INT202
Complexity of Algorithms

Minimum Spanning Tree and Network Flow

Xi'an Jiaotong-Liverpool University
2020-2021

Minimum Spanning Tree (MST)

Let G be an undirected graph.

We are also often interested in finding a spanning tree T, i.e. an acyclic spanning subgraph of G, which *minimizes* the sum of the weights of the edges of T.

Computing a spanning tree T with smallest total weight is the problem of constructing a *Minimum Spanning Tree* or MST for short.

Prim's Algorithm

Prim's algorithm is a greedy type of algorithm designed to find MST.

The idea is that we build the MST starting at a root vertex.

At each step, we "grow" the tree by adding an additional edge to a vertex not in the current tree, selecting one that minimizes the "attachment cost" while ensuring that we do not create a cycle with the edges already selected.

Prim's Algorithm

The pseudocode for prim's algorithm shows how we create two sets of vertices U and V-U. U contains the list of vertices that have been visited and V-U the list of vertices that haven't. One by one, we move vertices from set V-U to set U by connecting the least weight edge.

```
T = ∅;
U = { 1 };
while (U ≠ V)
  let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
  T = T ∪ {(u, v)};
  U = U ∪ {v}
```
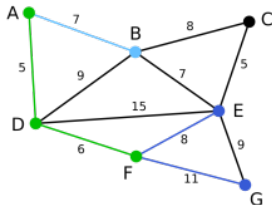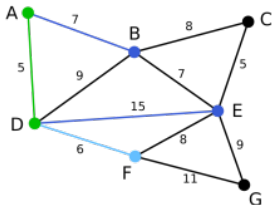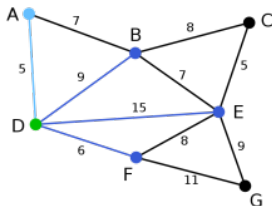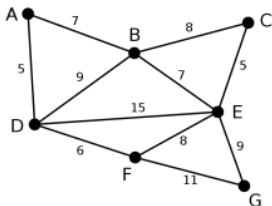
Prim's Algorithm

```
T = ∅;
U = { r };
while (U ≠ V)
   let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
   T = T ∪ {(u, v)};
   U = U ∪ {v}
```
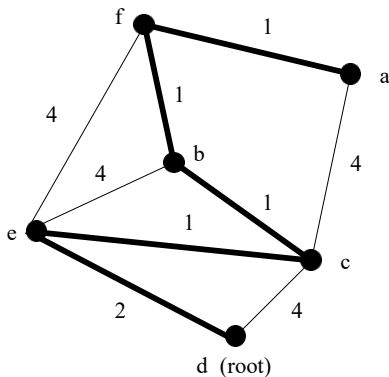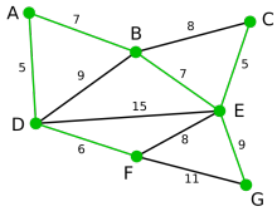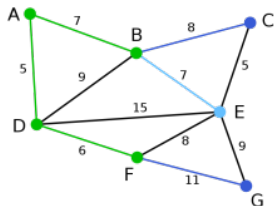
Prim's Algorithm
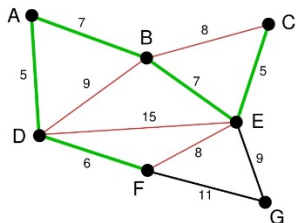
Prim's Algorithm
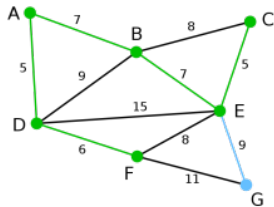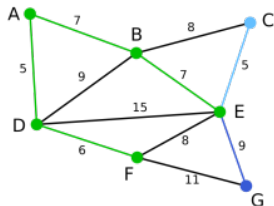
Kruskal's Algorithm for MST

Kruskal's algorithm, like Prim's, is a greedy algorithm.

Unlike Prim's which "grows" a MST starting from some vertex, Kruskal's algorithm works by considering the edges in order from smallest to largest. At each step, we greedily pick the smallest edge available that does not create a cycle together with those chosen so far.

The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high;

2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge;

3. Keep adding edges until we reach all vertices.

Kruskal's Algorithm for MST
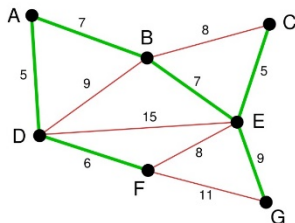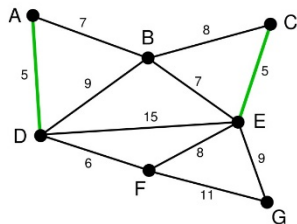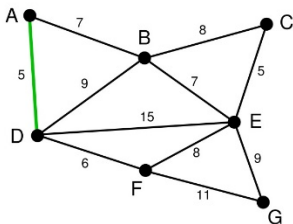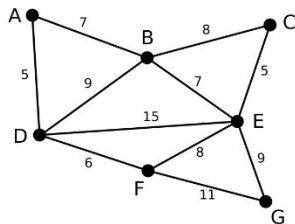
Borůvka's algorithm

Another greedy algorithm to find the minimum spanning tree for a graph:

1) Input is a connected, weighted and un-directed graph.

2) Initialize all vertices as individual components.

3) Initialize MST as empty.

4) While there are more than one components, do following

  for each component.

    a) Find the closest weight edge that connects this

      component to any other component.

    b) Add this closest edge to MST if not already added.

5) Return MST.

Borůvka's algorithm
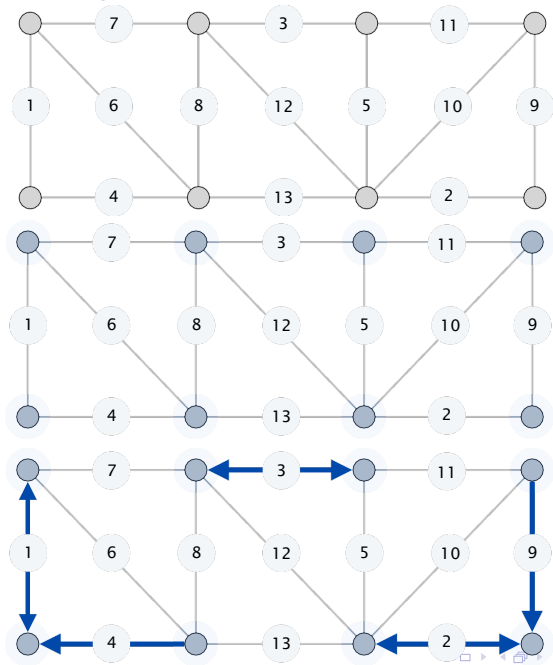
Borůvka's algorithm
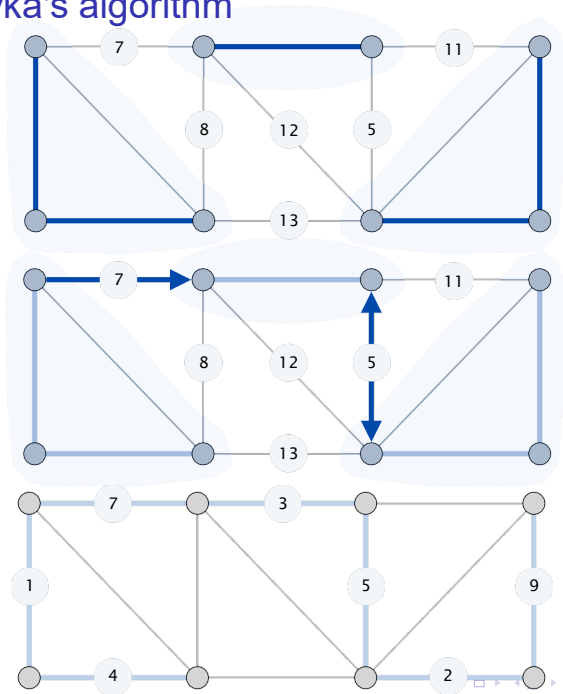
Flow Network

▷ A flow network (or just network) *N* consists of

- ▷ A weighted digraph *G* with nonnegative integer edge weights, where the weight of an edge *e* is called the capacity *c(e)* of *e*.
- ▷ Two distinguished vertices, *s* and *t* of *G*, called the *source* and *sink*, respectively, such that *s* has no incoming edges and *t* has no outgoing edges.

▷ An example
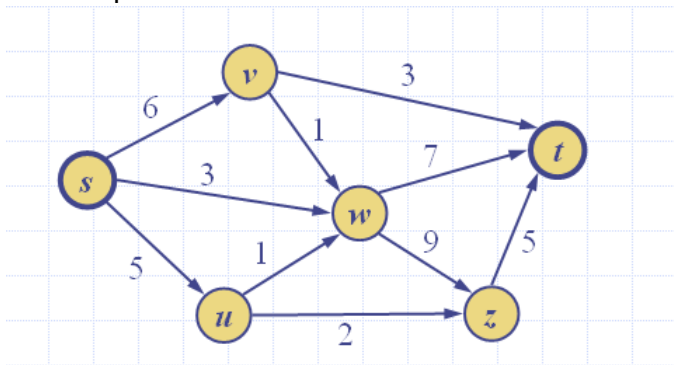
Flow

▷ A flow *f* for a network *N* is is an assignment of an integer value *f* (*e*) to each edge *e* that satisfies the following properties:

1. Capacity Rule: For each edge *e*, $0 \leq f(e) \leq c(e)$

2. Conservation Rule: For each vertex $v \neq s, t$

$$\sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e)$$

where $E^-(v)$ and $E^+(v)$ are the incoming and outgoing edges of *v*, respectively.
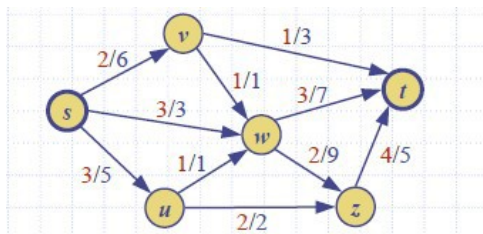
Flow

▷ The value of a flow *f*, denoted |*f*|, is the total flow from the source, which is the same as the total flow into the sink.

▷ Problem: Find a flow of maximum value?
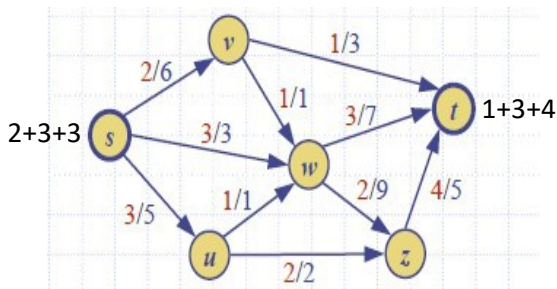
Maximum Flow

▷ A flow for a network *N* is said to be maximum if its value is the largest of all flows for *N*.

▷ The maximum flow problem consists of finding a maximum flow for a given network *N*

Flow of value 8 = 2 + 3 + 3 = 1 + 3 + 4
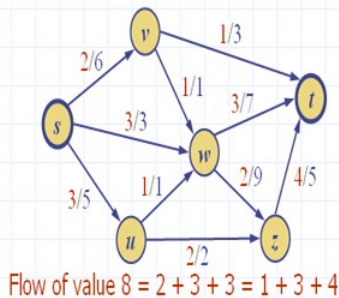
Cut - Definitions

Flows are closely related to another concept, known as *cuts*.

▷ A *cut* of a network N with source s and sink t is a partition $\chi = (V_s, V_t)$ of the vertices of N such that $s \in V_s$ and $t \in V_t$.

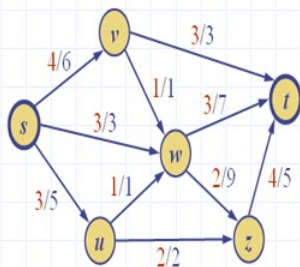 ▷ Forward edge of cut χ: origin in V_s and destination in V_t

  ▷ Backward edge of cut χ: origin in V_t and destination in V_s

▷ Flow $f(\chi)$ across a cut χ: total flow of forward edges minus total flow of backward edges

▷ Capacity $c(\chi)$ of a cut χ: total capacity of forward edges

Cut - Example
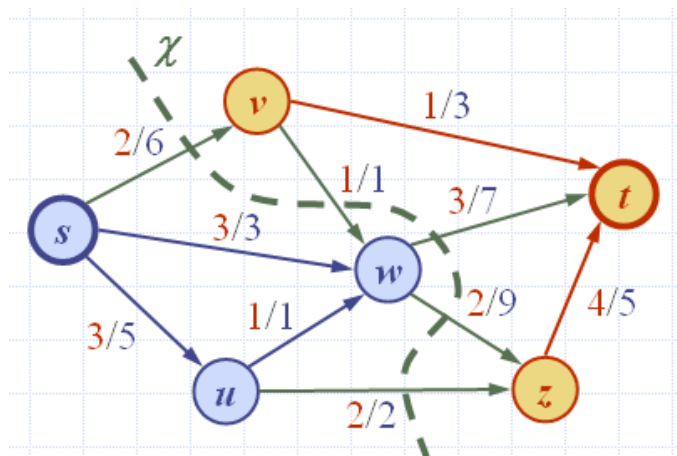
The *total flow/capacity* of the cut are
  ▹ $f(\chi) = 8$
  ▹ $c(\chi) = 24$

Flow and Cut

The *total flow/capacity* of the cut are
  ▷ $c(\chi_1) = 12 = 6 + 3 + 1 + 2$
  ▷ $c(\chi_2) = 21 = 3 + 7 + 9 + 2$
  ▷ For both cuts, the value of the flow is $|f| = 8$

Flow and Cut

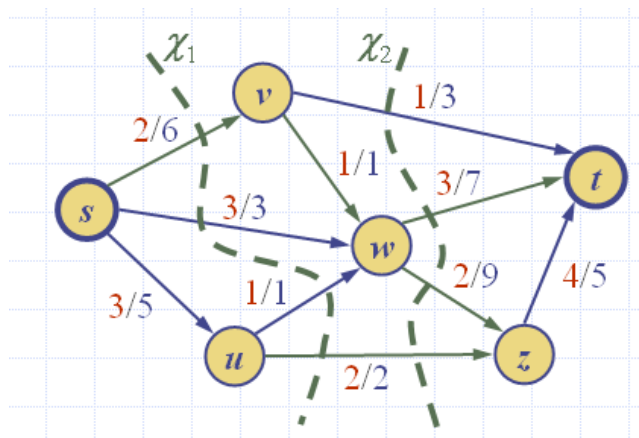**Lemma 1:** Let N be a flow network, and let f be a flow for N. For any cut χ of N, the value of f is equal to the flow across cut χ, that is, $|f| = f(\chi)$.

Lemma 2: Let N be a flow network, and let χ be a cut of N. Given any flow f for N, the flow across cut does not exceed the capacity of χ, that is, $f(\chi) \leq c(\chi)$.

Theorem 1: The value of any flow is less than or equal to the capacity of any cut, i.e., for any flow f and any cut χ, we have $|f| \leq c(\chi)$.

$\Rightarrow$ The value of a maximum flow is no more than the capacity of a minimum cut.

Augmenting Path - Definitions

To construct a maximum flow we need the concept of *augmenting path*.

▷ Consider a flow *f* for a network *N*

▷ Let *e* be an edge from *u* to *v*:
  ▷ *Residual capacity* of *e* from *u* to *v*: $\Delta_f(u, v) = c(e) - f(e)$
  ▷ *Residual capacity* of *e* from *v* to *u*: $\Delta_f(v, u) = f(e)$

▷ Let *π* be a path from *s* to *t*, the *residual capacity* $\Delta_f(π)$ of *π* is the smallest of the residual capacities of the edges of *π* in the direction from *s* to *t*.

▷ A path *π* from *s* to *t* is an augmenting path if $\Delta_f(π) > 0$
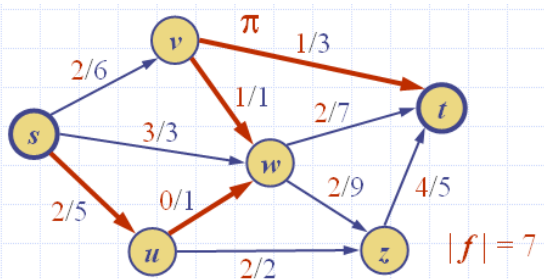
Augmenting Path - Example
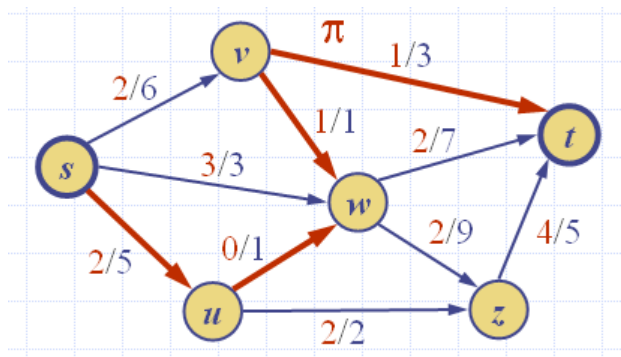
We get the following residual capacities and flow

▷ $\Delta_f(s,u) = 3$, $\Delta_f(u,w) = 1$, $\Delta_f(w,v) = 1$, $\Delta_f(v,t) = 2$.
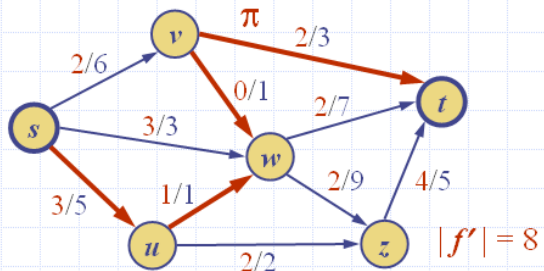
▷ $\Delta_f(\pi) = 1$

▷ $|f| = 7$

Flow Augmentation

We can always add the residual capacity of an augmenting path to an existing flow and get another valid flow.

Lemma 3: Let π be an augmenting path for flow f in network N. There exists a flow f' for N of value $|f'| = |f| + \Delta_f(\pi)$

Flow Augmentation

What if there is no augmenting path for a flow f in network N?

Lemma 4: If a network N does not have an augmenting path with respect to a flow f, then f is a maximum flow. Also, there is a cut χ of N such that $|f| = c(\chi)$

Lemma 4 and Theorem 1 gives us the Max-flow, Min-Cut Theorem also known as the *Ford-Fulkerson Theorem*

Theorem 2: The value of a maximum flow is equal to the capacity of a mimimum cut.

Max-Flow and Min-Cut

▷ Define
  ▷ V_s set of vertices reachable from s by augmenting paths
  ▷ V_t set of remaining vertices
▷ Cut $\chi = (V_s, V_t)$ has capacity $c(\chi) = |f|$
  ▷ Forward edge: $f(e) = c(e)$
  ▷ Backward edge: $f(e) = 0$
▷ Thus, flow f has maximum value and cut χ has minimum capacity. Below we have $c(\chi) = |f| = 10$.
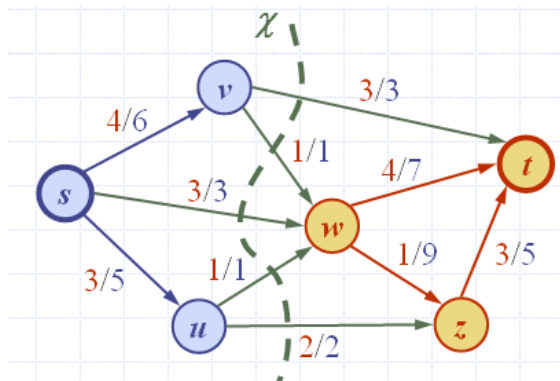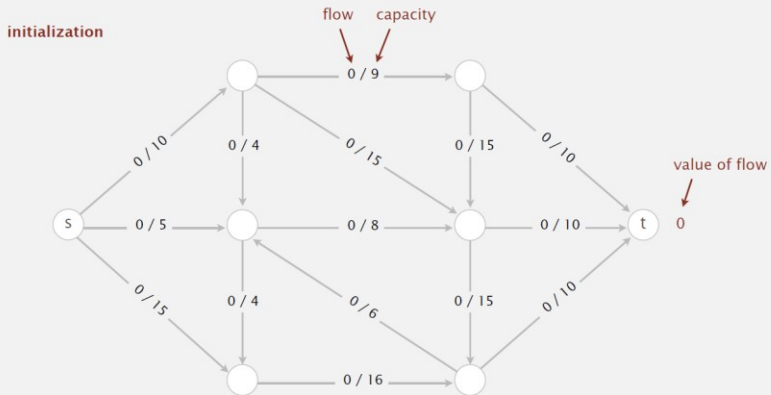
The Ford-Fulkerson Algorithm

▷ Initially, $f(e) = 0$ for each edge e

▷ Repeatedly
  ▷ Search for an augmenting path π
  ▷ Compute bottleneck capacity $\Delta_f(\pi)$
  ▷ Increase flow along the edges of π by bottleneck capacity $\Delta_f(\pi)$

Ford-Fulkerson algorithm

Initialization. Start with 0 flow.

Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

1st augmenting path

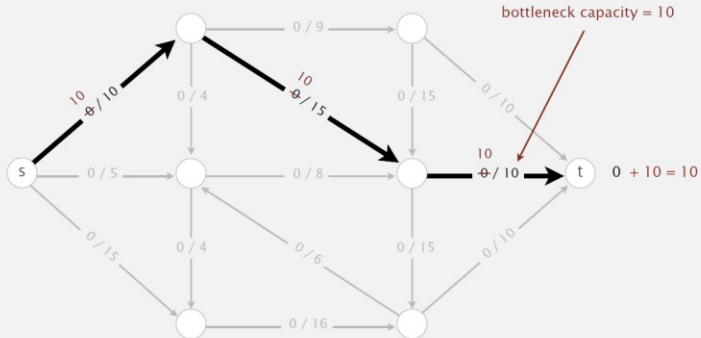bottleneck capacity = 10

$0 + 10 = 10$

Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

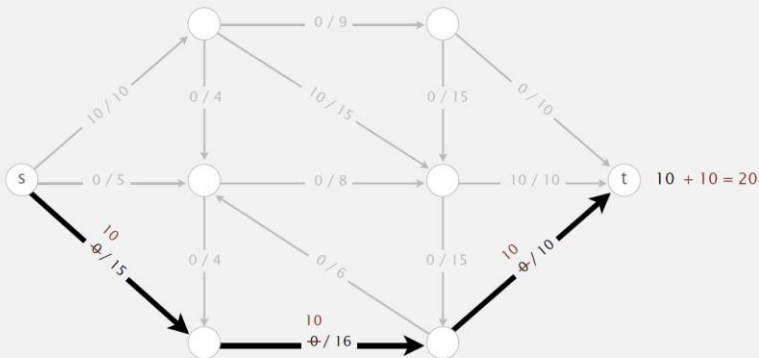**2nd augmenting path**

$10 + 10 = 20$

Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

3rd augmenting path

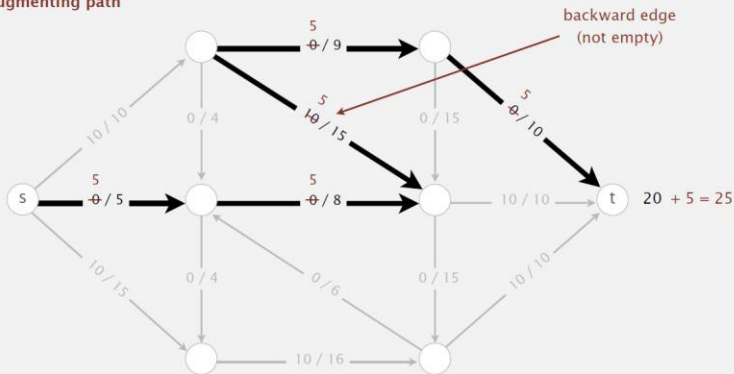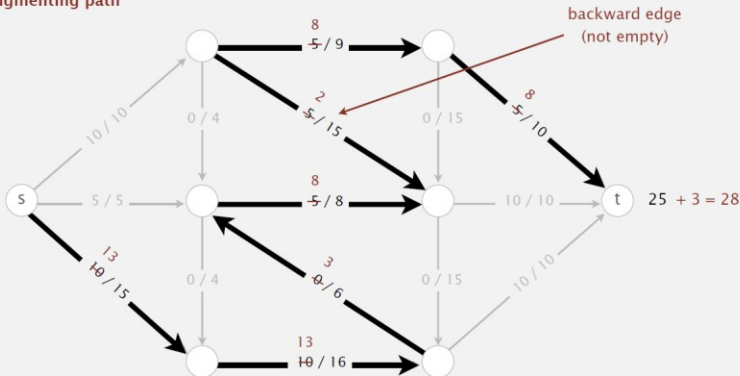backward edge
(not empty)

$20 + 5 = 25$

Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:
- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

4th augmenting path

backward edge
(not empty)

8
5 / 9

2
5 / 15

8
5 / 10

10 / 10

0 / 4

0 / 15

s

5 / 5

8
5 / 8

10 / 10

t   25 + 3 = 28

13
10 / 15

0 / 4

3
0 / 6

0 / 15

10 / 10

13
10 / 16

Idea: increase flow along augmenting paths

Termination. All paths from s to t are blocked by either a
- Full forward edge.
- Empty backward edge.

no more augmenting paths

full forward edge

empty backward edge