Max Flow, Min Cut

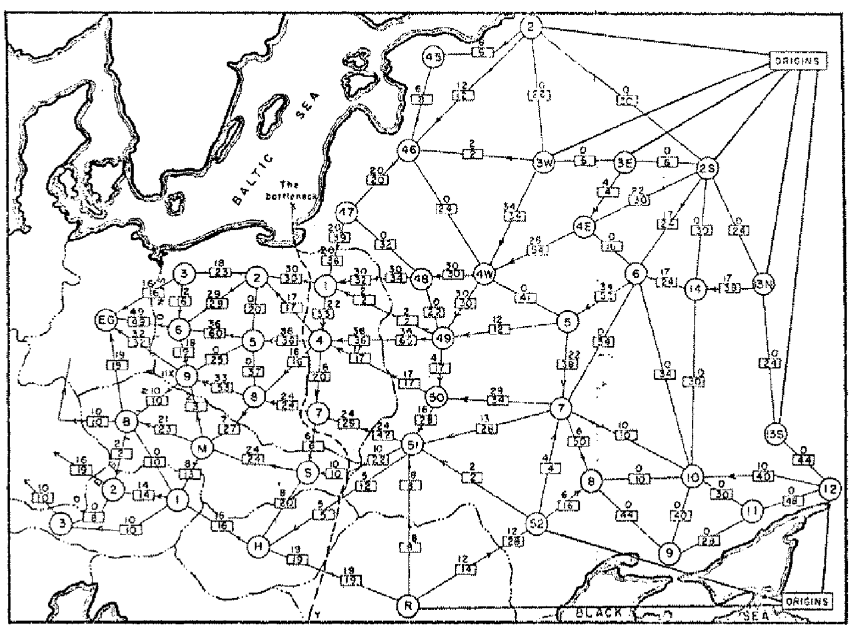
Minimum cut
Maximum flow
Max-flow min-cut theorem
Ford-Fulkerson augmenting path algorithm
Edmonds-Karp heuristics
Bipartite matching

Maximum Flow and Minimum Cut

Max flow and min cut.
- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions.
- Network connectivity.
- Bipartite matching.
- Data mining.
- Open-pit mining.
- Airline scheduling.
- Image processing.
- Project selection.
- Baseball elimination.
- Network reliability.
- Security of statistical data.
- Distributed computing.
- Egalitarian stable matching.
- Distributed computing.
- Many many more . . .

Soviet Rail Network, 1955

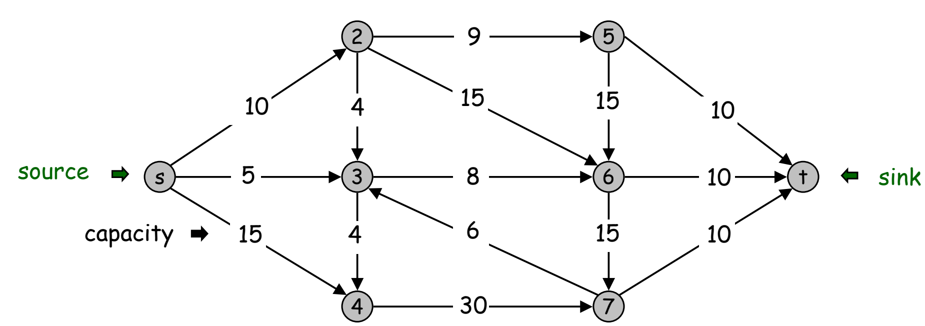


Source: *On the history of the transportation and maximum flow problems.* Alexander Schrijver in Math Programming, 91: 3, 2002.

Minimum Cut Problem

Network: abstraction for material FLOWING through the edges.
- Directed graph.
- Capacities on edges.
- Source node s, sink node t.

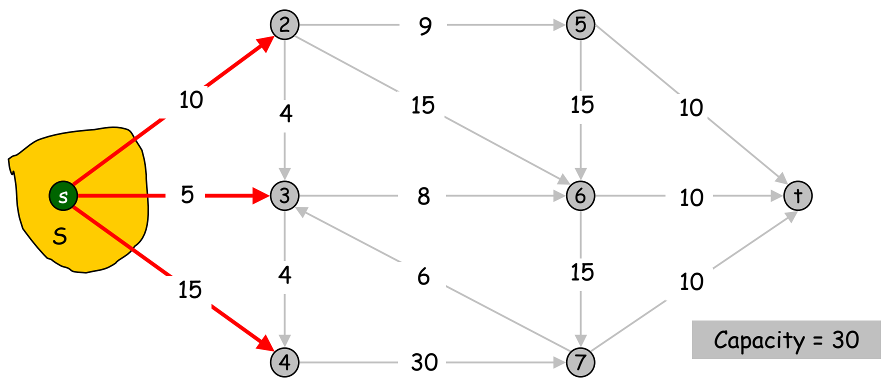
Min cut problem. Delete "best" set of edges to disconnect t from s.



Cuts

A cut is a node partition (S, T) such that s is in S and t is in T.

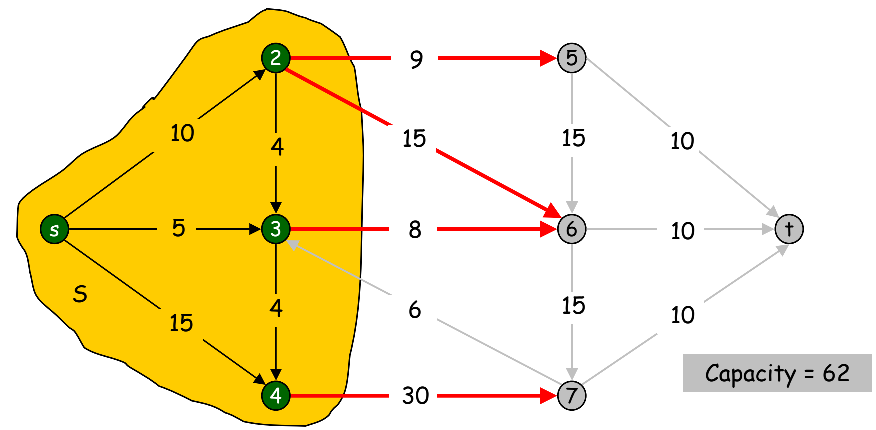
- capacity(S, T) = sum of weights of edges leaving S.



Cuts

A cut is a node partition (S, T) such that s is in S and t is in T.

- capacity(S, T) = sum of weights of edges leaving S.

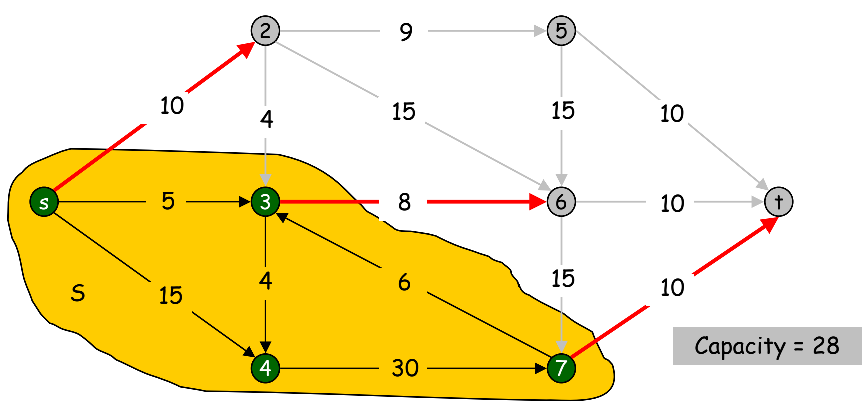


Minimum Cut Problem

A cut is a node partition (S, T) such that s is in S and t is in T.

- capacity(S, T) = sum of weights of edges leaving S.

Min cut problem. Find an s-t cut of minimum capacity.



Maximum Flow Problem

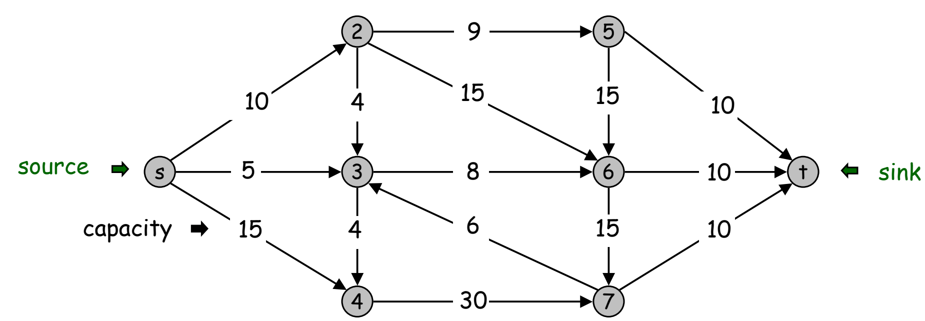
Network: abstraction for material FLOWING through the edges.

- Directed graph.
- Capacities on edges.
- Source node s, sink node t.

same input as min cut problem

Max flow problem. Assign flow to edges so as to:

- Equalize inflow and outflow at every intermediate vertex.
- Maximize flow sent from s to t.

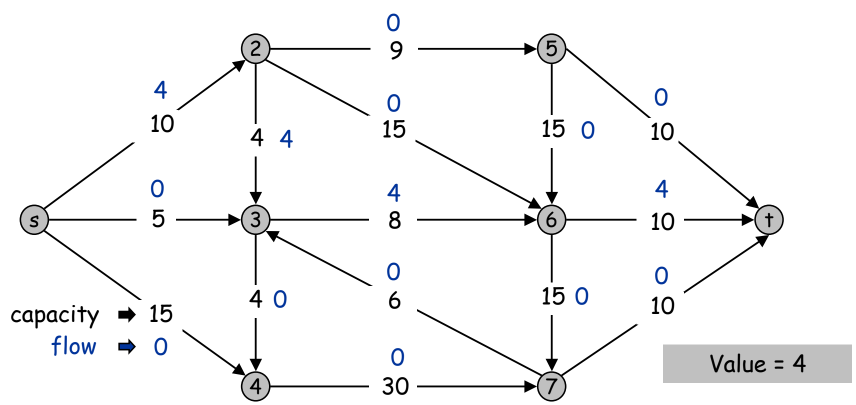


Flows

A flow f is an assignment of weights to edges so that:

- Capacity: $0 \leq f(e) \leq u(e)$.
- Flow conservation: flow leaving v = flow entering v.

except at s or t

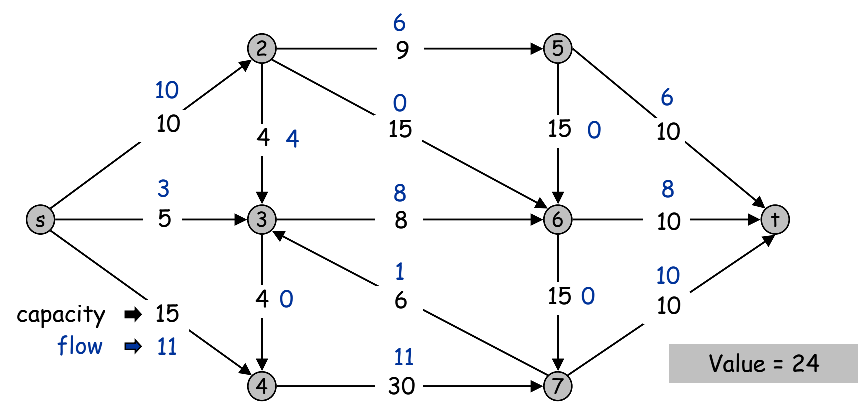


Flows

A flow f is an assignment of weights to edges so that:

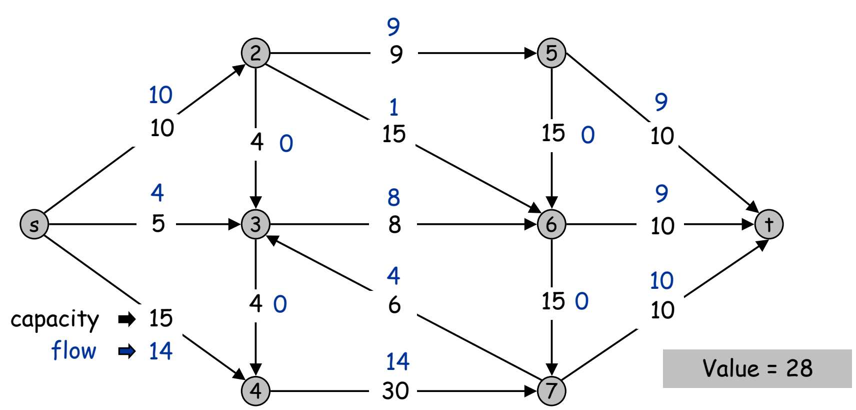
- Capacity: $0 \leq f(e) \leq u(e)$.
- Flow conservation: flow leaving v = flow entering v.

except at s or t



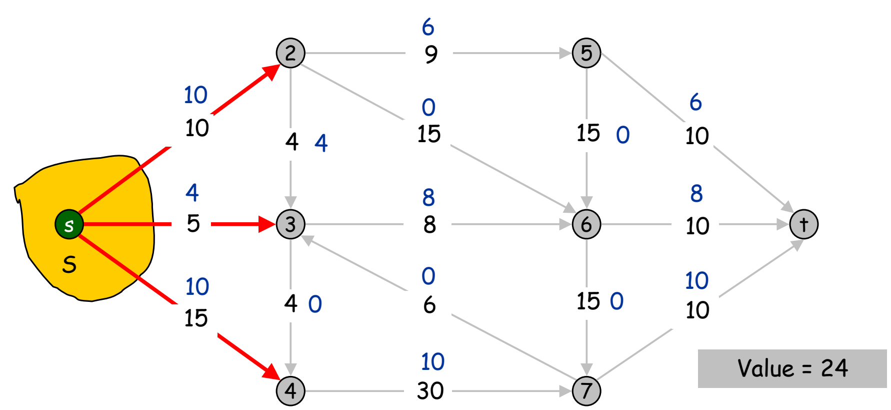
Maximum Flow Problem

Max flow problem: find flow that maximizes net flow into sink.



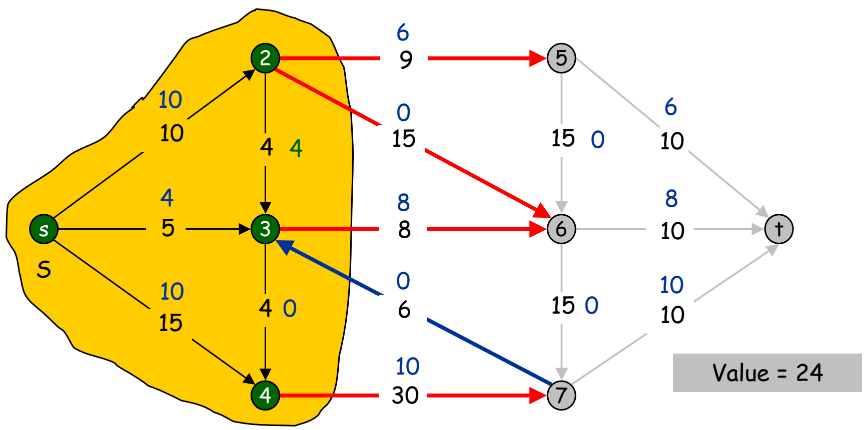
Flows and Cuts

Observation 1. Let f be a flow, and let (S, T) be any s-t cut. Then, the net flow sent across the cut is equal to the amount reaching t.



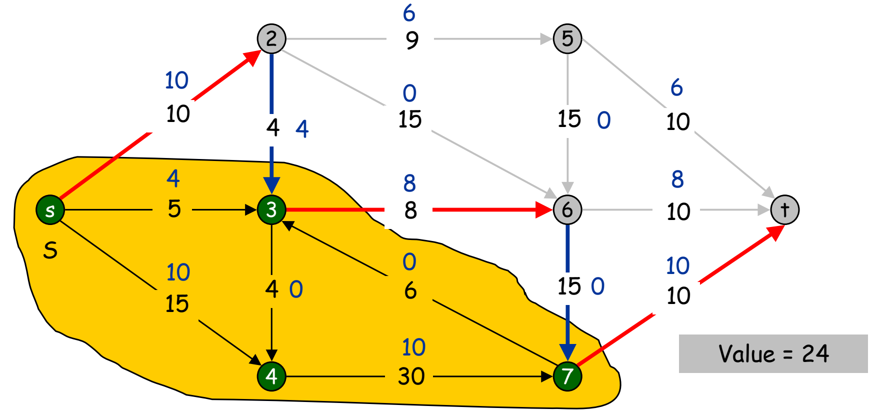
Flows and Cuts

Observation 1. Let f be a flow, and let (S, T) be any s-t cut. Then, the net flow sent across the cut is equal to the amount reaching t.



Flows and Cuts

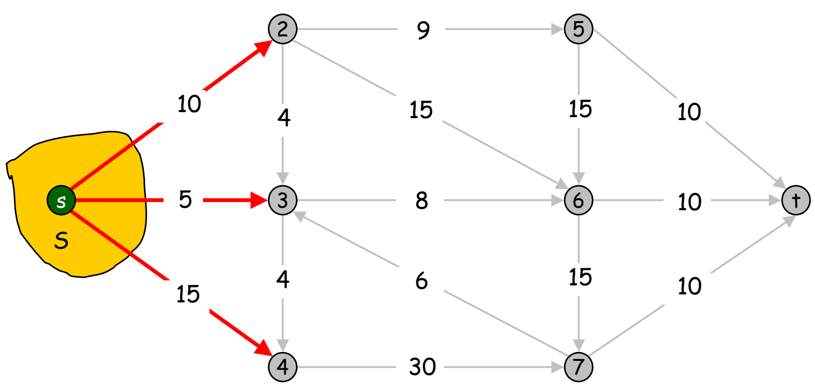
Observation 1. Let f be a flow, and let (S, T) be any s-t cut. Then, the net flow sent across the cut is equal to the amount reaching t.



Flows and Cuts

Observation 2. Let f be a flow, and let (S, T) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

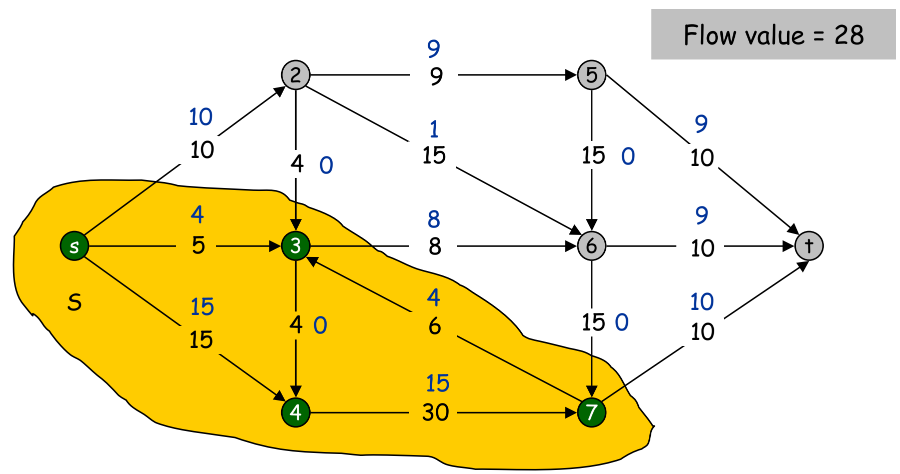
Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30



Max Flow and Min Cut

Observation 3. Let f be a flow, and let (S, T) be an s-t cut whose capacity equals the value of f. Then f is a max flow and (S, T) is a min cut.

Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28

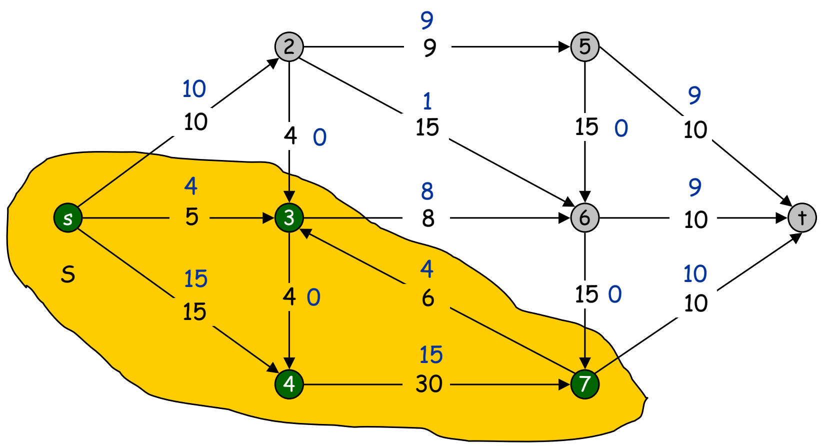


Max-Flow Min-Cut Theorem

Max-flow min-cut theorem. (Ford-Fulkerson, 1956): In any network, the value of max flow equals capacity of min cut.

- Proof IOU: we find flow and cut such that Observation 3 applies.

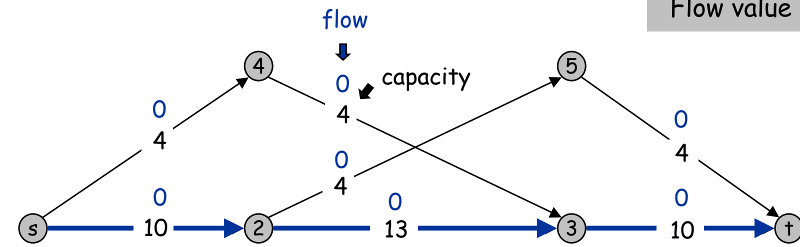
Min cut capacity = 28 ⇔ Max flow value = 28



Towards an Algorithm

Find s-t path where each arc has f(e) < u(e) and "augment" flow along it.

Flow value = 0

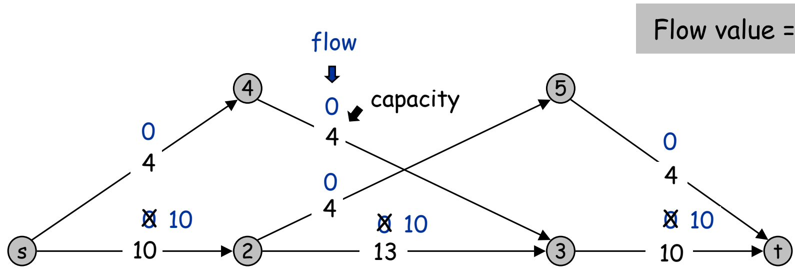


Towards an Algorithm

Find s-t path where each arc has f(e) < u(e) and "augment" flow along it.

- Greedy algorithm: repeat until you get stuck.

Flow value = 10



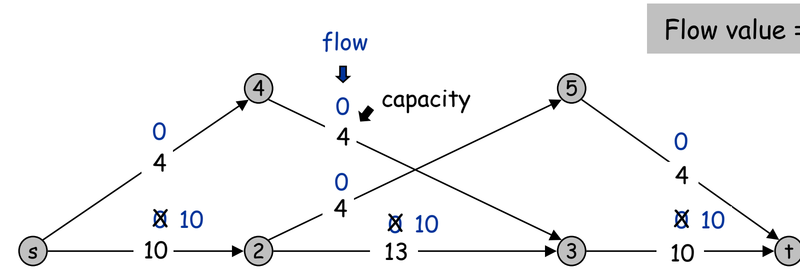
Bottleneck capacity of path = 10

Towards an Algorithm

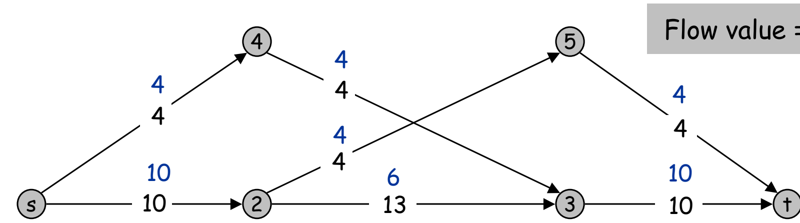
Find s-t path where each arc has f(e) < u(e) and "augment" flow along it.

- Greedy algorithm: repeat until you get stuck.
- Fails: need to be able to "backtrack."

Flow value = 10

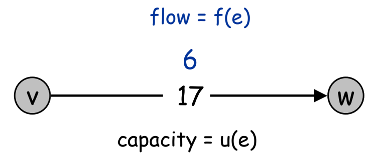


Flow value = 14



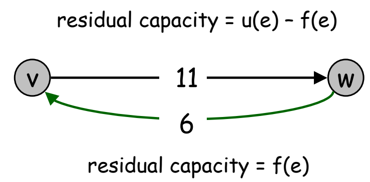
Residual Graph

Original graph.
- Flow f(e).
- Edge e = v-w



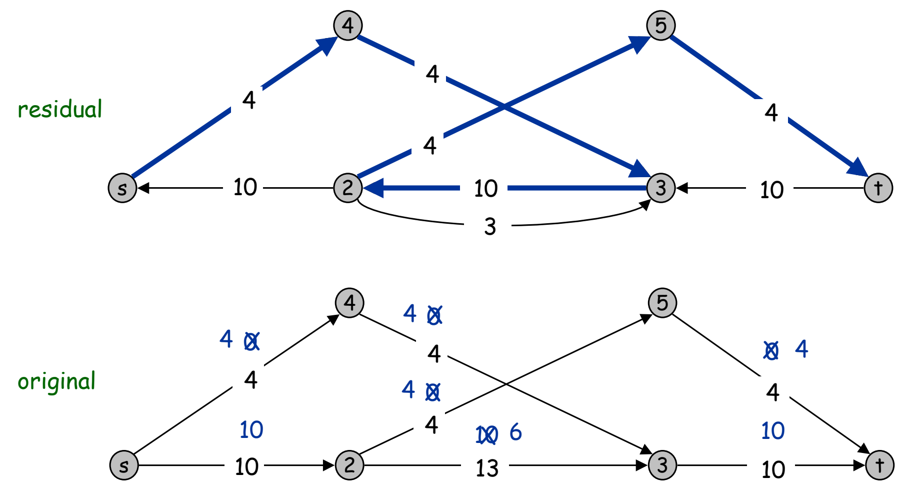
Residual edge.
- Edge e = v-w or w-v.
- "Undo" flow sent.

Residual graph.
- All the edges that have strictly positive residual capacity.



Augmenting Paths

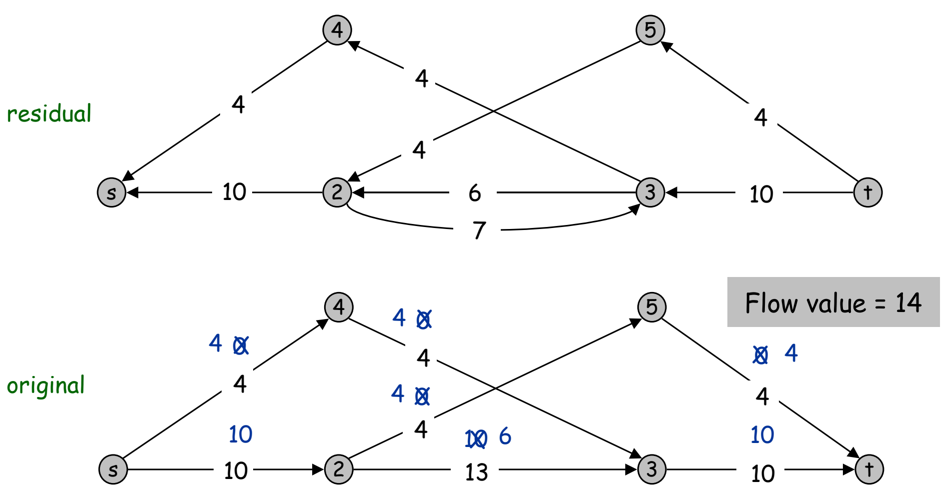
Augmenting path = path in residual graph.
- Increase flow along forward edges.
- Decrease flow along backward edges.



Augmenting Paths

Observation 4. If augmenting path, then not yet a max flow.

Q. If no augmenting path, is it a max flow?



Ford-Fulkerson Augmenting Path Algorithm

Ford-Fulkerson algorithm. Generic method for solving max flow.

```
while (there exists an augmenting path) {
   Find augmenting path P
   Compute bottleneck capacity of P
   Augment flow along P
}
```

Questions.
- Does this lead to a maximum flow? yes
- How do we find an augmenting path? s-t path in residual graph
- How many augmenting paths does it take?
- How much effort do we spending finding a path?

Max-Flow Min-Cut Theorem

Augmenting path theorem. A flow f is a max flow if and only if there are no augmenting paths.

> Max-flow min-cut theorem. The value of the max flow is equal to the capacity of the min cut.

We prove both simultaneously by showing the following are equivalent:

(i) f is a max flow.
(ii) There is no augmenting path relative to f.
(iii) There exists a cut whose capacity equals the value of f.

(i) $\Rightarrow$ (ii) equivalent to not (ii) $\Rightarrow$ not (i), which was Observation 4
(ii) $\Rightarrow$ (iii) next slide
(iii) $\Rightarrow$ (i) this was Observation 3

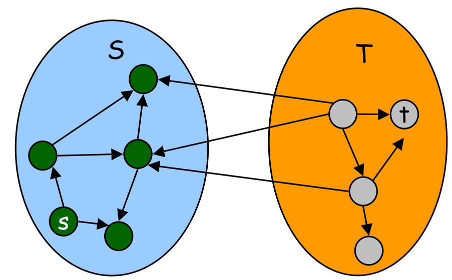
Proof of Max-Flow Min-Cut Theorem

(ii) $\Rightarrow$ (iii). If there is no augmenting path relative to f, then there exists a cut whose capacity equals the value of f.

Proof.

- Let f be a flow with no augmenting paths.
- Let S be set of vertices reachable from s in residual graph.
  - S contains s; since no augmenting paths, S does not contain t
  - all edges e leaving S in original network have f(e) = u(e)
  - all edges e entering S in original network have f(e) = 0

$$\begin{aligned} |f| &= \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) \\ &= \sum_{e \text{ out of } S} u(e) \\ &= \mathit{capacity}(S, T) \end{aligned}$$



residual network

Max Flow Network Implementation

Edge in original graph may correspond to 1 or 2 residual edges.

- May need to traverse edge e = v-w in forward or reverse direction.
- Flow = f(e), capacity = u(e).
- Insert two copies of each edge, one in adjacency list of v and one in w.

```
public class Edge {
   private int v, w;   // from, to
   private int cap;    // capacity from v to w
   private int flow;   // flow from v to w

   public Edge(int v, int w, int cap) { ... }

   public int cap()  { return cap;  }
   public int flow() { return flow; }

   public boolean from(int v) { return this.v == v;            }
   public int other(int v)    { return from(v) ? this.w : this.v;   }
   public int capRto(int v)   { return from(v) ? flow : cap - flow; }
   public void addflowRto(int v, int d) { flow += from(v) ? -d : d; }
}
```


Ford-Fulkerson Algorithm: Implementation

Ford-Fulkerson main loop.

```
// while there exists an augmenting path, use it
while (augpath()) {

   // compute bottleneck capacity
   int bottle = INFINITY;
   for (int v = t; v != s; v = ST(v))
      bottle = Math.min(bottle, pred[v].capRto(v));

   // augment flow
   for (int v = t; v != s; v = ST(v))
      pred[v].addflowRto(v, bottle);

   // keep track of total flow sent from s to t
   value += bottle;
}
```

Ford-Fulkerson Algorithm: Analysis

Assumption: all capacities are integers between 1 and U.

Invariant: every flow value and every residual capacities remain an integer throughout the algorithm.

Theorem: the algorithm terminates in at most $| f^* | \leq V U$ iterations.

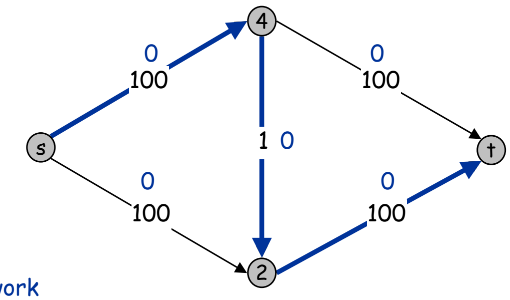
not polynomial in input size!

Corollary: if U = 1, then algorithm runs in $\leq$ V iterations.

Integrality theorem: if all arc capacities are integers, then there exists a max flow f for which every flow value is an integer.

Choosing Good Augmenting Paths

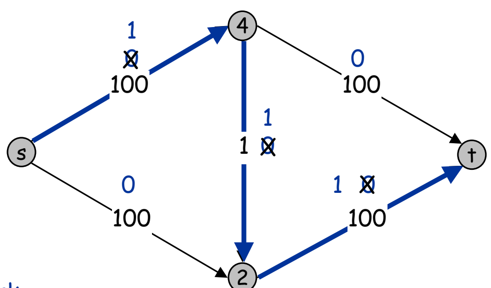
Use care when selecting augmenting paths.



Original Network

Choosing Good Augmenting Paths

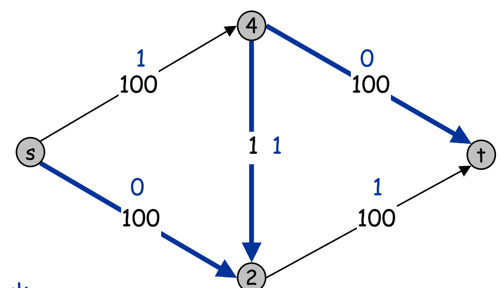
Use care when selecting augmenting paths.



Original Network

Choosing Good Augmenting Paths

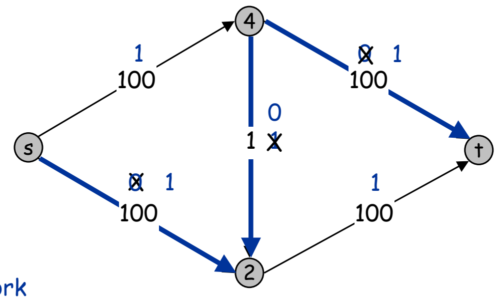
Use care when selecting augmenting paths.



Original Network

Choosing Good Augmenting Paths

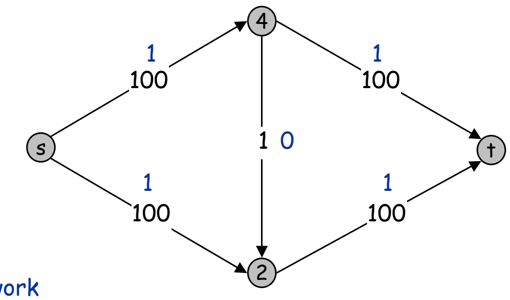
Use care when selecting augmenting paths.



Original Network

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.



Original Network

200 iterations possible!

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- Optimal choices for real world problems ???

Design goal is to choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting path with: Edmonds-Karp (1972)

- Fewest number of arcs. (shortest path)
- Max bottleneck capacity. (fattest path)

Shortest Augmenting Path

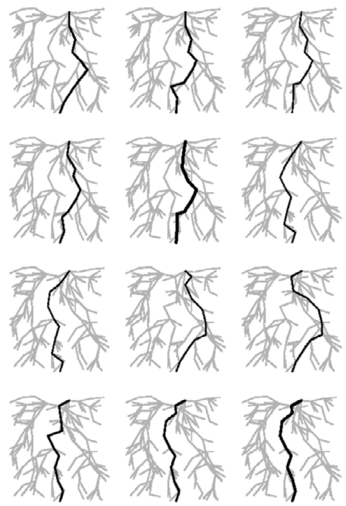
Shortest augmenting path.

- Easy to implement with BFS.
- Finds augmenting path with fewest number of arcs.

```
while (!q.isEmpty()) {
   int v = q.dequeue();
   IntIterator i = G.neighbors(v);
   while(i.hasNext()) {
      Edge e = i.next();
      int w = e.other(v);
      if (e.capRto(w) > 0) {  // is v-w a residual edge?
         if (wt[w] > wt[v] + 1) {
            wt[w] = wt[v] + 1;
            pred[w] = e;      // keep track of shortest path
            q.enqueue(w);
         }
      }
   }
}
return (wt[t] < INFINITY);    // is there an augmenting path?
```

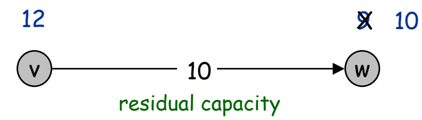
Shortest Augmenting Path Analysis

Length of shortest augmenting path increases monotonically.
- Strictly increases after at most E augmentations.
- At most E V total augmenting paths.
- $O(E^2 V)$ running time.



Fattest Augmenting Path

Fattest augmenting path.
- Finds augmenting path whose bottleneck capacity is maximum.
- Delivers most amount of flow to sink.
- Solve using Dijkstra-style (PFS) algorithm.




```
if (wt[w] < Math.min(wt[v], e.capRto(w)) {
    wt[w] = Math.min(wt[v], e.capRto(w));
    pred[w] = v;
}
```

Finding a fattest path. O(E log V) per augmentation with binary heap.
Fact. O(E log U) augmentations if capacities are between 1 and U.

Choosing an Augmenting Path

Choosing an augmenting path.
- Any path will do $\Rightarrow$ wide latitude in implementing Ford-Fulkerson.
- Generic priority first search.
- Some choices lead to good worst-case performance.
  - shortest augmenting path
  - fattest augmenting path
  - variation on a theme: PFS
- Average case not well understood.

Research challenges.
- Practice: solve max flow problems on real networks in linear time.
- Theory: prove it for worst-case networks.

History of Worst-Case Running Times

Year	Discoverer	Method	Asymptotic Time
1951	Dantzig	Simplex	$E V^2 U$ †
1955	Ford, Fulkerson	Augmenting path	$E V U$ †
1970	Edmonds-Karp	Shortest path	$E^2 V$
1970	Edmonds-Karp	Max capacity	$E \log U (E + V \log V)$ †
1970	Dinitz	Improved shortest path	$E V^2$
1972	Edmonds-Karp, Dinitz	Capacity scaling	$E^2 \log U$ †
1973	Dinitz-Gabow	Improved capacity scaling	$E V \log U$ †
1974	Karzanov	Preflow-push	V^3
1983	Sleator-Tarjan	Dynamic trees	$E V \log V$
1986	Goldberg-Tarjan	FIFO preflow-push	$E V \log (V^2 / E)$
. . .	. . .	. . .	. . .
1997	Goldberg-Rao	Length function	$E^{3/2} \log (V^2 / E) \log U$ † $EV^{2/3} \log (V^2 / E) \log U$ †

† Arc capacities are between 1 and U.

An Application

Jon placement.
- Companies make job offers.
- Students have job choices.

Can we fill every job?

Can we employ every student?

Alice-Adobe
Bob-Yahoo
Carol-HP
Dave-Apple
Eliza-IBM
Frank-Sun

Alice
- **Adobe**
- **Apple**
- **HP**

Bob
- **Adobe**
- **Apple**
- **Yahoo**

Carol
- **HP**
- **IBM**
- **Sun**

Dave
- **Adobe**
- **Apple**

Eliza
- **IBM**
- **Sun**
- **Yahoo**

Frank
- **HP**
- **Sun**
- **Yahoo**

Adobe
- Alice
- Bob
- Dave

Apple
- Alice
- Bob
- Dave

HP
- Alice
- Carol
- Frank

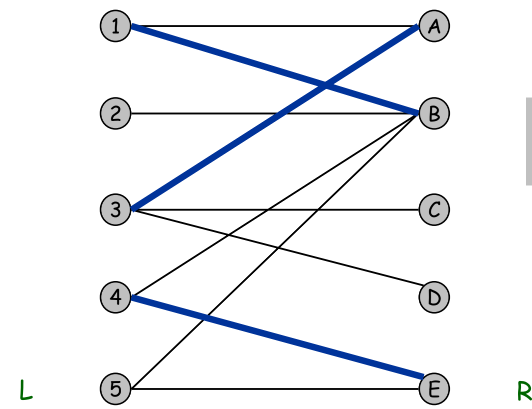
IBM
- Carol
- Eliza

Sun
- Carol
- Eliza
- Frank

Yahoo
- Bob
- Eliza
- Frank

Bipartite Matching

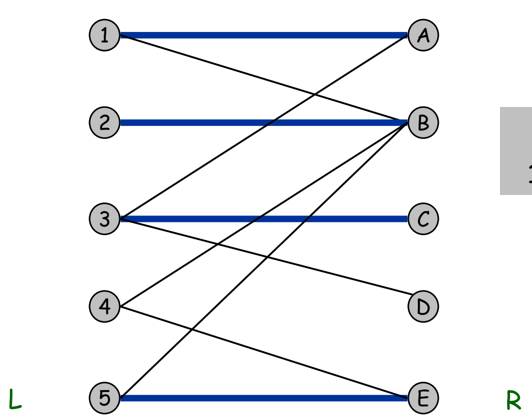
Bipartite matching.
- Input: undirected and bipartite graph G.
- Set of edges M is a matching if each vertex appears at most once.
- Max matching: find a max cardinality matching.



Matching M
1-B, 3-A, 4-E

Bipartite Matching

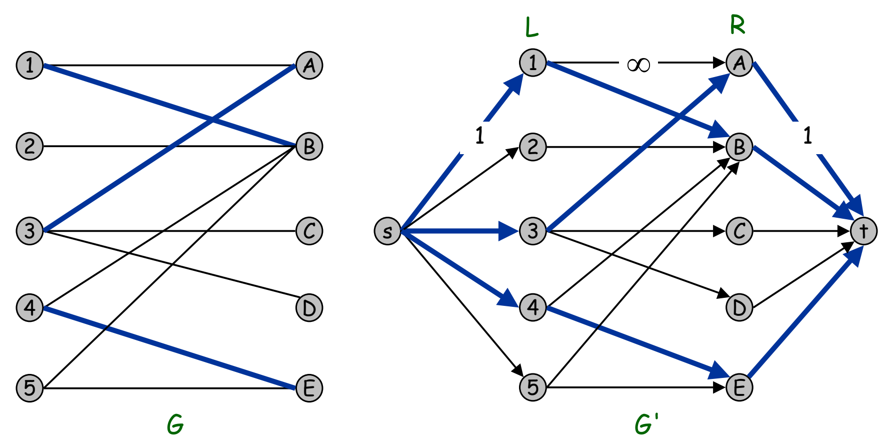
Bipartite matching.
- Input: undirected and bipartite graph G.
- Set of edges M is a matching if each vertex appears at most once.
- Max matching: find a max cardinality matching.



Matching M
1-A, 2-B, 3-C, 4-D

Bipartite Matching

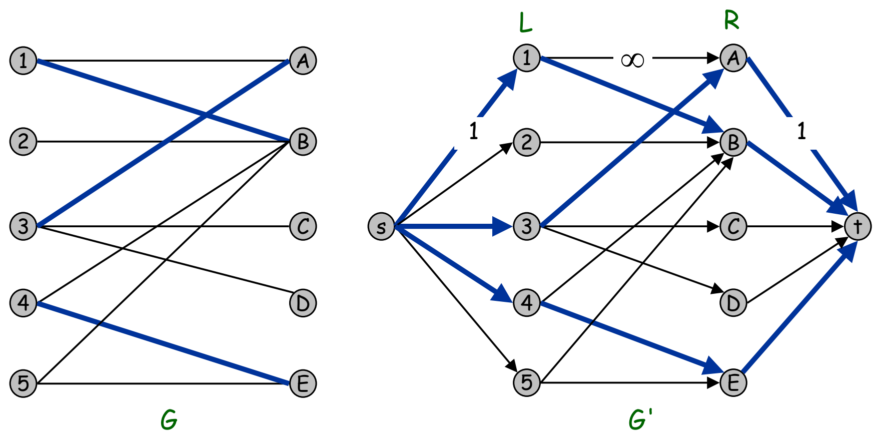
Reduces to max flow.
- Create a directed graph G'.
- Direct all arcs from L to R, and give infinite (or unit) capacity.
- Add source s, and unit capacity arcs from s to each node in L.
- Add sink t, and unit capacity arcs from each node in R to t.



Bipartite Matching: Proof of Correctness

Claim. Matching in G of cardinality k induces flow in G' of value k.

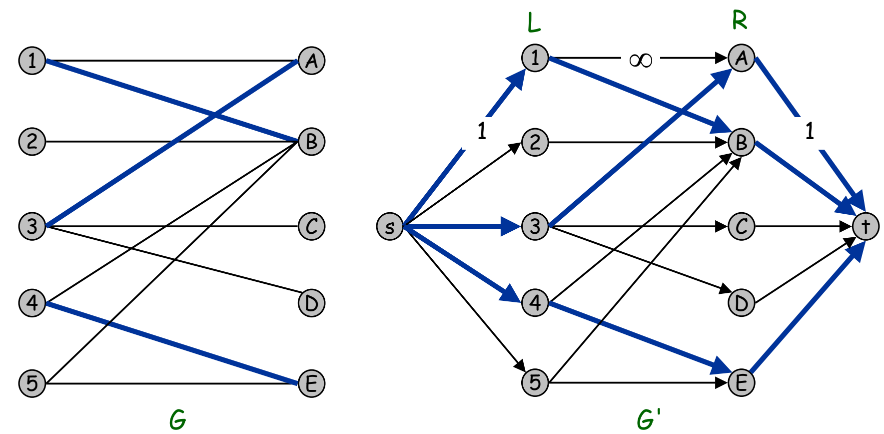
- Given matching M = { 1-B, 3-A, 4-E } of cardinality 3.
- Consider flow f that sends 1 unit along each of 3 paths:
  s-1-B-t s-3-A-t s-4-E-t.
- f is a flow, and has cardinality 3.



Bipartite Matching: Proof of Correctness

Claim. Flow f of value k in G' induces matching of cardinality k in G.

- By integrality theorem, there exists 0/1 valued flow f of value k.
- Consider M = set of edges from L to R with f(e) = 1.
  - each node in L and R incident to at most one edge in M
  - |M| = k



Reduction

Reduction.

- Given an instance of bipartite matching.
- Transform it to a max flow problem.
- Solve max flow problem.
- Transform max flow solution to bipartite matching solution.

Issues.

- How expensive is transformation? O(E + V)
- Is it better to solve problem directly? $O(E\ V^{1/2})$ bipartite matching

Bottom line: max flow is an extremely rich problem-solving model.

- Many important practical problems reduce to max flow.
- We know good algorithms for solving max flow problems.