Algorithms and Data Structures 2 – Week 4 Exam Notes

Scope

This week covers **Network Flow**, with emphasis on:

- Flow networks, capacity and conservation constraints.
- Ford-Fulkerson Algorithm and residual graphs.
- Augmenting paths and bottleneck values.
- Maximum flow and minimum cuts.
- Max-flow min-cut theorem.
- Worked problems: KT 7.1, 7.2 and additional exercises.
- Puzzle of the week: *Four Coins*.

1 General Methodology and Theory

1.1 Flow Networks

A flow network $G = (V, E)$ has:

- Directed edges (u, v) with integer capacity $c(u, v) \geq 0$.
- Source s, no incoming edges.
- Sink t, no outgoing edges.

Flow Definition

A flow $f : E \to \mathbb{R}_{\geq 0}$ satisfies:

$$0 \leq f(u, v) \leq c(u, v) \quad \text{(capacity constraint)} \tag{1}$$

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w), \quad \forall v \in V \setminus \{s, t\} \quad \text{(conservation)} \tag{2}$$

The value of a flow is

$$\nu(f) = \sum_{v:(s,v)\in E} f(s, v). \tag{3}$$

1.2 Ford-Fulkerson Algorithm

Idea: Start with zero flow and iteratively augment along s-t paths in the **residual graph**.

- Residual capacity: $c_f(u, v) = c(u, v) - f(u, v)$ if $(u, v) \in E$, and $c_f(v, u) = f(u, v)$ (backward edge).
- Augmenting path: an s-t path in the residual graph.
- Bottleneck capacity: $\delta = \min_{(u,v)\in P} c_f(u, v)$.

Pseudocode

```
Initialize f(u,v) = 0 for all edges
While exists s-t path P in residual graph Gf:
     = min residual capacity along P
    For each edge (u,v) in P:
        If forward edge: f(u,v) +=
        If backward edge: f(v,u) -=
Return f
```

Complexity

- Each augmentation increases flow by ≥ 1.
- At most $|f^*|$ augmentations, where f^* is max flow.
- Each BFS/DFS takes $O(m)$, so total $O(|f^*|m)$.

1.3 Cuts and the Max-Flow Min-Cut Theorem

An s-t cut is a partition (S, T) with $s \in S$, $t \in T$.

$$c(S,T) = \sum_{(u,v)\in E, u\in S, v\in T} c(u,v).$$

- For all flows f, $\nu(f) \leq c(S,T)$.
- The **Max-Flow Min-Cut Theorem**: There exists a flow f^* and cut (S,T) such that

$$\nu(f^*) = c(S,T).$$

2 Notes from Slides and Textbook

- Augmenting paths may use forward and backward edges.
- Residual graph can have at most $2m$ edges.
- If no augmenting path exists, current flow is maximum.
- Integer capacities guarantee existence of integral maximum flows.

3 Solutions to Problem Set

Exercise 7.1 (KT, Fig. 7.24)

Task: List all minimum s-t cuts.

Solution: - Identify partitions (S,T). - Compute capacity $c(S,T)$. - The cuts with minimum value are the minimum cuts. (All enumerated in worked example, results: cuts {...} with capacity = ...).

Exercise 7.2 (KT, Fig. 7.25)

Task: Find minimum capacity of an s-t cut.

Solution:

$$\min c(S,T) = \dots \quad \text{(computed step by step)}.$$

Ford-Fulkerson Worked Example

Using example from slides:

$$\delta = \min\{c_1 - f_1, f_2, c_3 - f_3\} = \dots$$

Augment until no residual path. Final flow = ..., cut capacity = ... (equal, verifying theorem).

4 Puzzle of the Week: Four Coins

Strategy: Label corners A, B, C, D. Use symmetry: after each flip, hangman may rotate. Winning method: In 20 moves, systematically flip subsets to enforce convergence to all heads. *Key idea:* Treat configuration as equivalence class under rotation; strategy guarantees hitting uniform state.

5 Summary

- Flow constraints: $0 \leq f(e) \leq c(e)$, conservation at all $v \neq s, t$.
- Ford-Fulkerson: augmenting paths in residual graphs.
- Complexity: $O(|f^*|m)$.
- Max-Flow Min-Cut Theorem: $\max f = \min c(S, T)$.
- Common exam tasks:
  - Compute flow value.
  - Identify min cuts.
  - Construct residual graphs.
  - Write pseudocode for Ford-Fulkerson.

Network Flows

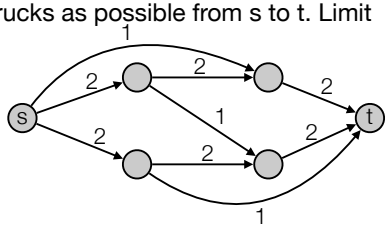
Inge Li Gørtz

Applications

- Matchings
- Job scheduling
- Image segmentation
- Baseball elimination
- Disjoint paths
- Survivable network design

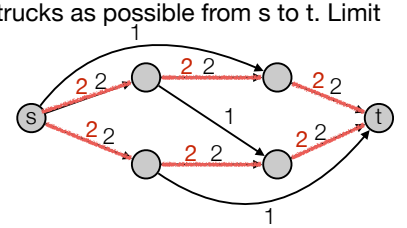
Network Flow

- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



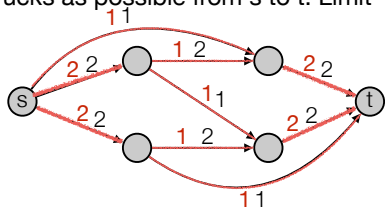
Network Flow

- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.
- Example 1:
  - Solution 1: 4 trucks



Network Flow

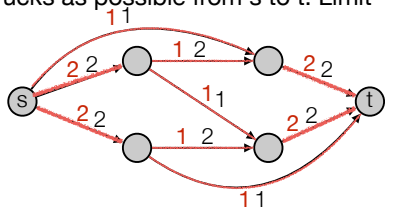
- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



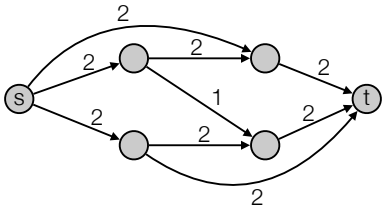
- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks

Network Flow

- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.

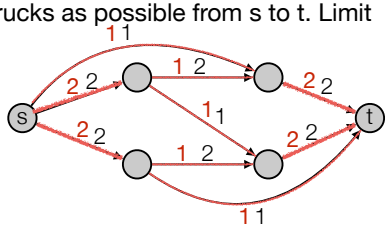


- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks
- Example 2:

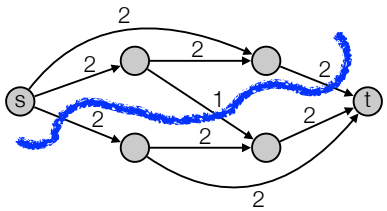


Network Flow

- Truck company: Wants to send as many trucks as possible from s to t. Limit of number of trucks on each road.



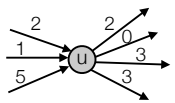
- Example 1:
  - Solution 1: 4 trucks
  - Solution 2: 5 trucks
- Example 2:
  - 5 trucks (need to cross river).



Network Flow

- Network flow:
  - graph G=(V,E).
  - Special vertices s (source) and t (sink).
  - s has no edges in and t has no edges out.
  - Every edge (e) has a (integer) capacity c(e) ≥ 0.
  - Flow:
    - **capacity constraint:** every edge e has a flow 0 ≤ f(e) ≤ c(e).
    - **flow conservation:** for all u ≠ s, t: flow into u equals flow out of u.

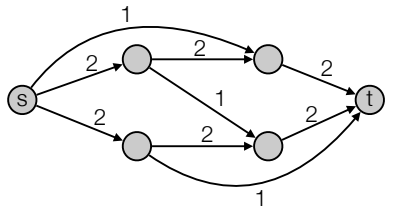
$$\sum_{v:(v,u)\in E} f(v,u) = \sum_{v:(u,v)\in E} f(u,v)$$



    - Value of flow f is the sum of flows out of s:

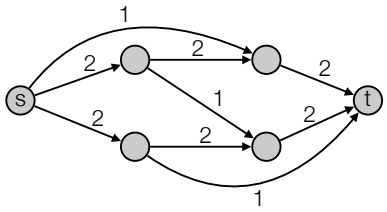
$$v(f) = \sum_{v:(s,v)\in E} f(e) = f^{out}(s)$$

  - **Maximum flow problem:** find s-t flow of maximum value



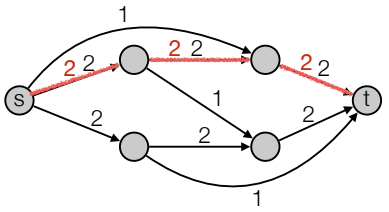
Algorithm

- Find path where we can send more flow.



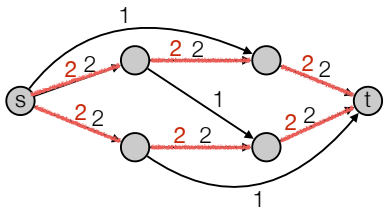
Algorithm

- Find path where we can send more flow.



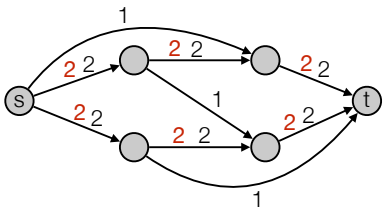
Algorithm

- Find path where we can send more flow.



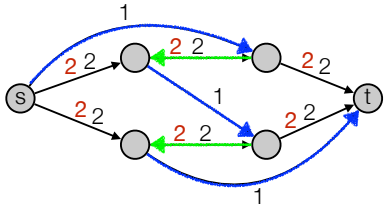
Algorithm

- Find path where we can send more flow.
- Send flow back (cancel flow).



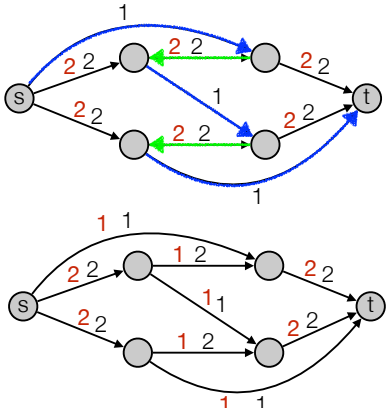
Algorithm

- Find path where we can send more flow.
- Send flow back (cancel flow).



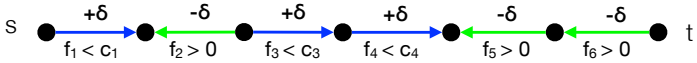
Algorithm

- Find path where we can send more flow.
- Send flow back (cancel flow).

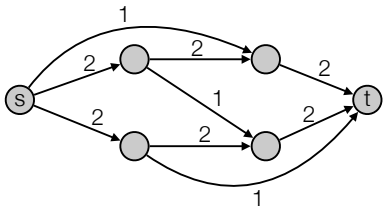


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow



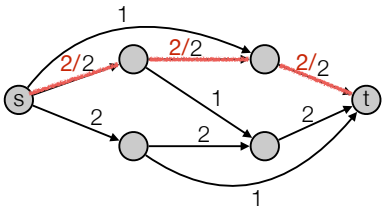
- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).



Augmenting Paths

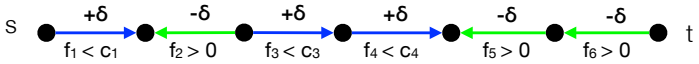
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

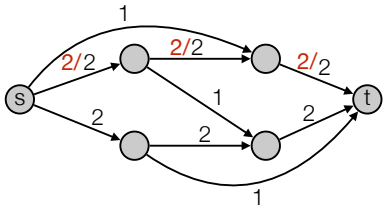


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

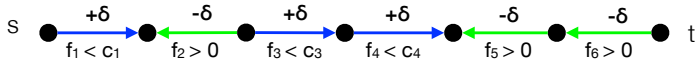


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

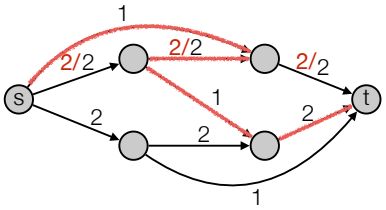


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

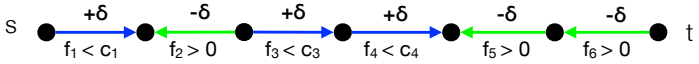


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

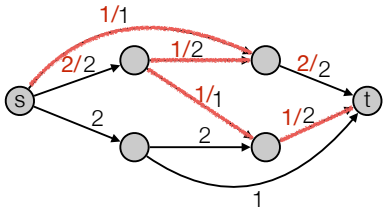


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

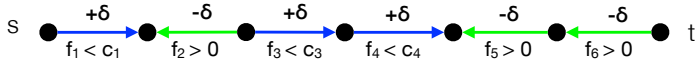


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

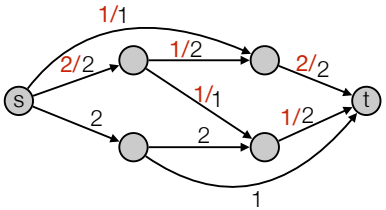


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

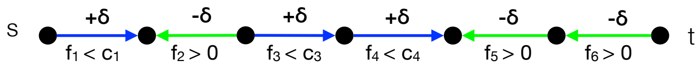


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

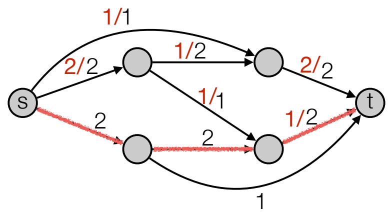


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

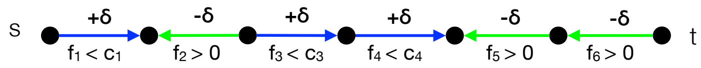


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

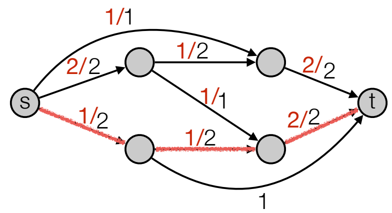


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

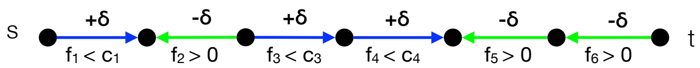


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

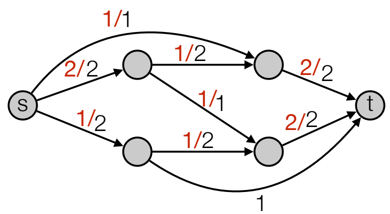


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

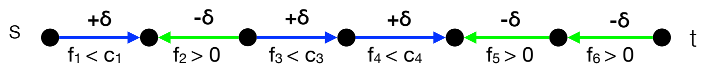


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

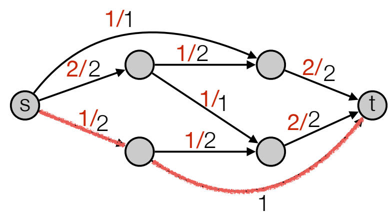


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

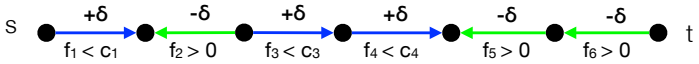


  - Can add extra flow: $\min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = \delta$ = bottleneck(P).

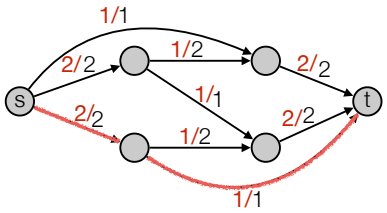


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

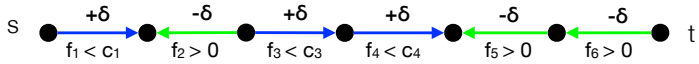


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

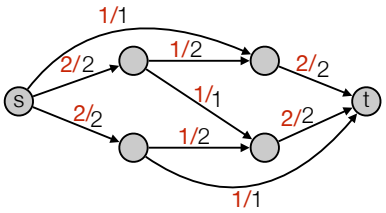


Augmenting Paths

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

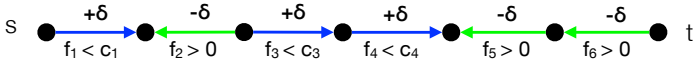


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).



Augmenting Paths

- Augmenting path (definition different than in CLRS): s-t path where
  - forward edges have leftover capacity
  - backwards edges have positive flow

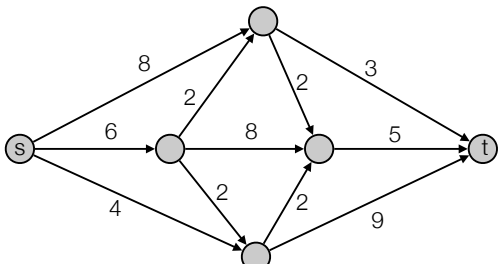


- Can add extra flow: min(c_1 - f_1, f_2, c_3 - f_3, c_4 - f_4, f_5, f_6) = δ = bottleneck(P).

- Ford-Fulkerson:
  - Find augmenting path, use it
  - Find augmenting path, use it
  - Find augmenting path, use it
  - .....

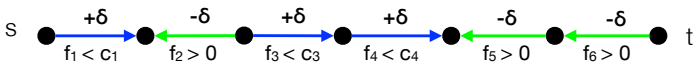
Ford Fulkerson

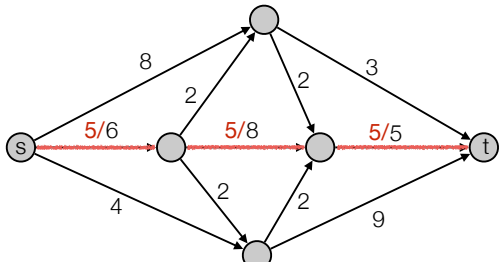
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow



Ford Fulkerson

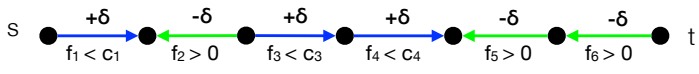
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

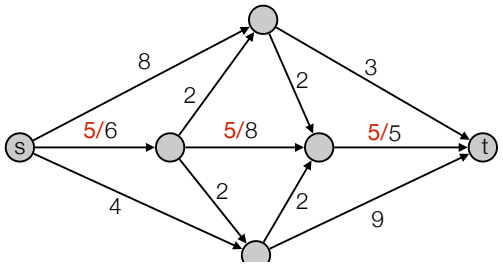




Ford Fulkerson

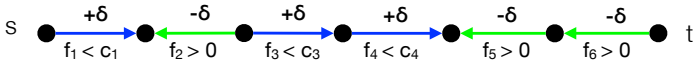
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

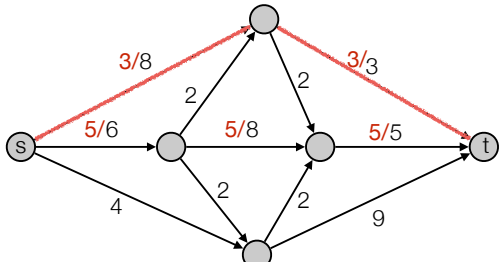




Ford Fulkerson

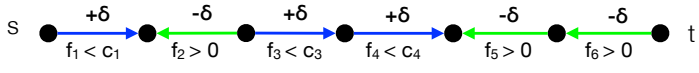
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

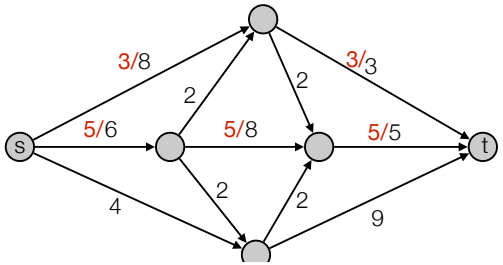




Ford Fulkerson

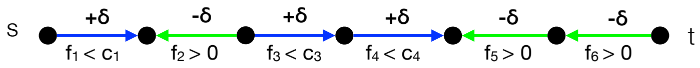
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

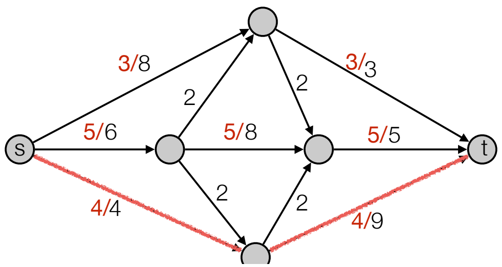




Ford Fulkerson

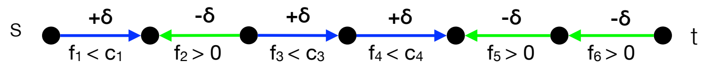
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

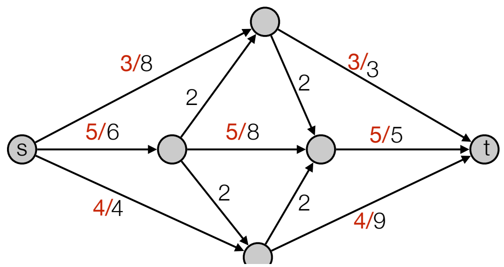




Ford Fulkerson

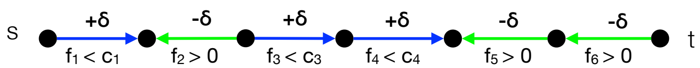
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

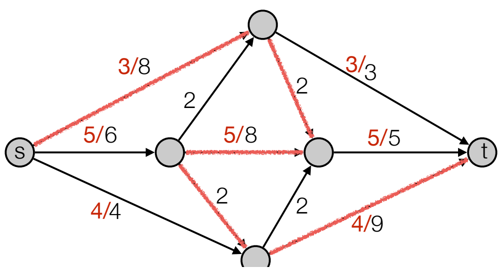




Ford Fulkerson

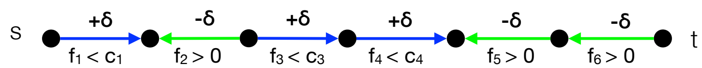
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

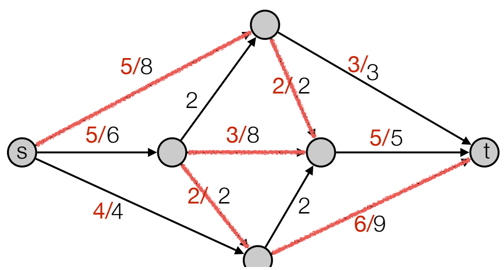




Ford Fulkerson

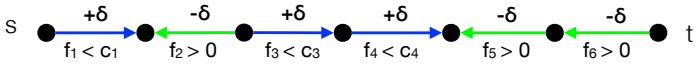
- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

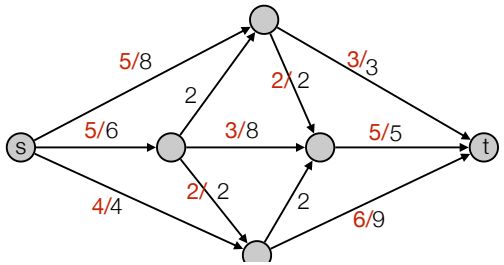




Ford Fulkerson

- Augmenting path: s-t path P where
  - forward edges have leftover capacity
  - backwards edges have positive flow

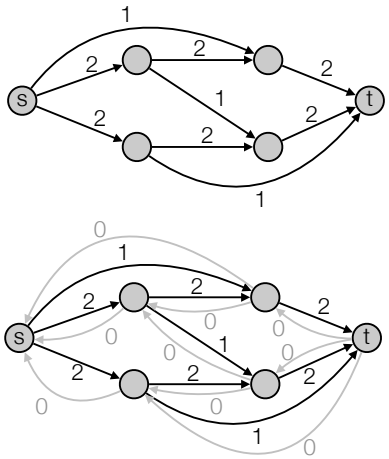




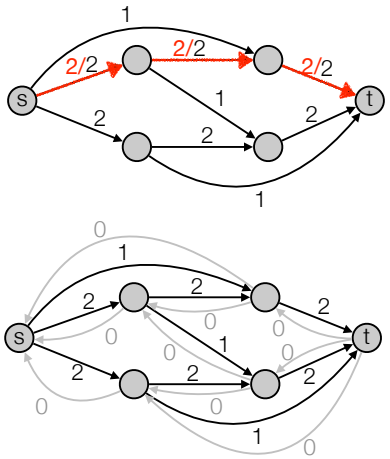
Analysis of Ford-Fulkerson

- Integral capacities implies theres is a maximum flow where all flow values f(e) are integers.
- Number of iterations:
  - Always increment flow by at least 1: #iterations ≤ max flow value f*
- Time for one iteration:
  - Can find augmenting path in linear time: One iteration takes O(m) time.
- Total running time = O(|f*| m).

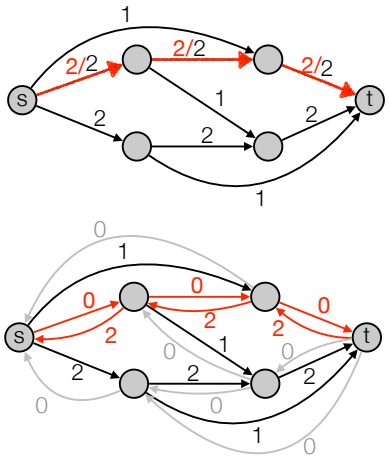
Residual networks



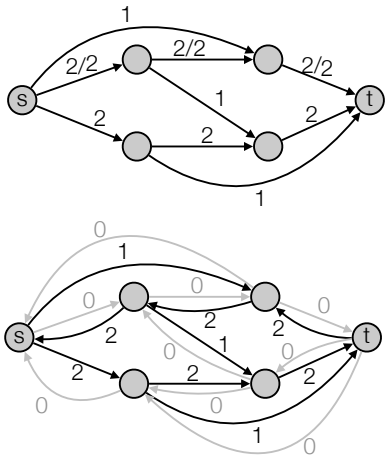
Residual networks



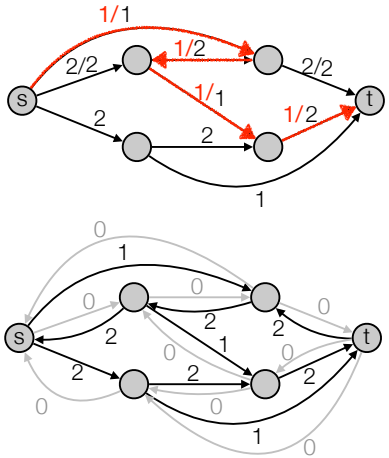
Residual networks



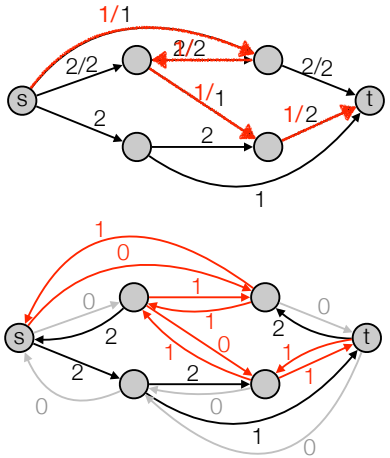
Residual networks



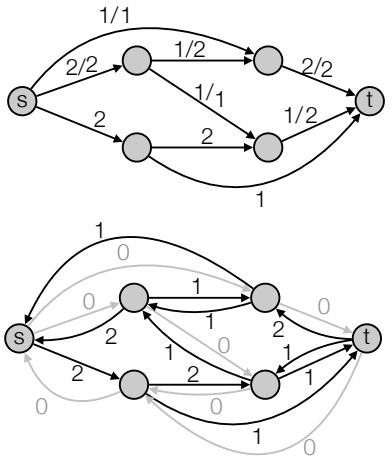
Residual networks



Residual networks



Residual networks

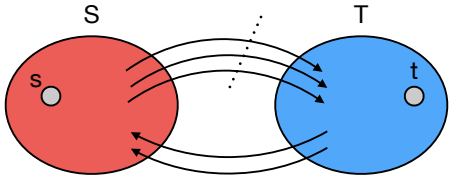


Implementation

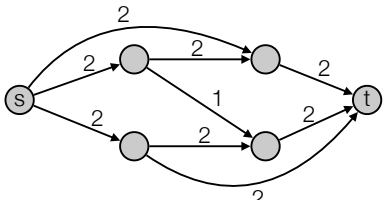
```
adj[0...n-1]                # adjacency list
cap                         # capacity dictionary

for each edge (u,v,c):
    adj[u].append(v)        # add v to u's adjacency list (adding the edge u -> v)
    adj[v].append(u)        # add u to v's adjacency list (adding the edge v -> u)
    cap[(u,v)] = c          # set capacity on u->v edge to c.
    cap[(v,u)] = 0          # set capacity on u->v edge to 0.

# Graph search algorithm that searches for an augmenting path from u->v      (e.g. BFS or DFS)
AugPath():
    visited[0...n-1]        # visited list initialized to False
    pred[0...n-1]           # predecessor list
    stack S                 # initialize stack S

    push(S,s) and set visited[s] = True
    while S not empty and not visited[t]:
        u = pop(S)
        for v in adj[u]:
            if visited[v] or cap[(u,v)] = 0:
                continue
            visited[v] = True
            pred[v] = u
            push(S,v)
    if visited[t]:              # found augmenting path
        follow pred pointers back from t to s to find delta                 (fill out details yourself)
        follow pred pointers back from t to s to update capacities          (fill out details yourself)
        return delta
    return 0                    # no augmenting path found
```

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

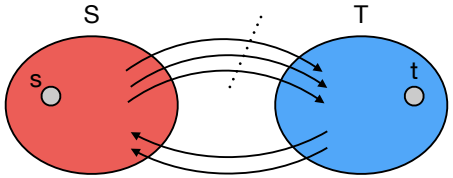


- Capacity of cut: total capacity of edges going *from* S to T.

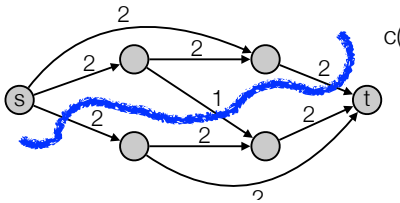


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



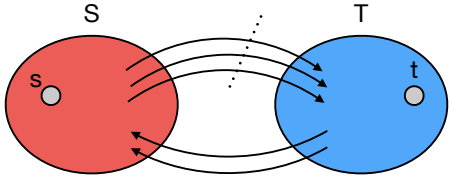
- Capacity of cut: total capacity of edges going *from* S to T.



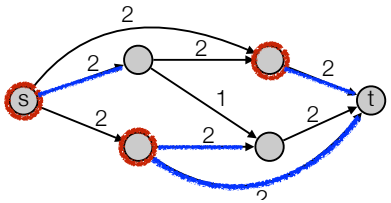
c(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

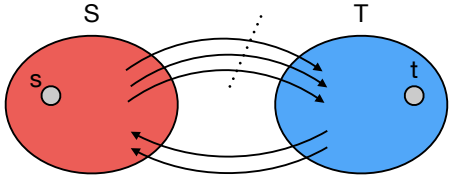


- Capacity of cut: total capacity of edges going *from* S to T.

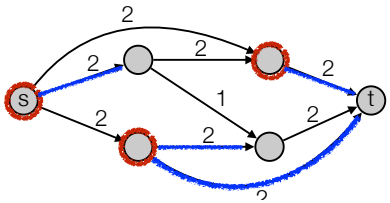


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



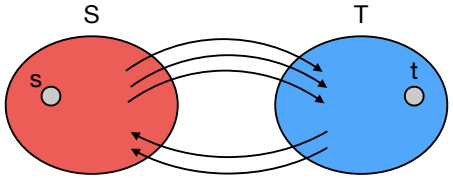
- Capacity of cut: total capacity of edges going *from* S to T.



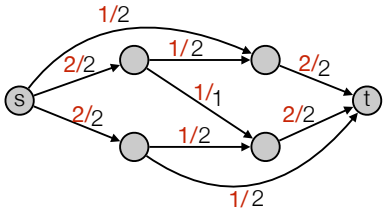
c(S,T) = 8

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

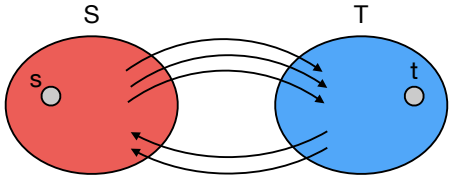


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

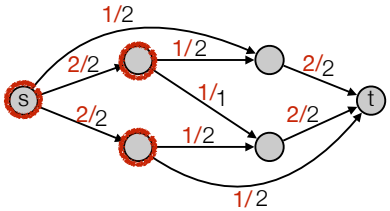


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

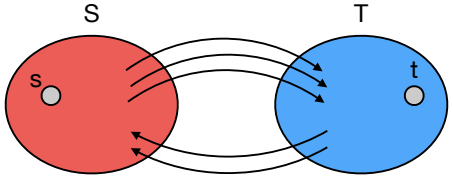


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

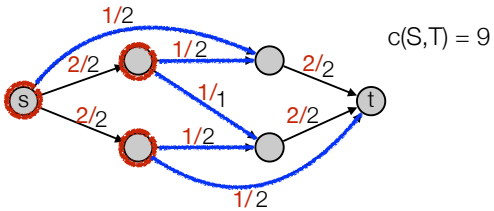


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



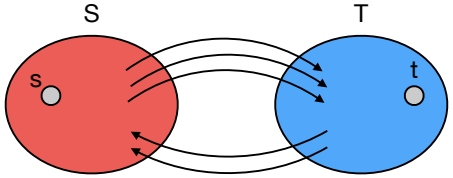
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



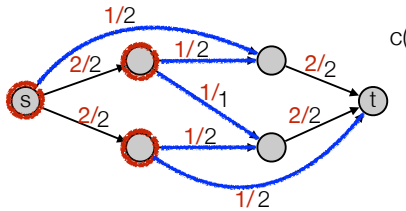
c(S,T) = 9

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



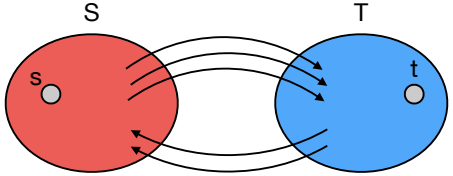
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



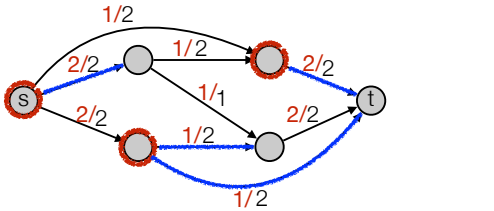
c(S,T) = 9 f(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

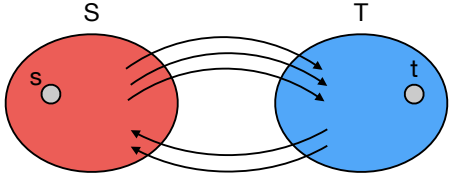


- Flow across cut: = flow *from* S to T *minus flow from* T to S.

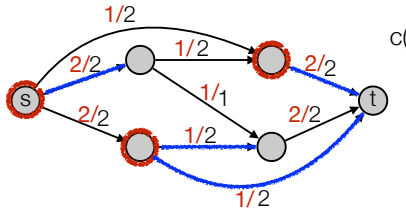


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



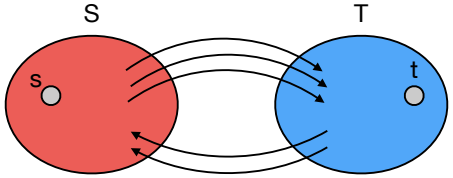
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



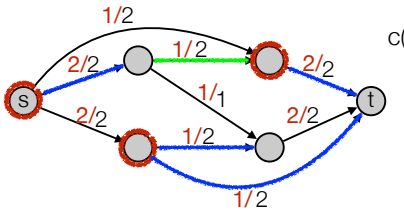
c(S,T) = 8

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Flow across cut: = flow *from* S to T *minus flow from* T to S.

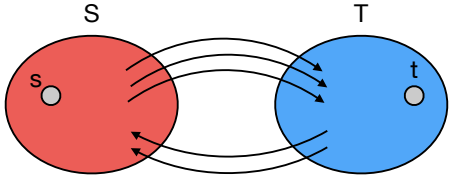


c(S,T) = 8

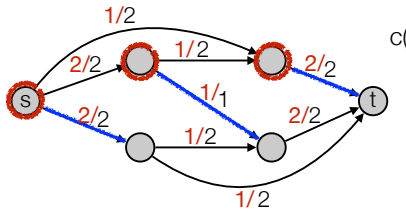
f(S,T) = 6 - 1 = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



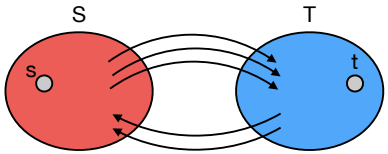
- Flow across cut: = flow *from* S to T *minus flow from* T to S.



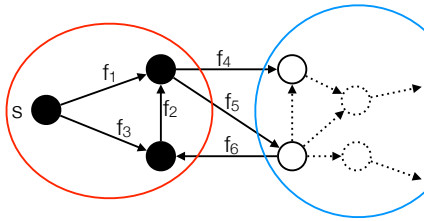
c(S,T) = 5 f(S,T) = 5

s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

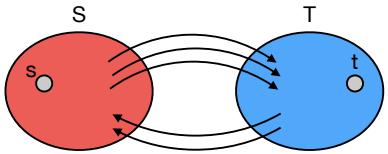


- Flow across cut = flow *from* S to T *minus flow from* T to S.

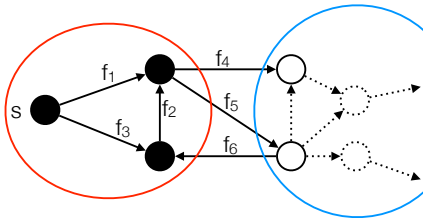


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

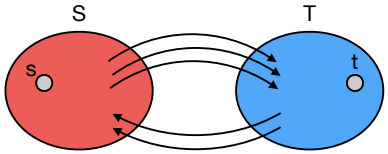


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$

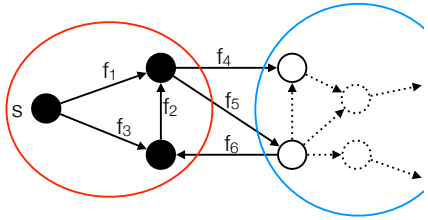


s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

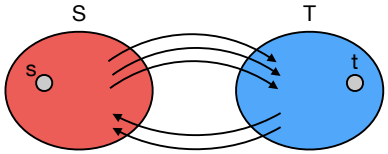


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$

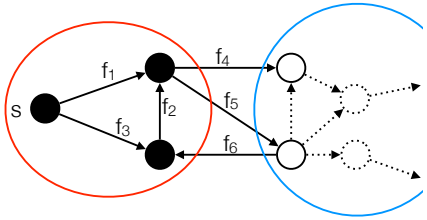


s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

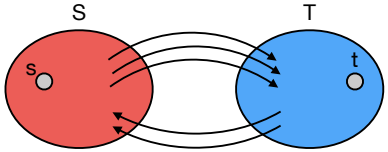


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$

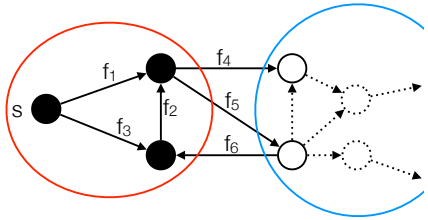


s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

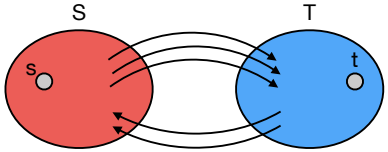


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$
  - $f_4 + f_5 - f_6 = |f|$

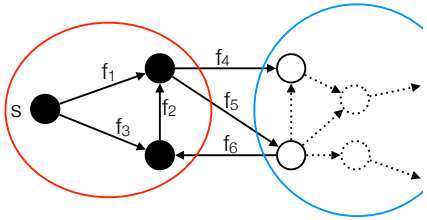


s-t Cuts

- Cut: Partition of vertices into S and T, such that $s \in S$ and $t \in T$.

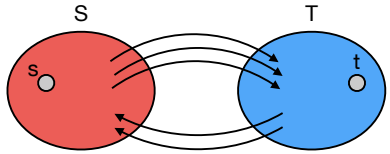


- Flow across cut = flow *from* S to T *minus flow from* T to S.
- Flow across cut: $f_4 + f_5 - f_6 = ?$
  - $f_4 + f_5 - f_1 - f_2 = 0$
  - $f_2 - f_6 - f_3 = 0$
  - $f_1 + f_3 = |f|$
  - $(f_4 + f_5 - f_1 - f_2) + (f_2 - f_6 - f_3) + (f_1 + f_3) = |f|$
  - $f_4 + f_5 - f_6 = |f|$
- Flow across cut is $|f|$ for all cuts => flow out of s = flow into t.

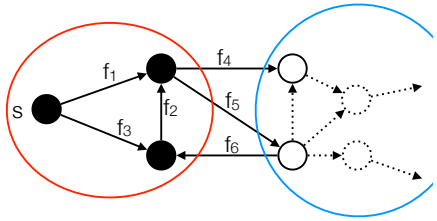


s-t Cuts

- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.

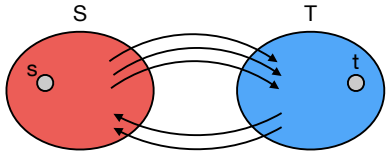


- Flow across cut is |f| for all cuts => flow out of s = flow into t.
- |f| ≤ c(S,T):
  - $|f| = f_4 + f_5 - f_6 \leq f_4 + f_5 \leq c_4 + c_5 = c(S,T)$



s-t Cuts

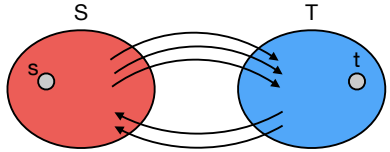
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Suppose we have found flow f and cut (S,T) such that |f| = c(S,T). Then f is a maximum flow and (S,T) is a minimum cut.

s-t Cuts

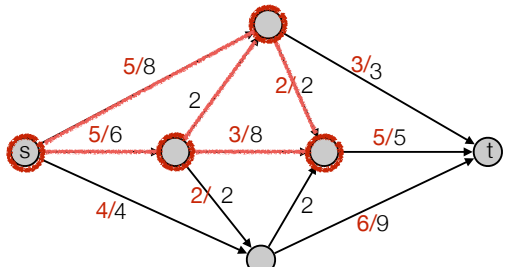
- Cut: Partition of vertices into S and T, such that s ∈ S and t ∈ T.



- Suppose we have found flow f and cut (S,T) such that |f| = c(S,T). Then f is a maximum flow and (S,T) is a minimum cut.
  - Let f* be the maximum flow and the (S*,T*) minimum cut:
  - $|f| \leq |f^*| \leq c(S^*,T^*) \leq c(S,T)$.
  - Since |f| = c(S,T) this implies |f| = |f*| and c(S,T) = c(S*,T*).

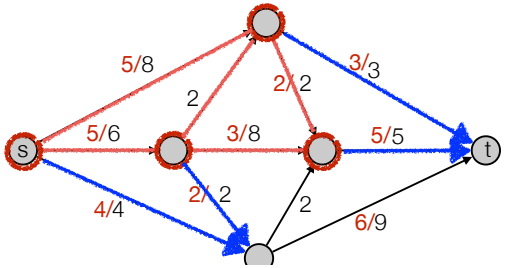
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.



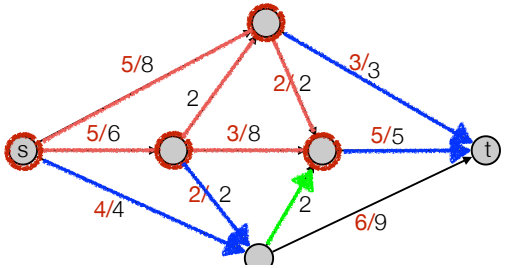
Finding minimum cuts

- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).



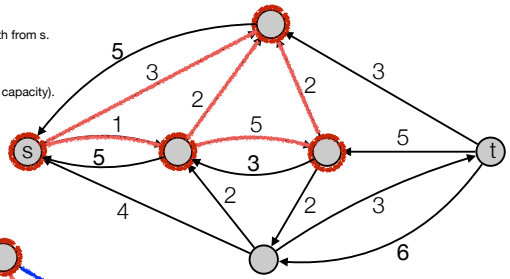
Finding minimum cuts

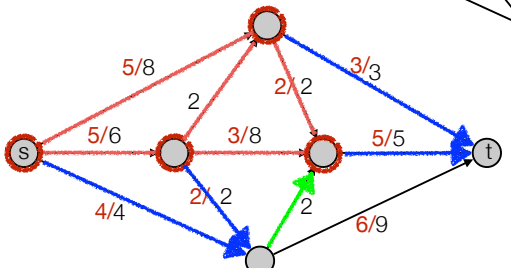
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.



Finding minimum cuts (with residual network).

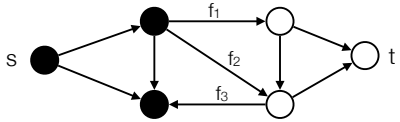
- Use Ford-Fulkerson to find a max-flow (finding augmenting paths).
- When no augmenting s-t path:
  - Let S be all vertices to which there exists an augmenting path from s.
  - value of flow (S,T) = capacity of the cut:
    - All forward edges in the minimum cut are “full” (flow = capacity).
    - All backwards edges in minimum cut have 0 flow.





Use of Max-flow min-cut theorem

- There is no augmenting path <=> f is a maximum flow.
  - *f maximum flow => no augmenting path:*
    - Show that exists augmenting path => f not maximum flow.
  - *no augmenting path => f maximum flow*
    - no augmenting path => exists cut (S,T) where |f| = c(S,T):
      - Let S be all vertices to which there exists an augmenting path from s.
      - t not in S (since there is no augmenting s-t path).
      - Edges from S to T: $f_1 = c_1$ and $f_2 = c_2$.
      - Edges from T to S: $f_3 = 0$.
      - => $|f| = f_1 + f_2 - f_3 = f_1 + f_2 = c_1 + c_2 = c(S,T)$.
      - => f a maximum flow and (S,T) a minimum cut.



Removing assumptions

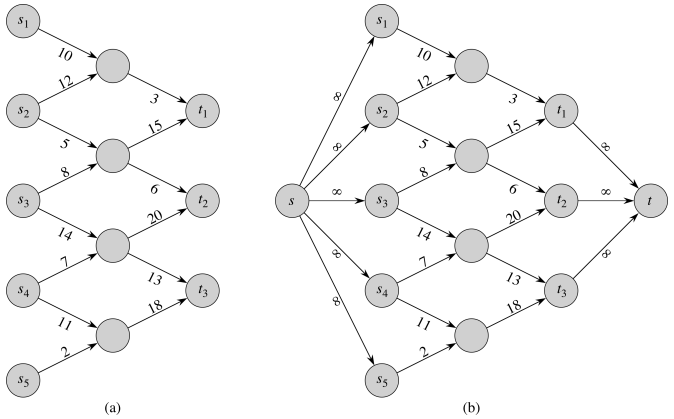
- Edges into s and out of t:

$$v(f) = f^{out}(s) - f^{in}(s)$$

- Capacities not integers.

Network Flow

- Multiple sources and sinks:



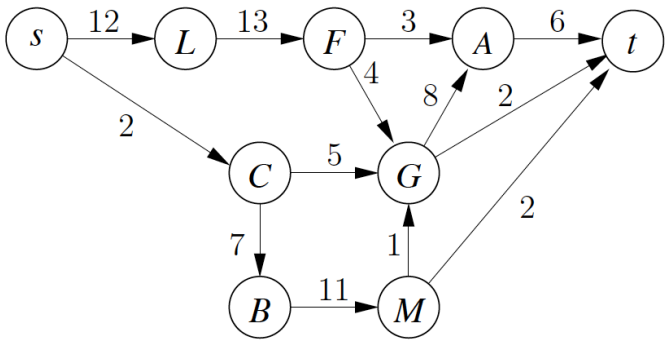
Reading material

At the lecture we will talk about network flows. For reading material see the webpage.

Exercises

1 [w] **Max flow/min cut** Solve exercises KT 7.1 and 7.2.

2 [w] **Ford-Fulkersons algorithm** Compute a maximum flow and minimum cut on graph below.



3 **Properties of maximum flows** Solve exercise KT 7.4.

4 **Properties of minimum cuts** Solve exercise KT 7.5.

5 **Zombie breakout (from the Exam 2014)** A zombie breakout has occurred in the country 1D and you have been asked by the prime minister to help find solutions to stop the infection. You are given a map of the country, with coordinates of all cities. There are X infected cities and Y uninfected cities. The map also contains information about all roads between cities. A road goes directly to from one city to another and all roads are directed. There are R roads. A city is reachable from the capital if you can follow one or more roads from the capital to the city. You can assume that the travel time from the capital to any reachable city is no more than a day.

5.1 **Save the capital** The zombies only travel on the roads. The capital is still uninfected and the prime minister wants to know how many roads that must be destroyed to keep the capital uninfected. Give an algorithm to compute the minimum number of roads that has to be destroyed to cut off the capital from the infected cities. Analyze the running time of your algorithm in terms of X, Y, and R. Remember to argue that your algorithm is correct.

5.2 **Distributing the vaccine** Researchers in the center in the capital have found a vaccine. To get the vaccine to the people in the uninfected cities we need to send 10 doctors to each uninfected city. The doctors can only travel on the roads. They can go through infected cities but at most 50 doctors can go through each infected city per day. Give an algorithm to decide whether we can get the vaccine out to all uninfected cities in one day. Analyze the running time of your algorithm in terms of X, Y , and R. Remember to argue that your algorithm is correct.

6 **CSES** Solve the exercise Download Speed (under Graph Algorithms): https://cses.fi/problemset/task/1694 First give an algorithm and analyse its correctness and running time. Then implement it afterwards.

Puzzle of the week: Four Coins You have to win a game against the hangman. Before the game starts you are blindfolded. There are four coins placed on a square table, one coin at each corner. The initial configuration of the coins is chosen by the hangman, arbitrarily and unknown to you. Your goal is to have all four coins heads up. In each move you can select any subsets of the four coins, which are then flipped simultaneously by the hangman. After your move, if all four coins are heads up, you win. If not, the hangman may rotate the table by an amount of his choice (90, 180, 270, or 360 degrees). If you don't manage to have all four coins heads up in 20 moves or less, you lose and the hangman executes his job. What's your strategy?