Defining network flow

- Consider a directed, weighted graph $G(V, E)$
  - Weights are applied to edges to state their capacity
    - $c(u, w)$ is the capacity of edge $(u, w)$
    - if there is no edge from $u$ to $w$, $c(u, w) = 0$
- Consider two vertices, a source $s$ and a sink $t$
  - Let’s determine the maximum flow that can run from $s$ to $t$ in the graph $G$
Let the $f(u, w)$ be the amount of flow being carried along the edge $(u, w)$

Some rules on the flow running through an edge:

- $\forall (u, w) \in E \ f(u, w) \leq c(u, w)$
- $\forall u \in (V - \{s, t\}) \ (\sum_{w \in V} f(w, u) - \sum_{w \in V} f(u, w)) = 0$

An edge $(u, w)$ is considered saturated if $f(u, w) = c(u, w)$
The Max Flow Problem

Given a graph $G$, a source vertex $s$, and a sink vertex $t$, find the maximum possible flow rate from $s$ to $t$. 
Ford Fulkerson

- Let all edges in G have an allocated flow of 0
- While there is path $p$ from $s$ to $t$ in G s.t. all edges in $p$ have some residual capacity (i.e., $\forall (u, w) \in p \ f(u, w) < c(u, w)$):
  - (Such a path is called an augmenting path)
  - Compute the residual capacity of each edge in $p$
    - Residual capacity of edge $(u, w)$ is $c(u, w) - f(u, w)$
  - Find the edge with the minimum residual capacity in $p$
    - We’ll call this residual capacity $new\_flow$
  - Increment the flow on all edges in $p$ by $new\_flow$
Ford Fulkerson example
Another Ford Fulkerson example
To find the max flow we will have need to consider re-routing flow we had previously allocated.

This means, when finding an augmenting path, we will need to look not only at the edges of $G$, but also at backwards edges that allow such re-routing.

For each edge $(u, w) \in E$, a backwards edge $(w, u)$ must be considered during pathfinding if $f(u, w) > 0$.

- The capacity of a backwards edge $(w, u)$ is equal to $f(u, w)$.
We will perform searches for an augmenting path not on G, but on a residual graph built using the current state of flow allocation on G.

The residual graph is made up of:

- $V$
- An edge for each $(u, w) \in E$ where $f(u, w) < c(u, w)$
  - $(u, w)$'s mirror in the residual graph will have 0 flow and a capacity of $c(u, w) - f(u, w)$
- A backwards edge for each $(u, w) \in E$ where $f(u, w) > 0$
  - $(u, w)$'s backwards edge has a capacity of $f(u, w)$
  - All backwards edges have 0 flow
Residual graph example
Another example

Graph:
- Source: s
- Intermediate: A
- Sink: t

Edges:
- s -> A: 1 /1000
- A -> t: 1 /1000
- s -> B: 1 /1000
- A -> B: 0 /1
- B -> t: 2 /1000
- B -> s: 2 /1000
Edmonds Karp

- How the augmenting path is chosen affects the performance of the search for max flow

- Edmonds and Karp proposed a shortest path heuristic for Ford Fulkerson
  - Use BFS to find augmenting paths
Another example

A

s

1000 /1000

1000 /1000

B

1000 /1000

t

1000 /1000

/1
Edmonds-Karp only uses BFS

- Used to find spanning trees and shortest paths for *unweighted* graphs
- Why do we not use some measure of priority to find augmenting paths?
Implementation concerns

- Representing the graph:
  - Similar to a directed graph
  - Can store an adjacency list of directed edges
    - Actually, more than simply directed edges
      - Flow edges
For each edge, we need to store:

- Start point, the “from” vertex
- End point, the “to” vertex
- Capacity
- Flow
- Residual capacities (for forwards and backwards edges)
FlowEdge class

class FlowEdge:
    def __init__(self, v, w, c):
        self.v = v          # from
        self.w = w          # to
        self.capacity = c   # capacity
        self.flow = 0       # flow

    def residualCapacityTo(self, vertex):
        if vertex == self.v:
            return self.flow
        elif vertex == self.w:
            return self.capacity - self.flow

    ...
BFS search for an augmenting path

```python
degreeTo = [None for i in range(len(self.adj_list))]
marked = [False for i in range(len(self.adj_list))]
q = [s]
marked[s] = True
while len(q) > 0:
    vertex = q.pop(0)
    for edge in self.adj_list[vertex]:
        w = edge.other(vertex)
        if edge.residualCapacityTo(w) > 0:
            if not marked[w]:
                edgeTo[w] = edge;
                marked[w] = True;
                q.append(w);
```

Each FlowEdge object is stored in the adjacency list twice:

- Once for its forward edge
- Once for its backwards edge

18
An example to review
The Min Cut Problem

Given a graph $G$, a source vertex $s$, and a sink vertex $t$, find a set of edges that, if removed from the graph, would separate $s$ from $t$, and which have the minimum possible sum of their edge weights.
How do we find the min st-cut?

- We could examine residual graphs
  - Specifically, try and allocate flow in the graph until we get to a residual graph with no existing augmenting paths
  - A set of saturated edges will make a minimum st-cut
    - Recall: \((u,w)\) is saturated if \(f(u,w) = c(u,w)\)
Min cut example
A special case of duality

- i.e., you can look at an optimization problem from two angles
  - In this case to find the maximum flow or minimum cut
  - In general, dual problems do not necessarily have equal solutions
    - The difference between the solutions to the two related problems is referred to as the *duality gap*
      - If the duality gap = 0, *strong duality* holds
        - Max flow/min cut uphold strong duality
      - If the duality gap > 0, *weak duality* holds

Max flow == min cut
Determining a minimum st-cut

- First, run Ford Fulkerson to produce a residual graph with no further augmenting paths
- The last attempt to find an augmenting path will visit every vertex reachable from s
  - Saturated edges with only one endpoint in this set comprise a minimum st-cut
Determining the min cut

\[ A \]
\[ B \]
\[ C \]
\[ s \]
\[ t \]

\[ \frac{3}{3} \]
\[ \frac{2}{7} \]
\[ \frac{5}{5} \]
\[ \frac{1}{1} \]

\[ \frac{3}{7} \]
\[ \frac{3}{9} \]

Min Cut
Will max flow/min cut always be near s/t?
Max flow / min cut on unweighted graphs

- Is it possible?
- How would we measure the Max flow / min cut?
- What would an algorithm to solve this problem look like?
Unweighted network flow