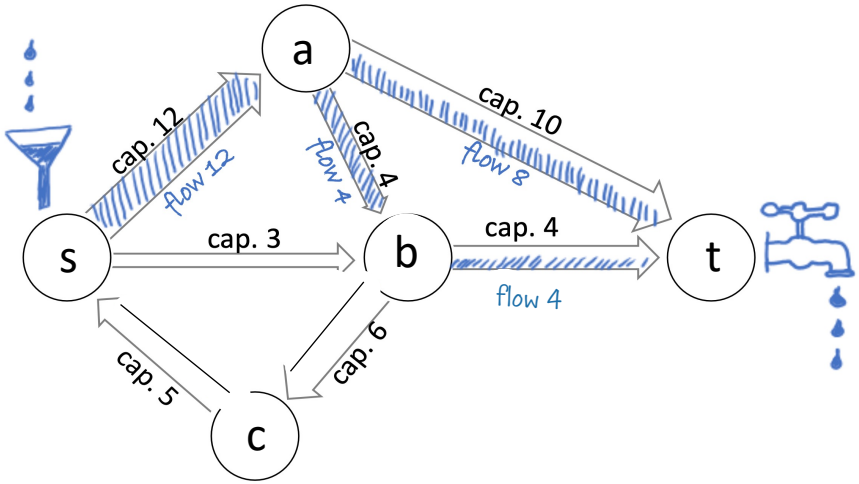


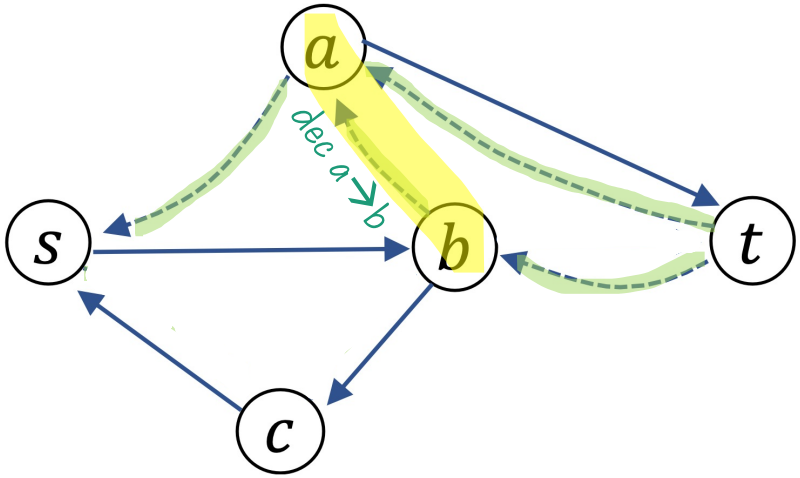


Ford-Fulkerson algorithm

Step 2a. Build the **residual graph**, which has the same vertices as the flow network, and

- If $f(u \rightarrow v) < c(u \rightarrow v)$ in the flow network, let the residual graph have an edge $u \rightarrow v$ with the label "increase flow $u \rightarrow v$".
- If $0 < f(u \rightarrow v)$ in the flow network, let the residual graph have an edge $v \rightarrow u$ (i.e. in the opposite direction) with the label "decrease flow $u \rightarrow v$".

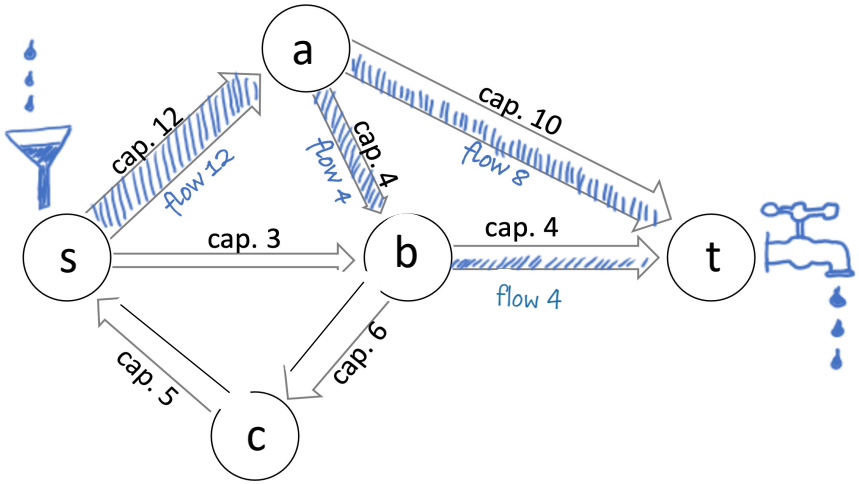


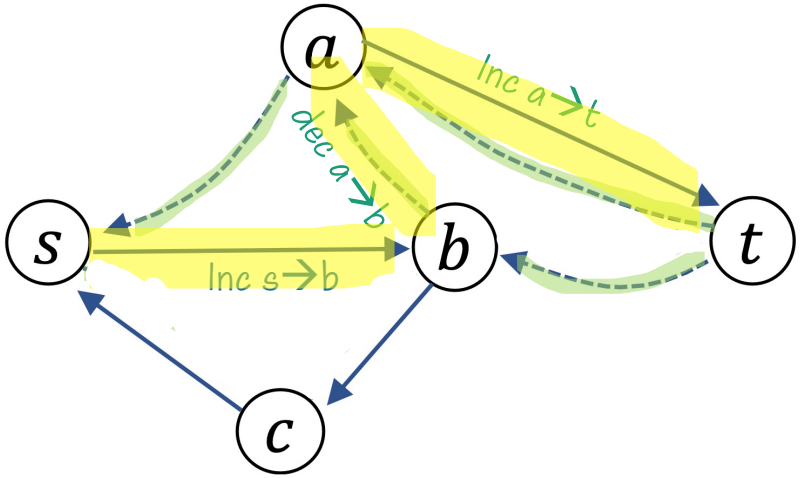


Ford-Fulkerson algorithm

Step 2a. Build the **residual graph**, which has the same vertices as the flow network, and

Step 2b. Look for a path from s to t in the residual graph. This is called an **augmenting path**



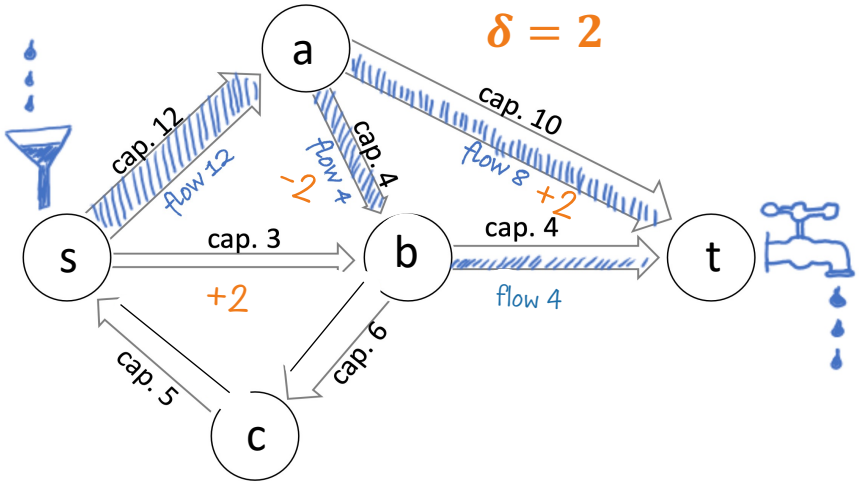


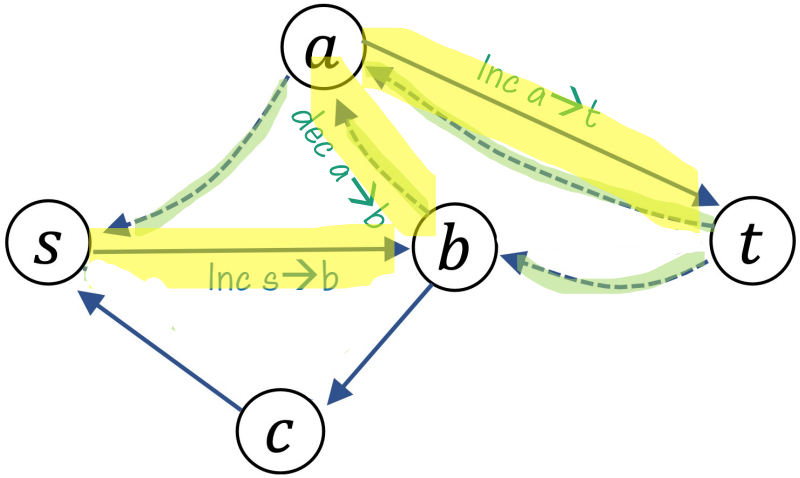
Ford-Fulkerson algorithm

Step 2a. Build the **residual graph**, which has the same vertices as the flow network, and

Step 2b. Look for a path from s to t in the residual graph. This is called an **augmenting path**

Step 3. find an update amount $\delta > 0$ that can be applied to all the edges along the augmenting path. Apply it.





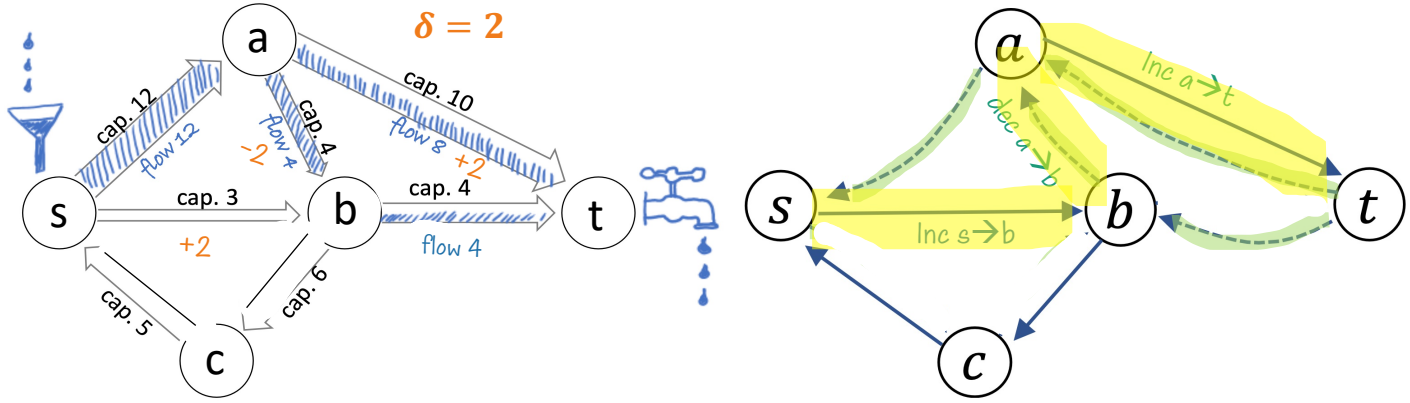
Ford-Fulkerson algorithm

Step 2a. Build the **residual graph**, which has the same vertices as the flow network, and

Step 2b. Look for a path from s to t in the residual graph. This is called an **augmenting path**

Step 3. find an update amount $\delta > 0$ that can be applied to all the edges along the augmenting path. Apply it.

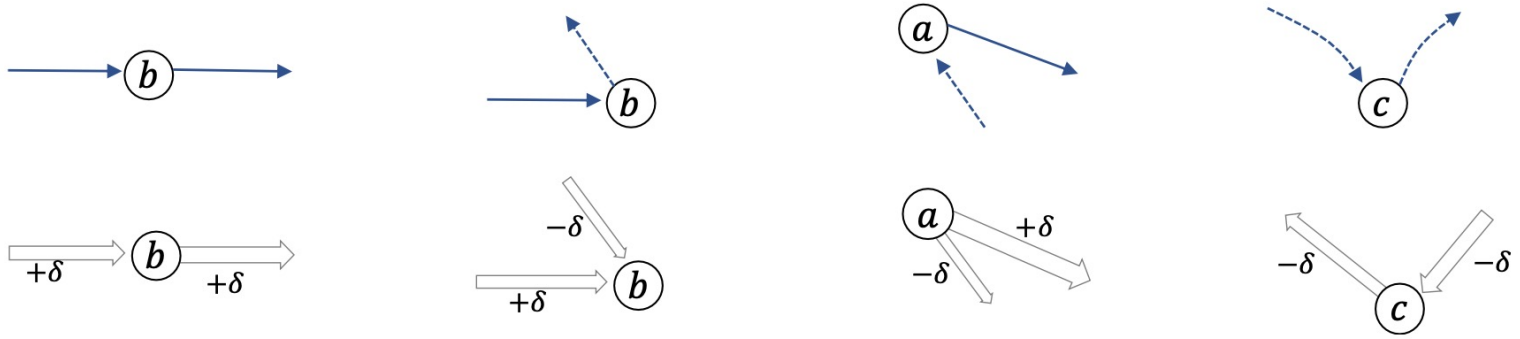
Lemma : After this update, we are left with a valid flow, and the flow value has increased by δ



Ford-Fulkerson algorithm

Lemma : After this update, we are left with a valid flow, and the flow value has increased by δ

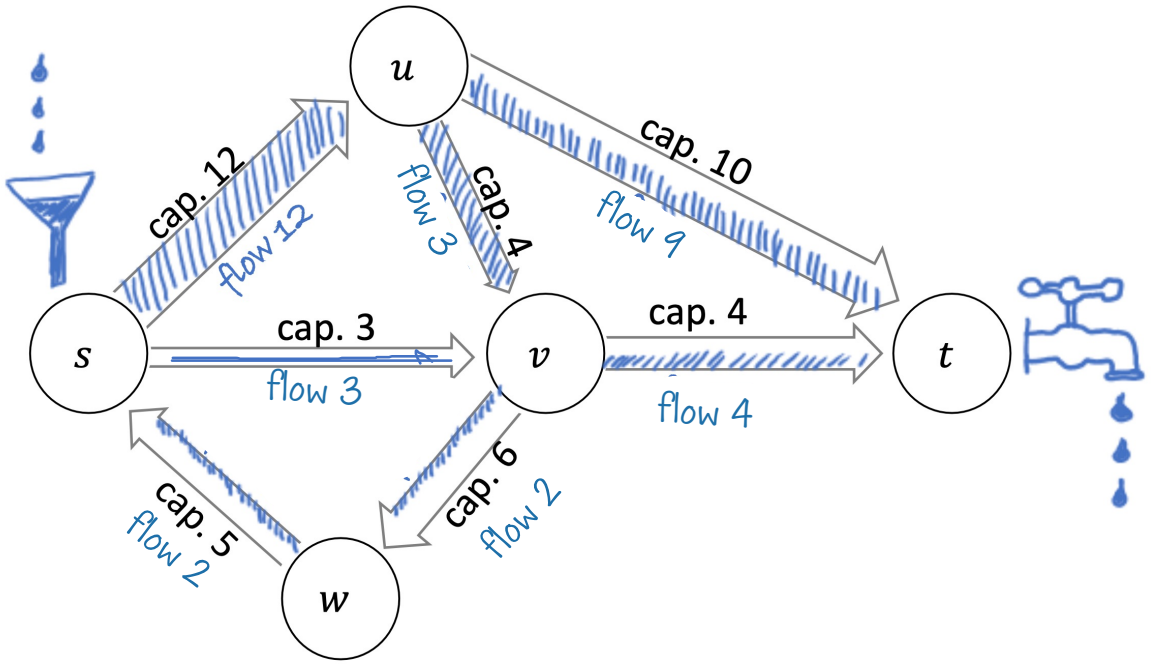
- Remember the two defining characteristics of a flow:
  - $0 < f(u \rightarrow v) < c(u \rightarrow v)$ on every edge
  - Total flow in = total flow out, at all vertices other than s and t.



Also, the flow value increases by δ :

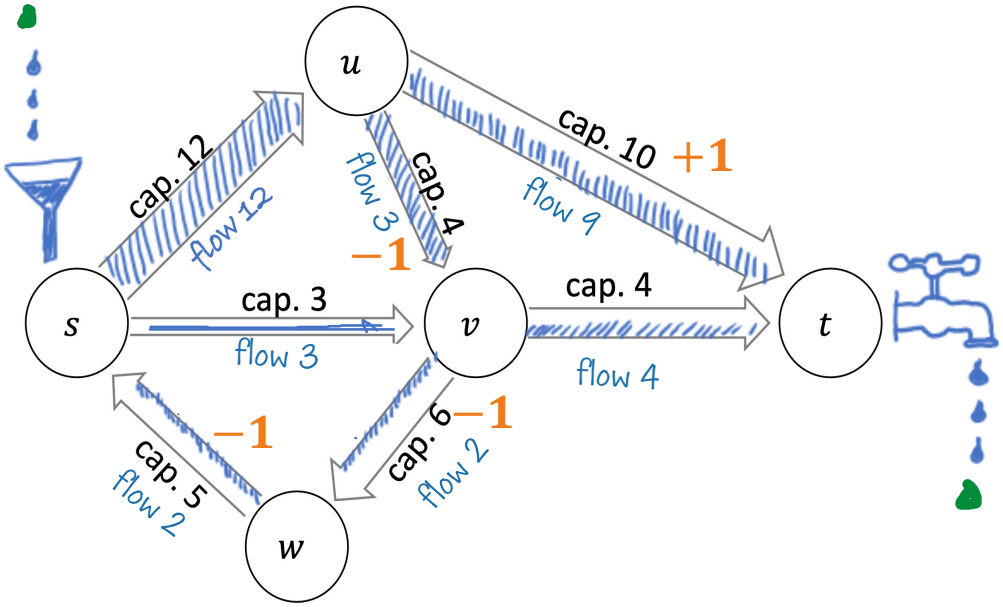
Ford-Fulkerson algorithm

- Back to our Question :
  - Can you find a way to increase the value of the flow shown here?

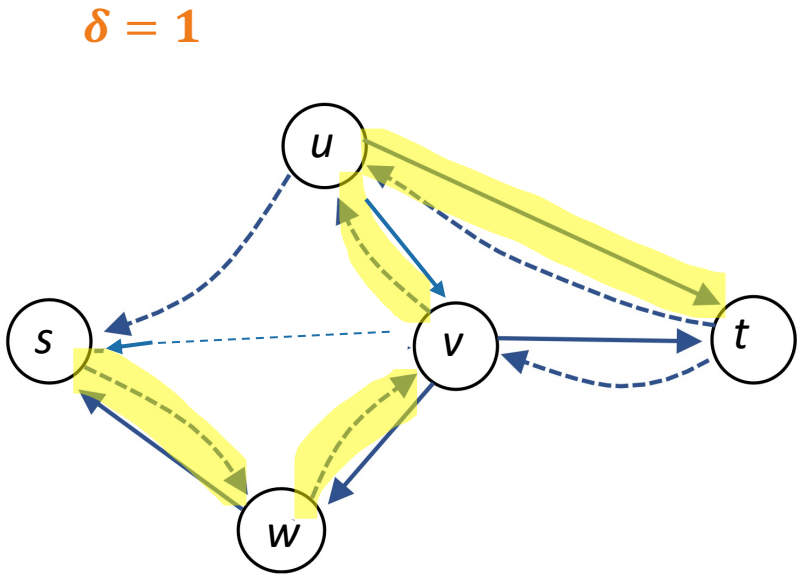


Ford-Fulkerson algorithm

- Back to our Question :
  - Can you find a way to increase the value of the flow shown here?



Flow value = 14



Ford-Fulkerson Algorithm

Pseudocode

```
def ford_fulkerson(g, s, t):
    # let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # define the residual graph h on the same vertices as g
        for each edge u → v in g:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc"
            if f(u → v) > 0: give h an edge v → u labelled "dec"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # There is a set of vertices that we can reach starting from s;
            # call this "the cut associated with flow f".
            # We'll use this in the analysis.
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break  # give up — can't find an augmenting path
        else:
            let the vertices of p be s = v_0, v_1, ..., v_k = t
            δ = ∞ # amount by which we'll augment the flow
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    δ = min(δ, c(v_i → v_{i+1}) − f(v_i → v_{i+1}))
                else the edge must have label "dec":
                    δ = min(δ, f(v_{i+1} → v_i))
            # assert: δ > 0
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    f(v_i → v_{i+1}) = f(v_i → v_{i+1}) + δ
                else the edge must have label "dec":
                    f(v_{i+1} → v_i) = f(v_{i+1} → v_i) − δ
            # assert: f is still a valid flow
```

- ‘pick the shortest path’ : Edmonds–Karp algorithm; it is a simple matter of running breadth first search on the residual graph.
- ‘pick the path that makes δ as large as possible’, also due to Edmonds and Karp.

```
def ford_fulkerson(g, s, t):
    # let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # define the residual graph h on the same vertices as g
        for each edge u → v in g:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc"
            if f(u → v) > 0: give h an edge v → u labelled "dec"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # There is a set of vertices that we can reach starting from s;
            # call this "the cut associated with flow f".
            # We'll use this in the analysis.
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break  # give up — can't find an augmenting path
        else:
            let the vertices of p be s = v_0, v_1, ..., v_k = t
            δ = ∞ # amount by which we'll augment the flow
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    δ = min(δ, c(v_i → v_{i+1}) − f(v_i → v_{i+1}))
                else the edge must have label "dec":
                    δ = min(δ, f(v_{i+1} → v_i))
            # assert: δ > 0
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    f(v_i → v_{i+1}) = f(v_i → v_{i+1}) + δ
                else the edge must have label "dec":
                    f(v_{i+1} → v_i) = f(v_{i+1} → v_i) − δ
            # assert: f is still a valid flow
```

Lemma: If all capacities are integers then the algorithm terminates, and the resulting flow on each edge is an integer.

Running time : $O(f^* \times E)$.

f^* = value of maximum flow

```
def ford_fulkerson(g, s, t):
    # let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # define the residual graph h on the same vertices as g
        for each edge u → v in g:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc"
            if f(u → v) > 0: give h an edge v → u labelled "dec"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # There is a set of vertices that we can reach starting from s;
            # call this "the cut associated with flow f".
            # We'll use this in the analysis.
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break  # give up — can't find an augmenting path
        else:
            let the vertices of p be s = v_0, v_1, ..., v_k = t
            δ = ∞ # amount by which we'll augment the flow
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    δ = min(δ, c(v_i → v_{i+1}) − f(v_i → v_{i+1}))
                else the edge must have label "dec":
                    δ = min(δ, f(v_{i+1} → v_i))
            # assert: δ > 0
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    f(v_i → v_{i+1}) = f(v_i → v_{i+1}) + δ
                else the edge must have label "dec":
                    f(v_{i+1} → v_i) = f(v_{i+1} → v_i) − δ
            # assert: f is still a valid flow
```

f starts with integer

Lemma: If all capacities are integers then the algorithm terminates, and the resulting flow on each edge is an integer.

Running time : $O(f^* \times E)$.
f^* = value of maximum flow

δ will be an integer

New f, f^* will be an integer

```
def ford_fulkerson(g, s, t):
    # let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # define the residual graph h on the same vertices as g
        for each edge u → v in g:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc"
            if f(u → v) > 0: give h an edge v → u labelled "dec"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # There is a set of vertices that we can reach starting from s;
            # call this "the cut associated with flow f".
            # We'll use this in the analysis.
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break  # give up — can't find an augmenting path
        else:
            let the vertices of p be s = v_0, v_1, ..., v_k = t
            δ = ∞ # amount by which we'll augment the flow
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    δ = min(δ, c(v_i → v_{i+1}) − f(v_i → v_{i+1}))
                else the edge must have label "dec":
                    δ = min(δ, f(v_{i+1} → v_i))
            # assert: δ > 0
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    f(v_i → v_{i+1}) = f(v_i → v_{i+1}) + δ
                else the edge must have label "dec":
                    f(v_{i+1} → v_i) = f(v_{i+1} → v_i) − δ
            # assert: f is still a valid flow
```

f starts with integer

New f, f^* will be an integer

δ will be an integer

Lemma: If all capacities are integers then the algorithm terminates, and the resulting flow on each edge is an integer.

Running time : $O(f^* \times E)$.
f^* = value of maximum flow

Flow value increases by at least 1

```
def ford_fulkerson(g, s, t):
    # let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # define the residual graph h on the same vertices as g
        for each edge u → v in g:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc"
            if f(u → v) > 0: give h an edge v → u labelled "dec"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # There is a set of vertices that we can reach starting from s;
            # call this "the cut associated with flow f".
            # We'll use this in the analysis.
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break  # give up — can't find an augmenting path
        else:
            let the vertices of p be s = v_0, v_1, ..., v_k = t
            δ = ∞ # amount by which we'll augment the flow
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    δ = min(δ, c(v_i → v_{i+1}) − f(v_i → v_{i+1}))
                else the edge must have label "dec":
                    δ = min(δ, f(v_{i+1} → v_i))
            # assert: δ > 0
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    f(v_i → v_{i+1}) = f(v_i → v_{i+1}) + δ
                else the edge must have label "dec":
                    f(v_{i+1} → v_i) = f(v_{i+1} → v_i) − δ
            # assert: f is still a valid flow
```

f starts with integer

New f, f^* will be an integer

δ will be an integer

Lemma: If all capacities are integers then the algorithm terminates, and the resulting flow on each edge is an integer.

Running time : $O(f^* \times E)$.
f^* = value of maximum flow

Flow value increases by at least 1

iterations ≤ maximum flow value

```
def ford_fulkerson(g, s, t):
    # let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # define the residual graph h on the same vertices as g
        for each edge u → v in g:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc"
            if f(u → v) > 0: give h an edge v → u labelled "dec"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # There is a set of vertices that we can reach starting from s;
            # call this "the cut associated with flow f".
            # We'll use this in the analysis.
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break  # give up — can't find an augmenting path
        else:
            let the vertices of p be s = v_0, v_1, ..., v_k = t
            δ = ∞ # amount by which we'll augment the flow
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    δ = min(δ, c(v_i → v_{i+1}) − f(v_i → v_{i+1}))
                else the edge must have label "dec":
                    δ = min(δ, f(v_{i+1} → v_i))
            # assert: δ > 0
            for each edge v_i → v_{i+1} along p:
                if the edge has label "inc":
                    f(v_i → v_{i+1}) = f(v_i → v_{i+1}) + δ
                else the edge must have label "dec":
                    f(v_{i+1} → v_i) = f(v_{i+1} → v_i) − δ
            # assert: f is still a valid flow
```

f starts with integer

Can be done using BFS in $O(E)$.

Lemma: If all capacities are integers then the algorithm terminates, and the resulting flow on each edge is an integer.

Running time : $O(f^* \times E)$.
f^* = value of maximum flow

δ will be an integer

New f, f^* will be an integer

Flow value increases by at least 1

iterations ≤ maximum flow value