CMPSC 465: LECTURE XX

Max-flow Min-cut Theorem

Ke Chen

October 15, 2025

The Ford-Fulkerson algorithm

Input: Flow network $G = (V, E, c)$
Output: Maximum flow f

Augment(P, f)

- $b = bottleneck(P, f)$
- **foreach** *edge* $(x, y) \in P$ **do**
  - **if** (x, y) *is a forward edge* **then** $f(x, y) = f(x, y) + b$
  - **if** (x, y) *is a backward edge* **then** $f(x, y) = f(x, y) - b$
- **return** f

Ford-Fulkerson(G)

- Initialize $f(e) = 0$ for all $e \in E$
- $G_f = G$
- **while** *there is an* $s - t$ *path* P *in* G_f **do**
  - $f =$ Augment(P, f)
  - Build new residual graph G_f
- Output f

The Ford-Fulkerson algorithm

Input: Flow network $G = (V, E, c)$

Output: Maximum flow f

Augment(P, f)

$b = bottleneck(P, f)$

foreach *edge* $(x, y) \in P$ **do**

  if (x, y) *is a forward edge* **then** $f(x, y) = f(x, y) + b$

  if (x, y) *is a backward edge* **then** $f(x, y) = f(x, y) - b$

return f

Ford-Fulkerson(G)

Initialize $f(e) = 0$ for all $e \in E$

$G_f = G$

while *there is an* $s - t$ *path* P *in* G_f **do**

  $f =$ Augment(P, f)

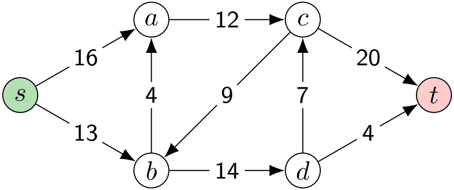
  Build new residual graph G_f

Output f

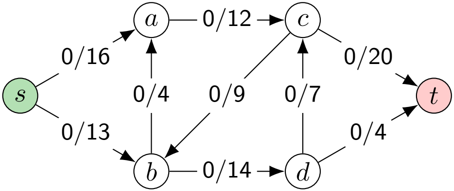
Correctness? Time complexity?

The Ford-Fulkerson algorithm

G_f :

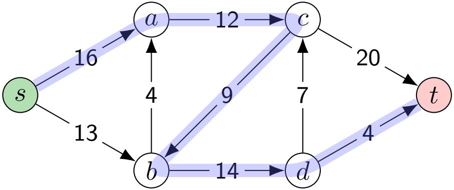


f :

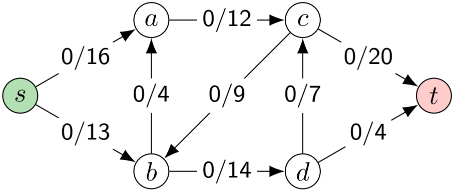


The Ford-Fulkerson algorithm

G_f :

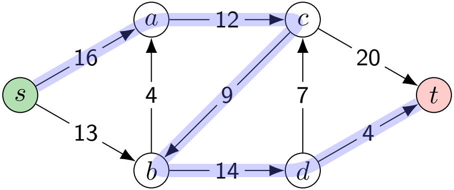


f :

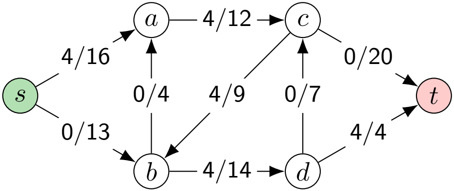


The Ford-Fulkerson algorithm

G_f :

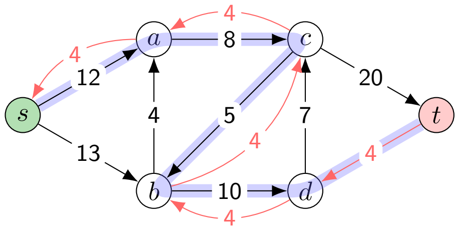


f :

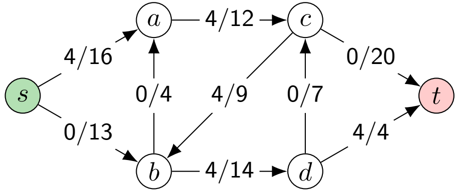


The Ford-Fulkerson algorithm

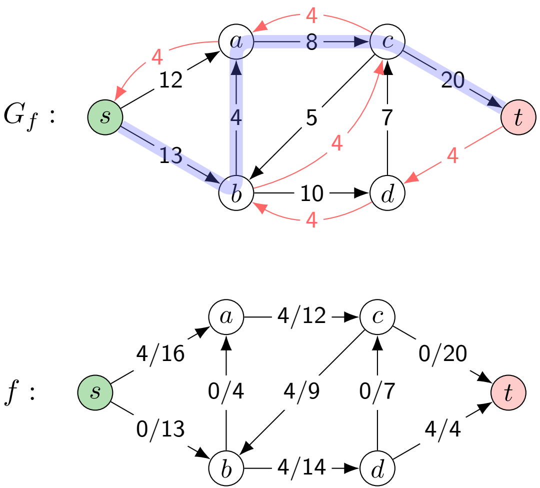
G_f :



f :

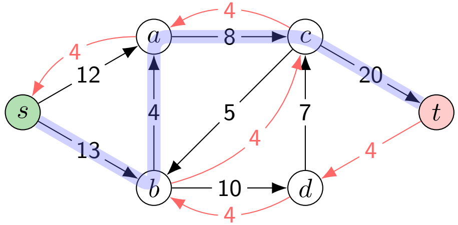


The Ford-Fulkerson algorithm

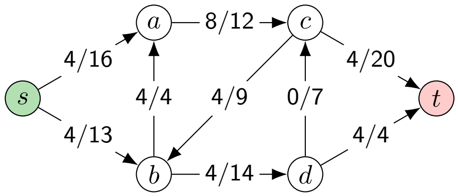


The Ford-Fulkerson algorithm

G_f :

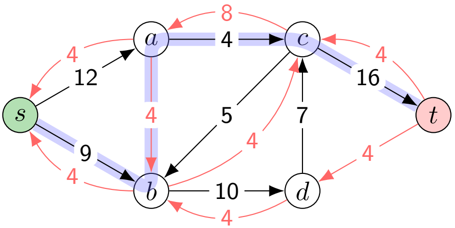


f :

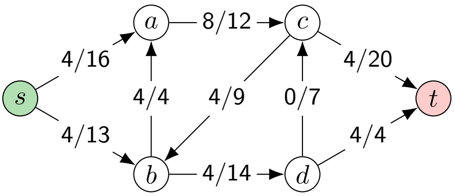


The Ford-Fulkerson algorithm

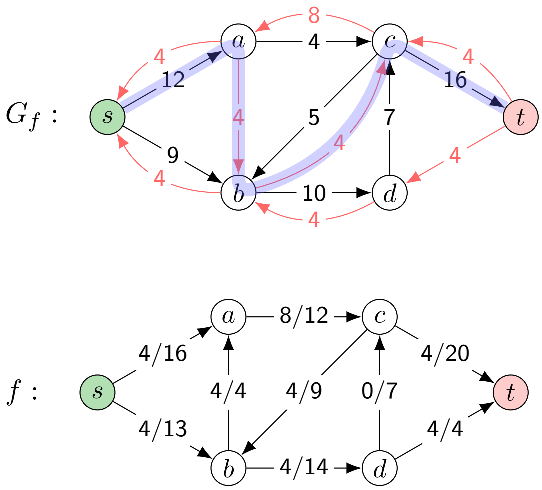
G_f :



f :

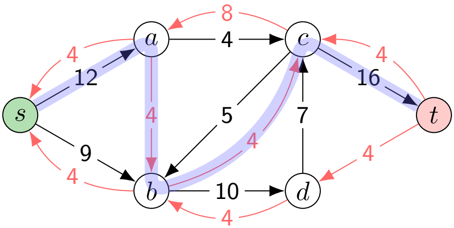


The Ford-Fulkerson algorithm

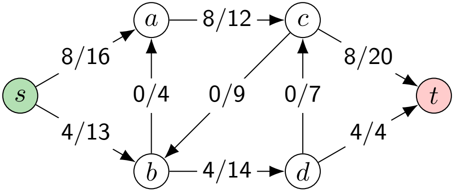


The Ford-Fulkerson algorithm

G_f :

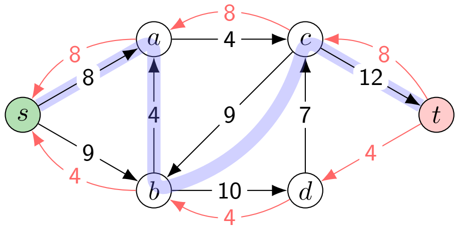


f :

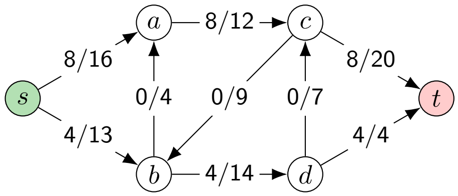


The Ford-Fulkerson algorithm

G_f :

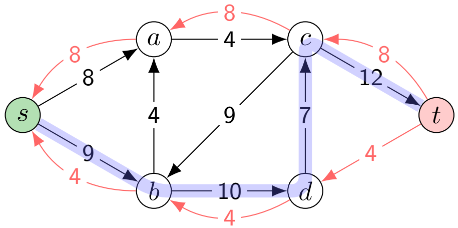


f :

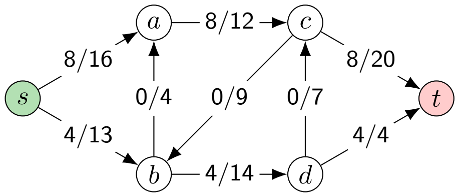


The Ford-Fulkerson algorithm

G_f :

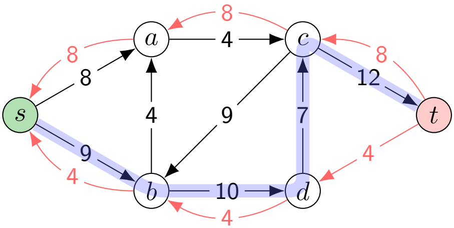


f :

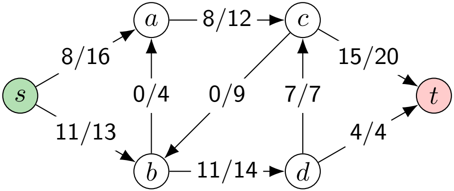


The Ford-Fulkerson algorithm

G_f :

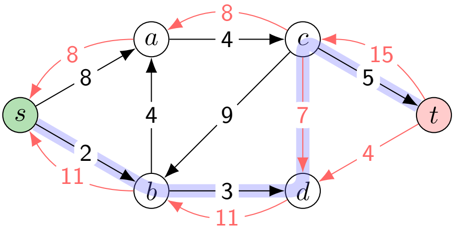


f :

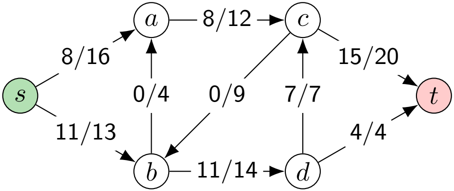


The Ford-Fulkerson algorithm

G_f :

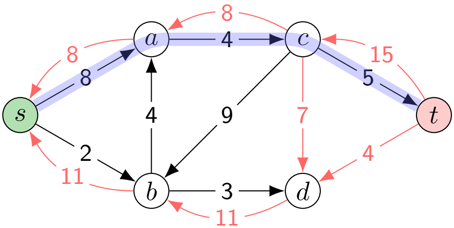


f :

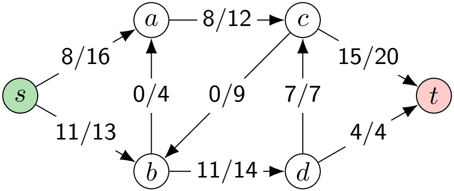


The Ford-Fulkerson algorithm

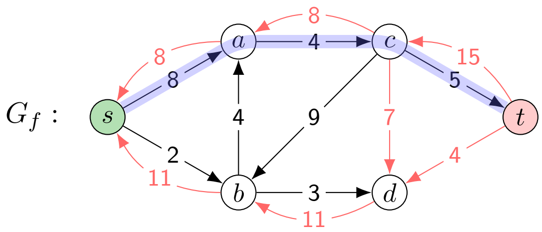
G_f :

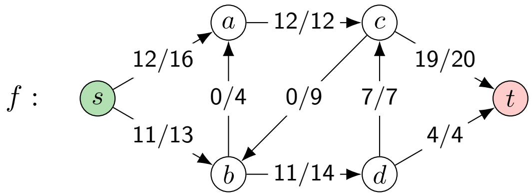


f :

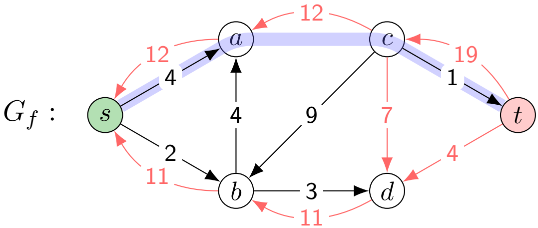


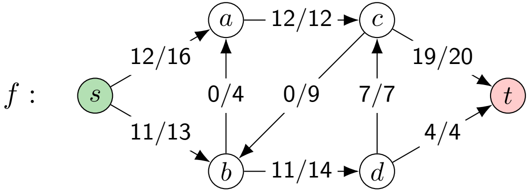
The Ford-Fulkerson algorithm





The Ford-Fulkerson algorithm





Augmenting path

Fact 1 If P is an augmenting $s - t$ path in G_f, then $f' = \text{Augment}(P, f)$ is a valid flow.

Augmenting path

Fact 1 If P is an augmenting $s - t$ path in G_f, then $f' = \text{Augment}(P, f)$ is a valid flow.

Idea Let $b = bottleneck(P, f)$, we only need to check capacity conditions and flow conservation constraints for f'.

Augmenting path

Fact 1 If P is an augmenting $s - t$ path in G_f, then $f' = \text{Augment}(P, f)$ is a valid flow.

Idea Let $b = \mathit{bottleneck}(P, f)$, we only need to check capacity conditions and flow conservation constraints for f'.

Proof:

Capacity condition holds by the construction of the residual graph G_f.

- $e \notin P$, then the flow on e does not change, $f'(e) = f'(e) \leq c(e)$.
- $e \in P$ is a forward edge, $f'(e) = f(e) + b \leq f(e) + (c(e) - f(e)) \leq c(e)$.
- $e \in P$ is a backward edge, $f'(e) = f(e) - b \geq f(e) - f(e) = 0$.

Augmenting path

Fact 1 If P is an augmenting $s - t$ path in G_f, then $f' = \text{Augment}(P, f)$ is a valid flow.

Idea Let $b = bottleneck(P, f)$, we only need to check capacity conditions and flow conservation constraints for f'.

Proof:

Capacity condition holds by the construction of the residual graph G_f.

- $e \notin P$, then the flow on e does not change, $f'(e) = f'(e) \leq c(e)$.
- $e \in P$ is a forward edge, $f'(e) = f(e) + b \leq f(e) + (c(e) - f(e)) \leq c(e)$.
- $e \in P$ is a backward edge, $f'(e) = f(e) - b \geq f(e) - f(e) = 0$.

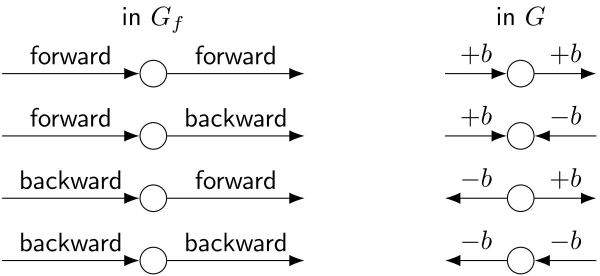
Augmenting path

Fact 1 If P is an augmenting $s - t$ path in G_f, then $f' = \text{Augment}(P, f)$ is a valid flow.

Idea Let $b = bottleneck(P, f)$, we only need to check capacity conditions and flow conservation constraints for f'.

Proof:

Flow conservation has four possible cases for vertices in P:



□

Running time analysis

Recall that we assume all capacities are integers.

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + bottleneck(P)$ and therefore $v(f') \geq v(f) + 1$.

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + bottleneck(P)$ and therefore $v(f') \geq v(f) + 1$.

Proof:

- We already know that f' is a valid flow (Fact 1).

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + bottleneck(P)$ and therefore $v(f') \geq v(f) + 1$.

Proof:

- We already know that f' is a valid flow (Fact 1).
- P is a simple path from s to t, so the first edge out of s must be a forward edge (why?).

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + \mathit{bottleneck}(P)$ and therefore $v(f') \geq v(f) + 1$.

Proof:

- We already know that f' is a valid flow (Fact 1).
- P is a simple path from s to t, so the first edge out of s must be a forward edge (why?).
- Flow on this edge increased by $\mathit{bottleneck}(P)$, so $v(f')$ increased by the same amount. $\square$

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + \mathit{bottleneck}(P)$ and therefore $v(f') \geq v(f) + 1$.

Proof:

- We already know that f' is a valid flow (Fact 1).
- P is a simple path from s to t, so the first edge out of s must be a forward edge (why?).
- Flow on this edge increased by $\mathit{bottleneck}(P)$, so $v(f')$ increased by the same amount. $\square$

Therefore, Ford-Fulkerson performs at most $C = \sum_{e \text{ out of } s} c(e)$ iterations.

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + \mathit{bottleneck}(P)$ and therefore $v(f') \geq v(f) + 1$.

Proof:

- We already know that f' is a valid flow (Fact 1).
- P is a simple path from s to t, so the first edge out of s must be a forward edge (why?).
- Flow on this edge increased by $\mathit{bottleneck}(P)$, so $v(f')$ increased by the same amount. $\square$

Therefore, Ford-Fulkerson performs at most $C = \sum_{e \text{ out of } s} c(e)$ iterations. Overall running time: $O(C(|V| + |E|))$.

Running time analysis

Recall that we assume all capacities are integers.

Fact 2 In every step of the algorithm, the flow and residual capacities are integers.

Fact 3 If f is a flow in G, P is an augmenting path in G_f, and $f' = \text{Augment}(P, f)$, then $v(f') = v(f) + bottleneck(P)$ and therefore $v(f') \geq v(f) + 1$.

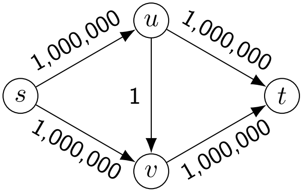
Proof:

- We already know that f' is a valid flow (Fact 1).
- P is a simple path from s to t, so the first edge out of s must be a forward edge (why?).
- Flow on this edge increased by $bottleneck(P)$, so $v(f')$ increased by the same amount. $\square$

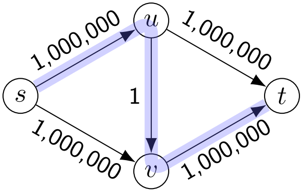
Therefore, Ford-Fulkerson performs at most $C = \sum_{e \text{ out of } s} c(e)$ iterations. Overall running time: $O(C(|V| + |E|))$.

Not polynomial in the input size!

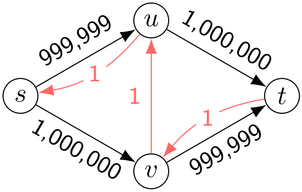
Running time analysis



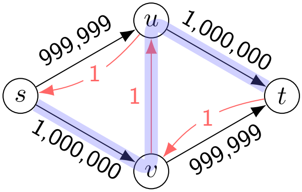
Running time analysis



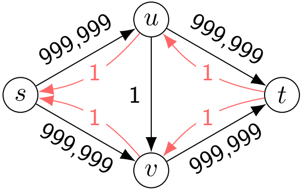
Running time analysis



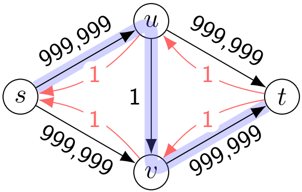
Running time analysis



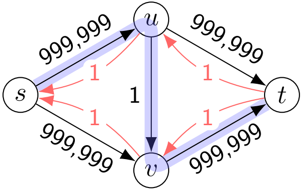
Running time analysis



Running time analysis

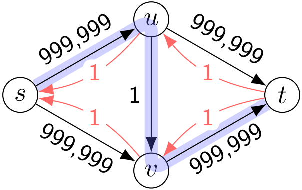


Running time analysis



Can we do better?

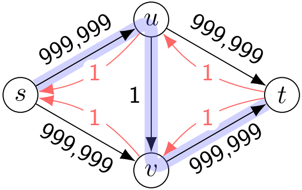
Running time analysis



Can we do better?

- Find the augmenting path with the largest bottleneck: $O(|E| \cdot \log C \cdot (|V| + |E|) \log |E|)$.

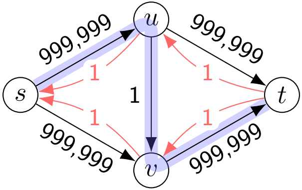
Running time analysis



Can we do better?

- Find the augmenting path with the largest bottleneck: $O(|E| \cdot \log C \cdot (|V| + |E|) \log |E|)$. Already polynomial!

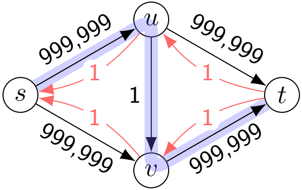
Running time analysis



Can we do better?

- Find the augmenting path with the largest bottleneck: $O(|E| \cdot \log C \cdot (|V| + |E|) \log |E|)$. Already polynomial!
- Find the augmenting path with the smallest number of edges (Edmonds-Karp (1972)): $O(|V| \cdot |E|^2)$.

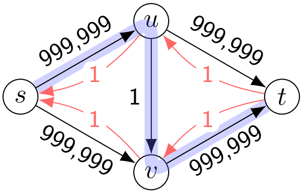
Running time analysis



Can we do better?

- Find the augmenting path with the largest bottleneck: $O(|E| \cdot \log C \cdot (|V| + |E|) \log |E|)$. Already polynomial!
- Find the augmenting path with the smallest number of edges (Edmonds-Karp (1972)): $O(|V| \cdot |E|^2)$.
- Dinitz (1970): $O(|V|^2 \cdot |E|)$.

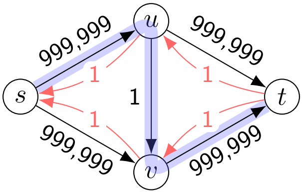
Running time analysis



Can we do better?

- Find the augmenting path with the largest bottleneck: $O(|E| \cdot \log C \cdot (|V| + |E|) \log |E|)$. Already polynomial!
- Find the augmenting path with the smallest number of edges (Edmonds-Karp (1972)): $O(|V| \cdot |E|^2)$.
- Dinitz (1970): $O(|V|^2 \cdot |E|)$.
- Orlin (2013): $O(|V| \cdot |E|)$.

Running time analysis



Can we do better?

- Find the augmenting path with the largest bottleneck: $O(|E| \cdot \log C \cdot (|V| + |E|) \log |E|)$. Already polynomial!
- Find the augmenting path with the smallest number of edges (Edmonds-Karp (1972)): $O(|V| \cdot |E|^2)$.
- Dinitz (1970): $O(|V|^2 \cdot |E|)$.
- Orlin (2013): $O(|V| \cdot |E|)$.
- And many more...

Cuts in graph

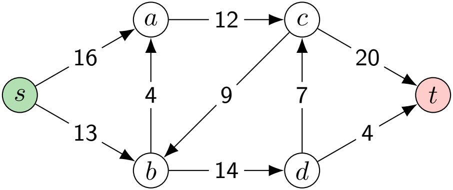
Definition An $s - t$ cut of $G = (V, E)$ is a partition of vertices (S, T) where $s \in S$, $t \in T = V - S$.

The capacity of the cut (S, T) is the total capacity out of S, denoted by $c(S, T)$.

Cuts in graph

Definition An $s - t$ cut of $G = (V, E)$ is a partition of vertices (S, T) where $s \in S$, $t \in T = V - S$.

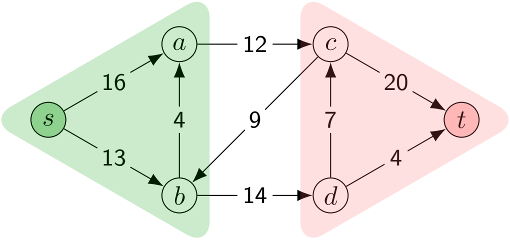
The capacity of the cut (S, T) is the total capacity out of S, denoted by $c(S, T)$.



Cuts in graph

Definition An $s - t$ cut of $G = (V, E)$ is a partition of vertices (S, T) where $s \in S$, $t \in T = V - S$.

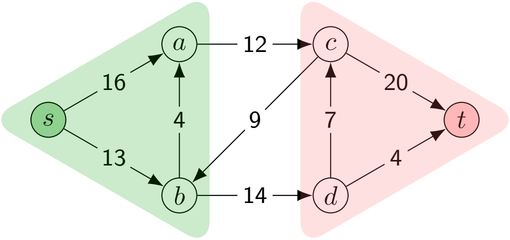
The capacity of the cut (S, T) is the total capacity out of S, denoted by $c(S, T)$.



Cuts in graph

Definition An $s - t$ cut of $G = (V, E)$ is a partition of vertices (S, T) where $s \in S$, $t \in T = V - S$.

The capacity of the cut (S, T) is the total capacity out of S, denoted by $c(S, T)$.

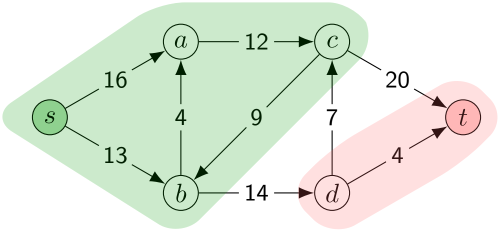


$$c(S, T) = 26$$

Cuts in graph

Definition An $s-t$ cut of $G=(V,E)$ is a partition of vertices (S, T) where $s \in S$, $t \in T = V - S$.

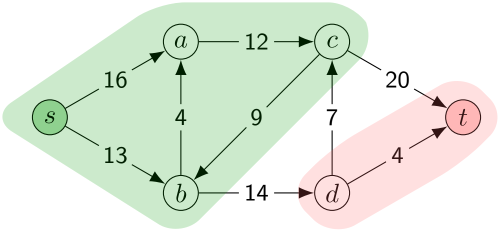
The capacity of the cut (S,T) is the total capacity out of S, denoted by $c(S,T)$.



Cuts in graph

Definition An $s - t$ cut of $G = (V, E)$ is a partition of vertices (S, T) where $s \in S$, $t \in T = V - S$.

The capacity of the cut (S, T) is the total capacity out of S, denoted by $c(S, T)$.



$$c(S, T) = 34$$

Flow vs Cut

Fact 4 The value of any $s - t$ flow f is bounded from above by the capacity of any $s - t$ cut (S, T): $v(f) \leq c(S, T)$.

Flow vs Cut

Fact 4 The value of any $s-t$ flow f is bounded from above by the capacity of any $s-t$ cut (S,T): $v(f) \leq c(S,T)$.

Idea Any flow depart from $s \in S$ needs to cross the cut (S,T) to get to $t \in T$.

Flow vs Cut

Fact 4 The value of any $s-t$ flow f is bounded from above by the capacity of any $s-t$ cut (S,T): $v(f) \leq c(S,T)$.

Idea Any flow depart from $s \in S$ needs to cross the cut (S,T) to get to $t \in T$.

For any vertex v, let $f^{\text{out}}(v) = \sum_{e=(v,w)} f(e)$ be the total flow out of v.

Let $f^{\text{in}}(v) = \sum_{e=(u,v)} f(e)$ be the total flow into v. Then

Flow vs Cut

Fact 4 The value of any $s - t$ flow f is bounded from above by the capacity of any $s - t$ cut (S, T): $v(f) \le c(S, T)$.

Idea Any flow depart from $s \in S$ needs to cross the cut (S, T) to get to $t \in T$.

For any vertex v, let $f^{\text{out}}(v) = \sum_{e=(v,w)} f(e)$ be the total flow out of v.

Let $f^{\text{in}}(v) = \sum_{e=(u,v)} f(e)$ be the total flow into v. Then

$$v(f) = \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v) \right)$$

Flow vs Cut

Fact 4 The value of any $s-t$ flow f is bounded from above by the capacity of any $s-t$ cut (S,T): $v(f) \le c(S,T)$.

Idea Any flow depart from $s \in S$ needs to cross the cut (S,T) to get to $t \in T$.

For any vertex v, let $f^{\text{out}}(v) = \sum_{e=(v,w)} f(e)$ be the total flow out of v.

Let $f^{\text{in}}(v) = \sum_{e=(u,v)} f(e)$ be the total flow into v. Then

$$\begin{aligned} v(f) &= \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) \\ &= \sum_{e=(v,w), v \in S} f(e) - \sum_{e=(u,v), v \in S} f(e) \end{aligned}$$

Flow vs Cut

Fact 4 The value of any $s-t$ flow f is bounded from above by the capacity of any $s-t$ cut (S,T): $v(f) \le c(S,T)$.

Idea Any flow depart from $s \in S$ needs to cross the cut (S,T) to get to $t \in T$.

For any vertex v, let $f^{\text{out}}(v) = \sum_{e=(v,w)} f(e)$ be the total flow out of v.

Let $f^{\text{in}}(v) = \sum_{e=(u,v)} f(e)$ be the total flow into v. Then

$$\begin{aligned} v(f) &= \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v) \right) \\ &= \sum_{e=(v,w), v \in S} f(e) - \sum_{e=(u,v), v \in S} f(e) \\ &= \sum_{e=(v,w), v \in S, w \notin S} f(e) - \sum_{e=(u,v), u \notin S, v \in S} f(e) \end{aligned}$$

Flow vs Cut

Fact 4 The value of any $s-t$ flow f is bounded from above by the capacity of any $s-t$ cut (S,T): $v(f) \le c(S,T)$.

Idea Any flow depart from $s \in S$ needs to cross the cut (S,T) to get to $t \in T$.

For any vertex v, let $f^{\text{out}}(v) = \sum_{e=(v,w)} f(e)$ be the total flow out of v.

Let $f^{\text{in}}(v) = \sum_{e=(u,v)} f(e)$ be the total flow into v. Then

$$
\begin{aligned}
v(f) &= \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) \\
&= \sum_{e=(v,w), v \in S} f(e) - \sum_{e=(u,v), v \in S} f(e) \\
&= \sum_{e=(v,w), v \in S, w \notin S} f(e) - \sum_{e=(u,v), u \notin S, v \in S} f(e) \\
&\le c(S,T).
\end{aligned}
$$

Flow vs Cut

Fact 5 If for a flow f, there is no $s - t$ path in G_f, then all the nodes reachable from s and all the nodes that can reach t forms a cut (S, T) such that $v(f) = c(S, T)$.

Flow vs Cut

Fact 5 If for a flow f, there is no $s - t$ path in G_f, then all the nodes reachable from s and all the nodes that can reach t forms a cut (S, T) such that $v(f) = c(S, T)$.

Proof:

- For any edge $e = (v, w)$ out of S, we must have $c(e) = f(e)$, otherwise we could reach w in G_f.

Flow vs Cut

Fact 5 If for a flow f, there is no $s-t$ path in G_f, then all the nodes reachable from s and all the nodes that can reach t forms a cut (S,T) such that $v(f)=c(S,T)$.

Proof:

- For any edge $e=(v,w)$ out of S, we must have $c(e)=f(e)$, otherwise we could reach w in G_f.
- For any edge $e=(u,v)$ into S, we must have $f(e)=0$, otherwise we could reach u in G_f.

Flow vs Cut

Fact 5 If for a flow f, there is no $s - t$ path in G_f, then all the nodes reachable from s and all the nodes that can reach t forms a cut (S, T) such that $v(f) = c(S, T)$.

Proof:

- For any edge $e = (v, w)$ out of S, we must have $c(e) = f(e)$, otherwise we could reach w in G_f.
- For any edge $e = (u, v)$ into S, we must have $f(e) = 0$, otherwise we could reach u in G_f.

Therefore,

$$c(S, T) = \sum_{e \text{ out of } S} c(e) = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ into } S} f(e) = v(f).$$

$\square$

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).
- Value of any flow $\leq$ capacity of any cut (Fact 4).

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).
- Value of any flow $\leq$ capacity of any cut (Fact 4).
- So f must be a maximum flow, .

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).
- Value of any flow $\leq$ capacity of any cut (Fact 4).
- So f must be a maximum flow, and (S, T) is a minimum cut.

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).
- Value of any flow $\leq$ capacity of any cut (Fact 4).
- So f must be a maximum flow, and (S, T) is a minimum cut.

This is known as the max-flow min-cut theorem.

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).
- Value of any flow $\leq$ capacity of any cut (Fact 4).
- So f must be a maximum flow, and (S, T) is a minimum cut.

This is known as the max-flow min-cut theorem.

It has many applications such as Menger's Theorem in number of edge-disjoint paths, Hall's Theorem in bipartite matching, scheduling, baseball elimination, etc.

Max-flow min-cut

In summary, we know:

- Ford-Fulkerson finds a flow f where there is no $s - t$ path in G_f.
- This yields a cut (S, T) in G_f and $v(f) = c(S, T)$ (Fact 5).
- Value of any flow $\leq$ capacity of any cut (Fact 4).
- So f must be a maximum flow, and (S, T) is a minimum cut.

This is known as the max-flow min-cut theorem.

It has many applications such as Menger's Theorem in number of edge-disjoint paths, Hall's Theorem in bipartite matching, scheduling, baseball elimination, etc.

Take CMPSC 565 to learn more :)