NETWORK FLOW - FLOW NETWORKS

network = directed graph

$S$: supply: unbounded input

$T$: target: output limited only by network

Each edge has a capacity $\geq$ flow

(pipe cross-section, number of lanes, etc)

$\circ$ Vertices: $\Sigma$ input = $\Sigma$ output $\Rightarrow$ keep things flowing
\[ c(x,y) = \text{capacity of edge } \overrightarrow{xy} \quad \rightarrow \text{part of problem input} \]

\[ f(x,y) = \text{actual flow in edge } \overrightarrow{xy} \quad \rightarrow \text{TBD} \]

**Capacity constraint:** \[ 0 \leq f(x,y) \leq c(x,y) \]

**Flow conservation:** For all \( u \neq S, T \): \[ \sum_{\text{all } v} f(v,u) = \sum_{\text{all } v} f(u,v) \]
changed direction

$v$: should be removed

Every vertex should be on $\geq 1$ directed path $S \rightarrow T$.

Both directions allowed, but we can simulate.

Note: w/ 2 directions, one will always have zero flow.
Can we simplify this?
If any cycle has flow on all edges, we can get an equivalent solution with \( \geq 1 \) edge having zero flow.

Notice any edge \( \overrightarrow{vS} \) must be part of a cycle.

Any solution has an equivalent with \( f(v, S) = 0 \).
If any cycle has flow on all edges, we can get an equivalent solution with at least one edge having zero flow.

Notice any edge \( v \in S \) must be part of a cycle.

Any solution has an equivalent with \( f(v, S) = 0 \).
If any cycle has flow on all edges, we can get an equivalent solution with at least one edge having zero flow.

Notice any edge $\vec{vS}$ must be part of a cycle.

Any solution has an equivalent with $f(v, S) = 0$.
If any cycle has flow on all edges, we can get an equivalent solution with \( \geq 1 \) edge having zero flow.

Notice any edge \( \vec{S} \) must be part of a cycle.

Any solution has an equivalent with \( f(v, S) = 0 \) (intuitive... why send product back to limitless source?)

In any case assume \( S \nless \rightarrow T \) (can also resolve multisource/target networks)
Goal: maximize Flow VALUE = \sum_{all \, v} f(S,v) = \sum_{all \, v} f(v,T)

# capacities
# flows
Goal: maximize \[ \text{Flow Value} = \sum_{\text{all } v} f(S, v) = \sum_{\text{all } v} f(v, T) \]

# capacities
# flows
Goal: maximize flow value \( \sum_{\text{all } v} \mathcal{f}(S,v) = \sum_{\text{all } v} \mathcal{f}(v,T) \)

# capacities
# flows
Goal: maximize \[ \text{Flow Value} = \sum_{\forall \, i} f(S_i) = \sum_{\forall \, i} f(i, T) \]

# of capacities

# of flows
Goal: maximize \( \text{Flow Value} = \sum_{\text{all } v} f(S, v) = \sum_{\text{all } v} f(v, T) \)
Goal: maximize \( \text{Flow Value} = \sum_{all v} f(S, v) = \sum_{all v} f(v, T) \)

# capacities
# flows
Goal: maximize \( \text{Flow Value} = \sum_{\text{all } v} f(S,v) = \sum_{\text{all } v} f(v,T) \)

\[\begin{array}{c}
S \\
V_2 \\
V_1 \\
V_3 \\
V_4 \\
T
\end{array}\]

#: capacities

#: flows
Goal: maximize \( \text{Flow Value} = \sum_{v \in V} f(S, v) = \sum_{v \in V} f(v, T) \)

# capacities
# flows
Goal: maximize \( \text{Flow Value} = \sum_{\forall v} f(S,v) = \sum_{\forall v} f(v,T) \)

Graph with nodes and edges labeled with capacities and flows:

- Ideas:
  - Simplify cycles
  - Find positive paths from S to T
CUTS

- partition vertices into 2 groups.
- $S$ & $T$ separated.
- edges: either not cut, or cut once $ightarrow$ from $S$ to $T$, from $T$ to $S$
\[ \text{Flow}(C) = \sum \text{flow} - \sum \text{flow} \]

\[ = 11 + 1 + 11 + 15 - 12 - 7 \]
\[ \text{Flow}(C) = \sum \text{flow} - \sum \text{flow} \]
\[ = 11 + 1 + 1 + 15 - 12 - 7 = 19 \]

\[ \text{Flow}(C) = \text{Flow}(C_2) \]
\[ = 11 + 8 \]
\[ \text{Flow}(C) = \sum \text{flow} - \sum \text{flow} = 11 + 11 + 15 - 12 - 7 = 19 \]

\[ \text{Flow}(C) = \text{Flow}(C_2) = \text{Flow}(C_3) = 11 + 12 - 4 \]
\[
\text{Flow}(C) = \Sigma \text{flow} - \Sigma \text{flow} = 11 + 1 + 11 + 15 - 12 - 7 = 19
\]
Flow(C) = \sum \text{flow} - \sum \text{flow} \\
= 11 + 1 + 1 + 15 - 12 - 7 = 19 \\

Flow(C) = Flow(C_2) = Flow(C_3) = Flow(\text{any cut}) = \text{Flow value}
Flow(C) = \( \sum \text{flow} - \sum \text{flow} \)
= 11 + 11 + 15 - 12 - 7 = 19

Capacity(C) = \( \sum \text{capacity} \)
= 16 + 4 + 14 + 20 = 54

Flow(C) = Flow(C_2) = Flow(C_3) = Flow(any cut) = Flow value
\[ \text{Flow}(C) = \sum \text{flow} - \sum \text{flow} = 11 + 1 + 11 + 15 - 12 - 7 = 19 \]

\[ \text{Capacity}(C) = \sum \text{capacity} = 16 + 4 + 14 + 20 = 54 \]

(i) \[ \text{Flow}(C) = \text{Flow}(C_2) = \text{Flow}(C_3) = \text{Flow(}\text{any cut}\text{)} = \text{FLOW VALUE} \]

(ii) \[ \text{Flow}(C) \leq \text{Capacity}(C) \]

\[ \text{FLOW VALUE} \leq \text{Capacity}(C) \]
\[ \text{Flow}(C) = \sum \text{flow} - \sum \text{flow} \]
\[ = 11 + 1 + 11 + 15 - 12 - 7 = 19 \]

\[ \text{Capacity}(C) = \sum \text{capacity} \]
\[ = 16 + 4 + 14 + 20 = 54 \]

(i) \[ \text{Flow}(C) = \text{Flow}(C_2) = \text{Flow}(C_3) = \text{Flow}(\text{any cut}) = \text{FLOW VALUE}\]

(ii) \[ \text{Flow}(C) \leq \text{Capacity}(C) \]

\[ \text{FLOW VALUE} \leq \min \{ \text{Capacity} \} \]
\[ \text{MAX FLOW} \leq \text{MIN CUT} \]
Reverse flow
Unused capacities, if positive

Residual network (all potential flow changes)
Residual network $\rightarrow$ all potential flow changes

Blue edges = reverse flow = potential flow decrease
Black edges (pink labels) = potential flow increase

Every edge in network is "represented" here, at least once.
Any directed $S \rightarrow T$ path in the residual network means we can improve the solution.

$S \rightarrow V_2 \rightarrow V_3 \rightarrow T$: AUGMENTING PATH
The residual network also has a min. cut \( X \). (min capacity of all cuts)

This gives a "residual capacity" (=4) = max. increase in flow.
Actual network

Residual network & an augmenting path
Updated flow

Residual network reconstructed
Residual capacity = 0
no augmenting paths

Updated flow
Residual network reconstructed
If there exists an augmenting path, then flow can increase.

If MAX flow then no augmenting path.

If flow equals capacity of some cut, then we have MAX flow.

To prove:

If no augmenting path

Then flow equals capacity of some cut

Implies MAX flow = MIN cut
no augmenting path \[?\] \(\rightarrow\) flow equals capacity of some cut

In residual network, let \(X: \{\text{vertices reachable from } S\}\), \(Y = V - X\)

no augmenting path \(\rightarrow\) \(T\) is in \(Y\)

\(\Rightarrow\) \(X\) \& \(Y\) define a cut, \(C\)
no augmenting path \( \rightarrow \) flow equals capacity of some cut

Look at any edge \( \overline{pq} \) crossing \( C \) in flow network.

- \( \overline{pq} \) is not in residual.
  (otherwise \( q \) would be in \( X \))
no augmenting path $\Rightarrow$ flow equals capacity of some cut

Look at any edge $\overrightarrow{pq}$ crossing $C$ in flow network.

- $\overrightarrow{pq}$ is not in residual. (otherwise $q$ would be in $X$)

if $\overrightarrow{pq}$ in flow network then $\text{Flow}(\overrightarrow{pq}) = \text{Capacity}(\overrightarrow{pq})$
no augmenting path \( \rightarrow \) flow equals capacity of some cut

Look at any edge \( \overrightarrow{pq} \) crossing \( C \) in flow network.

- \( \overrightarrow{pq} \) is not in residual.
  (otherwise \( q \) would be in \( X \))

\[ \text{if } \overrightarrow{pq} \text{ in flow network then } \text{Flow}(\overrightarrow{pq}) = \text{Capacity}(\overrightarrow{pq}) \]

\[ \text{if } \overrightarrow{pq} \text{ in flow network then } \text{Flow}(\overrightarrow{pq}) = 0 \]
no augmenting path $\implies$ flow equals capacity of some cut

$$\text{Flow}(C) = \sum \text{flow} - \sum \text{flow}$$

$$\text{Capacity}(C) = \sum \text{capacity}$$

For edges crossing $C(x,y)$
- if $pq$ in flow network then $\text{Flow}(pq) = \text{Capacity}(pq)$
- if $pq$ in flow network then $\text{Flow}(pq) = 0$
FORD-FULKERSON method

- start w/ zero flow
- while augmenting path exists
  - increase flow on an augmenting path
    - find min-weight \( w \) on aug. path
    - for every edge \( pq \) on aug. path
      - if \( pq \) in flow network
        - add \( w \) to Flow(\( pq \))
      - else subtract \( w \) from Flow(\( pq \))

  - how many times?
  - how do we find one?

forms basis of more advanced algorithms (e.g. Edmonds-Karp)
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)

Residual graph
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)
#: capacity (normal edge)
#: flow
#: leftover (residual edge)
#: reverse (residual edge)
*: augmenting path

S → V₁ → V₃ → V₂ → V₄ → t

16 12 9 14 4
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)

ለማቅለያ (ደረጋulers path)

S → V₁ → V₃ → V₂ → V₄ → T

16 12 9 14 4
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)

Residual graph
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)
~: augmenting path

S \rightarrow v_2 \rightarrow v_1 \rightarrow v_3 \rightarrow t

13 4 8 20
# : capacity (normal edge)
#: flow
#: leftover (residual edge)
#: reverse (residual edge)

~*: augmenting path

S → v₂ → v₁ → v₃ → t

13 4 8 20
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)

Residual graph
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)

↗️: augmenting path

S → V₁ → V₂ → V₃ → t

12  4  4  16
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)

~*: augmenting path

S → V₁ → V₂ → V₃ → t

12 4 4 16
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)

Residual graph
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)

$\Rightarrow$ : augmenting path

$S \rightarrow V_2 \rightarrow V_4 \rightarrow V_3 \rightarrow t$
\#: capacity (normal edge)
\#: flow
\#: leftover (residual edge)
\#: reverse (residual edge)

\textbf{знак}: augmenting path

$S \rightarrow V_2 \rightarrow V_4 \rightarrow V_3 \rightarrow t$

9 10 7 12
#: capacity (normal edge)
#: flow
#: leftover (residual edge)
#: reverse (residual edge)

Residual graph
 #: capacity (normal edge)
 #: flow
 #: leftover (residual edge)
 #: reverse (residual edge)
# capacity (normal edge)
# flow
# leftover (residual edge)
# reverse (residual edge)

augmenting path

S → v₁ → v₃ → t

8 4 5
# = capacity (normal edge)
# = flow
# = leftover (residual edge)
# = reverse (residual edge)
*N* = augmenting path

S → V₁ → V₃ → t

8 4 5
# : capacity (normal edge)
# : flow
# : leftover (residual edge)
# : reverse (residual edge)

Residual graph
No augmenting path.

DONE
**Ford-Fulkerson Method**

While augmenting path exists, increase flow on an augmenting path.

- How many times?
- How do we find one?
  - Increases flow value.
  - $O(\text{path length})$

Arbitrarily?

\[ M = 10^6 \]
*Ford-Fulkerson* method

- while augmenting path exists
- increase flow on an augmenting path

> how many times?

> how do we find one?

- increases flow value.
  - $O(path \ length)$

**Arbitrarily?**

**Residual**

M = $10^6$
**FORD-FULKERSON method**

- While augmenting path exists, increase flow on an augmenting path.

How do we find one? Increases flow value in $O(|\text{path length}|)$.

Arbitrarily?

Residual

$M = 10^6$
**Ford-Fulkerson method**

- While augmenting path exists, increase flow on an augmenting path.

- How many times?
- How do we find one?
  - Flow value increases in $O(\text{path length})$.

Arbitrarily?

$M = 10^6$
**Ford-Fulkerson Method**

- While augmenting path exists
  - Increase flow on an augmenting path

- How many times?
  - How do we find one?
    - Increases flow value,
      - $O(\text{path length})$

Arbitrarily?

![Residual Graph](image)

$M = 10^6$
**Ford-Fulkerson method**

- While augmenting path exists,
  increase flow on an augmenting path

**How many times?**

- How do we find one?
  - Increases flow value
    - $O(\text{path length})$

**Arbitrarily?**

**Residual**

$M = 10^6$
**Ford-Fulkerson method**

- While augmenting path exists, increase flow on an augmenting path.

Arbitrarily? \( \Rightarrow \) \#iterations = \( M \)

(or even DFS)

\( M = 10^6 \)

How many times?

How do we find one?

Increases flow value. \( O(\text{path length}) \)

Residual

Etc.
FORD-FULKERSON method

- while augmenting path exists
  increase flow on an augmenting path

how many times?
how do we find one?

increases
  flow value.
  $O(path \ length)$

Arbitrarily? No. In fact if capacities are irrational
  there are networks causing non-termination
  & no convergence. (see links)

For integer capacities, $\# \text{iterations} \leq \text{MAX \ flow}$
For rational, transform to integer (multiply)
The Ford-Fulkerson method

- While augmenting path exists, increase flow on an augmenting path.

How many times?

Use BFS to find one, increases flow value. $O(path\ length)$.

Residual
**Ford-Fulkerson** method

- while augmenting path exists
  - increase flow on an augmenting path

How many times?

Use BFS to find one

increases flow value.

$O(\text{path length})$

Residual
**Ford-Fulkerson** method

- While augmenting path exists, increase flow on an augmenting path.

**How many times?**

Use BFS to find one, increases flow value. $O(path\ length)$.

**Residual**
Ford-Fulkerson method

- While augmenting path exists, increase flow on an augmenting path

How many times?

Use BFS to find one

\[ O(\text{path length}) \]

Residual

Done
Ford-Fulkerson method

- while augmenting path exists
  - increase flow on an augmenting path

how many times?

use BFS to find one: $O(E)$
- flow value increases
  - $O($path length$)$

Using BFS: Edmonds-Karp algorithm: $O(V \cdot E)$ iterations
  - so $O(V \cdot E^2)$ time

$O(V^2E)$ & $O(V^3)$ algorithms also exist.
Finding a Maximum Bipartite Matching

![Graph diagram](image)
Finding a Maximum Bipartite Matching

All capacities = 1
Finding a maximum bipartite matching

All capacities = 1 (integer)

- MAX FLOW = $O(V)$

iterations BFS

so $O(V \cdot E)$ time
Finding a maximum bipartite matching

- All capacities = 1 (integer)
- \( \text{MAX FLOW} = O(V) \) iterations
- \( \text{BFS} \)
- so \( O(V\cdot E) \) time

* it can be shown that integer input gives integer flow on edges