COL 351 : Analysis & Design of Algorithms

LECTURE 28

All Pairs Shortest Paths Problem & Introduction to Network Flow

Oct 14, 2026 | Rohit Vaish

ALL PAIRS SHORTEST PATHS PROBLEM

ALL PAIRS SHORTEST PATHS PROBLEM

input: A directed graph $G = (V, E)$,
a possibly negative length ℓ_e for each edge $e \in E$

ALL PAIRS SHORTEST PATHS PROBLEM

input: A directed graph $G = (V, E)$,
a possibly negative length ℓ_e for each edge $e \in E$

no distinguished "source" vertex

ALL PAIRS SHORTEST PATHS PROBLEM

input: A directed graph $G = (V, E)$,
a possibly negative length ℓ_e for each edge $e \in E$

output:

ALL PAIRS SHORTEST PATHS PROBLEM

input: A directed graph $G = (V, E)$,
a possibly negative length ℓ_e for each edge $e \in E$

output: (1) compute the length of the shortest $u - v$ path for all pairs of vertices $u, v \in V$, or

ALL PAIRS SHORTEST PATHS PROBLEM

input: A directed graph $G = (V, E)$,
a possibly negative length ℓ_e for each edge $e \in E$

output: (1) compute the length of the shortest $u - v$ path for all pairs of vertices $u, v \in V$, or

(2) output a negative cycle.

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source ?

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source ?

If non negative lengths $\Longrightarrow$ run Dijkstra n times

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$

- $O(n^2 \log n)$ sparse
- $O(n^3 \log n)$ dense

values in the output

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

Is $O(n^{3-\varepsilon})$ possible for any $\varepsilon > 0$?

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

Is $O(n^{3-\varepsilon})$ possible for any $\varepsilon > 0$? Open problem!

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

Is $O(n^{3-\varepsilon})$ possible for any $\varepsilon > 0$? Open problem!

until Oct 4th, 2026

Truly Subquadratic 3SUM and Truly Subcubic APSP via Triangles in Sparse Lopsided Graphs

Josh Alman
Columbia University

Virginia Vassilevska Williams
MIT

arXiv Oct 5th, 2026

Acknowledgments and Methodology

How the result was originally found and shared with the authors. An Anthropic employee used an internal research model to investigate open problems in the theory of cryptography. One of them was about cryptographic constructions based on the average-case hardness of Zero-k-Clique [LLV19, AHY25]. Claude was tasked with verifying and improving the constructions, but instead developed this algorithm, first for the average case, then for the worst case. The session used 16M output tokens with no human input.

Anthropic shared the algorithm with the authors in September 2026 under a confidentiality agreement, offered compensation, and provided access to the public version of Claude.

Shared algorithm. The algorithm that was shared with the authors is essentially the algorithm in Section 2, although presented differently and with other numerical parameters. A different reduction from Exact Triangle to the problem of Section 2 was also shared, although the authors did not include it here, and instead used known reductions to All-Edges Sparse Triangle, which suffice and strengthen the connection. The authors derived the data structure version in Section 4 and its connection with Hinted OMv.

Lean certification. After the paper was completely written, Anthropic used an internal research model to certify this paper's main results using the Lean 4 proof assistant with the Mathlib library. The formalization is available at `https://github.com/anthropics/formal-math/tree/main/3sum-apsp`. The following are formalized there: Theorem 19 (Exact Triangle in $O(n^{2.9983})$ time), Theorem 22 (3SUM in $O(n^{1.9992})$ time, and the $(\min,+)$-product and APSP in $O(n^{2.99942})$ time), and the Zero-Weight case of Corollary 39 (Zero-Weight k-Clique in $O(n^{k-0.0017\lfloor k/3\rfloor})$ time for every $k \geq 3$). All lemmas and prior results that these statements rely on are also formalized there.

Further acknowledgments. The authors used Claude to help with writing, figure creation, and checking mathematical details throughout this project. They have attempted to make the entire paper accessible and well-written, and take full responsibility for its contents.

The authors' contribution to this publication was done in their individual capacities, not in connection with their duties or responsibilities to Columbia University or to MIT. Any opinions expressed in this work are opinions of the authors; they are not official positions or endorsements of Columbia University or of MIT, and do not represent the research of the authors at Columbia University or MIT.

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

Is $O(n^{3-\varepsilon})$ possible for any $\varepsilon > 0$? Open problem!

Isn't it "obvious" that one needs $\Omega(n^3)$ time? Afterall, # edges/path $\sim O(n)$

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

Is $O(n^{3-\varepsilon})$ possible for any $\varepsilon > 0$? Open problem!

Isn't it "obvious" that one needs $\Omega(n^3)$ time? Afterall, # edges/path $\sim O(n)$

Maybe some computation is simultaneously relevant to multiple problems

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

Is $O(n^{3-\varepsilon})$ possible for any $\varepsilon > 0$? Open problem!

Isn't it "obvious" that one needs $\Omega(n^3)$ time? Afterall, # edges/path $\sim O(n)$

Maybe some computation is simultaneously relevant to multiple problems

remember matrix multiplication?

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

If possibly negative lengths

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

If possibly negative lengths $\Rightarrow$ run Bellman-Ford n times

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow$ $O(mn \log n)$ $\rightarrow$ $O(n^2 \log n)$ sparse

$\rightarrow$ $O(n^3 \log n)$ dense

If possibly negative lengths $\Rightarrow$ run Bellman-Ford n times

$\Rightarrow$ $O(mn^2)$

ALL PAIRS SHORTEST PATHS PROBLEM



Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

If possibly negative lengths $\Rightarrow$ run Bellman-Ford n times

$\Rightarrow O(mn^2)$ → $O(n^3)$ sparse

→ $O(n^4)$ dense

ALL PAIRS SHORTEST PATHS PROBLEM

🤔 Why not solve single source shortest path problem with every vertex as a source?

If non negative lengths $\Rightarrow$ run Dijkstra n times

$\Rightarrow O(mn \log n)$ → $O(n^2 \log n)$ sparse

→ $O(n^3 \log n)$ dense

If possibly negative lengths $\Rightarrow$ run Bellman-Ford n times

$\Rightarrow O(mn^2)$ → $O(n^3)$ sparse

→ $O(n^4)$ dense ☹️

FLOYD - WARSHALL ALGORITHM

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP
* Works even with negative edge lengths l_e

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP
* Works even with negative edge lengths l_e
* at least as good as n invocations of Bellman-Ford, even on sparse graphs

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP
* Works even with negative edge lengths l_e
* at least as good as n invocations of Bellman-Ford, even on sparse graphs
* for nonnegative edge lengths, competitive with n Dijkstra's in dense graphs

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP
* Works even with negative edge lengths l_e
* at least as good as n invocations of Bellman-Ford, even on sparse graphs
* for nonnegative edge lengths, competitive with n Dijkstra's in dense graphs
* Applications:

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP
* Works even with negative edge lengths l_e
* at least as good as n invocations of Bellman-Ford, even on sparse graphs
* for nonnegative edge lengths, competitive with n Dijkstra's in dense graphs
* Applications: ① finding transitive closure of binary relation

FLOYD - WARSHALL ALGORITHM

* $O(n^3)$ algorithm for APSP
* Works even with negative edge lengths l_e
* at least as good as n invocations of Bellman-Ford, even on sparse graphs
* for nonnegative edge lengths, competitive with n Dijkstra's in dense graphs
* Applications: (1) finding transitive closure of binary relation
  (2) Schulze voting rule



OPTIMAL SUBSTRUCTURE

OPTIMAL SUBSTRUCTURE

Recall: can be tricky to define ordering on subproblems in graph problems

OPTIMAL SUBSTRUCTURE

Recall: can be tricky to define ordering on subproblems in graph problems



Order the vertices $V = \{1, 2, \ldots, n\}$ arbitrarily

OPTIMAL SUBSTRUCTURE

Recall: can be tricky to define ordering on subproblems in graph problems



Order the vertices $V = \{1, 2, \ldots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \ldots, k\}$

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



Order the vertices $V = \{1, 2, \ldots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \ldots, k\}$

Lemma: Suppose graph G has no negative cycle.

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



Order the vertices $V = \{1, 2, \ldots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \ldots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



Order the vertices $V = \{1, 2, \ldots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \ldots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



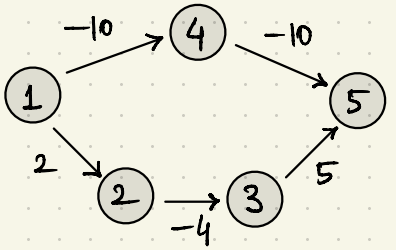
Order the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \dots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Example: $i=1$, $j=5$



OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



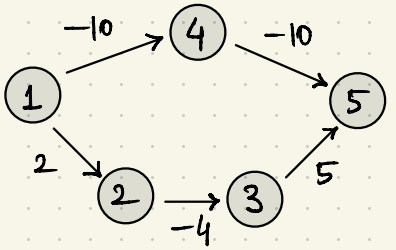
Order the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \dots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Example: $i=1$, $j=5$



$k \in \{0, 1, 2\}$:

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



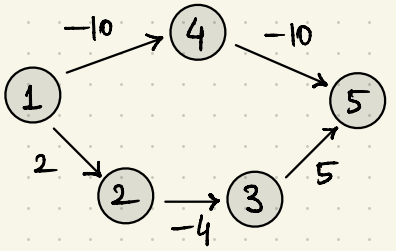
Order the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \dots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Example: $i=1$, $j=5$



$k \in \{0, 1, 2\}$: No path

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



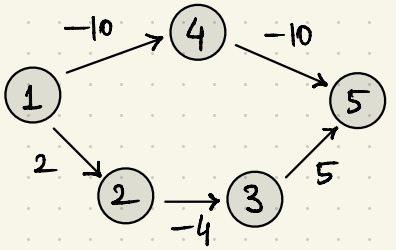
Order the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \dots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Example: $i=1$, $j=5$



$k \in \{0, 1, 2\}$: No path

$k = 3$:

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



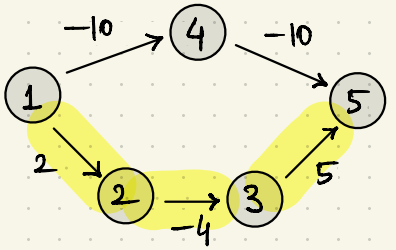
Order the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \dots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Example: $i=1$, $j=5$



$k \in \{0, 1, 2\}$: No path

$k = 3$: $1 \to 2 \to 3 \to 5$

OPTIMAL SUBSTRUCTURE

Recall: Can be tricky to define ordering on subproblems in graph problems



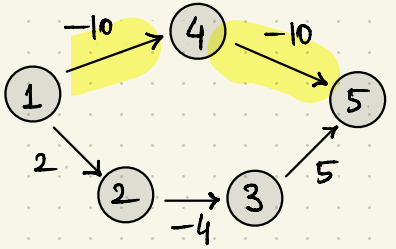
Order the vertices $V = \{1, 2, \dots, n\}$ arbitrarily

Let $V^{(k)} := \{1, 2, \dots, k\}$

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Example: $i=1$, $j=5$



$k \in \{0,1,2\}$: No path

$k = 3$: $1 \to 2 \to 3 \to 5$

$k = 4$: $1 \to 4 \to 5$

OPTIMAL SUBSTRUCTURE

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

OPTIMAL SUBSTRUCTURE

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

OPTIMAL SUBSTRUCTURE

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k-1)}$.

OPTIMAL SUBSTRUCTURE

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k-1)}$.

Case II: If k is internal to P

OPTIMAL SUBSTRUCTURE

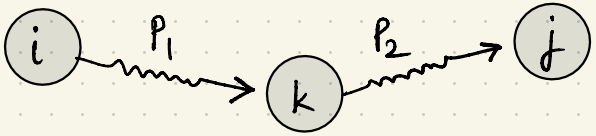
Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k-1)}$.

Case II: If k is internal to P



OPTIMAL SUBSTRUCTURE

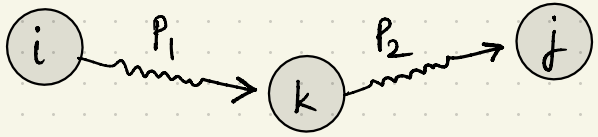
Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k-1)}$.

Case II: If k is internal to P



cycle-free path

OPTIMAL SUBSTRUCTURE

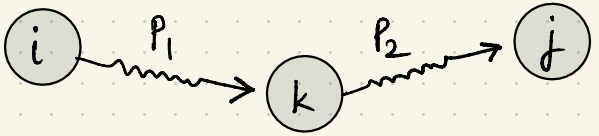
Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k-1)}$.

Case II: If k is internal to P



$\Rightarrow P_1$ is the shortest (cycle-free) $i-k$ path with all internal nodes in $V^{(k-1)}$

OPTIMAL SUBSTRUCTURE

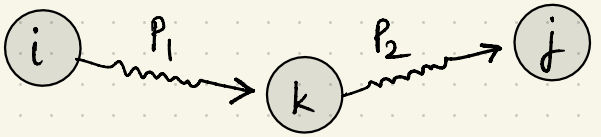
Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and prefix length k.

$P :=$ shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k)}$.

Case I: If k is not internal to P.

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with all internal nodes in $V^{(k-1)}$.

Case II: If k is internal to P



$\Rightarrow$ P_1 is the shortest (cycle-free) $i-k$ path with all internal nodes in $V^{(k-1)}$, and

P_2 is the shortest (cycle-free) $k-j$ path with all internal nodes in $V^{(k-1)}$.

A digression ...

WHY NOT USE EDGE BUDGET AGAIN?

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

Case I: If P has at most $(k-1)$ edges

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

Case I: If P has at most $(k-1)$ edges

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with at most $k-1$ edges

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

Case I: If P has at most $(k-1)$ edges

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with at most $k-1$ edges

Case II: If P has k edges

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

Case I: If P has at most $(k-1)$ edges

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with at most $k-1$ edges

Case II: If P has k edges



WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

Case I: If P has at most $(k-1)$ edges

$\Rightarrow$ P is the shortest (cycle-free) $i-j$ path with at most $k-1$ edges

Case II: If P has k edges



$\Rightarrow P'$ is the shortest (cycle-free) $i-h$ path with at most $k-1$ edges

WHY NOT USE EDGE BUDGET AGAIN?

Lemma: Suppose graph G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and edge budget k.

$P :=$ shortest (cycle-free) $i-j$ path with at most k edges.

Case I: If P has at most $(k-1)$ edges

$\Rightarrow P$ is the shortest (cycle-free) $i-j$ path with at most $k-1$ edges

Case II: If P has k edges



$\Rightarrow P'$ is the shortest (cycle-free) $i-h$ path with at most $k-1$ edges

Check: For each i, j, $O(1 + \text{in-deg}(j))$ time per update $\Rightarrow O(n^4)$ ☹

Back to prefixes

The Recurrence

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i, j, k]$ = length of shortest i–j path with all internal vertices in $V^{(k)}$.

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i, j, k]$ = length of shortest $i-j$ path with all internal vertices in $V^{(k)}$.

Base cases:

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i,j,k]$ = length of shortest i–j path with all internal vertices in $V^{(k)}$.

Base cases: What is $A[i,j,0]$ if

* $i = j$?
* $(i,j) \in E$?
* $i \neq j$ and $(i,j) \notin E$?

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i,j,k]$ = length of shortest i–j path with all internal vertices in $V^{(k)}$.

Base cases: What is $A[i,j,0]$ if

* $i = j$? 0 (empty path)
* $(i,j) \in E$?
* $i \neq j$ and $(i,j) \notin E$?

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i,j,k]$ = length of shortest i–j path with all internal vertices in $V^{(k)}$.

Base cases: What is $A[i,j,0]$ if

* $i = j$? 0 (empty path)
* $(i,j) \in E$? ℓ_{ij}
* $i \neq j$ and $(i,j) \notin E$?

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i,j,k]$ = length of shortest i–j path with all internal vertices in $V^{(k)}$.

Base cases: What is $A[i,j,0]$ if

* $i=j$? 0 (empty path)
* $(i,j) \in E$? ℓ_{ij}
* $i \neq j$ and $(i,j) \notin E$? $+\infty$ (no path)

THE RECURRENCE

Setup: Let A be a 3D array (indexed by i, j, k)

Intent: $A[i, j, k]$ = length of shortest i–j path with all internal vertices in $V^{(k)}$.

Recurrence: For all $i \in V$, $j \in V$ and $k \leq |V|$,

$$A[i, j, k] = \min\{A[i, j, k-1],\ A[i, k, k-1] + A[k, j, k-1]\}.$$

FLOYD - WARSHALL ALGORITHM

$A :=$ $n \times n \times (n+1)$ size array // indexed by i, j, k

// base cases

for all $i, j \in V$: $A[i, j, 0] = \begin{cases} 0 & \text{if } i = j \\ \ell_{ij} & \text{if } (i,j) \in E \\ +\infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$

// recurrence

```
for k = 1 to n
    for i ∈ V
        for j ∈ V
```

$$A[i, j, k] = \min \begin{cases} A[i, j, k-1] & \text{// Case I} \\ A[i, k, k-1] + A[k, j, k-1] & \text{// Case II} \end{cases}$$

Running time: $O(1)$ per subproblem, $O(n^3)$ overall.

FAQs

FAQs

(1) What about graphs with negative cycles ?

FAQs

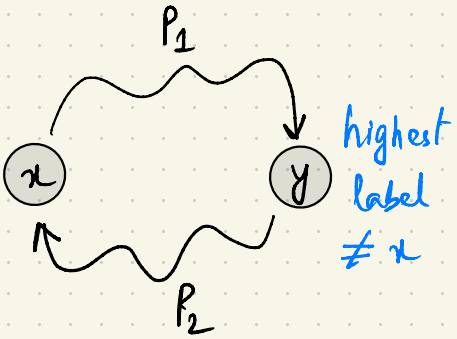
(1) What about graphs with negative cycles?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

FAQs

(1) What about graphs with negative cycles?

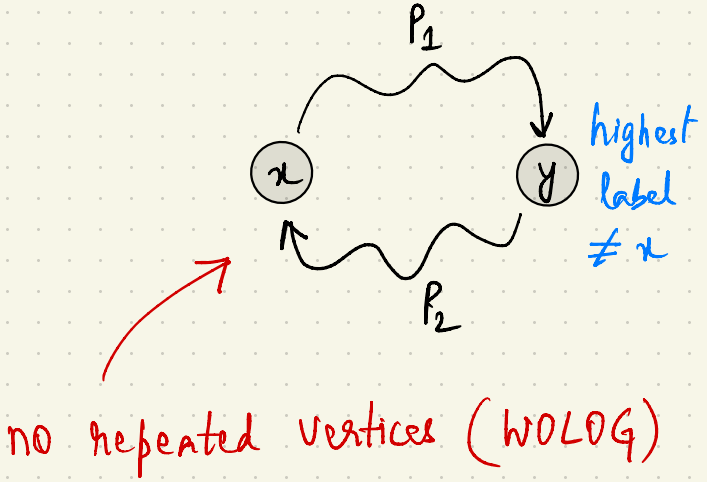
Then, at least one "diagonal" entry $A[i,i,n]$ is negative.



FAQs

(1) What about graphs with negative cycles?

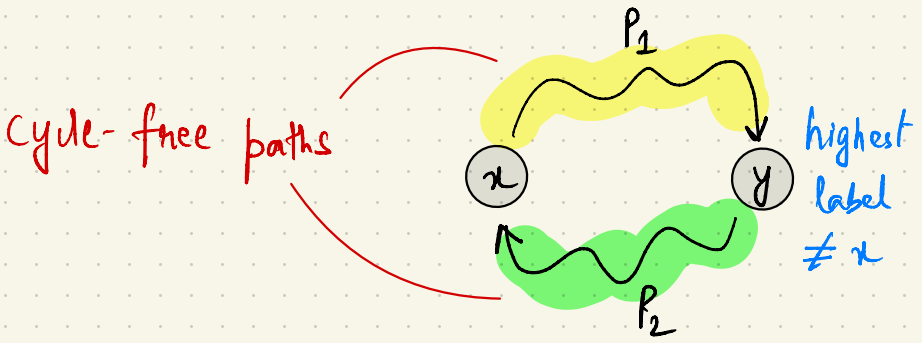
Then, at least one "diagonal" entry $A[i,i,n]$ is negative.



FAQs

(1) What about graphs with negative cycles?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

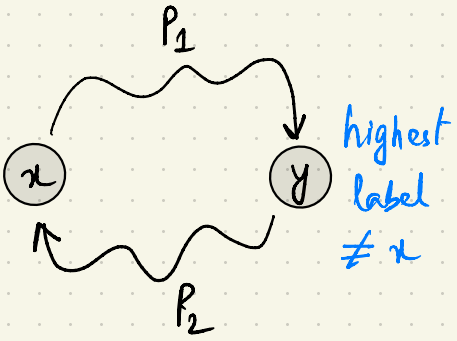


FAQs

① What about graphs with negative cycles?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$A[x,x,y] =$

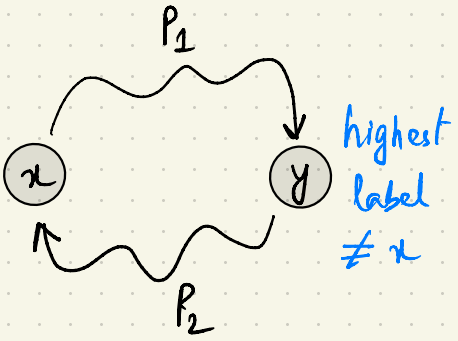


FAQs

① What about graphs with negative cycles ?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$

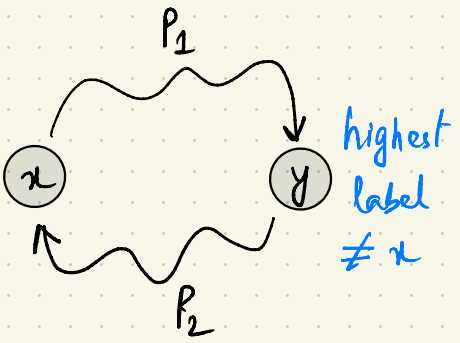


FAQs

① What about graphs with negative cycles ?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$



at most the length of shortest cycle-free path from x to y with internal nodes in $V^{(y-1)}$.

at most the length of shortest cycle-free path from y to x with internal nodes in $V^{(y-1)}$.

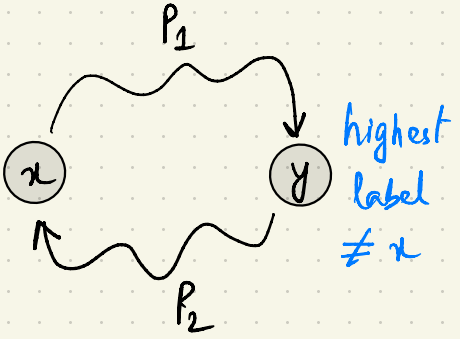
FAQs

① What about graphs with negative cycles ?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$

$< \text{wt of cycle} < 0$



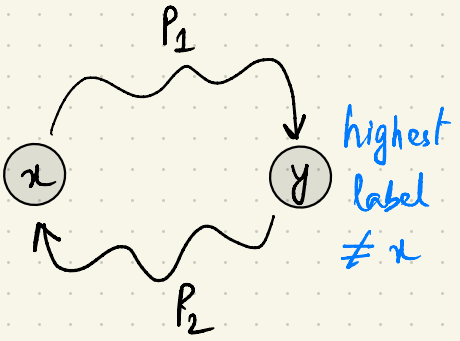
FAQs

(1) What about graphs with negative cycles ?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$

$< \text{wt of cycle} < 0$



(2) How to reconstruct shortest paths ?

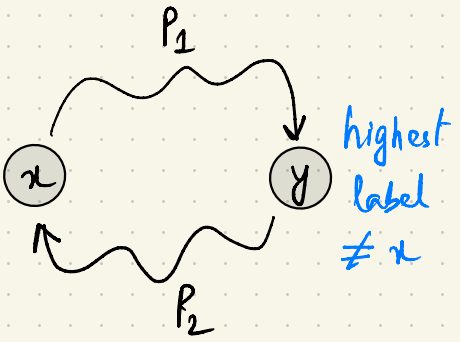
FAQs

① What about graphs with negative cycles ?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$

$< \text{wt of cycle} < 0$



② How to reconstruct shortest paths ?

$B[i,j]$ = max label, say l, of an internal node on a shortest $i-j$ path for all $i,j \in V$.

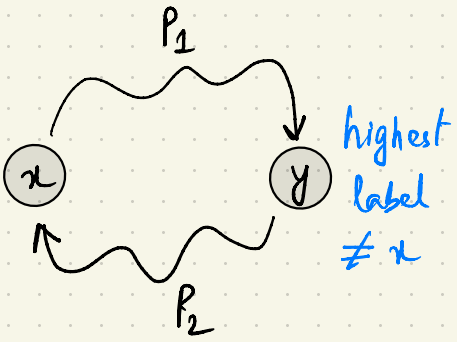
FAQs

(1) What about graphs with negative cycles?

Then, at least one "diagonal" entry $A[i,i,n]$ is negative.

$$A[x,x,y] = \min \begin{cases} A[x,x,y-1] \\ A[x,y,y-1] + A[y,x,y-1] \end{cases}$$

$< \text{wt of cycle} < 0$



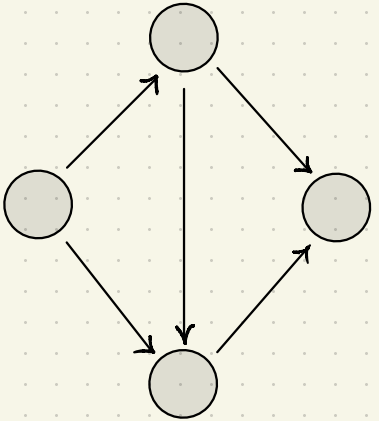
(2) How to reconstruct shortest paths?

$B[i,j]$ = max label, say ℓ, of an internal node on a shortest $i-j$ path for all $i,j \in V$.

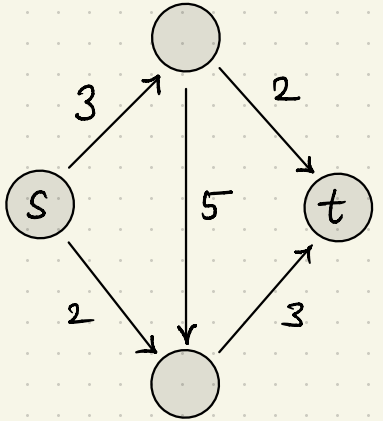
Recurse on $B[i,\ell]$ and $B[\ell,j]$.

NETWORK FLOWS

Maximum Flow Problem

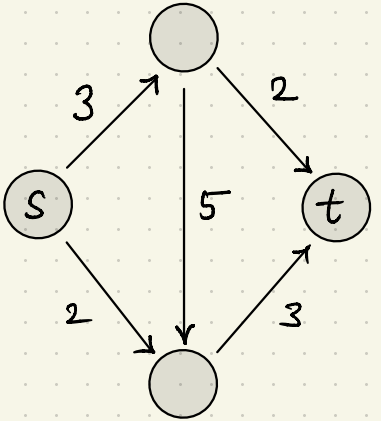


Maximum Flow Problem



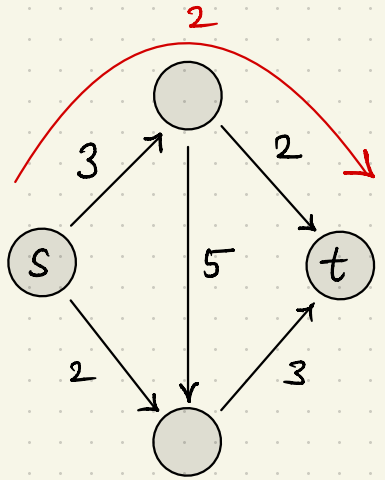
Capacity = maximum amount of stuff an edge can carry

Maximum Flow Problem



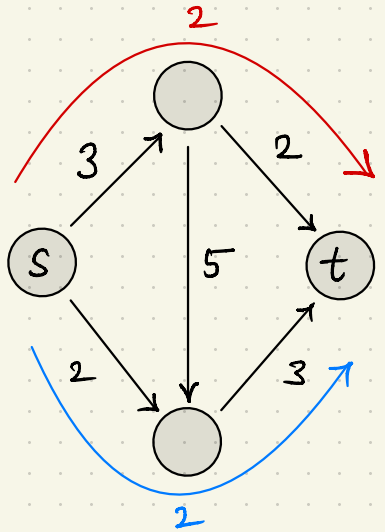
How much stuff can be pushed from s to t ?

Maximum Flow Problem



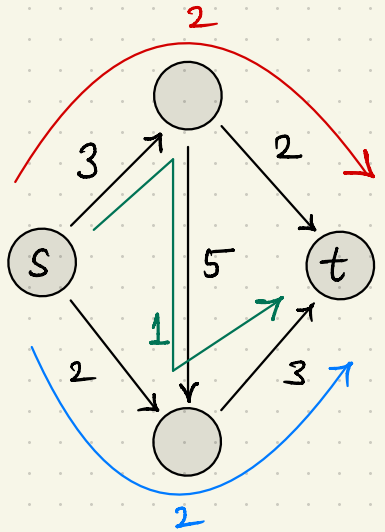
How much stuff can be pushed from s to t?

Maximum Flow Problem



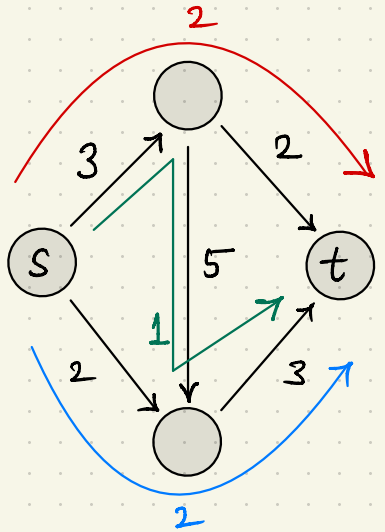
How much stuff can be pushed from s to t ?

Maximum Flow Problem



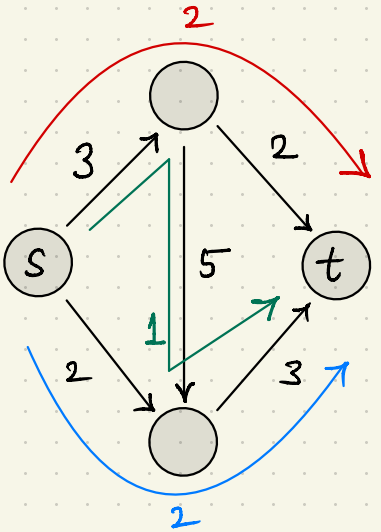
How much stuff can be pushed from s to t ?

Maximum Flow Problem



How much stuff can be pushed from s to t? 5 units.

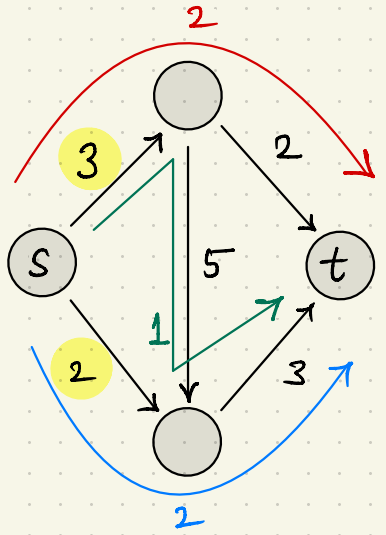
Maximum Flow Problem



How much stuff can be pushed from s to t? 5 units.

Can we do better?

Maximum Flow Problem



How much stuff can be pushed from s to t? 5 units.

Can we do better? NO!

Maximum Flow Problem

an instance of max flow problem is defined by:

Maximum Flow Problem

an instance of max flow problem is defined by:

a directed graph $G = (V, E)$

Maximum Flow Problem

an instance of max flow problem is defined by:

- a directed graph $G = (V, E)$
- a source vertex $s \in V$
- a sink vertex $t \in V$

Maximum Flow Problem

an instance of max flow problem is defined by:

- a directed graph $G = (V, E)$
- a source vertex $s \in V$
- a sink vertex $t \in V$
- a nonnegative and integral capacity u_e for each edge $e \in E$

Maximum Flow Problem

an instance of max flow problem is defined by:

- a directed graph $G = (V, E)$
- a source vertex $s \in V$
- a sink vertex $t \in V$
- a nonnegative and integral capacity u_e for each edge $e \in E$

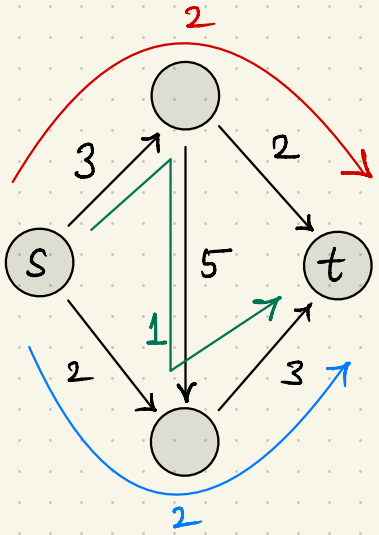
Why not c_e?

Maximum Flow Problem

Feasible Solutions : Flows

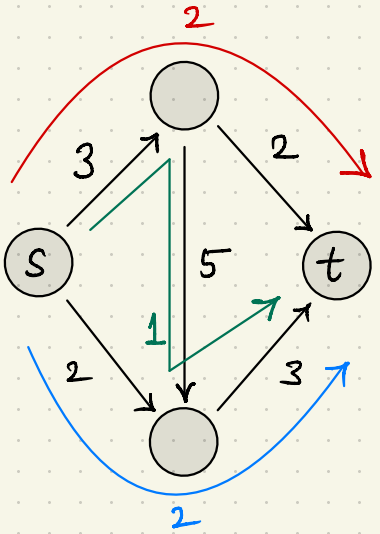
Maximum Flow Problem

Feasible Solutions : Flows

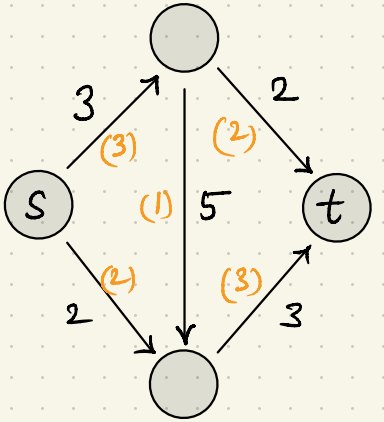


Maximum Flow Problem

Feasible Solutions : Flows



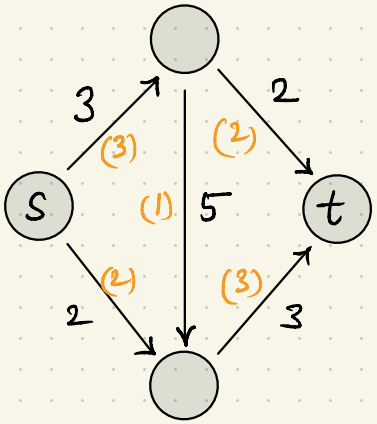
≡



Superposition of paths

Maximum Flow Problem

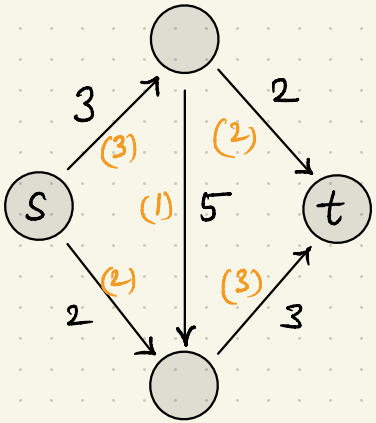
Feasible Solutions : Flows



Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$

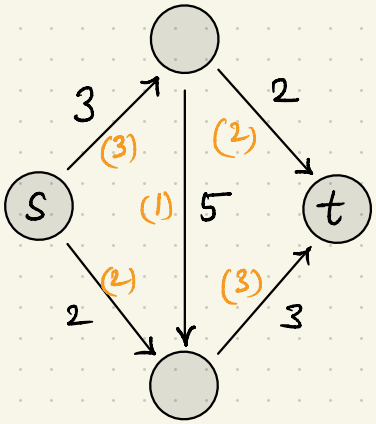


Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints



2. Conservation constraints

Maximum Flow Problem

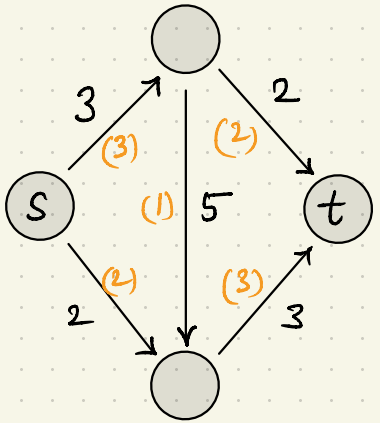
Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints

$$f_e \leq u_e \quad \forall e \in E$$

2. Conservation constraints



Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

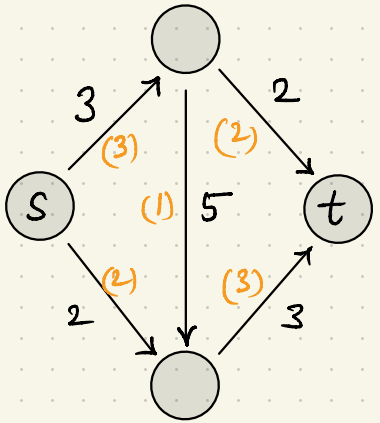
1. Capacity constraints

$$f_e \leq u_e \quad \forall e \in E$$

2. Conservation constraints

$$\text{flow in} = \text{flow out} \quad \forall v \neq s, t$$





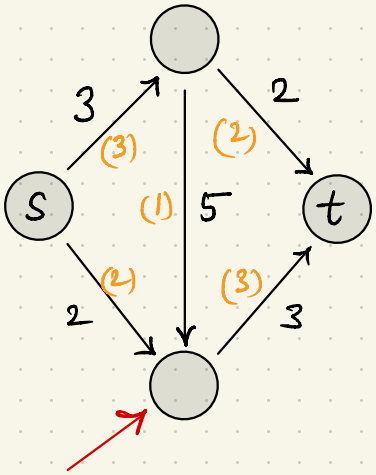
Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints

$$f_e \leq u_e \quad \forall e \in E$$



2. Conservation constraints

$$\text{flow in} = \text{flow out} \quad \forall v \neq s, t$$



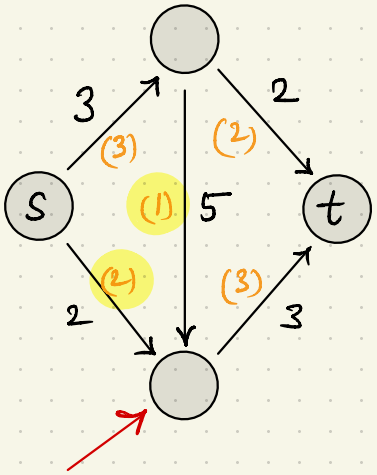
Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints

$$f_e \leq u_e \quad \forall e \in E$$



2. Conservation constraints

$$\text{flow in} = \text{flow out} \quad \forall v \neq s, t$$



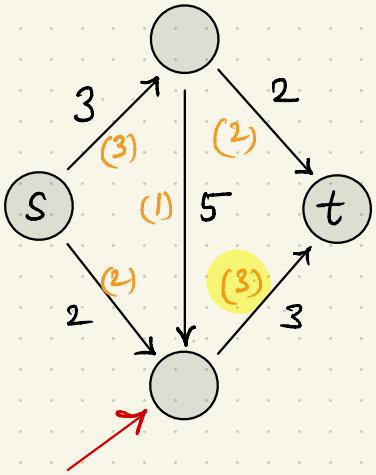
Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints

$$f_e \leq u_e \quad \forall e \in E$$



2. Conservation constraints

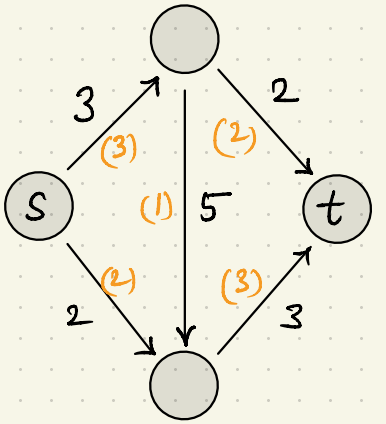
$$\text{flow in} = \text{flow out} \quad \forall v \neq s, t$$



Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying



1. Capacity constraints

$$f_e \leq u_e \quad \forall e \in E$$

2. Conservation constraints

$$\text{flow in} = \text{flow out} \quad \forall v \neq s, t$$



$f_e = 0 \ \forall e$ is feasible

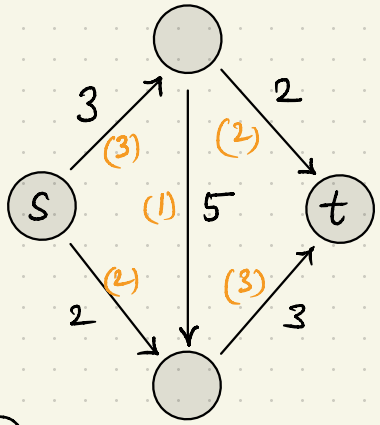
Maximum Flow Problem

Claim: Flow out of s = Flow into t

Proof: Define Type I edges : (s, v) for $v \notin \{s, t\}$

Type II edges : (v, t) for $v \notin \{s, t\}$

Type III edges : (u, v) for $u, v \notin \{s, t\}$



$$\sum_{v \neq s,t} \text{flow-in}(v) = \sum_{\text{Type I}} f_e + \sum_{\text{Type III}} f_e \quad \text{——} \quad (1)$$

$$\sum_{v \neq s,t} \text{flow-out}(v) = \sum_{\text{Type II}} f_e + \sum_{\text{Type III}} f_e \quad \text{——} \quad (2)$$

Capacity constraints $\Rightarrow$ (1) = (2) $\Rightarrow$ flow out of s = flow into t.

(Type I) (Type II)

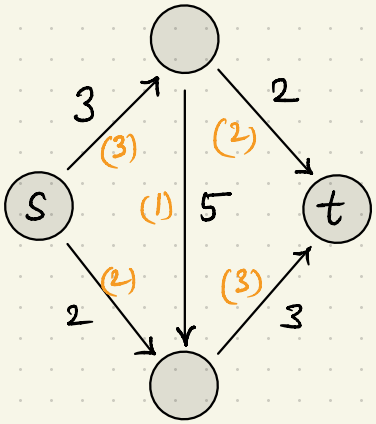
∎

Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints
2. Conservation constraints

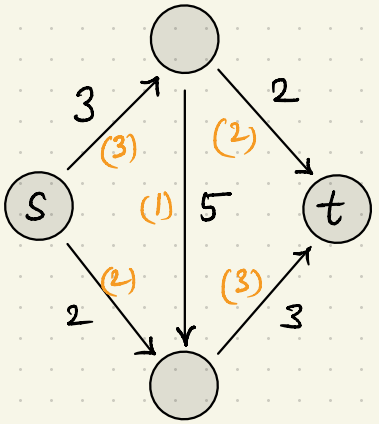


Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints
2. Conservation constraints



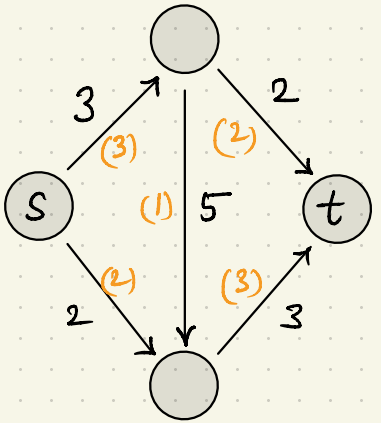
Objective : maximize flow value := flow going out of s

Maximum Flow Problem

Feasible Solutions : Flows

A nonnegative vector $\{f_e\}_{e \in E}$ satisfying

1. Capacity constraints
2. Conservation constraints



Objective : maximize flow value := flow going out of s

Assume : no edges going into s or coming out of t. (WOLOG)

ALGORITHMS

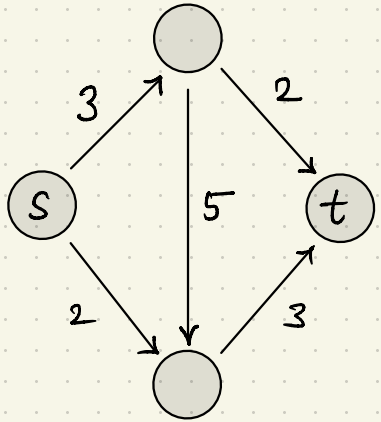
Idea #1 : Naive Greedy Approach

Idea #1: Naive Greedy Approach

Compute a sequence of flows with progressively higher values

IDEA # 1 : NAIVE GREEDY APPROACH

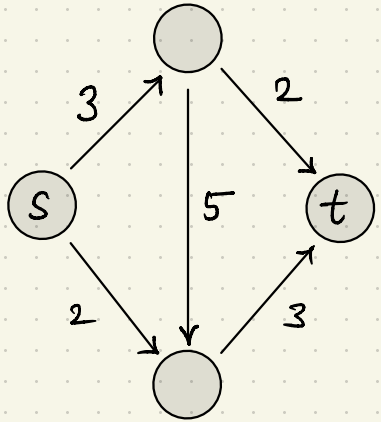
Compute a sequence of flows with progressively higher values



IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values

① initialize $f_e = 0 \quad \forall e$

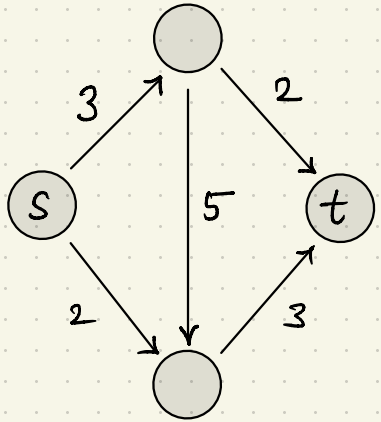


IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values

① initialize $f_e = 0 \quad \forall e$

② repeat :



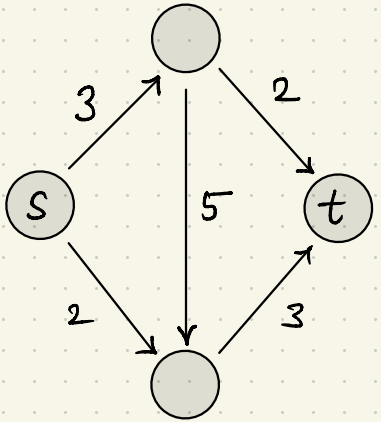
IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values

① initialize $f_e = 0 \quad \forall e$

② repeat :

* search for an $s-t$ path P such that $f_e < u_e \; \forall e \in P$.



IDEA # 1 : NAIVE GREEDY APPROACH

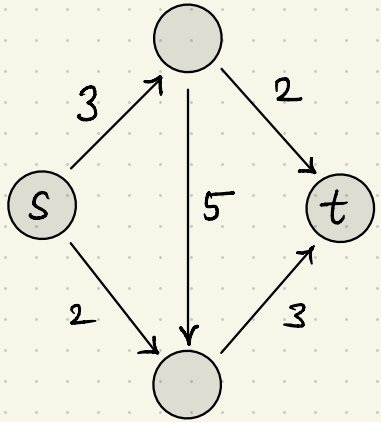
Compute a sequence of flows with progressively higher values

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

// possible in $O(|E|)$ time via BFS / DFS



IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values

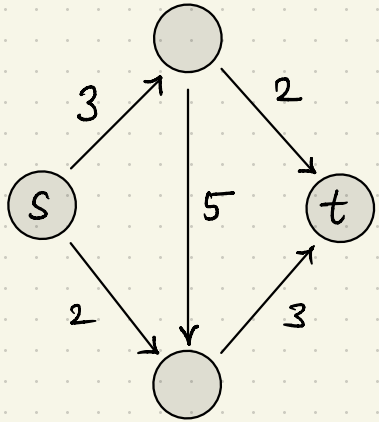
① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

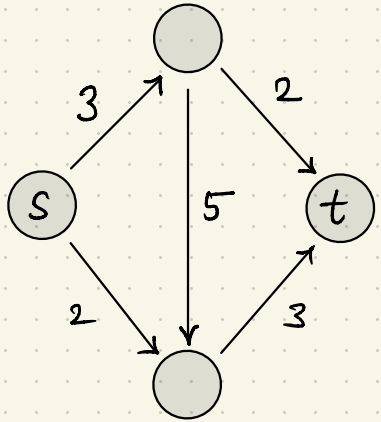
// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$



IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values



① initialize $f_e = 0 \quad \forall e$

② repeat :

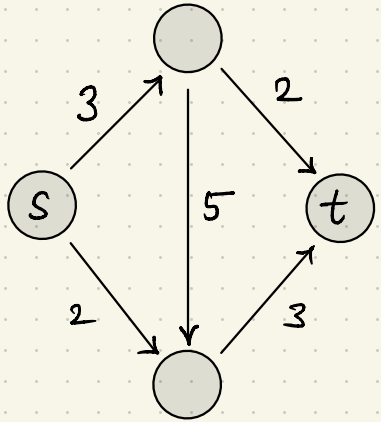
* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else,

IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values



① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

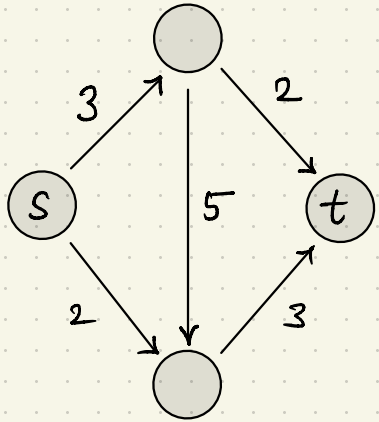
// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P} (u_e - f_e)$

└

IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values



① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

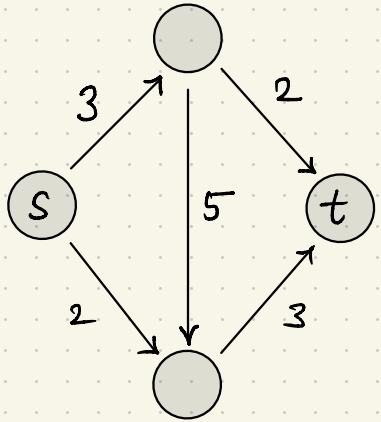
// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P} \underbrace{(u_e - f_e)}_{\text{room on } e}$

└

IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values



① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$

* else, let $\Delta := \min_{e \in P} (u_e - f_e)$

$(u_e - f_e)$: room on e

$\min_{e \in P}$: room on P

└

Idea #1 : Naive Greedy Approach

Compute a sequence of flows with progressively higher values

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P such that $f_e < u_e \ \forall e \in P$.

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P} (u_e - f_e)$

  for all edges $e \in P$, increase f_e by Δ.

