MAX FLOW

CS302, Spring 2013 David Kauchak

Admin

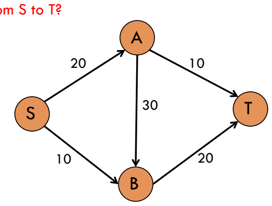
Max Power

http://www.youtube.com/watch?v=vDA-SAwz2VQ

Student networking

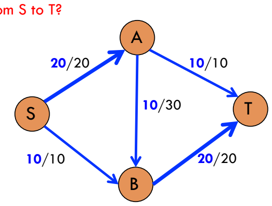
You decide to create your own campus network:

- You get three of your friends and string some network cables
- Because of capacity (due to cable type, distance, computer, etc) you can only send a certain amount of data to each person
- If edges denote capacity, what is the maximum throughput you can you send from S to T?



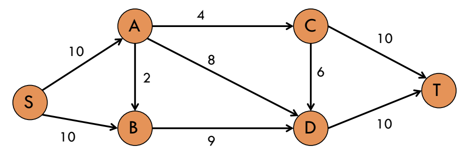
Student networking

- You decide to create your own campus network:
  - You get three of your friends and string some network cables
  - Because of capacity (due to cable type, distance, computer, etc) you can only send a certain amount of data to each person
  - If edges denote capacity, what is the maximum throughput you can you send from S to T?



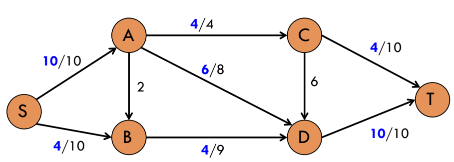
30 units

Another flow problem



How much water flow can we continually send from s to t?

Another flow problem

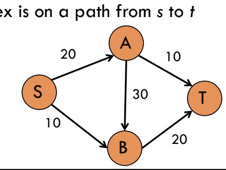


14 units

Flow graph/networks

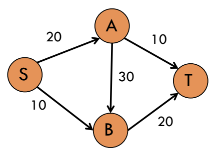
Flow network

- directed, weighted graph (V, E)
- positive edge weights indicating the "capacity" (generally, assume integers)
- contains a single source $s \in V$ with no incoming edges
- contains a single sink/target $t \in V$ with no outgoing edges
- every vertex is on a path from s to t



Flow

What are the constraints on flow in a network?

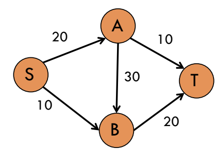


Flow

in-flow = out-flow for every vertex (except s, t)

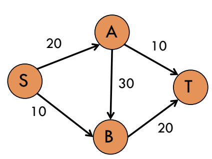
flow along an edge cannot exceed the edge capacity

flows are positive



Max flow problem

Given a flow network: *what is the maximum flow we can send from s to t that meet the flow constraints?*



Applications?

network flow
- water, electricity, sewage, cellular…
- traffic/transportation capacity

bipartite matching

sports elimination

…

Max flow origins

Rail networks of the Soviet Union in the 1950's

The US wanted to know how quickly the Soviet Union could get supplies through its rail network to its satellite states in Eastern Europe.

In addition, the US wanted to know which rails it could destroy most easily to cut off the satellite states from the rest of the Soviet Union.

These two problems are closely related, and that solving the **max flow problem** also solves the **min cut problem** of figuring out the cheapest way to cut off the Soviet Union from its satellites.

Source: Ibackstrom, The Importance of Algorithms, at www.topcoder.com

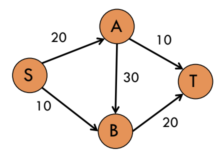
Algorithm ideas?

graph algorithm?
- BFS, DFS, shortest paths…
- MST

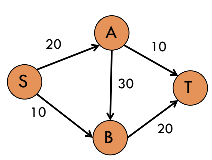
divide and conquer?

greedy?

dynamic programming?

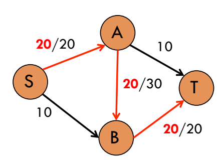


Algorithm idea



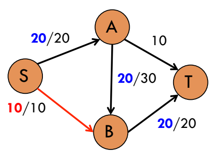
Algorithm idea

send some flow down a path



Algorithm idea

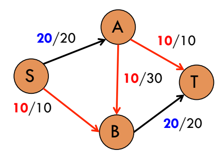
send some flow down a path



Now what?

Algorithm idea

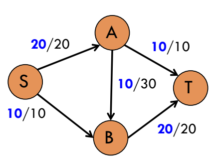
reroute some of the flow



Total flow?

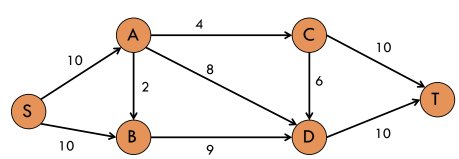
Algorithm idea

reroute some of the flow



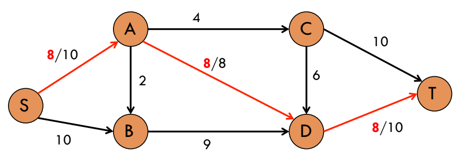
30

Algorithm idea



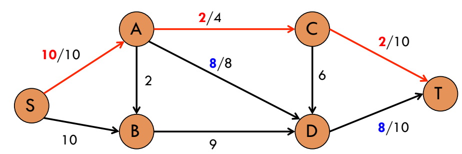
Algorithm idea

send some flow down a path



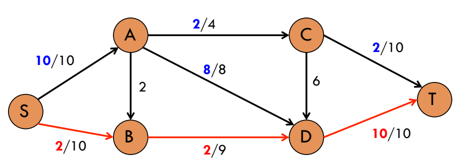
Algorithm idea

send some flow down a path



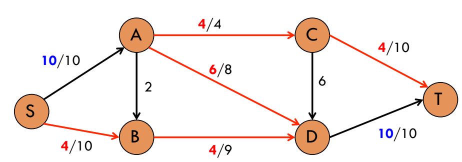
Algorithm idea

send some flow down a path

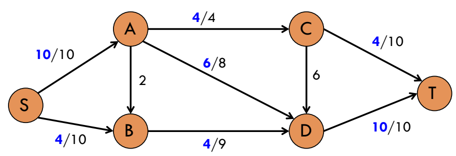


Algorithm idea

reroute some of the flow



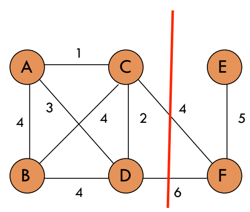
Algorithm idea



Are we done?
Is this the best we can do?

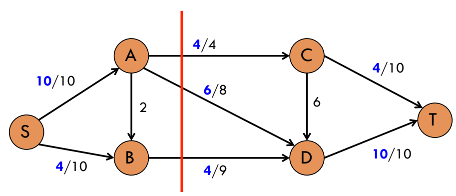
Cuts

A cut is a partitioning of the vertices into two sets A and B = V-A



Flow across cuts

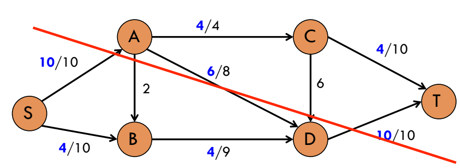
In flow graphs, we're interested in cuts that separate s from t, that is s ∈ A and t ∈ B



Flow across cuts

The flow "across" a cut is the total flow from nodes in A to nodes in B *minus* the total from from B to A

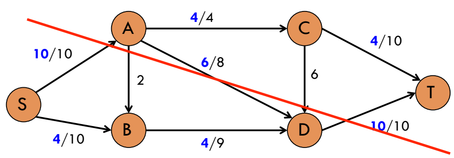
What is the flow across this cut?



Flow across cuts

The flow "across" a cut is the total flow from nodes in A to nodes in B *minus* the total from from B to A

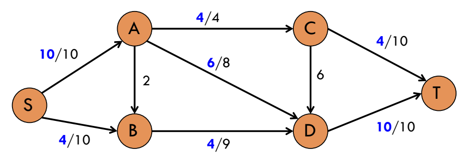
10+10-6 = 14



Flow across cuts

Consider any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

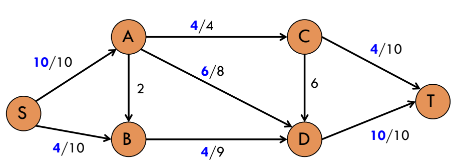
What do we know about the flow across the any such cut?



Flow across cuts

Consider any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

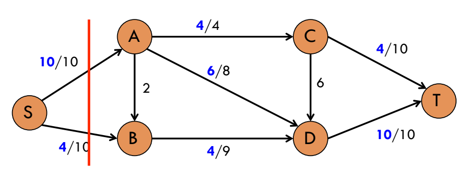
The flow across ANY such cut is the same and is the current flow in the network



Flow across cuts

Consider any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

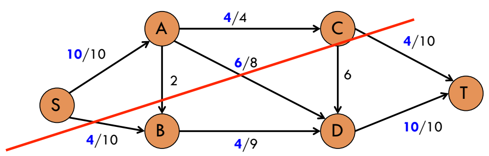
4+10 = 14



Flow across cuts

Consider any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

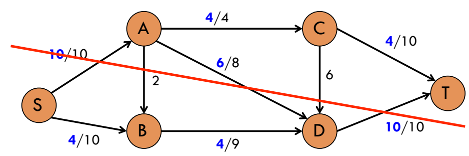
4+6+4 = 14



Flow across cuts

Consider any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

10+10-6 = 14

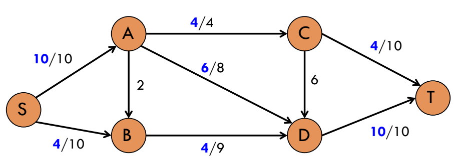


Flow across cuts

Consider any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

The flow across ANY such cut is the same and is the current flow in the network

Why? Can you prove it?



Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Inductively?

- every vertex is on a path from *s* to *t*
- in-flow = out-flow for every vertex (except s, t)
- flow along an edge cannot exceed the edge capacity
- flows are positive

Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Base case: A = s

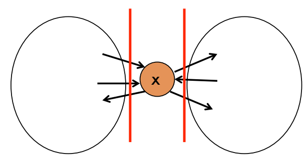
- Flow is total from from s to t: therefore total flow out of s should be the flow
- All flow from s gets to t
  - every vertex is on a path from s to t
  - in-flow = out-flow

Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Inductive case: Consider moving a node *x* from A to B

Is the flow across the different partitions the same?

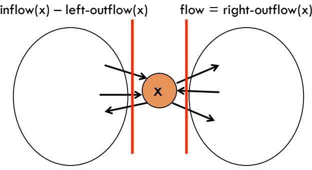


Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Inductive case: Consider moving a node *x* from A to B

flow = left-inflow(x) – left-outflow(x)   flow = right-outflow(x) – right-inflow(x)



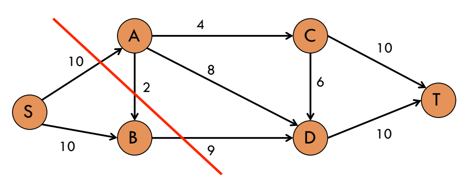
left-inflow(x) + right-inflow(x) = left-outflow(x) + right-outflow(x)   in-flow = out-flow

left-inflow(x) - left-outflow(x) = right-outflow(x) – right-inflow(x)

Capacity of a cut

The "**capacity of a cut**" is the maximum flow that we *could* send from nodes in A to nodes in B (i.e. across the cut)

How do we calculate the capacity?

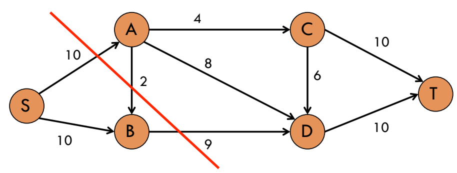


Capacity of a cut

The "**capacity of a cut**" is the maximum flow that we *could* send from nodes in A to nodes in B (i.e. across the cut)

Capacity is the sum of the edges from A to B

Why?



Capacity of a cut

The "**capacity of a cut**" is the maximum flow that we *could* send from nodes in A to nodes in B (i.e. across the cut)

Capacity is the sum of the edges from A to B

- Any more and we would violate the edge capacity constraint
- Any less and it would not be maximal, since we could simply increase the flow

Quick recap

A cut is a partitioning of the vertices into two sets A and B = V-A

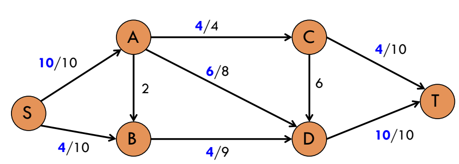
For any cut where s ∈ A and t ∈ B, i.e. the cut partitions the source from the sink

- the flow across any such cut is the same
- the maximum capacity (i.e. flow) across the cut is the sum of the capacities for edges from A to B

Maximum flow

For any cut where s ∈ A and t ∈ B

- the flow across the cut is the same
- the maximum capacity (i.e. flow) across the cut is the sum of the capacities for edges from A to B



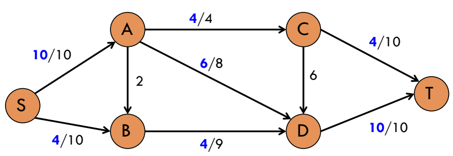
Are we done?

Is this the best we can do?

Maximum flow

For any cut where s ∈ A and t ∈ B

- the flow across the cut is the same
- the maximum capacity (i.e. flow) across the cut is the sum of the capacities for edges from A to B

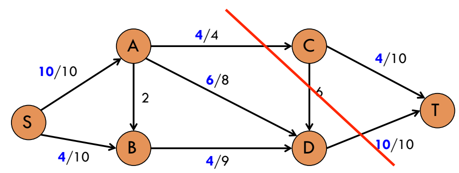


We can do no better than the minimum capacity cut!

Maximum flow

What is the minimum capacity cut for this graph?

Capacity = 10 + 4

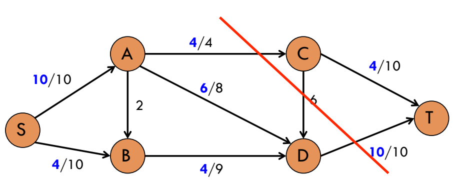


Is this the best we can do?

Maximum flow

What is the minimum capacity cut for this graph?

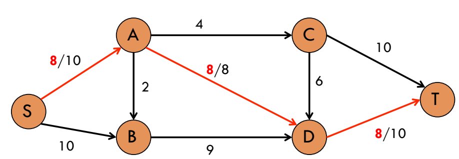
Capacity = 10 + 4



flow = minimum capacity, so we can do no better

Algorithm idea

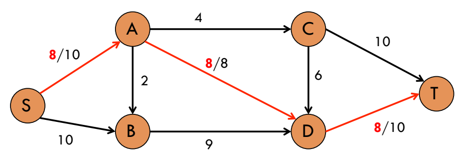
send some flow down a path



How do we determine the path to send flow down?

Algorithm idea

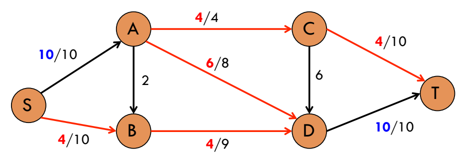
send some flow down a path



Search for a path with
remaining capacity from s to t

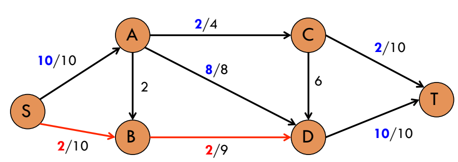
Algorithm idea

reroute some of the flow



How do we handle
"rerouting" flow?

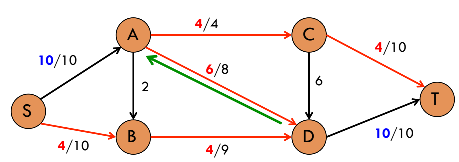
Algorithm idea



During the search, if an edge
has some flow, we consider
"reversing" some of that flow

Algorithm idea

reroute some of the flow



During the search, if an edge
has some flow, we consider
"reversing" some of that flow

The residual graph

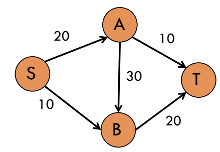
The *residual graph* G_f is constructed from G

For each edge e in the original graph (G):

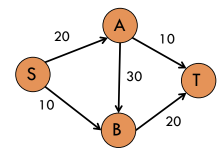
- if *flow(e)* < capacity(e)
  - introduce an edge in G_f with capacity = capacity(e)-flow(e)
  - this represents the remaining flow we can still push
- if flow(e) > 0
  - introduce an edge in G_f in the *opposite direction* with capacity = flow(e)
  - this represents the flow that we can reroute/reverse

Algorithm idea

G



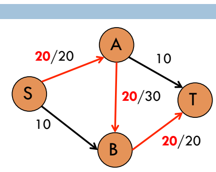
G_f



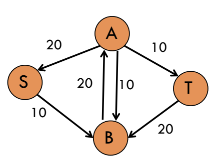
Find a path from s to t in G_f

Algorithm idea

G



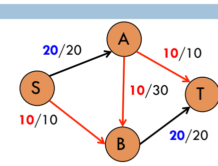
G_f



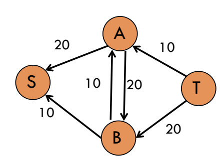
Find a path from s to t in G_f

Algorithm idea

G



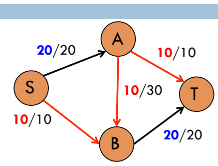
G_f



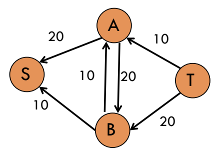
Find a path from s to t in G_f

Algorithm idea

G



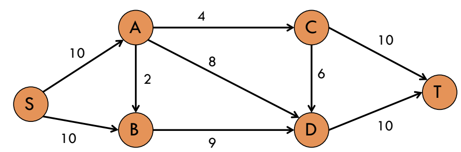
G_f



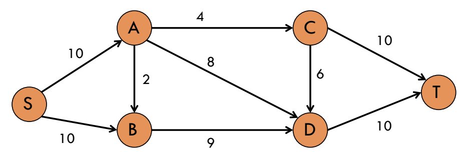
None exist… done!

Algorithm idea

G

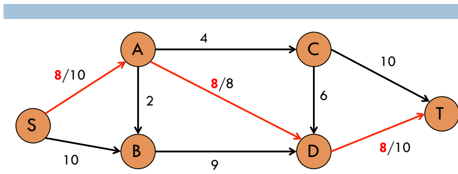


G_f



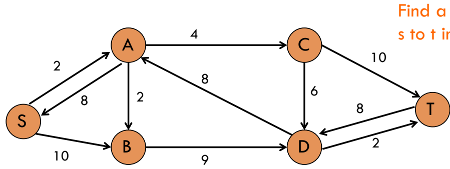
Algorithm idea

G



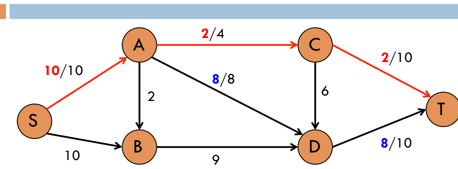
Find a path from s to t in G_f

G_f



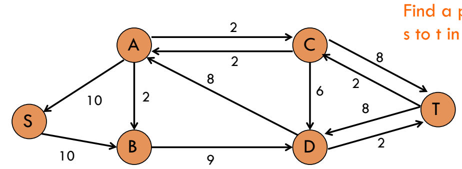
Algorithm idea

G

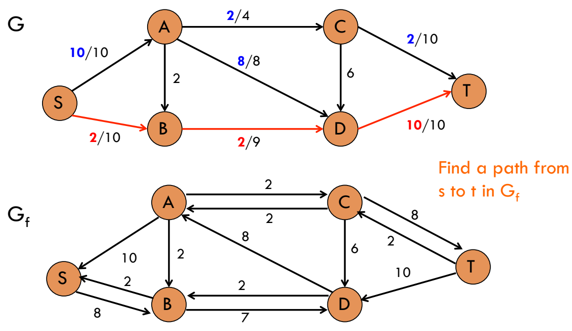


Find a path from s to t in G_f

G_f

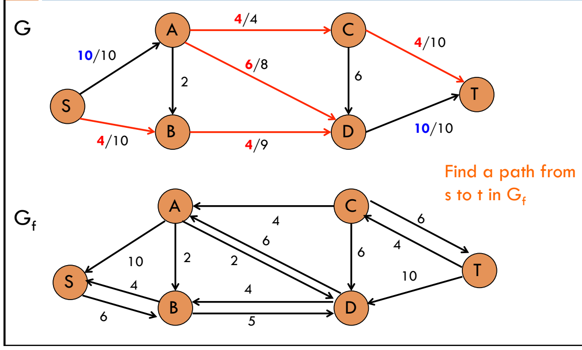


Algorithm idea



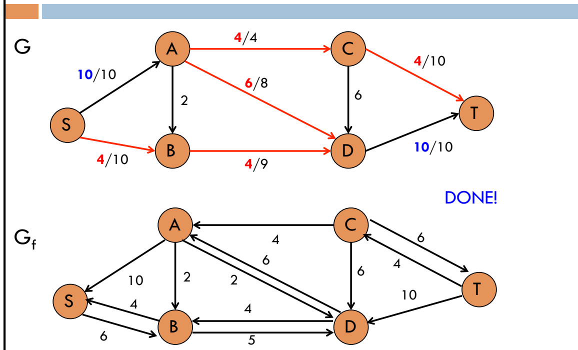
Find a path from s to t in G_f

Algorithm idea



Find a path from s to t in G_f

Algorithm idea



DONE!

Ford-Fulkerson

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along the path as possible
    G_f = residualGraph(G)
  return flow

a simple path contains no repeated vertices

Ford-Fulkerson: is it correct?

Does the function terminate?

Every iteration increases the flow from s to t

- Every path must start with s
- The path has positive flow (or it wouldn't exist)
- The path is a simple path (so it cannot revisit s)
- conservation of flow

```
Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  Gf = residualGraph(G)
  while a simple path exists from s to t in Gf
    send as much flow along path as possible
    Gf = residualGraph(G)
  return flow
```

Ford-Fulkerson: is it correct?

Does the function terminate?

- Every iteration increases the flow from s to t
- the flow is bounded by the min-cut

```
Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  Gf = residualGraph(G)
  while a simple path exists from s to t in Gf
    send as much flow along path as possible
    Gf = residualGraph(G)
  return flow
```

Ford-Fulkerson: is it correct?

When it terminates is it the maximum flow?

```
Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  Gf = residualGraph(G)
  while a simple path exists from s to t in Gf
    send as much flow along path as possible
    Gf = residualGraph(G)
  return flow
```

Ford-Fulkerson: is it correct?

When it terminates is it the maximum flow?

Assume it didn't

- We know then that the flow < min-cut
- therefore, the flow < capacity across EVERY cut
- therefore, across each cut there must be a forward edge in G_f
- thus, there must exist a path from s to t in G_f
  - start at s (and A = s)
  - repeat until t is found
    - pick one node across the cut with a forward edge
    - add this to the path
    - add the node to A (for argument sake)
- However, the algorithm would not have terminated… a contradiction

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

- traverse the graph
- at most add 2 edges for original edge
- O(V + E)

Can we simplify this expression?

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

- traverse the graph
- at most add 2 edges for original edge
- O(V + E) = O(E)
- (all nodes exists on paths exist from s to t)

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

- BFS or DFS
- O(V + E) = O(E)

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

- max-flow!
- increases ever iteration
- integer capacities, so integer increases

Can we bound the number of times the loop will execute?

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

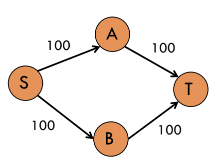
- max-flow!
- increases ever iteration
- integer capacities, so integer increases

Overall runtime?   O(max-flow * E)

O(max-flow * E)

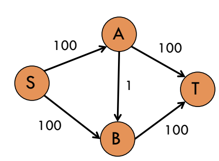
Can you construct a graph that *could get* this running time?

Hint:



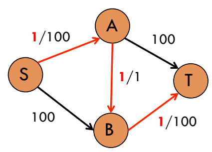
O(max-flow * E)

Can you construct a graph that *could get* this running time?



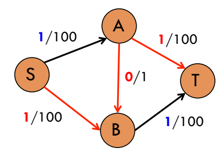
O(max-flow * E)

Can you construct a graph that *could get* this running time?



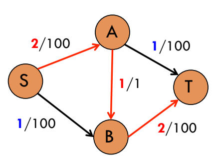
O(max-flow * E)

Can you construct a graph that *could get* this running time?



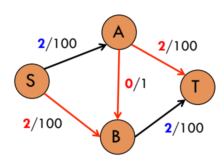
O(max-flow * E)

Can you construct a graph that *could get* this running time?



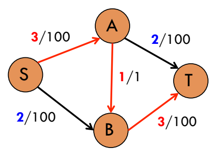
O(max-flow * E)

Can you construct a graph that *could get* this running time?



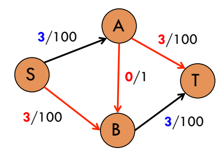
O(max-flow * E)

Can you construct a graph that *could get* this running time?



O(max-flow * E)

Can you construct a graph that *could get* this running time?



What is the problem here?
Could we do better?

Faster variants

Edmunds-Karp

- Select the *shortest path* (in number of edges) from s to t in G_f
  - How can we do this?
    - use BFS for search
- Running time: $O(V E^2)$
  - avoids issues like the one we just saw
  - see the book for the proof
  - or http://www.cs.cornell.edu/courses/CS4820/2011sp/handouts/edmondskarp.pdf

preflow-push (aka push-relabel) algorithms

- $O(V^3)$

Other variations…

Method	Complexity
Linear programming	
Ford–Fulkerson algorithm	$O(E \max\|f\|)$
Edmonds–Karp algorithm	$O(VE^2)$
Dinitz blocking flow algorithm	$O(V^2E)$
General push-relabel maximum flow algorithm	$O(V^2E)$
Push-relabel algorithm with *FIFO* vertex selection rule	$O(V^3)$
Dinitz blocking flow algorithm with dynamic trees	$O(VE \log(V))$
Push-relabel algorithm with dynamic trees	$O(VE \log(V^2/E))$
Binary blocking flow algorithm [1]	$O(E \min(V^{2/3}, \sqrt{E}) \log(V^2/E) \log U)$
MPM (Malhotra, Pramodh-Kumar and Maheshwari) algorithm	$O(V^3)$

http://en.wikipedia.org/wiki/Maximum_flow

TABLE I. POLYNOMIAL-TIME ALGORITHMS FOR THE MAXIMUM FLOW PROBLEM[a]

Algorithm no.	Date	Discoverer	Running time	References
1	1969	Edmonds and Karp	$O(nm^2)$	[5]
2	1970	Dinic	$O(n^2m)$	[4]
3	1974	Karzanov	$O(n^3)$	[18]
4	1977	Cherkasky	$O(n^2m^{1/2})$	[3]
5	1978	Malhotra, Pramodh Kumar, and Maheshwari	$O(n^3)$	[21]
6	1978	Galil	$O(n^{5/3}m^{2/3})$	[11]
7	1978	Galil and Naamad; Shiloach	$O(nm(\log n)^2)$	[12, 25]
8	1980	Sleator and Tarjan	$O(nm \log n)$	[27, 28]
9	1982	Shiloach and Vishkin	$O(n^3)$	[26]
10	1983	Gabow	$O(nm \log U)$	[10]
11	1984	Tarjan	$O(n^3)$	[31]
12	1985	Goldberg	$O(n^3)$	[14]
13	1986	Goldberg and Tarjan	$O(nm \log(n^2/m))$	[16, 15]
14	1986	Ahuja and Orlin	$O(nm + n^2 \log U)$	[1]

[a] Algorithm 13 is presented in this paper.

http://akira.ruc.dk/~keld/teaching/algoritmedesign_f03/Artikler/08/Goldberg88.pdf