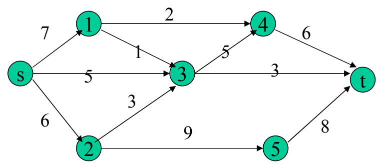
Maximum Flow Problem

- Given a weighted directed graph
  - Each edge is a pipe whose weight denotes its capacity: the maximum amount it can transport
    - Use *c(e)* for the capacity of edge *e*
  - Given a source, *s*, and a sink, *t*, find the maximum amount (flow) can transfer from *s* to *t*



Flow

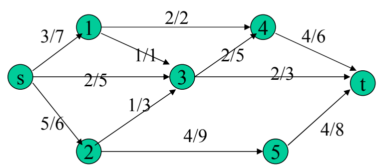
- A flow network *N* consists of
  - A connected directed graph *G* with weighted edge denoting capacity
  - A source (no incoming edges) and a sink (no outgoing edges).
- A flow for network *N* is an assignment of an integer value *f*(*e*) to each edge *e* of *G*, such that
  - Capacity rule $\forall e \in G, \quad 0 \le f(e) \le c(e)$
  - Conservation rule

$$\forall v \in G, v \neq s,t, \ \sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e)$$

$E^-(v)$: incoming edges of v

$E^+(v)$: outgoing edges of v

An Example Flow

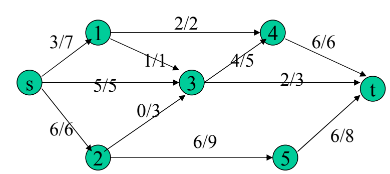


Flow value and Maximum flow

- The *value* of a flow *f*, denoted by |*f*| is

$$|f| = \sum_{e \in E^-(t)} f(e) = \sum_{e \in E^+(s)} f(e)$$

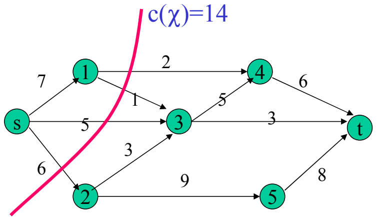
- A *maximum flow* is a flow with maximum value



$|f^*| = 14$

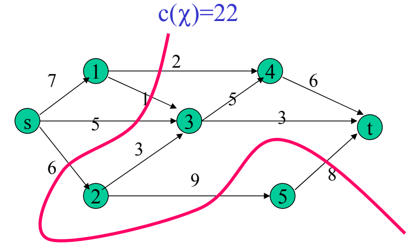
Cuts

- A cut of N is a partition $\chi=(V_s, V_t)$ of the vertices of N such that $s\in V_s$ and $t\in V_t$
  - (u, v) is a ***forward* edge** if $u\in V_s$ and $v\in V_t$
  - (u, v) is a ***backward* edge** if $v\in V_s$ and $u\in V_t$
- Capacity of a cut: total capacity of forward edges



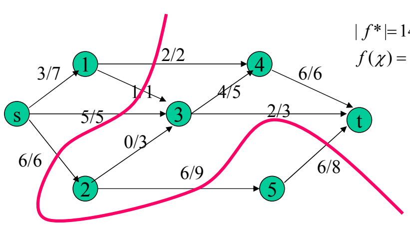
Cuts: the other example

- A cut of N is a partition $\chi=(V_s, V_t)$ of the vertices of N such that $s\in V_s$ and $t\in V_t$
  - (u, v) is a forward edge if $u\in V_s$ and $v\in V_t$
  - (u, v) is a backward edge if $v\in V_s$ and $u\in V_t$
- Capacity of a cut: total capacity of *forward* edges



Network Flow v.s. Flow Across Cut

- Given a flow f for N, the flow across cut χ, denoted $f(\chi)$, is equal to the sum of the flows in the forward edges minus the sum of the flows in the backward edges.



$|f^*| = 14$

$f(\chi) = 2+1+5+6+6-6 = 14$

Network Flow v.s. Flow Across Cut

- Lemma and Theorem
  - Let N be a flow network, and let f be a flow for N. For any cut χ of N, the value of f is equal to the flow across cut , that is, $|f| = f(\chi)$.
  - Let N be a flow network, and let χ be a cut of N. Given any flow f for N, the flow across cut χ does not exceed the capacity of χ. that is, $f(\chi) \leq c(\chi)$
  - Let N be a flow network. Given any flow f and any cut χ, the value of f does not exceed the capacity of χ, that is $|f| \leq c(\chi)$.
- Minimum cut of N is the cut with minimum capacity.

Residual Capacity

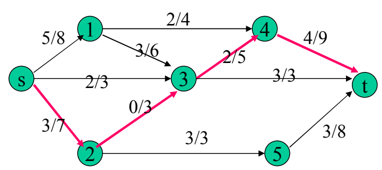
- Let N be a flow network, and let f be a flow for N. Let $e = (u, v)$ be an edge in G for N. The residual capacity from u to v with respect to the flow f is defined as $\Delta_f(u,v) = c(e) - f(e)$. The residual capacity from v to u w.r.t. f is defined as $\Delta_f(v,u) = f(e)$
  - Intuition: the residual capacity is the capacity that f has not fully take advantage of
- Let π be a path from s to t, and edges can be traversed in either forward or backward direction
  - Forward edge: origin of the edge encountered first
  - Backward edge: destination encountered first

$$\Delta_f(e) = \begin{cases} c(e) - f(e) & \text{e is forward} \\ f(e) & \text{e is backward} \end{cases}$$

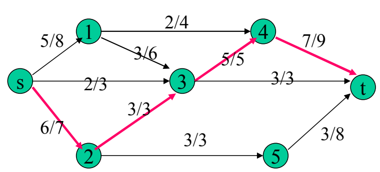
Augmenting Paths

- The residual capacity of a path π is the minimum residual capacity of the edges: $\Delta_f(\pi) = \min_{e\in\pi} \Delta_f(e)$
- An *augmenting path* for flow f is a path π from the source s to sink t with nonzero residual capacity
  - For each edge e of π
    - $f(e) < c(e)$, if e is a forward edge
    - $f(e) > 0$, if e is a backward edge

Augmenting Path Example

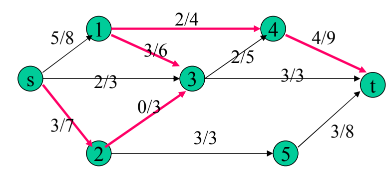


Augmenting path

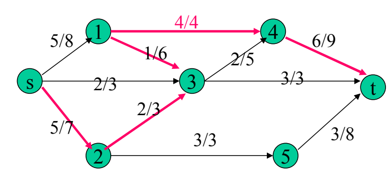


New flow obtained after pursing 3 units from s to t along the path

Augmenting Path Example

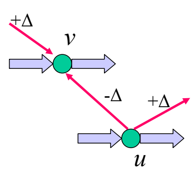


Augmenting path



New flow obtained after pursing 2 units from s to t along the path

Understanding backward edges



Conservation rule still holds after pushing residual capacity

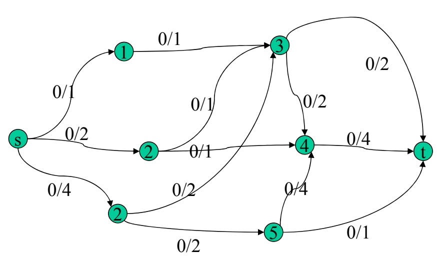
The Max-Flow, Min-Cut Theorem

- Lemma 1:
  - Let π be a augmenting path for flow f in network N, there exists a flow f' for N of value $|f'|=|f|+\Delta_f(\pi)$
- Lemma 2:
  - If a network N does not have an augmenting path with respect to a flow f, then f is a maximum flow. Also there is a cut of N such that $|f|=c(\chi)$.
- Theorem:
  - The value of a maximum flow is equal to the capacity of a minimum cut

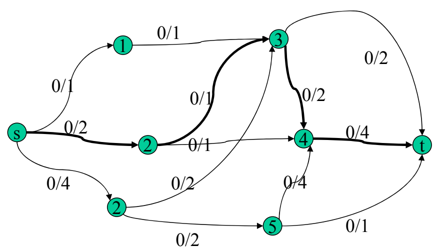
The Ford-Fulkerson Algorithm

```
maxFlowFordFulkerson(N)
 // N =(G, c, s, t)
 {
    for each edge e in N do {
      f(e) = 0;
      stop = false;
    }
    while (!stop) {
        traverse G starting at s to find an augmenting path for f;
        if an augmenting π path exists {
            Δ = minimum Δf(e) along π;
            for each edge e in π {
               if (e is an forward edge) f(e) += Δ; else f(e) -= Δ;
            }
        } else
           stop = true;
    }
 }
```

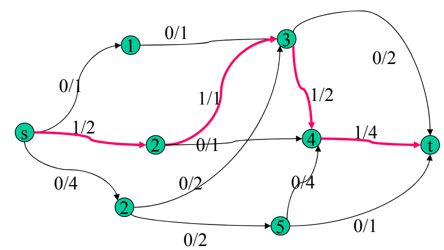
Example of the Ford-Fulkerson Algorithm



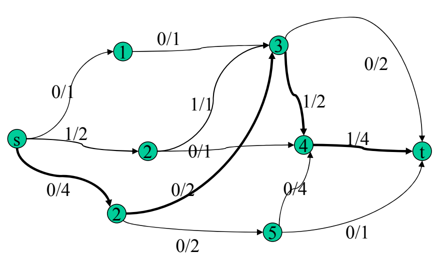
Example of the Ford-Fulkerson Algorithm



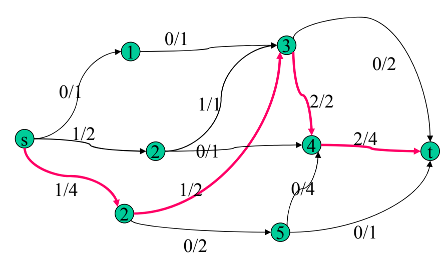
Example of the Ford-Fulkerson Algorithm



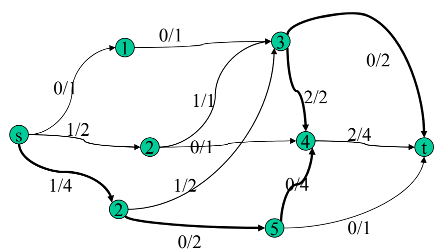
Example of the Ford-Fulkerson Algorithm



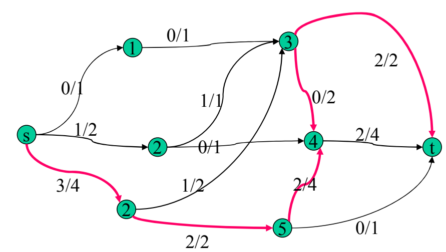
Example of the Ford-Fulkerson Algorithm



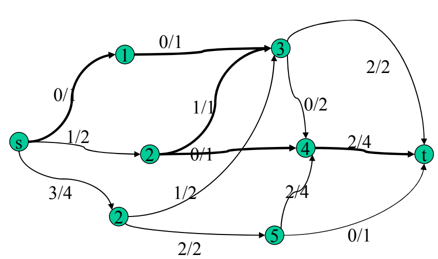
Example of the Ford-Fulkerson Algorithm



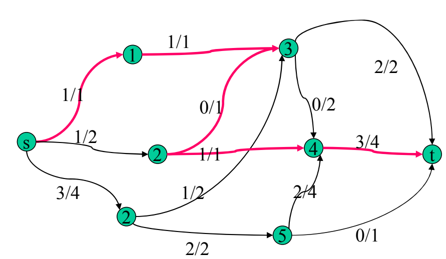
Example of the Ford-Fulkerson Algorithm



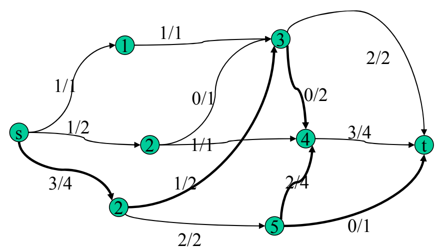
Example of the Ford-Fulkerson Algorithm



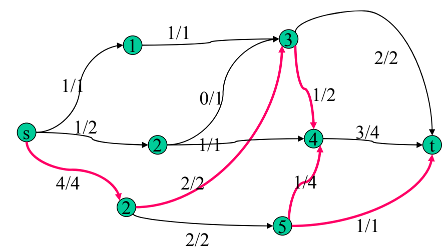
Example of the Ford-Fulkerson Algorithm



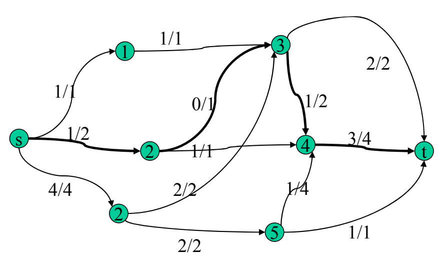
Example of the Ford-Fulkerson Algorithm



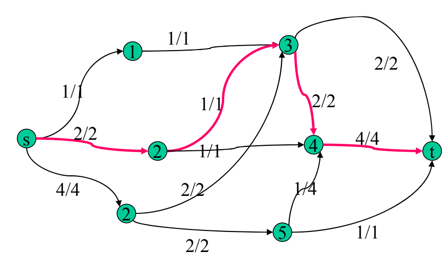
Example of the Ford-Fulkerson Algorithm



Example of the Ford-Fulkerson Algorithm



Example of the Ford-Fulkerson Algorithm



How to Compute Augmenting Paths

- Method 1
  - Modify BFS or DFS by considering only the following edges
    - Outgoing edges of v with flow less than the capacity
    - Incoming edges of v with nonzero flow
- Method 2
  - Construct a residual graph R_f with respect to flow f
    - $V(R_f) = V(G)$
    - Add edge (u, v) to R_f if $\Delta_f(u,v) > 0$
  - Traverse R_f use BFS

Depth-first search algorithm

```
DFS(G)
{
   for each node u ∈ N {
      color[u] = WHITE;
      π[u] = null;
   }

   time = 0;

//   for each node u ∈ N {
//      if (color[u] == WHITE)
//         DFS-Visit(u);
//   }
   DFS-Visit(s);
}
```

```
DFS-Visit(u)
{
   color[u] = GRAY;
   d[u] = ++time;
   for each v incident to u {
      if  (color[v] == WHITE
           && residual(u,v) > 0)
         )
      {
         π[v] = u;
         if (v == t) find an augment path;
         else DFS-Visit(v);
      }
   }
   color[u] = BLACK;
   f[u] = ++time;
}
```

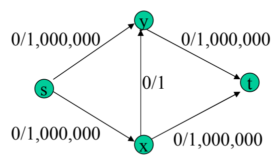
Analysis of the Ford-Fulkerson Algorithm

- Let n and m be #vertices and #edges (n<=m+1)
- Let f^* be a maximum flow
- Each loop increases the value of the flow by at least 1
  - The upper bound for the outer loop is $|f^*|$
- DFS or BFS takes time O(m)
- The algorithms takes O($m|f^*|$)

The Ford-Fulkerson Algorithm

```
maxFlowFordFulkerson(N)
 // N =(G, c, s, t)
{
   for each edge e in N do {
     f(e) = 0;
     stop = false;
   }
   while (!stop) {                                  At most |f*| iterations
O(m)  traverse G starting at s to find an augmenting path for f;
      if an augmenting π path exists {
        Δ = minimum Δf(e) along π;
        for each edge e in π {
          if (e is an forward edge) f(e) += Δ; else f(e) -= Δ;
        }
      } else
        stop = true;
   }
}
```

Example of a bad case



Runs slow when choose (s, x, y,t) and (s,y,x,t) alternatively

The Edmonds-Karp Algorithm

- Try to find a "good" augmenting path each time
  - Choose an augmenting path with the smallest number of edges
    - Can be implemented using BFS traversal

Breadth-first search algorithm

```
BFS(G,s)
{
   for each node u ∈ N – {s} {
      color[u] = WHITE;
      d[u] = ∞;
      π[u] = null;
   }

   color[s] = GRAY;
   d[s] =0
   enqueue(Q, s);
```

```
while (!empty(Q)) {
    u = dequeue(Q);
    for each v incident to u {
        if (color[v] == WHITE
        && residual(u,v) > 0) {
         color[v] = GRAY;
         d[v] = d[u] + 1;
         π[v] = u;
         if (v == t)
            find an augment path;
         enqueue(Q, v);
        }
    }
    color[u] = BLACK;
}
```

d[]: tracks shortest distance, assuming each edge's weight is 1
π[]: tracks the parent-child relationship in the breadth-first tree

Lemmas and Theorem

- Let g be the flow obtained from flow f with an augmentation along a path of minimum length then for each vertex v, $d_f(v) <= d_g(v)$
- When executing the Edmonds-Karp algorithm on a network with n vertices and m edges, the number of flow augmentations is no more than nm.
- Given a flow network with n vertices and m edges, the Edmonds-Karp algorithm computes a maximum flow in $O(nm^2)$ time

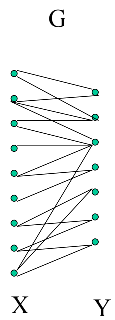
Maximum Bipartite Matching

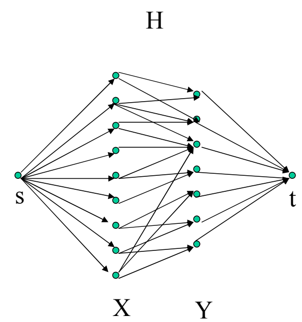
- Bipartite graph
  - a graph with vertices partitioned into two sets X and Y, such that every edge has one endpoint in X and the other in Y
- Matching in a bipartite graph
  - A set of edges that has no end points in common
- Maximum bipartite matching
  - The matching with the greatest number of edges

Reduction to the Maximum Flow Problem

- Let G be a partite graph whose vertices are partitioned into sets X and Y. Create a flow network H as follows
  - Add each vertex of G into H plus a source vertex s and a sink vertex t.
  - Add edges of G into H and make each edge orient from an endpoint in X to an endpoint in Y
  - Insert a directed edge from s to each vertex in X
  - Insert a directed edge from each vertex in Y to t
  - Assign each edge in H a capacity of 1

An example of reduction





All edges with capacity 1

Reduction to the Maximum Flow Problem

- Given the maximum flow f of H, define M as a set of edges such that e in M iff f(e) =1
  - M is a matching
  - M is a maximum matching
- Reverse transformation: given a matching M in H, define a flow f
  - For each edge e of H that is also in G, f(e) = 1 if e ∈M and f(e) = 0 otherwise.
  - For each edge of H incident to s or t and v be the other end point, f(e) = 1 if v is is an endpoint of some edge of M and f(e) = 0 otherwise

An example of reduction