中山大學
SUN YAT-SEN UNIVERSITY

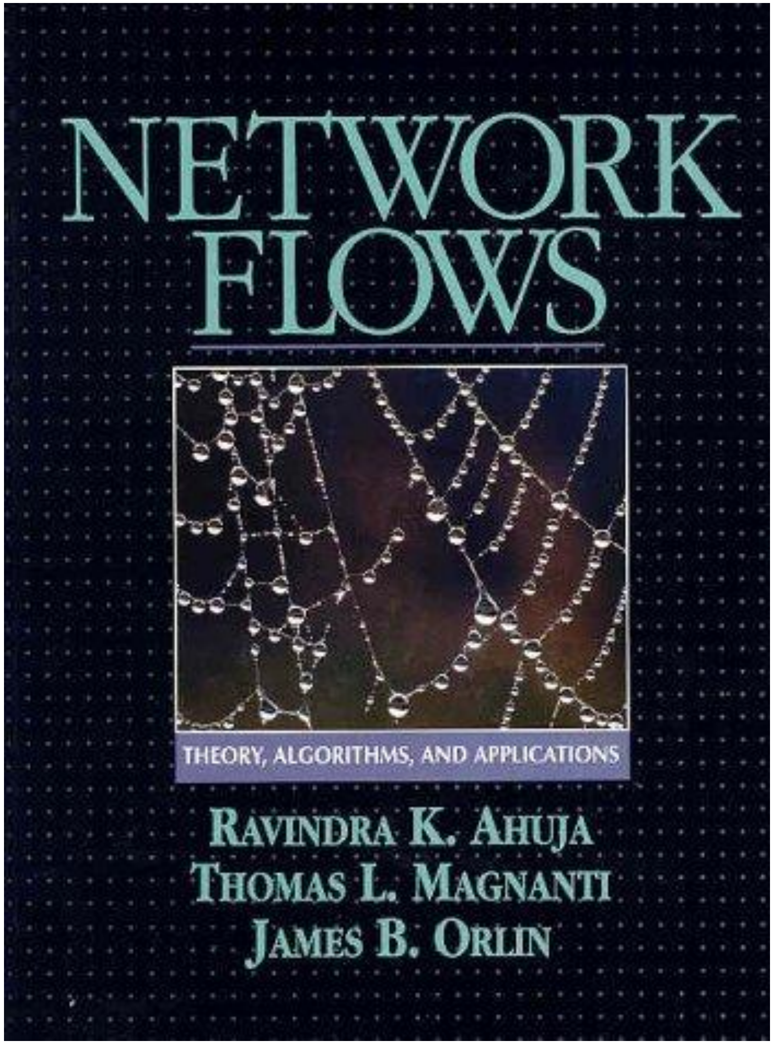
Lecture 15
Network Flows

Algorithm Design

zhangzizhen@gmail.com

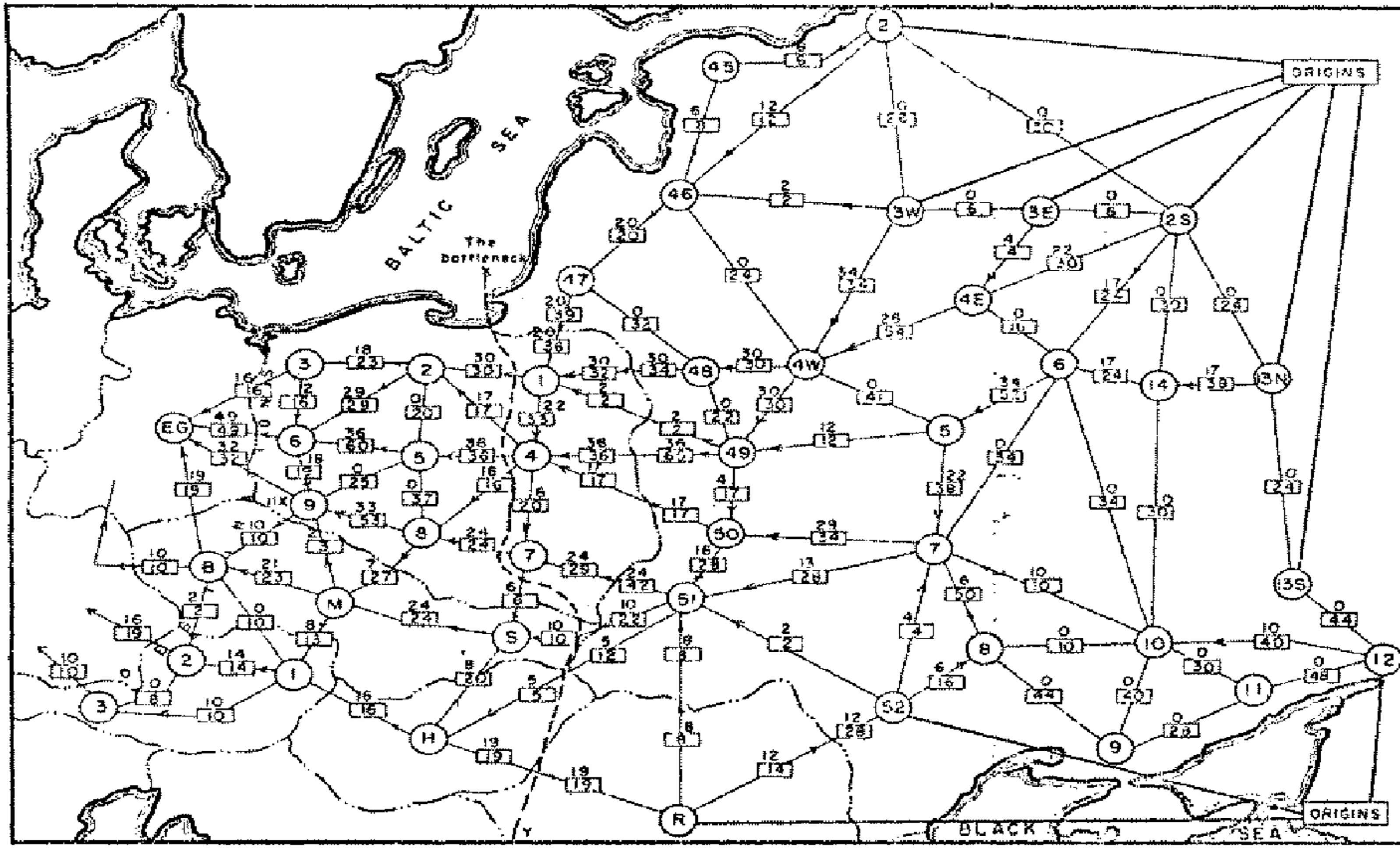
QQ group: 117282780

Recommended Book



A comprehensive introduction to network flows that brings together the classic and the contemporary aspects of the field, and provides an integrative view of theory, algorithms and applications.* presents in-depth, self-contained treatments of shortest path, maximum flow, and minimum cost flow problems, including descriptions of polynomial-time algorithms for these core models. * emphasizes powerful algorithmic strategies and analysis tools such as data scaling, geometric improvement arguments, and potential function arguments. * provides an easy-to-understand descriptions of several important data structures, including d-heaps, Fibonacci heaps, and dynamic trees. * devotes a special chapter to conducting empirical testing of algorithms. * features over 150 applications of network flows to a variety of engineering, management, and scientific domains. * contains extensive reference notes and illustrations.

Example: Soviet Rail Network, 1955



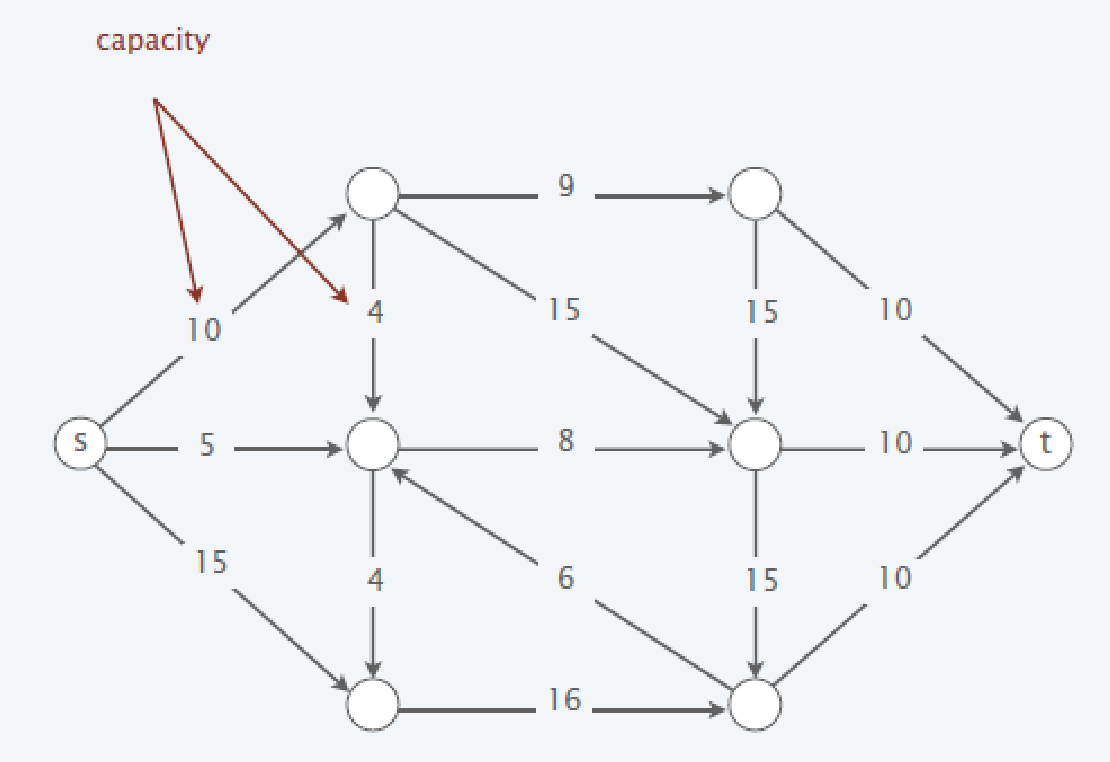
Reference: *On the history of the transportation and maximum flow problems*. Alexander Schrijver in Math Programming, 91: 3, 2002.

Sample Network

Network	Nodes	Arcs	Flow
communication	telephone exchanges, computers, satellites	cables, fiber optics, microwave relays	voice, video, packets
circuits	gates, registers, processors	wires	current
mechanical	joints	rods, beams, springs	heat, energy
hydraulic	reservoirs, pumping stations, lakes	pipelines	fluid, oil
financial	stocks, companies	transactions	money
transportation	airports, rail yards, street intersections	highways, railbeds, airway routes	freight, vehicles, passengers
chemical	sites	bonds	energy

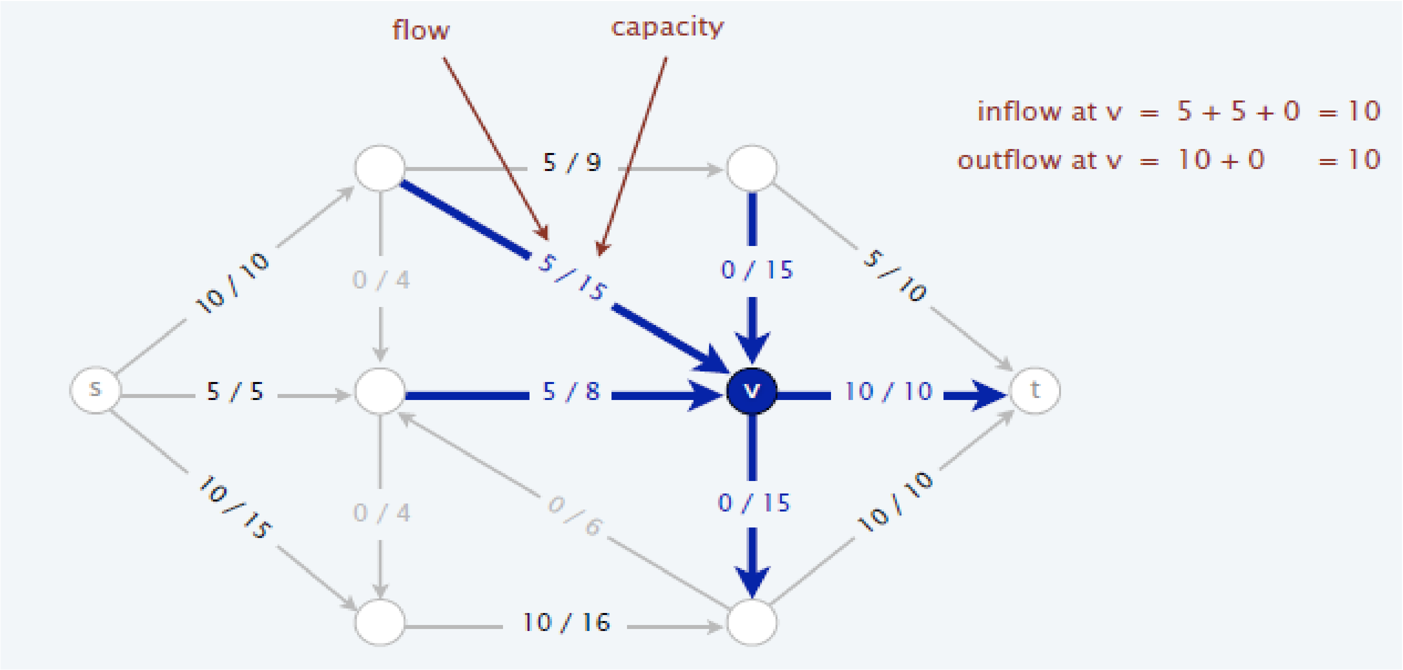
Flow Network

- Abstraction for material flowing through the edges.
- Digraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$.
- Nonnegative integer capacity $c(e)$ for each $e \in E$.



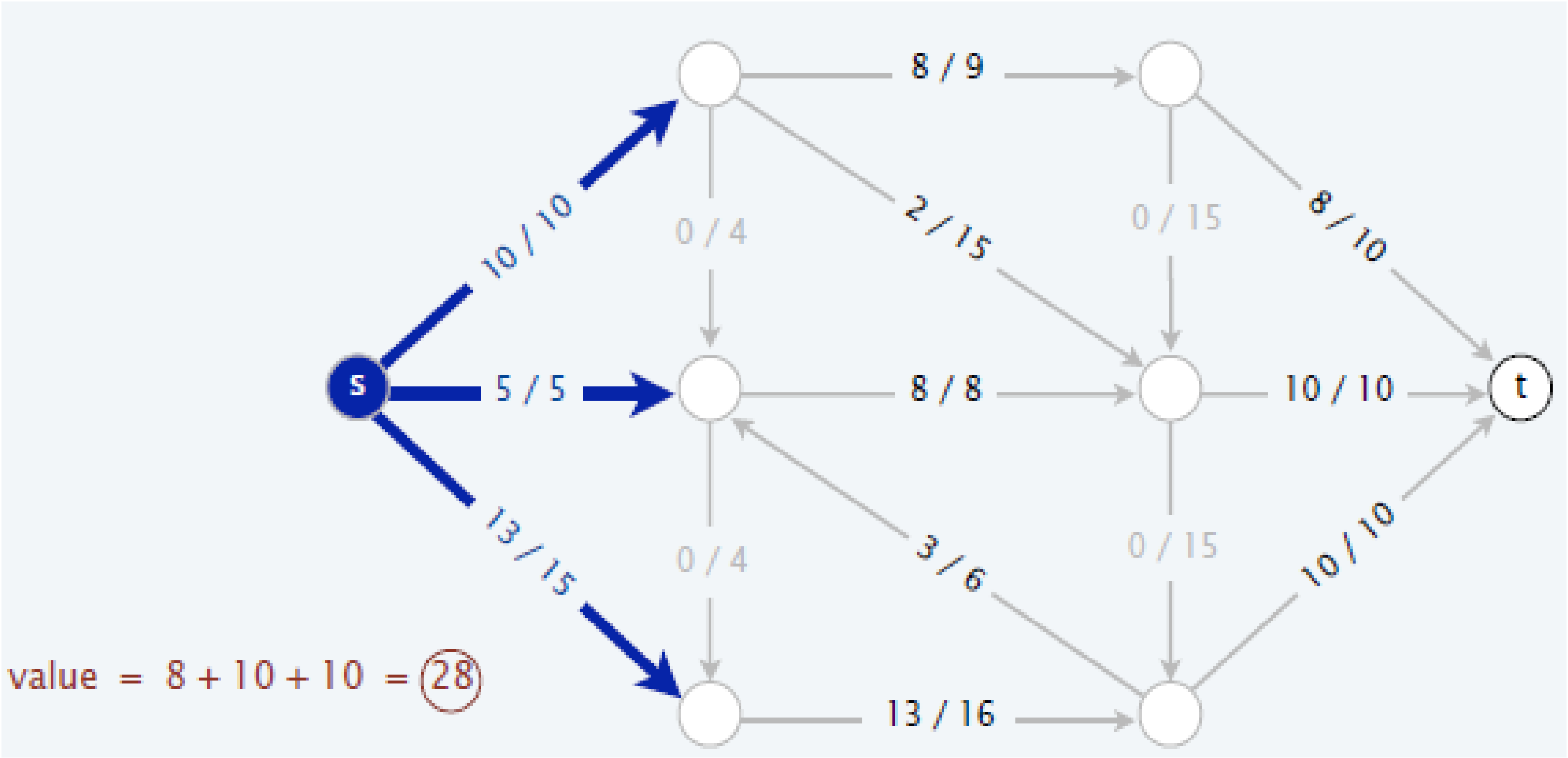
Maximum Flow Problem

- Definition: An *s-t flow f* is a function that satisfies:
  - For each $e \in E$: $0 \leq f(e) \leq c(e)$ (Capacity)
  - For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (Flow conservation，流平衡)



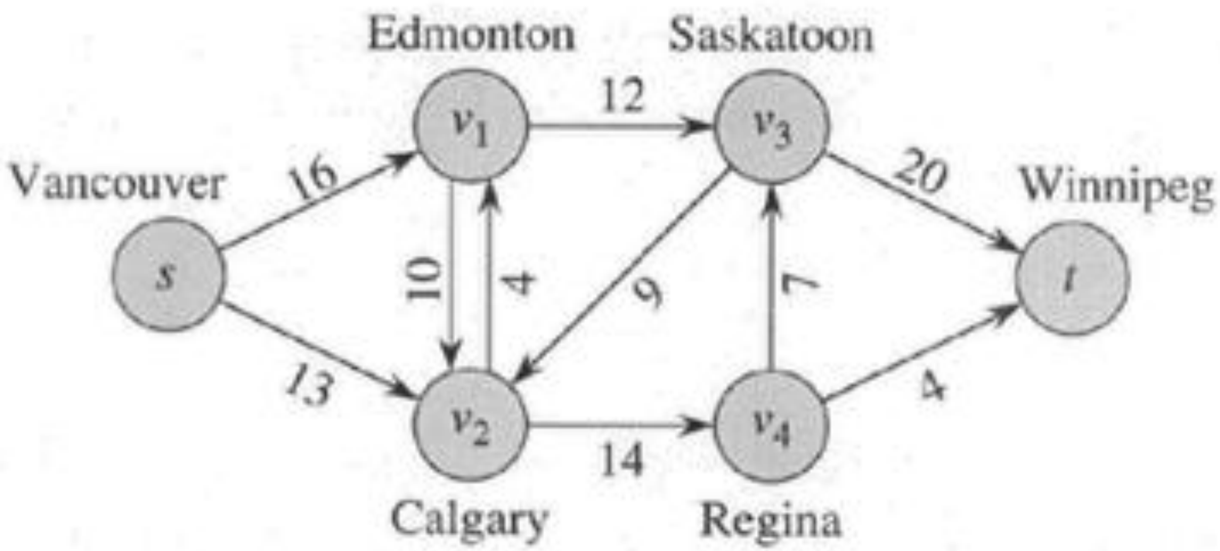
Maximum Flow Problem

- Definition: The value of a flow *f* is: $val(f) = \sum_{e \text{ out of } s} f(e)$.
- Max-flow problem: Find a flow of maximum value.



Exercise

- What is the maximum flow of the following graph?



Lucky Puck Distribution Network

Ford-Fulkerson Algorithm

- Basic idea:
  - Start with zero flow
  - Repeat until convergence:

1. Find an augmenting path, from s to t along which we can push more flow.
2. Augment flow along this path.

- Main components:
  - Residue networks （残量网络/残留网络）
  - Augmenting paths（增广路）

Residual Network

- Given a flow f in network $G = (V, E)$, consider $e = (u,v) \in E$.
- Residual capacity: amount of additional flow we can push directly from u to v.

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

≥ 0, since $f(u, v) \leq c(u, v)$

≥ 0, since $f(v, u) \geq 0$

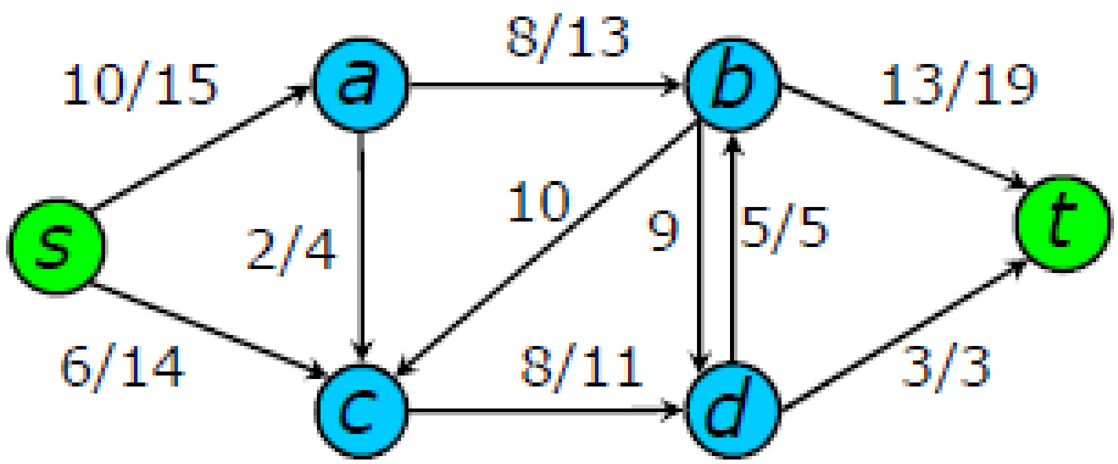
- Residual network $G_f = (V, E_f)$

  $E_f = \{ e \in V \times V \mid c_f(e) > 0 \}$
- Example:

  $c(u,v) = 16$, $f(u,v) = 5$, then $c_f(u, v) = 11$

Exercise

- Compute the residual graph of the graph with the following flow:



Augmenting Path

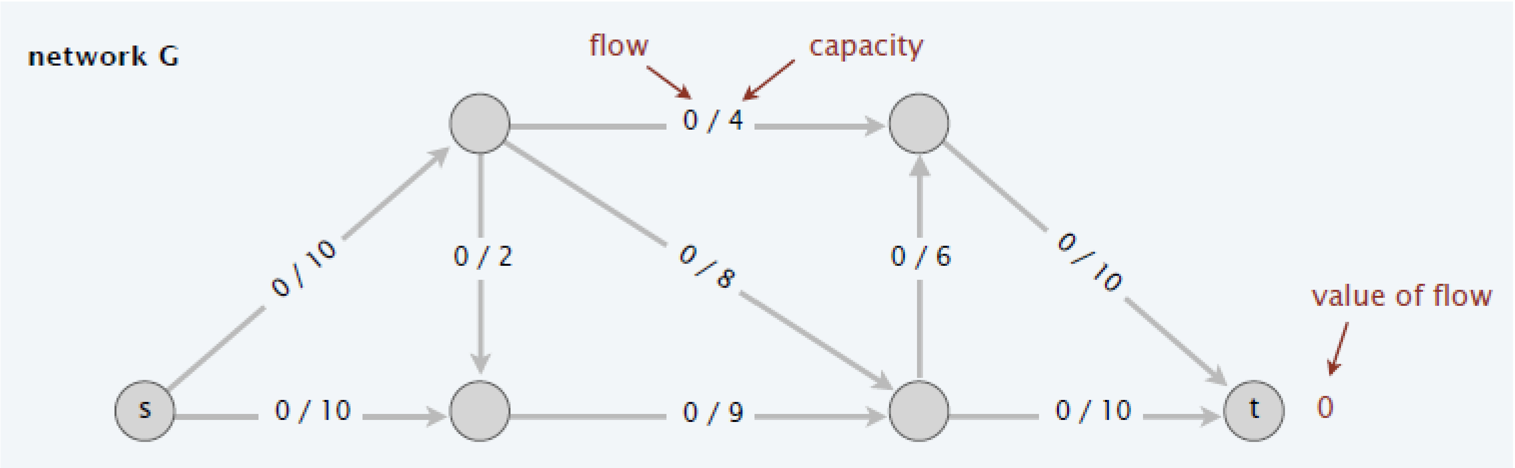
- An augmenting path is a simple $s \rightsquigarrow t$ path P in the residual graph G_f.
- The bottleneck capacity of an augmenting P is the minimum residual capacity of any edge in P.
- Key property:
  - Let f be a flow and let P be an augmenting path in G_f.
  - Then f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

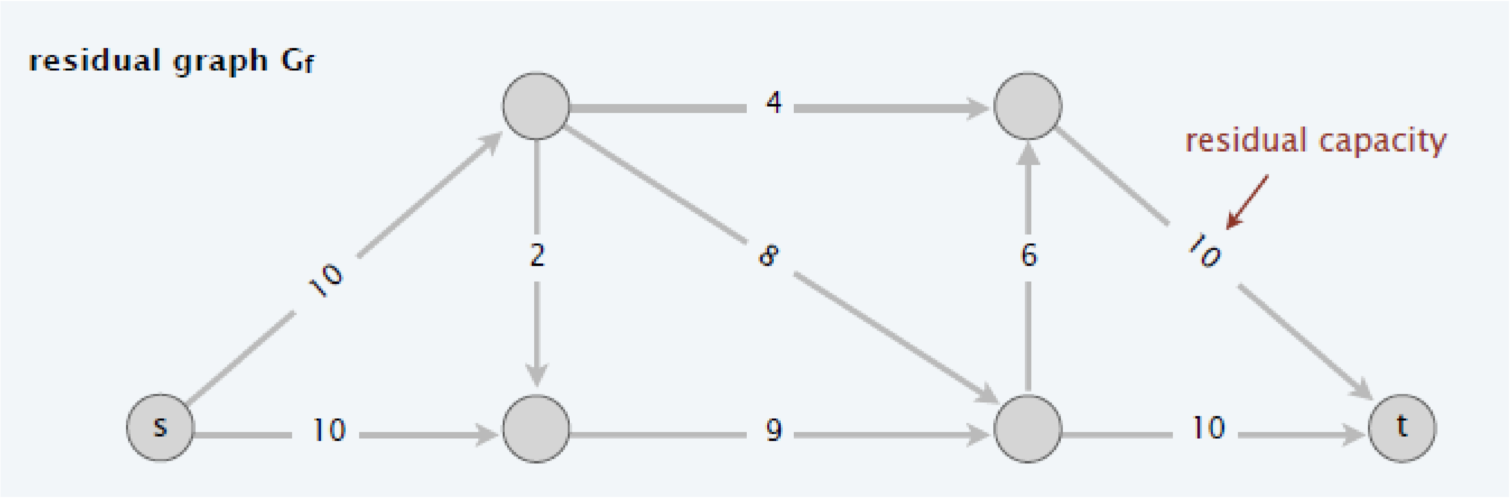
Finding an Augmenting Path

- Find a path from *s to* t in the residual graph.
- The residual capacity of a path P in G_f:

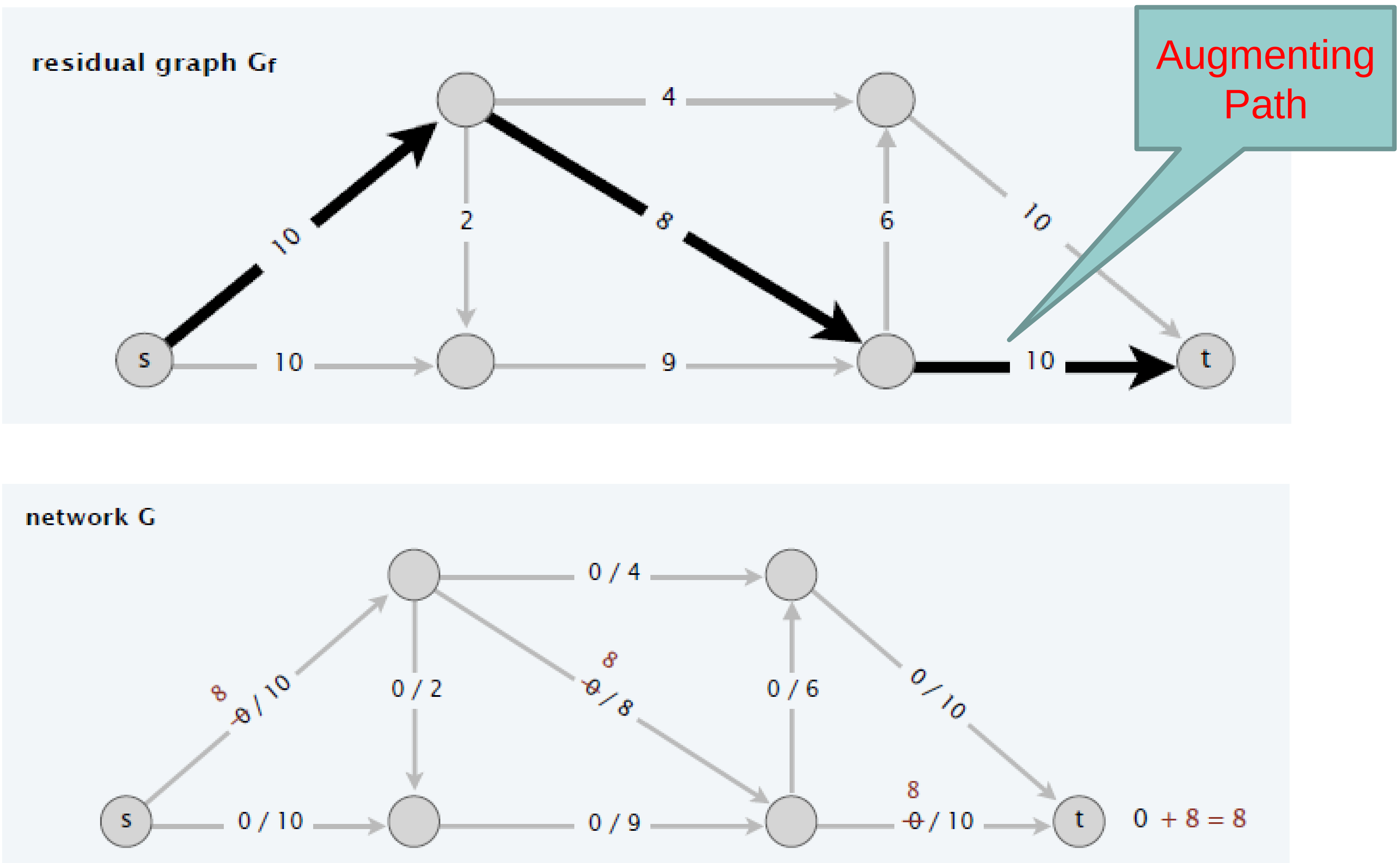
  $c_f(P) = \min\{ c_f(u,v): (u,v) \text{ is in } P \}$
- Doing augmentation: for all (u,v) in P, we just add this $c_f(p)$ to $f(u,v)$.
- Resulting flow is a valid flow with a larger value.

Ford-Fulkerson Algorithm Demo

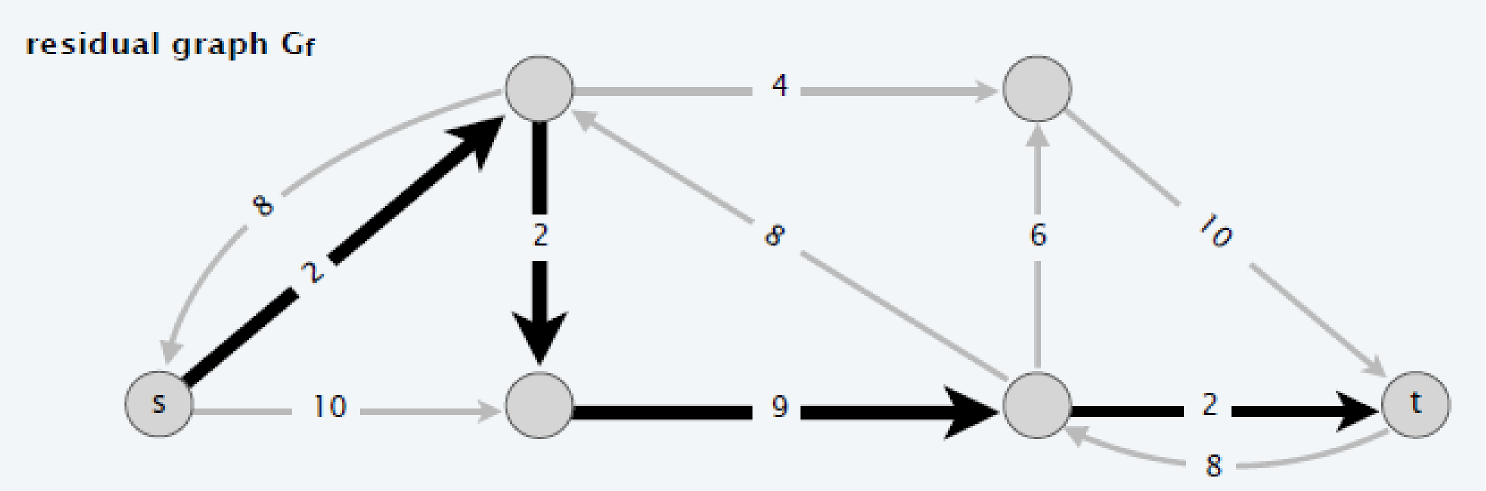


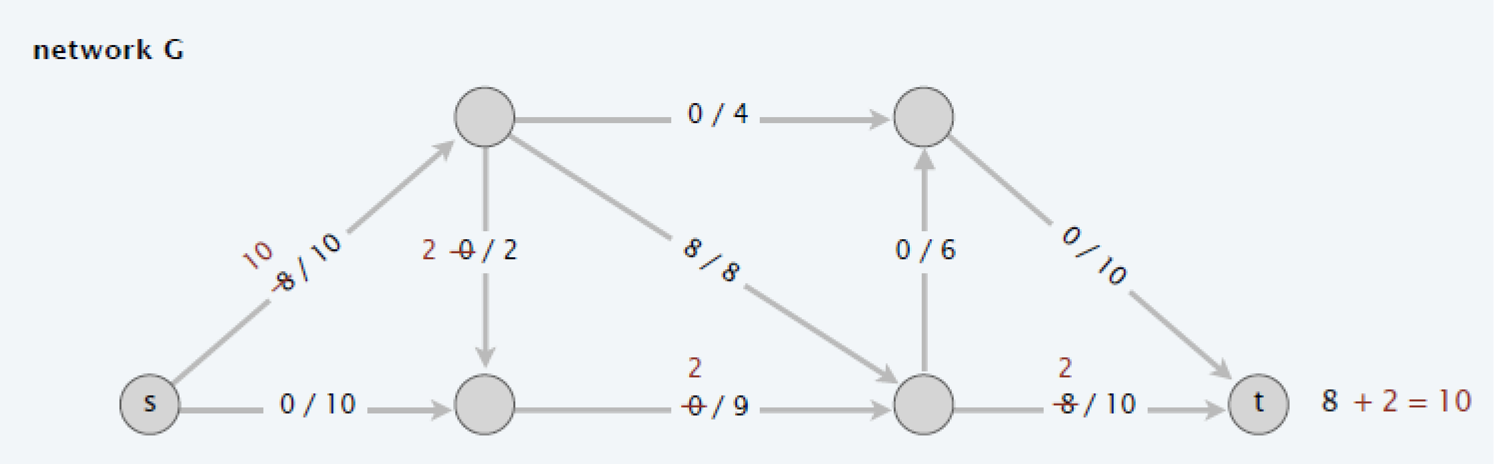


Ford-Fulkerson Algorithm Demo

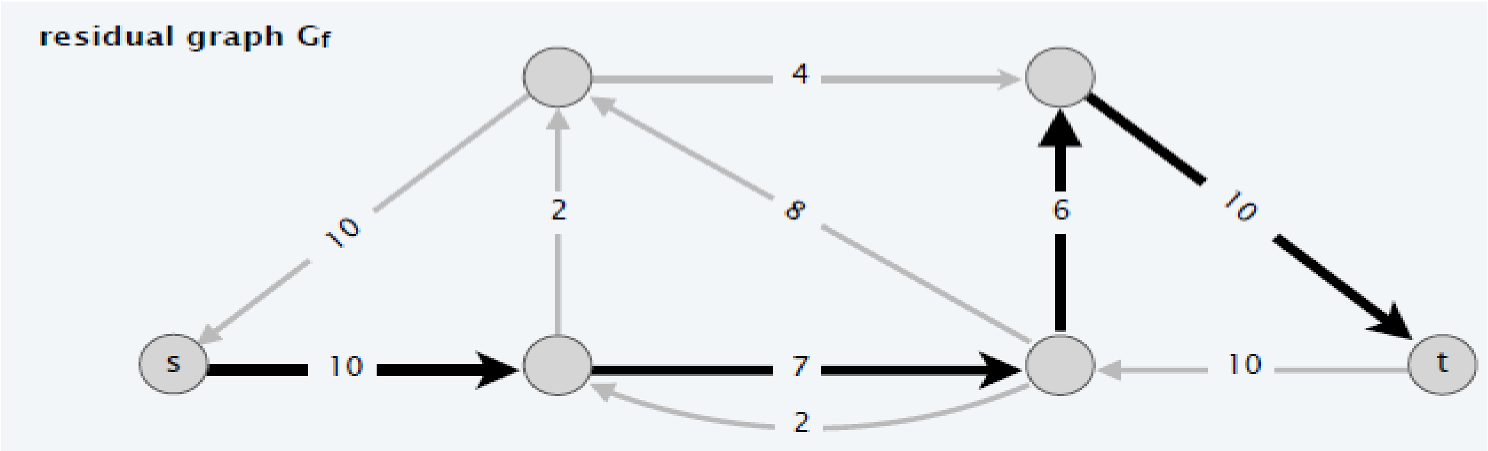


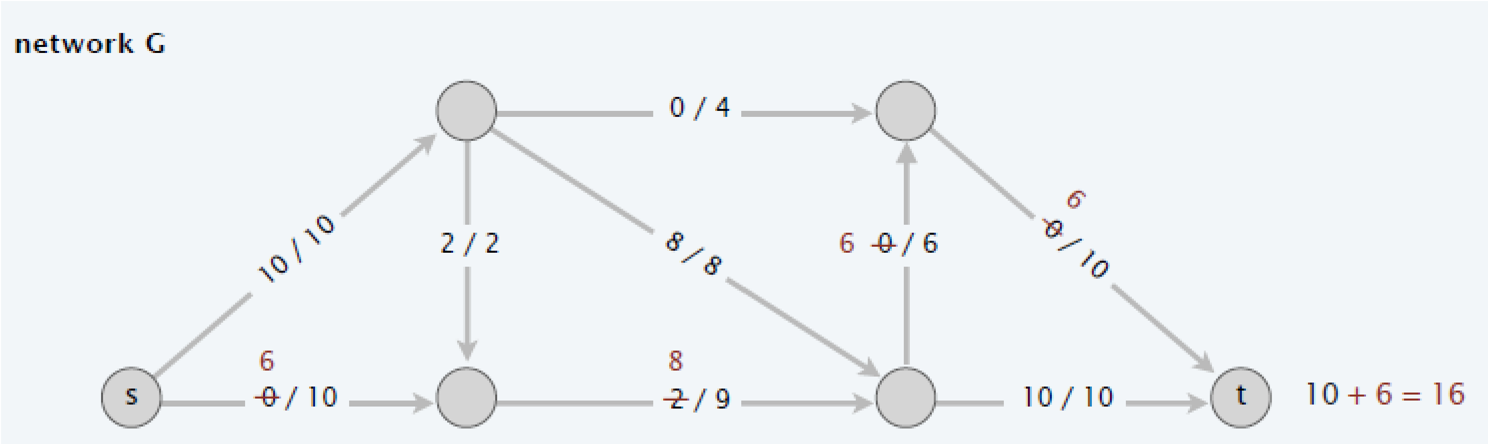
Ford-Fulkerson Algorithm Demo



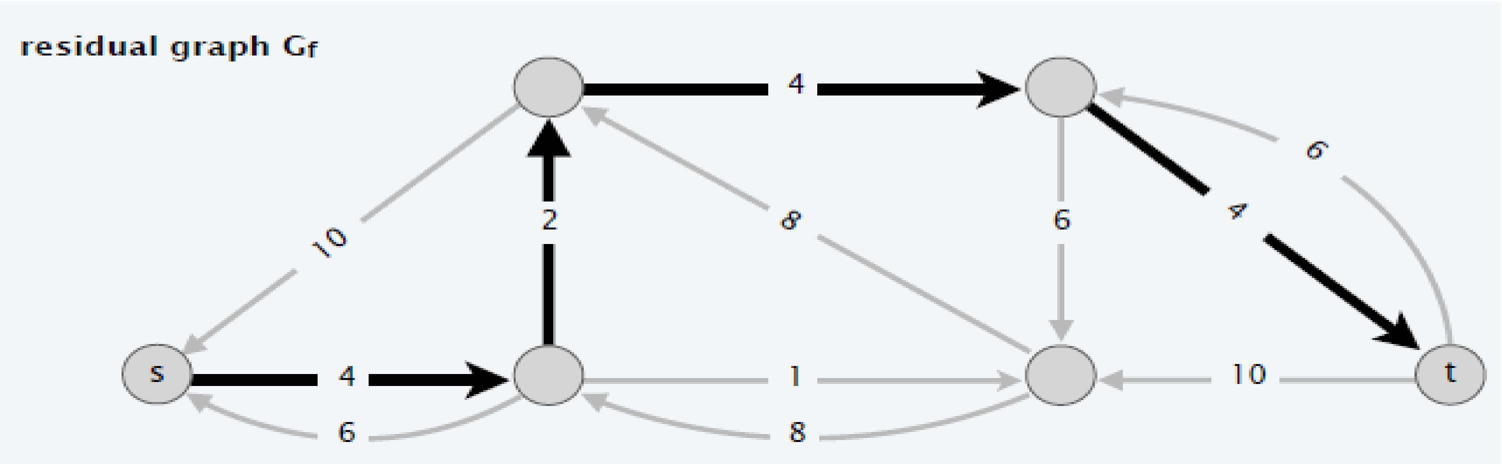


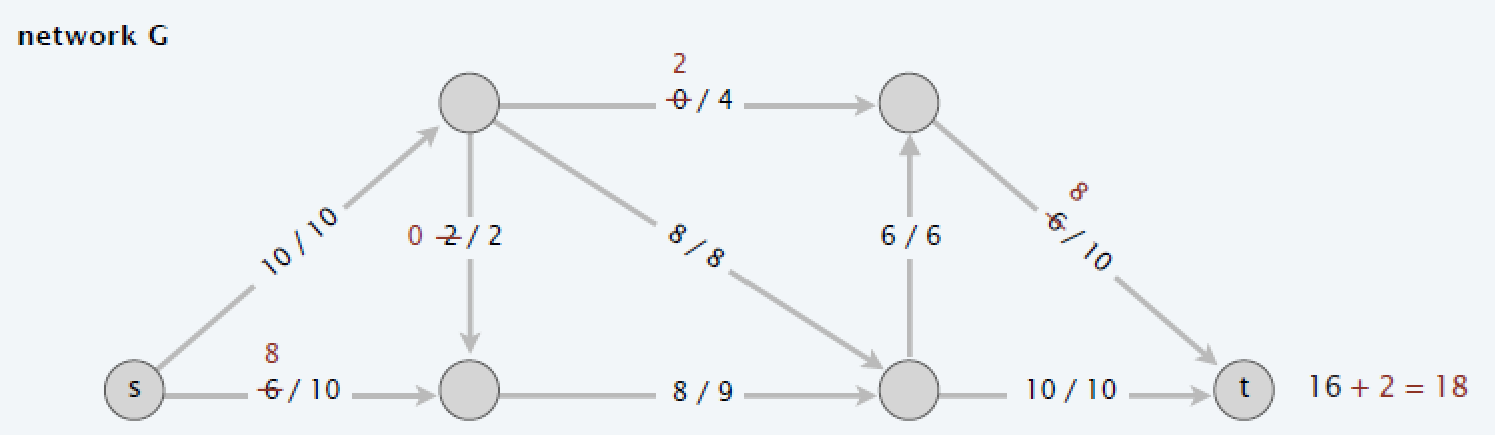
Ford-Fulkerson Algorithm Demo



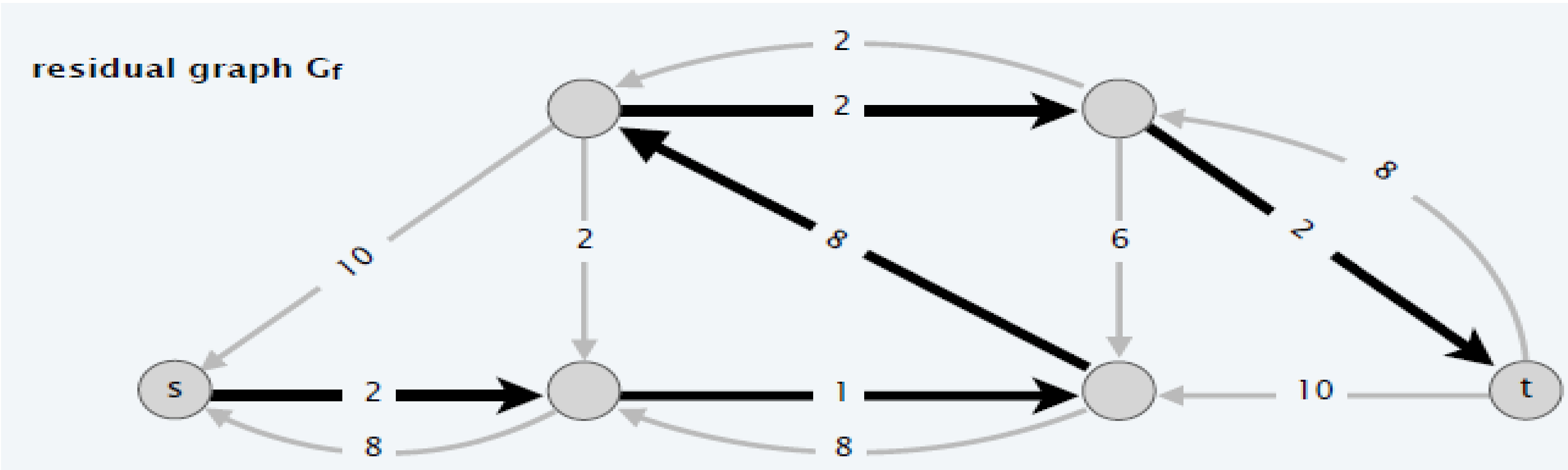


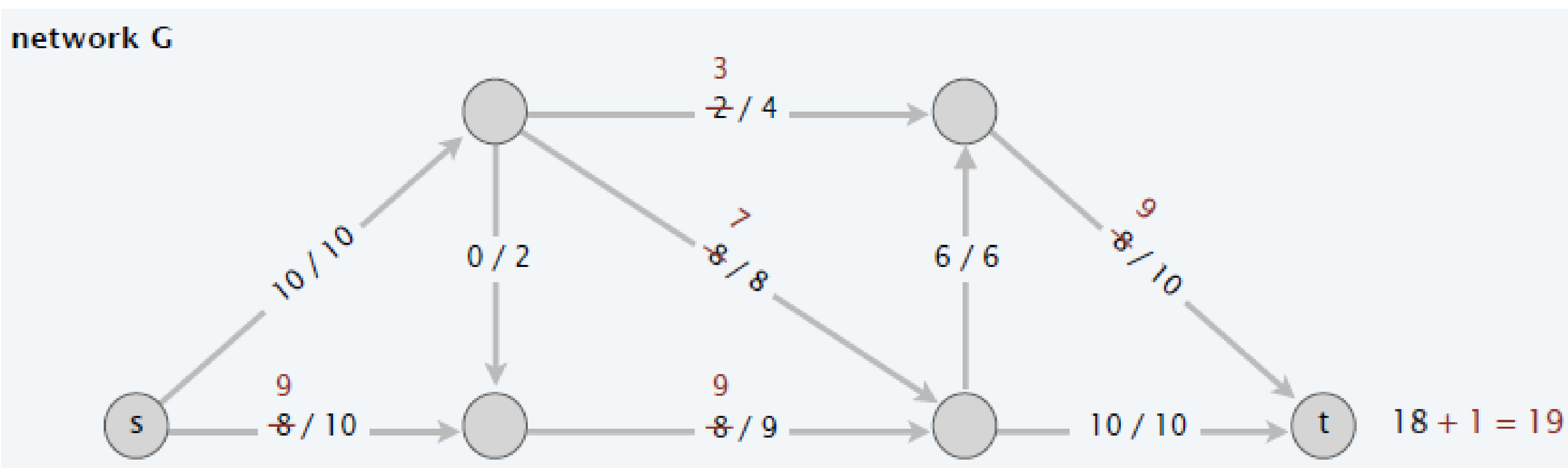
Ford-Fulkerson Algorithm Demo



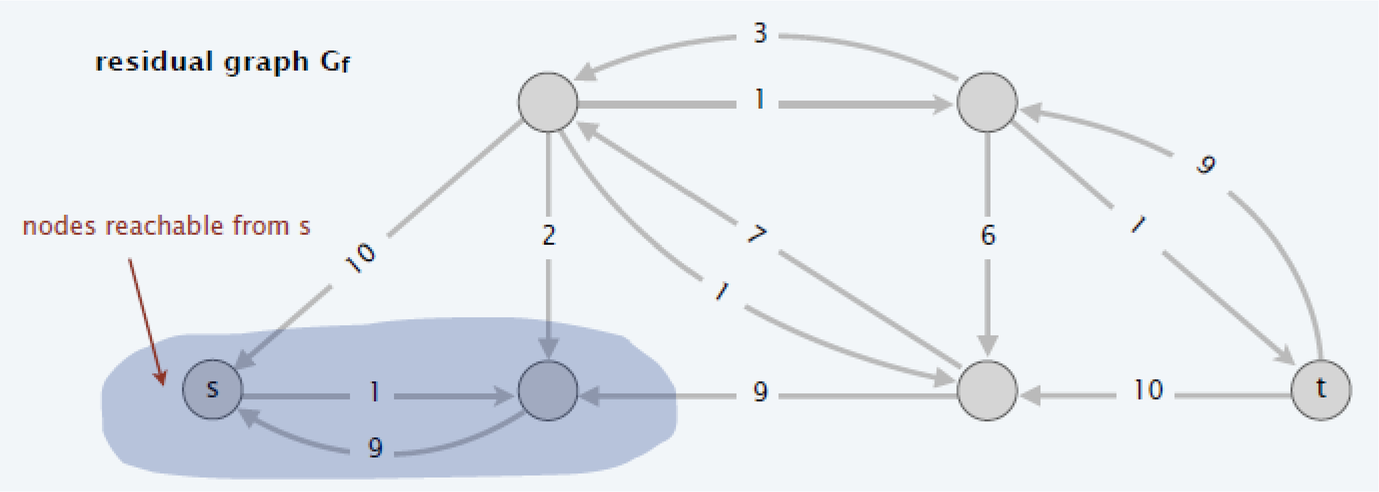


Ford-Fulkerson Algorithm Demo





Ford-Fulkerson Algorithm Demo



Ford-Fulkerson Algorithm

```
int fordFulkerson( int n, int s, int t )
{   // ASSUMES: cap[u][v] stores capacity of edge (u,v).  cap[u][v] = 0 for no edge.

    // Initialize the flow network so that fnet[u][v] = 0 for all u,v
    int flow = 0;  // no flow yet
    while( true ) {
        // Find an augmenting path, using BFS
        for( int i=0; i < n; i++ ) prev[i] = -1;
        queue< int > q;
        prev[s] = -2;
        q.push( s );
        while( !q.empty() && prev[t] == -1 ) {
            int u = q.front();
            q.pop();

            for( int v = 0; v < n; v++ ) {
                if( prev[v] == -1 ) {   // not seen yet
                    if ( fnet[v][u] || fnet[u][v] < cap[u][v] ) {
                        // either a backward edge (v,u) or a forward edge (u,v)
                        prev[v] = u;
                        q.push( v );
                    }
                }
            }
        }
```

Note these comments!

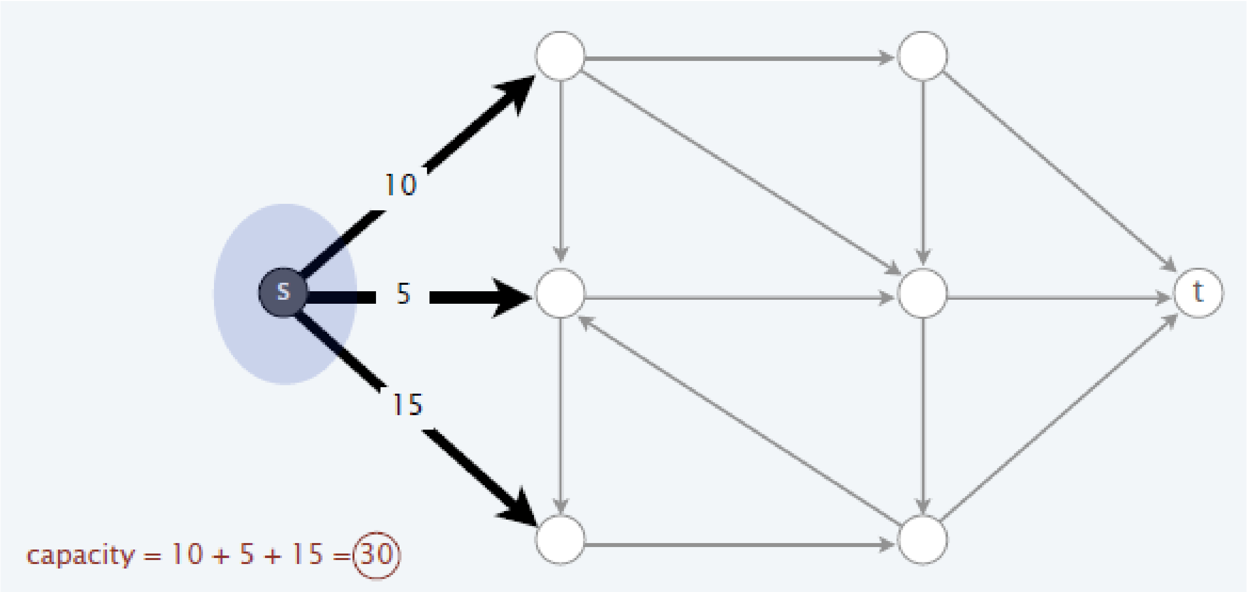
Ford-Fulkerson Algorithm

```
        // See if we couldn't find any path to t (t has no parents)
        if( prev[t] == -1 ) break;

        // Get the bottleneck capacity
        int bot = INT_MAX;
        for( int v = t, u = prev[v]; u >= 0; v = u, u = prev[v] ) {
            if ( fnet[v][u] ) // always use backward edge over forward
                bot = min( bot, fnet[v][u] );
            else // must be a forward edge otherwise
                bot = min( bot, cap[u][v] – fnet[u][v] );
        }

        // update the flow network
        for( int v = t, u = prev[v]; u >= 0; v = u, u = prev[v] ) {
            if ( fnet[v][u] ) // backward edge -> subtract
                fnet[v][u] -= bot;
            else // forward edge -> add
                fnet[u][v] += bot;
        }

        // Sent 'bot' amount of flow from s to t, so update flow
        flow += bot;
    }

    return flow;
}
```

Cuts（割）

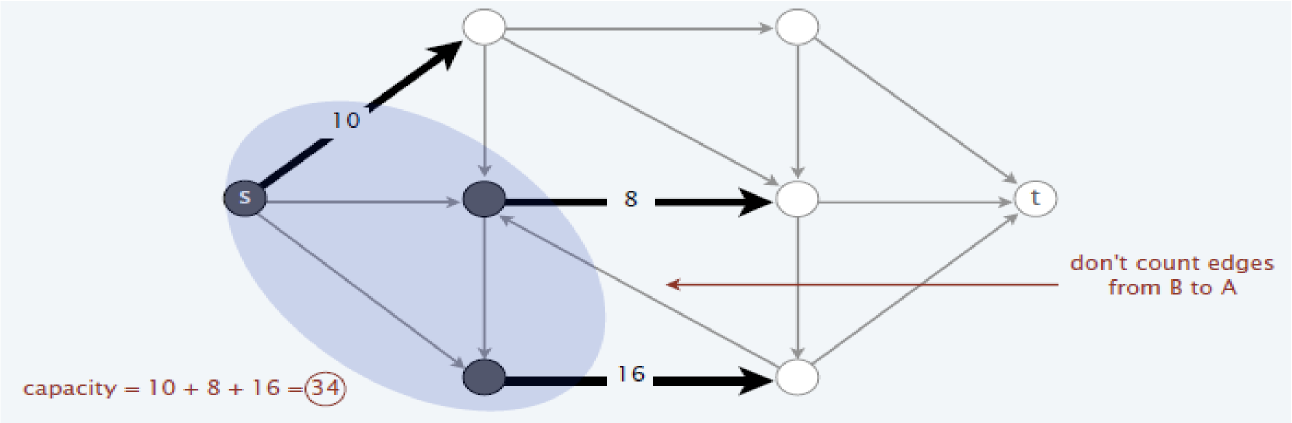
- Definition: An s-t cut is a partition (A, B) of V with s $\in$ A and t $\in$ B.
- Definition: The capacity of a cut (A, B) is:

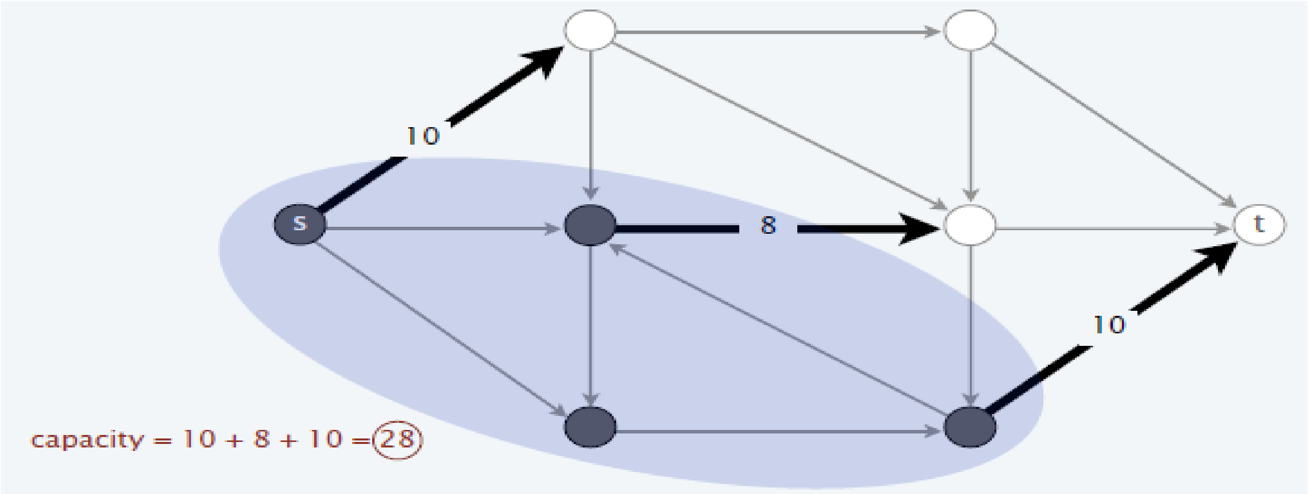
$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Minimum Cut Problem

- Goal: Find a cut of minimum capacity.

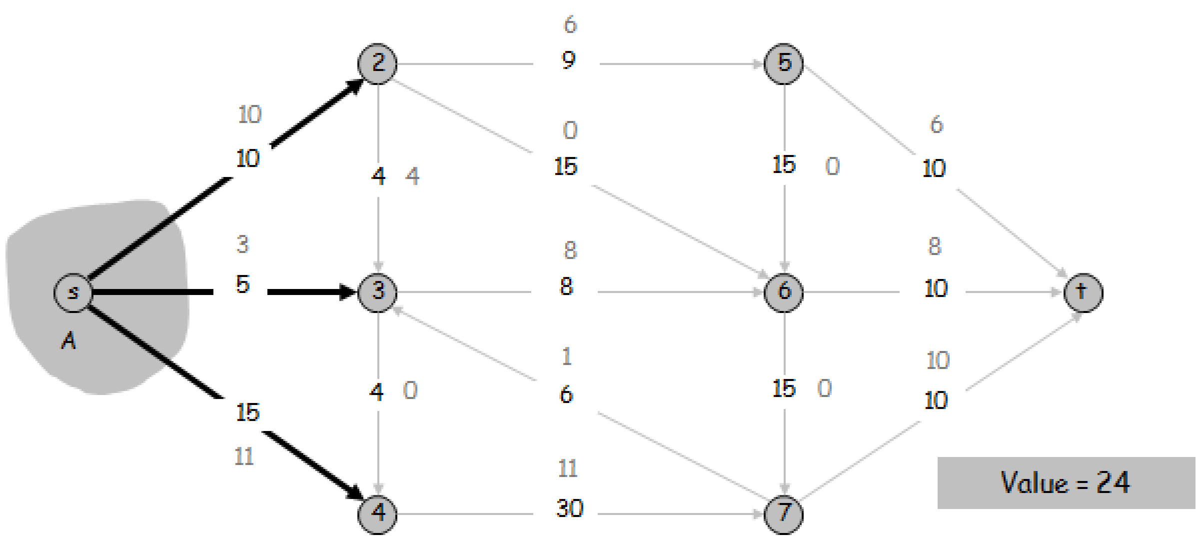




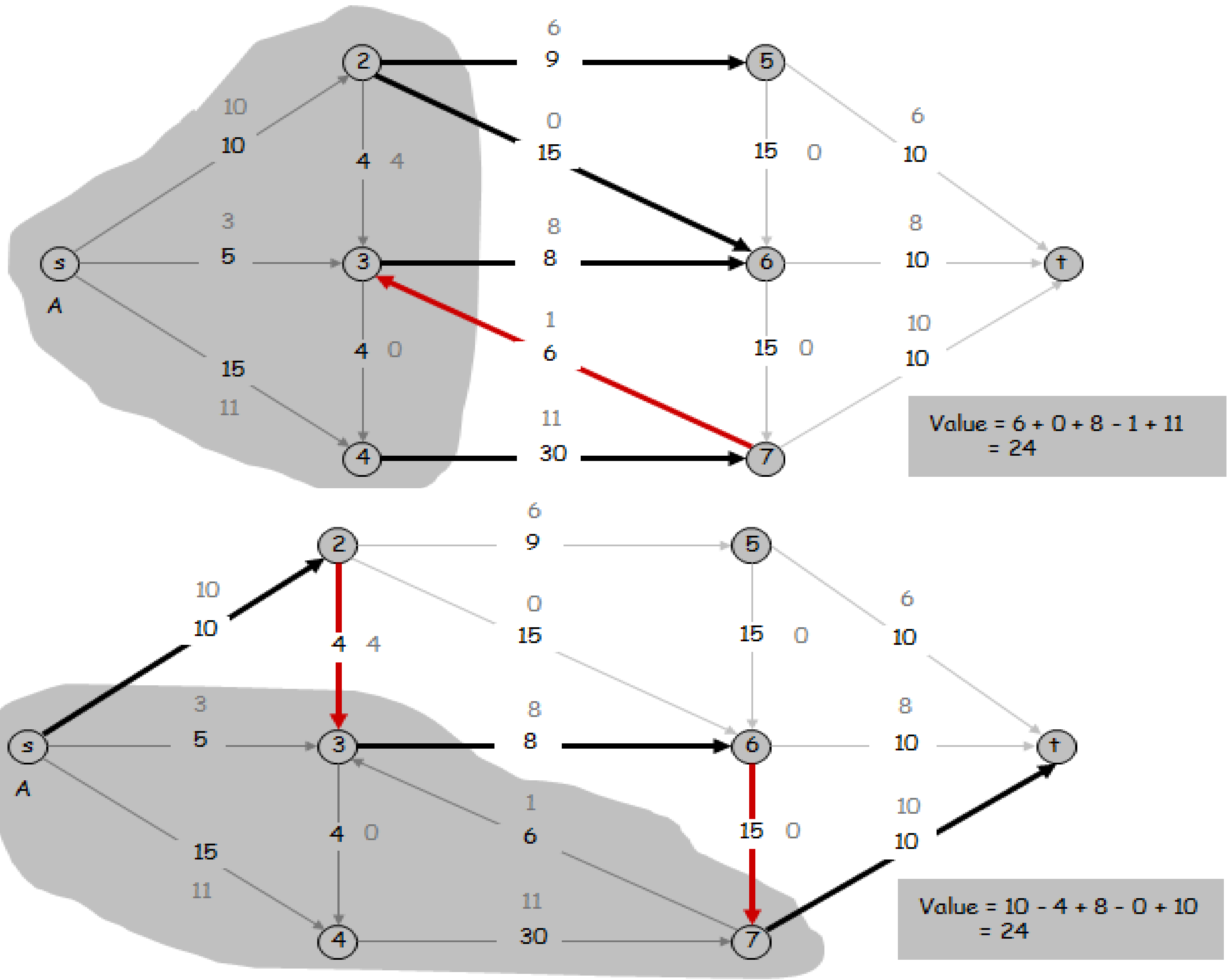
Flows and Cuts

- Flow value lemma: Let *f* be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



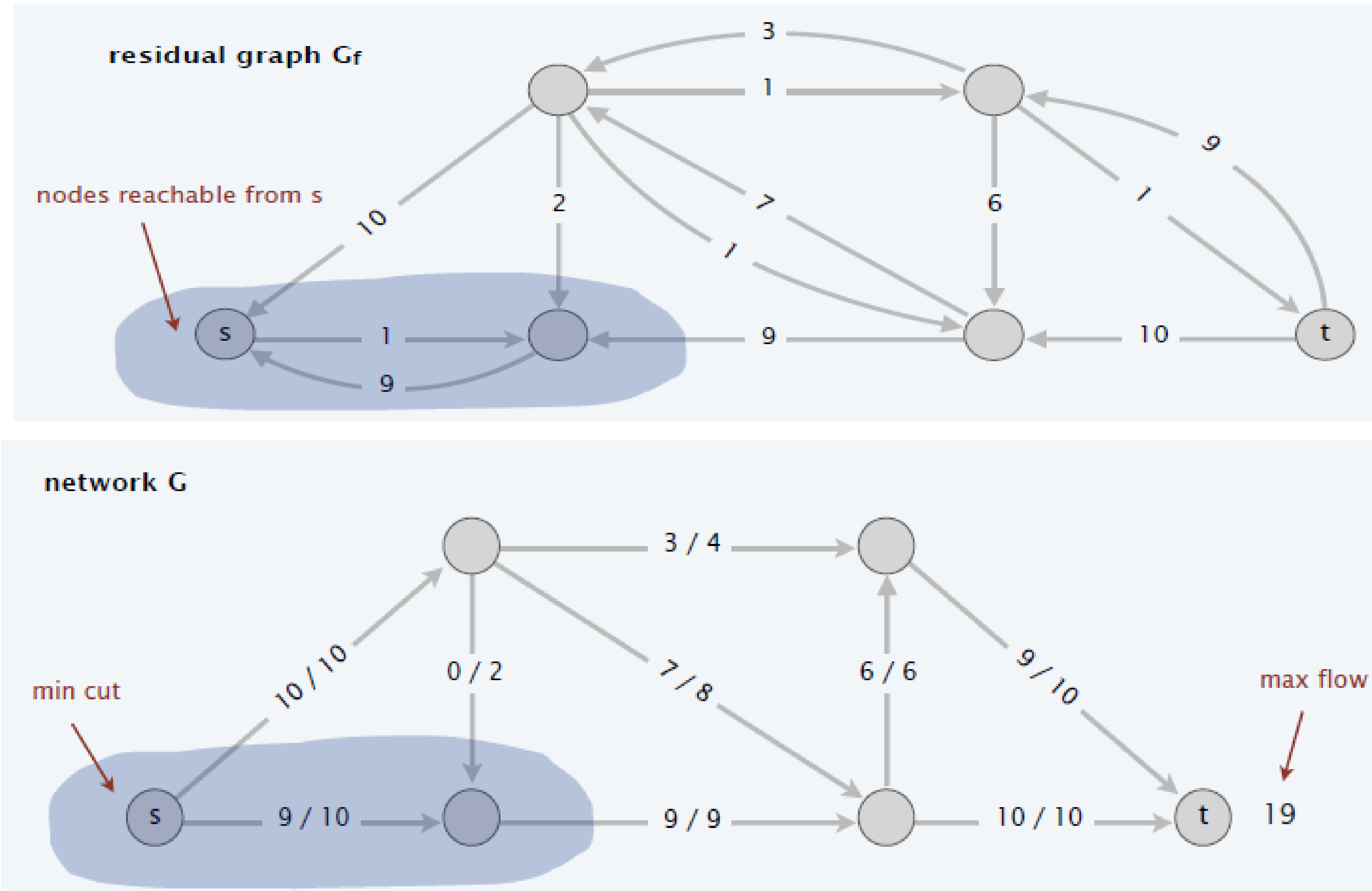
Flows and Cuts



Flows and Cuts

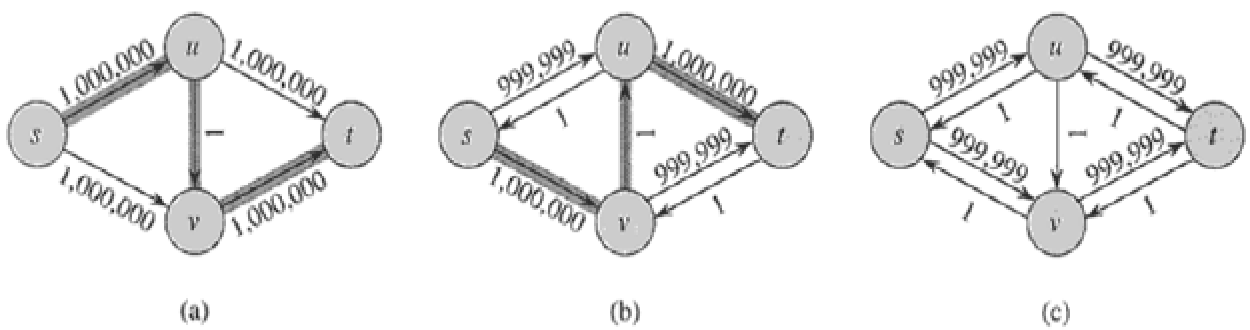
- Weak duality: Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut, i.e., $v(f) \leq cap(A, B)$.
- Corollary: Let f be any flow, and let (A, B) be any cut. If $v(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.
- Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.
- Max-flow Min-cut Theorem: [Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Recall: FFA Demo



Running Time Complexity

- Naive Implementation
  - Assuming integer flow. Each augmentation increases the value of the flow by some positive amount.
  - Augmentation can be done in O(E). Total worst-case running time O(E|f*|), where f* is the max-flow found by the algorithm.



Augmenting path of 1 | Resulting Residual Network | Resulting Residual Network

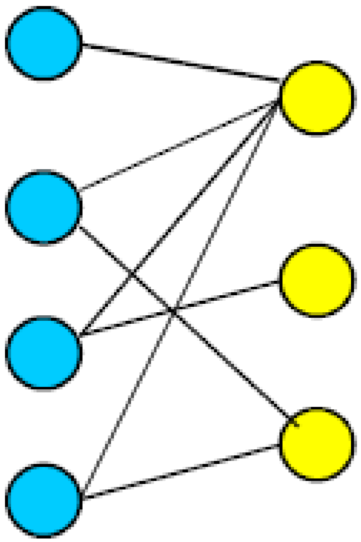
- Take shortest path (in terms of number of edges) as an augmenting path – Edmonds-Karp algorithm
  - How do we find such a shortest path?
  - Running time *O(VE²), because the number of* augmentations is *O(VE)*

Application – Bipartite Matching

- Given a community with *n men and m* women
- Assume we have a way to determine which couples (man/woman) are compatible for marriage
  - E.g. (Joe, Susan) or (Fred, Susan) but not (Frank, Susan)
- Problem: Maximize the number of marriages
  - No polygamy（多偶制） allowed
  - Can solve this problem by creating a flow network out of a bipartite graph

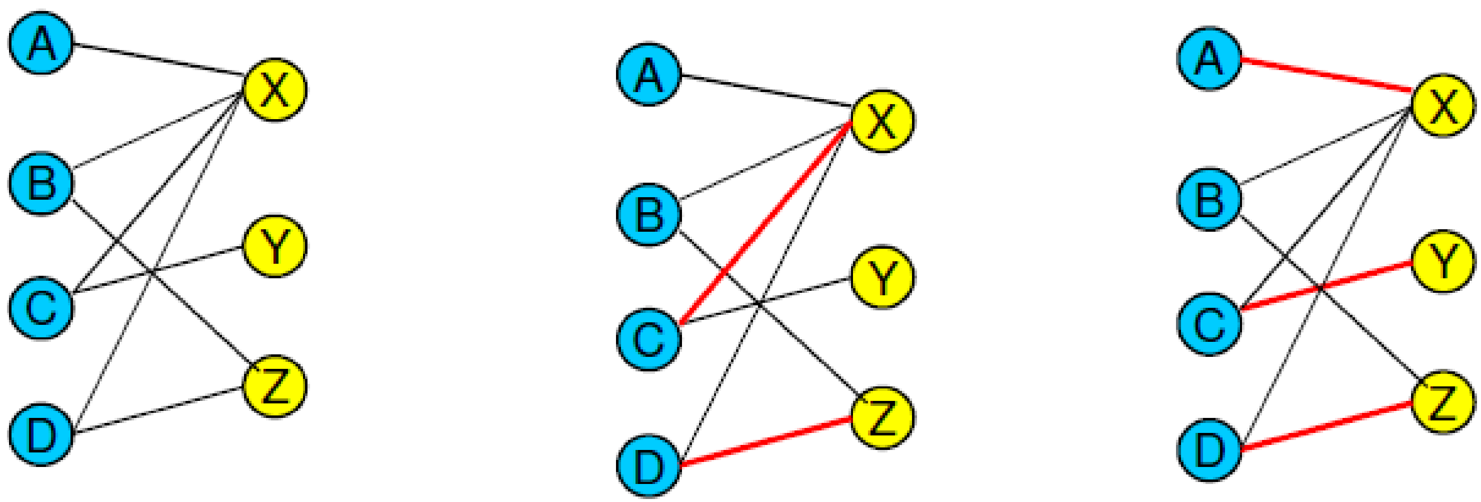
Bipartite Graph

- A bipartite graph is an undirected graph G=(V,E) in which V can be partitioned into two sets V_1 and V_2 such that $(u,v) \in E$ implies either $u \in V_1$ and $v \in V_2$ or vice versa.
- That is, all edges go between the two sets V_1 and V_2 and not within V_1 and V_2.



Model for Matching Problem

- Men on leftmost set, women on rightmost set, edges if they are compatible



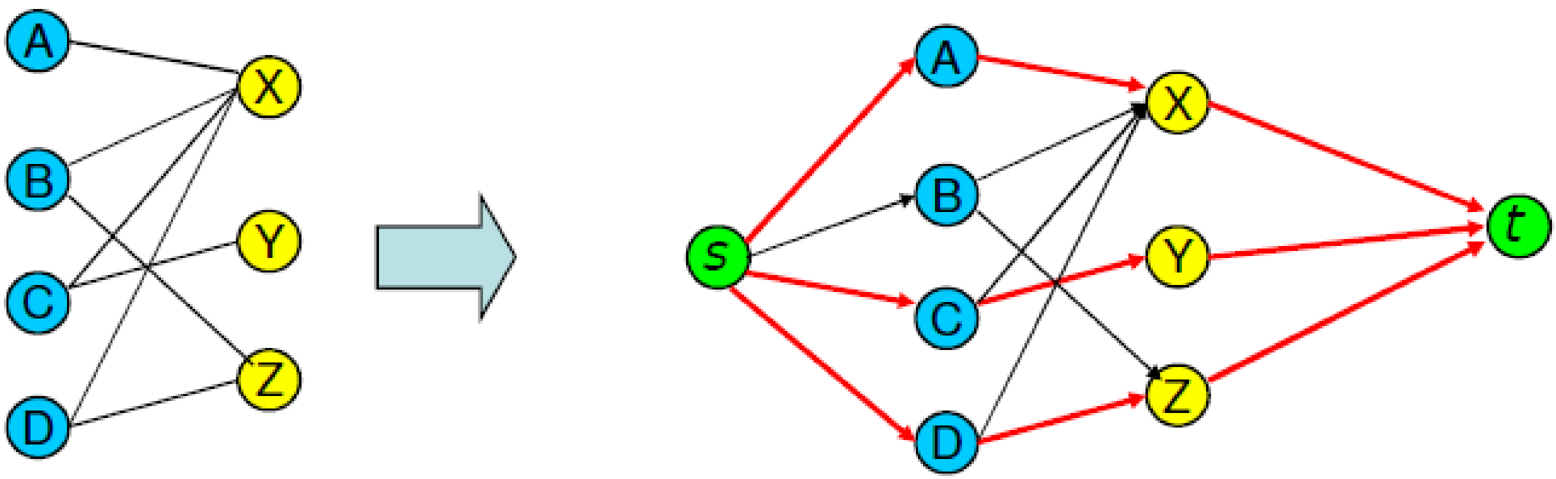
Men Women

A matching

Optimal matching

Solution Using Max Flow

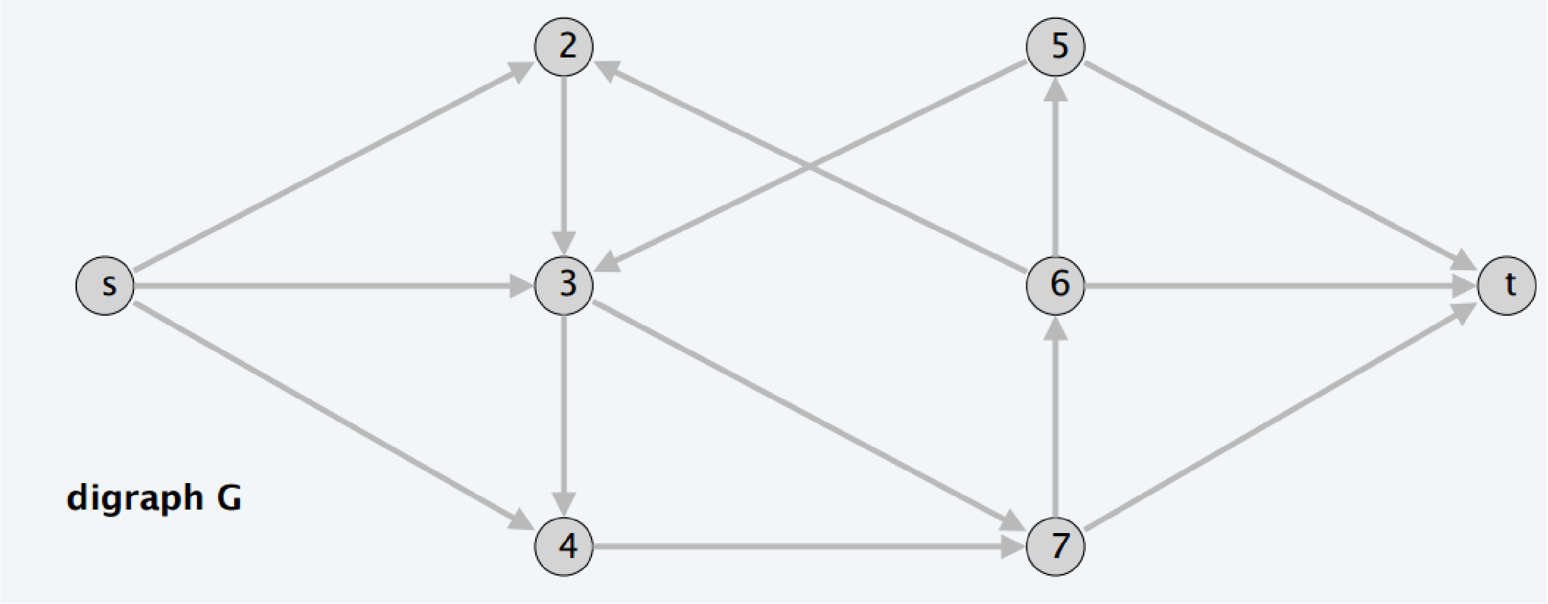
- Add a supersouce, supersink, make each undirected edge directed with a flow of 1



Since the input is 1, flow conservation prevents multiple matchings

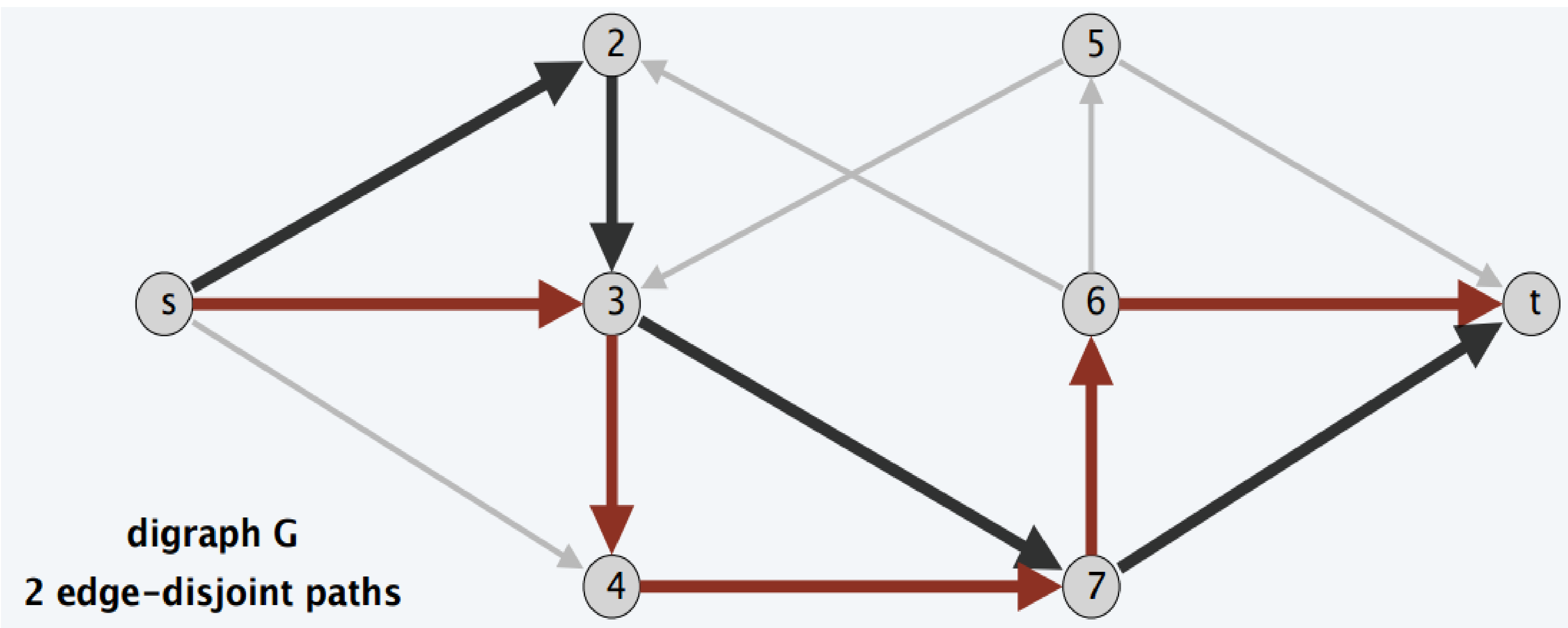
Application – Edge Disjoint Paths

- Definition: Two paths are edge-disjoint if they have no edge in common.
- Disjoint path problem. Given a digraph G = (V, E) and two nodes s and t, find the max number of edge-disjoint s↝t paths.



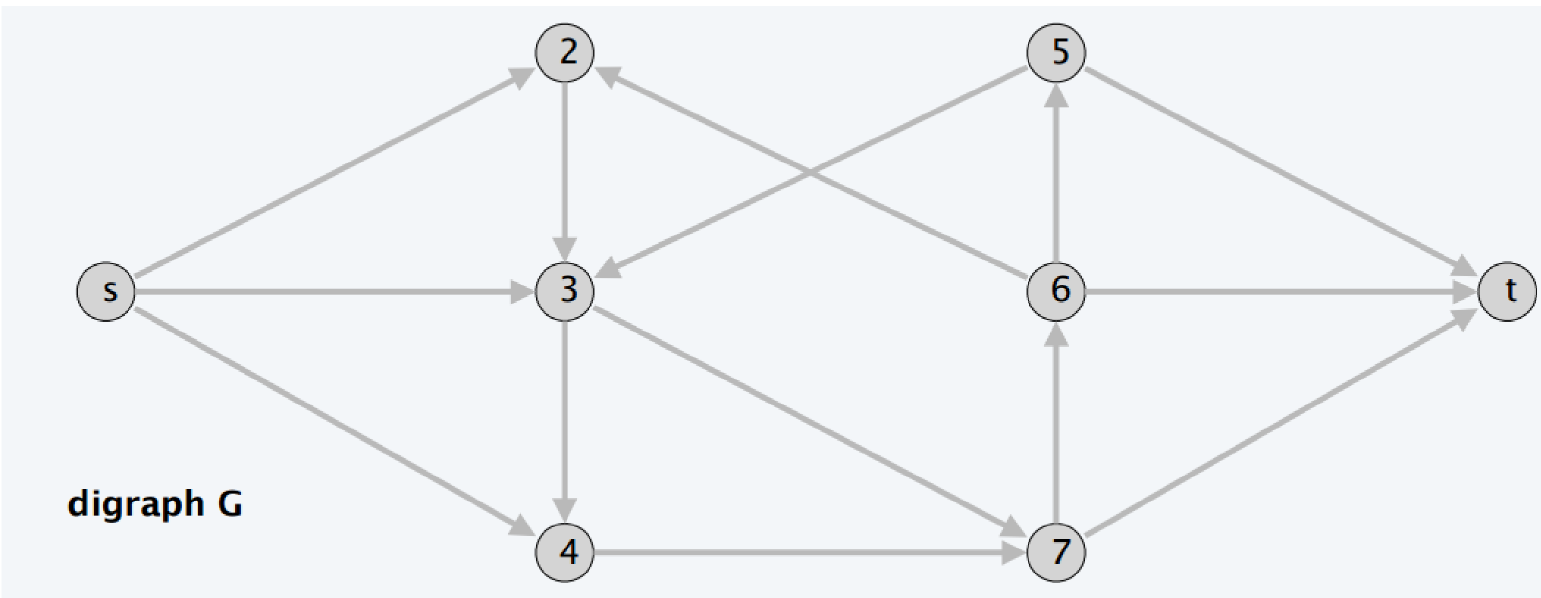
Edge Disjoint Paths

- Max flow formulation: Assign unit capacity to every edge.
- Max number edge-disjoint s↝t paths equals value of max flow.



Application – Node Disjoint Paths

- Definition: Two s-t paths P and P' f are said to be *node-disjoint* if the only nodes in common to P and P' are s and t).
- How can one determine the maximum number of node disjoint s-t paths? (Hint: node splitting)



Thank you!