**cs 473: Algorithms: Lecture 14**

**logistics:**
- part 8 due w/10
- rec 3 flw/edge
- rec 4 flw path
- rec 5 algo

**today:**
- flow algo

---

**def:** A **single-source/sink network** is a directed graph \( G = (V, E) \)

- **source:** \( s \in V \)
- **sink:** \( t \in V \)
- **capacity constraint:** \( c : E \rightarrow \mathbb{R}_{\geq 0} \)
- **flow constraint:** \( f : E \rightarrow \mathbb{R}_{\geq 0} \)

An **(s,t)-flow** is \( f : E \rightarrow \mathbb{R}_{\geq 0} \).

**capacity constraint:** \( V \in \mathbb{R}_{\geq 0} \)

**flow conservation constraint:**

\[
\begin{align*}
\forall v \in V \setminus \{s, t\}, \quad f(v) &= \sum_{e : u \rightarrow v} f^+(e) - \sum_{e : v \rightarrow u} f^-(e) \\
\end{align*}
\]

\( f(s) = 0 \quad f(t) = |f| \leq c(e) \)

**lem:**
- given network, find max flow

---

**def:** a **network** \( G = (V, E) \), \( s \neq t \in V \), \( c : E \rightarrow \mathbb{R}_{\geq 0} \)

An **(s,t)-cut** is a partition \( V = S \cup T \) \( s \in S \), \( t \notin T \)

The **capacity of the cut** is \( |C| = c(S, T) = \sum_{e : s \rightarrow t} c(e) \)

**ques:**
- given network find min capacity cut.

**prop:**
- \( (s,t) \) then, \( c(S, T) = |f| \leq |C| \)

**then:**
- \( f \) is a max flow.

**note:** flows and cuts are dual objects: \( c \)-min flow = \( c \)-max cut

**lem:**
- network \( G \) has no bidirected edges

**sketch:**
- show that \( f \) is a **value preserving bijection** of \( |C| \) and \( |C| \)

**ques:**
- what algorithmic paradigms to use?

- divide and conquer?
- dynamic programming?
- randomized algo?

is ready? 

Max flow = 2C

An edge (u,v) is selected only if at least one of the following conditions is met:
- For forward edges: there exists a flow path from s to v with capacity at least f(u,v) + p(p)
- For backward edges: there exists a flow path from v to s with capacity at least f(u,v) - p(p)

The residual network Gf = (V, Ef) is obtained from G by removing all edges with capacity 0.

An augmenting path is a (simple) s-t path p in Gf.

The capacity of p is defined as c(p) = \min \{ c(e) | e \in p \}

A flow f is increased by p if p is an augmenting path in Gf.

If p is a simple s-t path, assign flow p to each edge in p.

The value of the flow f is increased by \sum \{ p(e) | e \in p \}.

A flow f is maximum if there is no augmenting path in G.

key: 2 - G has no bidirectional edge, but G has no parallel edge.

- G has no edge = G has no edge.

Def: An augmenting path is a (simple) s-t path p in Gf.

It's capacity is |p| = \min_{e \in p} c(e).

Lemma: If p is an augmenting path in Gf, then f+p is a flow.

We value |f+p| = |f| + |p|.

If p is a path from s to t that satisfies the capacity constraints, we assign flow p to each edge in p.

\text{capacity constraints:}
- (u,v) \in E: c(u,v) \geq f(u,v) + p(p)
- (v,u) \in E: c(v,u) \geq f(u,v) - p(p)

For any simple s-t path p in Gf, we have:
- 0 \leq f(u,v) - p(p) \leq c(u,v)
- 0 \leq f(u,v) + p(p) \leq c(u,v)

Conservative: p is simple s-t path, hence v \notin S, but p enters v and p goes in the

hence, \forall s,v \in V \setminus \{s,t\},

\text{value} = \text{value increase by } p(p)

(f+p)^{\text{out}}(s) = f^{\text{out}}(s) + |p|

(f+p)^{\text{in}}(s) = f^{\text{in}}(s)
Algorithm (Ford, Fulkerson 1956):

Network \( G = (V, E) \), \( s \neq t \in V \), capacities \( c \)

Input: \( s \leq t \in E \)

Initialize: \( G_f \) by augmenting

While simple \( s \)-\( t \) path \( p \) in \( G_f \)

Compute \( l = \min_{e \in p} c_e \)

\( f' = f + p \leq O(n) \) shallow/successive

\( G_f = G_f - p \leq O(n) \) edge to update

Return \( f \)

\[ \text{Q: } \text{Bipartite?} \]

\[ \text{Q: } \text{Is } f \text{ a minimum?} \]

\[ \text{Q: } \text{Is } G_f \text{ valid?} \]

\[ \text{Q: } \text{Is } f \text{ a feasible flow?} \]

\[ \text{Q: } \text{Is } f \text{ a maximum flow?} \]

\[ \text{Q: } \text{What does a terminal "look like"?} \]

\[ \text{Q: } \text{Is } f \text{ a flow of max value?} \]

\[ \text{Q: } \text{Does a cut exist?} \]

\( f \) is flow of max value if \( S \cup \{s\} \) is a cut.
\[ \text{no } S \to T \text{ path in } G \Rightarrow \text{no edges } S \to T \text{ in } G \]

\[ \Rightarrow \begin{cases} f(u,v) = c(u,v) & (u,v) \in E \\ f(u,v) = 0 & (u,v) \notin E \end{cases} \]

\[ \text{Lemma: } |f| = \sum_{u \in S, v \in T} f(u,v) - \sum_{u \in S, v \in T} f(v,u) \]

\[ = c(S,T) = 0 \]

\[ \text{Corollary: } c(S,T) = c(S,v) \]

\[ \text{Proof: } S = \{ u \in V : \text{exists } u \to v \text{ in } G \} \]

\[ \text{with } G \text{ of integral capacities } \Rightarrow \max |f| = \text{max } c(S,v) \]

\[ \text{max flow = min cut still holds } \]

\[ \text{Ford-Fulkerson actually run in } O(m \log C) \text{ time.} \]

\[ \text{In } O(m C) \text{ time, } \max \text{flow value } \]

\[ \sum_{e \in E} c(e) \leq c(S,v) \]

\[ \text{Given } C(e) \text{ in binary representation, } \]

\[ \text{but input size is } O(m \log C) \]

\[ \text{hence: FF run 'poly time' else if } C \text{ is large!} \]

\[ \text{FF can achieve } (1/2) \text{ iteration, e.g. } \]

\[ \text{Q: polynomial time also? } \]

\[ \text{Lemma: } \text{pres-S edge WID} \]

\[ \text{today: Ford-Fulkerson} \]

\[ \text{next: max flow algo} \]

\[ \text{application of max flow} \]