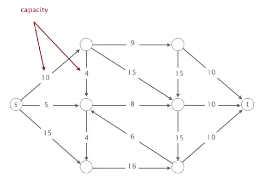
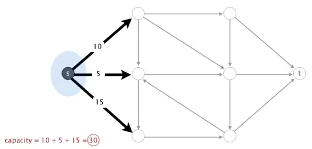
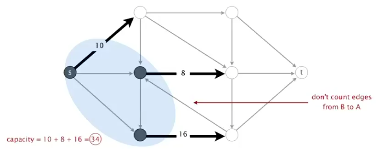
Maximum Flow and Minimum Cut

Input: An edge-weighted digraph (each edge has a positive capacity... weight), source vertex s, and target vertex t.

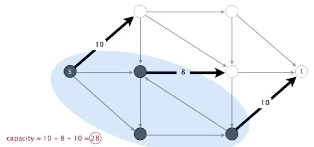


**An *st*-cut (**cut) is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B

The **capacity** is the sum of the capacities of the edges from A to B. Example 1: 

Example 2 (don't count from *t* to *s*; only s - > t): 

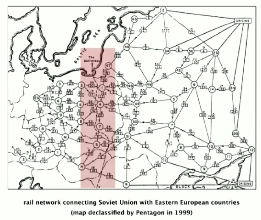
Example 3:



**The mincut problem: Find a cut of the minimum capacity.**

Mincut applications

"Free world" goal: cut supplies if cold war turns in to real war



Potential mincut application (2010s):

Government in-power's goal: cut off communication to a set of people

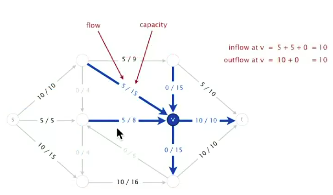


Maxflow problem

Input: An edge-weighted digraph (each edge has a positive capacity... weight), source vertex s, and target vertex t.

And *st-*flow (flow) is an assignment of values to the edges such that:

* Capacity constraint: 0 <= edges's flow <= edge's capacity
* Local equilibrium: inflow = outflow at every vertex  (except s and t). Except at:
  + source (everything leaves the source)
  + and the target (everything goes to the target)

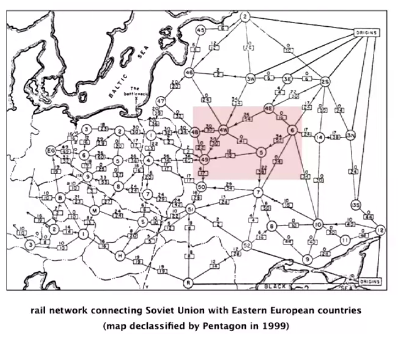


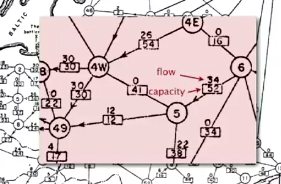
The value of a flow is the inflow at t.

**Maxflow st-flow (maxflow) problem: find a flow of maximum value.**

Maxflow application (1950s)

Soviet Union goal: maximize flow of supplies to Eastern Europe



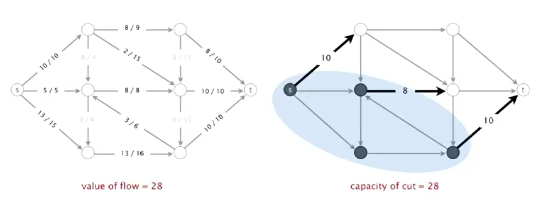


Potential maxflow application (2010s)

"Free world" goal: Maximize flow of information to specified set of people



Summary



Input: a weighted digraph, source vertex s, and target vertex t

Mincut problem: find the cut of minimum capacity

Maxflow problem: find a flow of maximum value

Interesting: these problems are dual (pretty much the same )

Running time analysis:

* How to compute a mincut? East
* How to find an augmenting path? BFS works well
* If FF terminates, does it always compute a maxflow? Yes
* Does FF always terminate? If so, after how many augmentations?
  + Yes, provided edge capacities are integers   
    (or augmenting paths are chosen carefully)
  + Number of augmentations requires clever analysis

Important special case: edge capacities are integers between 1 and U (some large maximum)

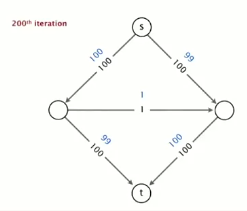
Invariant: the flow is integer-valued throughout Ford-Fulkerson  
Proof (by induction):

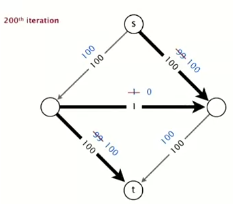
* Bottleneck capacity is an integer
* Flow on an edge increases/decreases by bottleneck capacity

Proposition: Number of augmentations <= the value of the maxflow  
Proof: Each augmentation increases the value by at least 1

Integrality theorem: There exists an integer-valued maxflow (and FF finds one).  
Proof: Ford-Fulkerson terminates and maxflow that it finds is integer-valued

Bad news: Even when edge capacities are integers, number of augmenting paths could be equal to the value (can be exponential in input size) of the maxflow. The below graph would take 200 iterations with base FF algorithm (see 200th iteration).

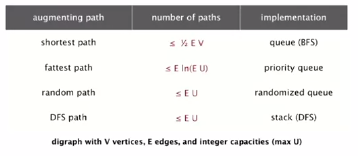




Good news: This case is easily avoided, using the shortest/fattest path.

Performance

FF performance depends on algorithm used to choose augmenting paths.



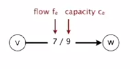
Difference between shortest and fattest paths:

* Shortest= fewest number of edges
* Fattest= maximum bottleneck capacity



Java implementation

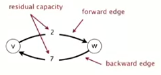
Flow edge data type: Associate flow fc and capacity ce with edge e = v-> w.



Flow network data type: need to process edge e – v -> w in either direction:  
 Include e in both v and w’s adjacency lists

Residual capacity:

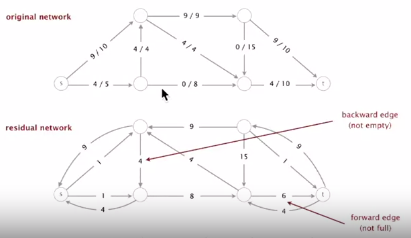
* Forward edge: residual capacity = ce - fe
* Backward edge: residual capacity = fe



Augment flow:

* Forward edge: add (delta) Δ
* Backward edge: subtract (delta) Δ

Residual network: a useful view of a flow network.

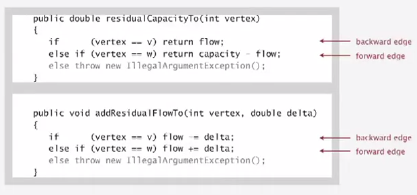
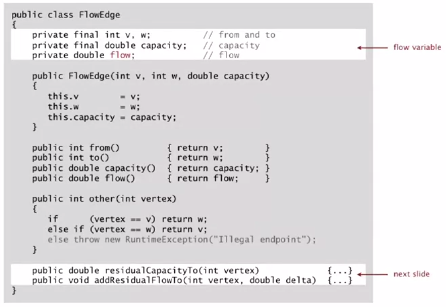


Key point: augmenting path in original network is equivalent to directed path in residual network.

Flow edge API

Public class API  
FlowEdge(int v, int w, double capacity) : create a flow edge v->w  
int from() : vertex this edge points from  
int to() : vertex this edge points to  
int other(int v) : other endpoint  
double capacity() : capacity of this edge  
double flow() : flow in this edge  
double residualCapacityTo(int v) : residual capacity toward v  
void addResidualFlowTo(int v, double delta) : add delta flow toward v  
String toString() : string representation

Implementation

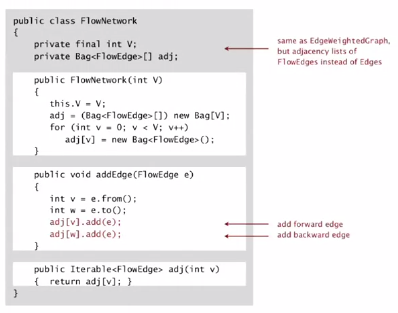


Flow network API

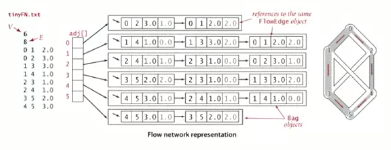
Public class FlowNetwork  
FlowNetwork(int V) : create an empty flow network with V vertices  
FlowNetwork(In in) : construct flow network input stream  
void addEdge(FlowEdge e) : add flow edge e to this flow network  
Iterable<FlowEdge> adj(int v) : forward and backward edges incident to v  
Iterable<FlowEdge> edges() : all edges to this flow network  
int V() : number of vertices  
int E() : number of edges  
String toString() : string representation

Conventions: allow self-loops and parallel edges

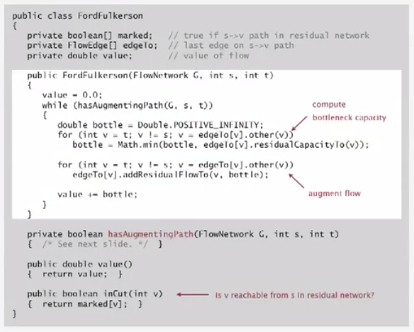
Flow network Implementation



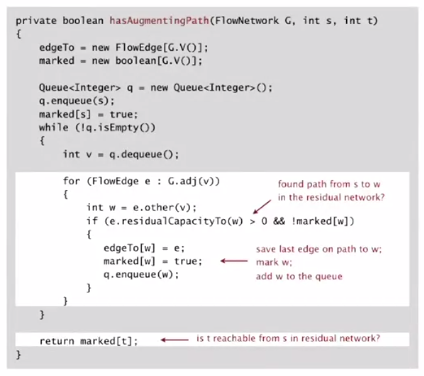
Maintain vertex-indexed array of FlowEdge lists (use Bag abstraction)



Ford-Fulkerson implementation



hasAugmentingPath implementation



Maxflow and mincut applications

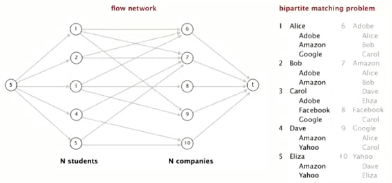
Very widely applicable problem solving model

Some applications:

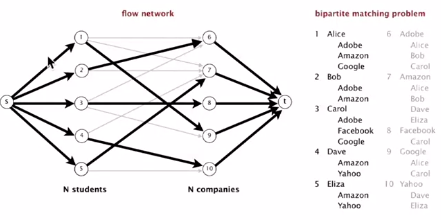
* Data-mining
* Bipartite matching
* Network reliability
* Baseball elimination
* Image segmentation
* Sensor placement for homeland security

Network flow formulation of **bipartite matching** (matching jobs to applicants)

* Create s, t, one vertex for each student, and one vertex for each job
* Add edge from s to each student (capacity 1)
* Add edge from each job to t (capacity 1)
* Add edge from each student to each job offered (infinite capacity)

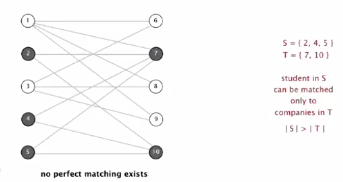


(Cont’d on next page)



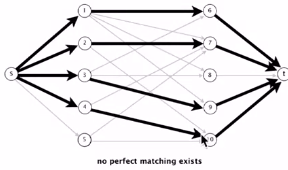
Mincut can also tell us both:

* No perfect match
* When no perfect match, explain why.
* Viz:



Mincut: Consider mincut (A, B)

* Let S = students on s side of cut
* Let T = companies on s side of cut
* Fact: | S | > | T |; students in S can be matched only to companies in T



When no perfect matching, mincut explains why

**Baseball elimination**

Which teams have a chance of finishing the season with the most wins?



In this above case:

Montreal is mathematically eliminated:

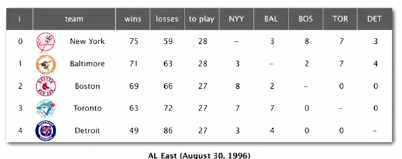
* Montreal finishes with <= 80 wins (only 3 games left)
* Atlanta already has 83 wins

Philadelphia is mathematically eliminated:

* Philadelphia finishes with <= 83 wins (only 3 games left)
* Either NY or ATL will finish with >= 84 wins

Observation: Answer depends not only on how many games already won and left to play, but on whom they’re against.

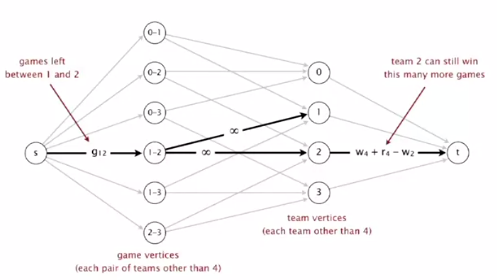
More complicated situation:



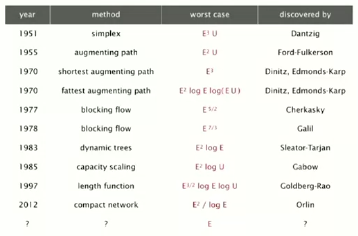
Detroit is mathematically eliminated:

* Detroit finishes with <= 76 wins
* Wins for R = { NYY, BAL, BOS, TOR } = 278
* Remaining games among { NYY, BAL, BOS, TOR } = 3 + 8 + 7 + 2 + 7 = 27
* Average team in R wins 305/4 = 76.25 games

Intuition: Remaining games flow from s to t.



Maximum flow algorithms: theory  
 *-> Holy grail for theoretical computer scientists*



Max flow algorithms: practice

Warning: worst-case order-of-growth is generally not useful for predicting or comparing maxflow algorithm performance in practice.

Best in practice: push-relabel method with gap relabeling: E3/2

**Summary Overall**

**Mincut problem**: find an st-cut of minimum capacity

**Maxflow problem**: find an st-flow of maximum value

**Duality**: Value of the maxflow = capacity of mincut

Proven successful approaches:

* **Ford-Fulkerson** (various augmenting-path strategies)
* **Preflow-push** (various versions)

Open research challenges:

* *Practice*: solve real-world maxflow/mincut problems in linear time
* *Theory*: prove it for worst-case inputs
* Still much to be learned ☺