COL 351 : Analysis & Design of Algorithms

LECTURE 13

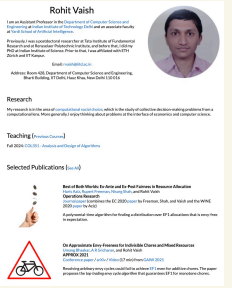
Graph Algorithms VI :
Dijkstra's Algorithm

Aug 23, 2024 | Rohit Vaish

Application of SCCs : Understanding the web graph

The Web Graph

The Web Graph



Rohit Vaish

I am an Assistant Professor in the Department of Computer Science and Engineering at Indian Institute of Technology Delhi and an associate faculty at Yardi School of Artificial Intelligence.

Previously, I was a postdoctoral research fellow at Tata Institute of Fundamental Research and at Rensselaer Polytechnic Institute, and before that, I did my PhD in Computer Science at Technion–Israel Institute of Technology. I was also affiliated with ETH Zürich and IST Vienna.

Email: rvaish@iitd.ac.in

Address: Room 430, Department of Computer Science and Engineering, Bharti Building, IIT Delhi, Hauz Khas, New Delhi 110016

Research

My research is in the area of computational social choice, which is the study of collective decision-making problems from a computational lens. More generally, I enjoy thinking about problems at the interface of economics and computer science.

Teaching [Previous Courses]

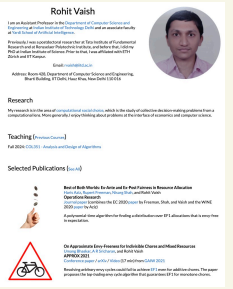
Fall 2024: COL351 - Analysis and Design of Algorithms

Selected Publications [See All]

Best of Both Worlds: Ex-Ante and Ex-Post Fairness in Resource Allocation
Operations Research
2024 paper by Hannaneh Akrami, Jugal Garg, Pieter Kleer, Mengqi Zhao, and Rohit Vaish
A polynomial-time algorithm for finding a distribution over EF1 allocations that is envy-free in expectation.

On Approximate Envy-Freeness for Indivisible Chores and Mixed Resources
APPROX 2021
Breaking a three-decade envy-cycle could help us achieve 1/2 + ε envy-free for additive chores. The paper proposes the bar-trading envy-cycle algorithm that guarantees EF1/3 for monotone chores.

The Web Graph



Rohit Vaish

I am an Assistant Professor in the Department of Computer Science and Engineering at Indian Institute of Technology Delhi and an associate faculty at Yardi School of Artificial Intelligence.

Previously, I was a postdoctoral research fellow at Tata Institute of Fundamental Research and at Rensselaer Polytechnic Institute, and before that, I did my PhD in Computer Science and Business Price-in-Hat. I was affiliated with ETH Zürich and IIT Kanpur.

Email: rvaish@iitd.ac.in

Address: Room 430, Department of Computer Science and Engineering, Bharti Building, IIT Delhi, Hauz Khas, New Delhi 110016

Research

My research is in the area of computational social choice, which is the study of collective decision-making problems from a computational lens. More generally, I enjoy thinking about problems at the interface of economics and computer science.

Teaching [Previous Courses]

Fall 2024: COL351 - Analysis and Design of Algorithms

Selected Publications [See All]

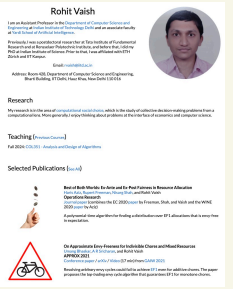
Best of Both Worlds: Ex-Ante and Ex-Post Fairness in Resource Allocation
Operations Research
2024 version by Arxiv

A polynomial-time algorithm for finding a distribution over EF1 allocations that is envy-free in expectation.

On Approximate Envy-Freeness for Indivisible Chores and Mixed Resources
APPROX 2021
2021 version / arXiv / Video [67 min] from GAMM 2021

Breaking a three-envy cycle could fail to achieve EF1 even for additive chores. The paper proposes the bar-trading envy cycle algorithm that guarantees EF1 for monotone chores.

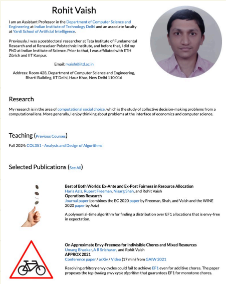
The Web Graph



COL351, Fall Semester 2024

Analysis and Design of Algorithms

The Web Graph



Rohit Vaish

I am an Assistant Professor in the Department of Computer Science and Engineering at Indian Institute of Technology Delhi and an associate faculty at Yardi School of Artificial Intelligence.

Previously, I was a postdoctoral researcher at Tata Institute of Fundamental Research and at Rensselaer Polytechnic Institute, and before that, I did my PhD at Indian Institute of Science. Prior to that, I was affiliated with ETH Zürich and IIT Kanpur.

Email: rvaish@iitd.ac.in

Address: Room 428, Department of Computer Science and Engineering, Bharti Building, IIT Delhi, Hauz Khas, New Delhi 110 016

Research

My research is in the area of computational social choice, which is the study of collective decision-making problems from a computational lens. More generally, I enjoy thinking about problems at the interface of economics and computer science.

Teaching (Previous Courses)

Fall 2024: COL351 - Analysis and Design of Algorithms

Selected Publications (See All)

Best of Both Worlds: Ex-Ante and Ex-Post Fairness in Resource Allocation
Haris Aziz, Rupert Freeman, Nisarg Shah, and Rohit Vaish
Operations Research
Journal paper (combines the EC 2020 paper by Freeman, Shah, and Vaish and the WINE 2020 paper by Aziz)

A polynomial-time algorithm for finding a distribution over EF1 allocations that is envy-free in expectation.

On Approximate Envy-Freeness for Indivisible Chores and Mixed Resources
Umang Bhaskar, A R Sricharan, and Rohit Vaish
APPROX 2021
Conference paper / arXiv / Video (17 min) from GAIW 2021

Resolving *arbitrary* envy cycles could fail to achieve EF1 even for additive chores. The paper proposes the *top-trading envy cycle algorithm* that guarantees EF1 for monotone chores.

COL351, Fall Semester 2024

Analysis and Design of Algorithms

. . .

awesome collaborators!

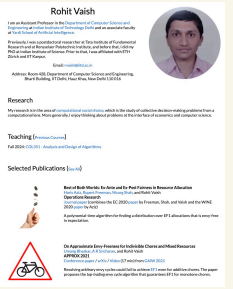
The Web Graph

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

Rohit Vaish
Assistant Professor and Pankaj Gupta Faculty Fellow
Ph.D. (IISc)

Algorithms and Economics, Artificial Intelligence, Computational Social Choice, Game Theory

COMSOC



COL351, Fall Semester 2024
Analysis and Design of Algorithms

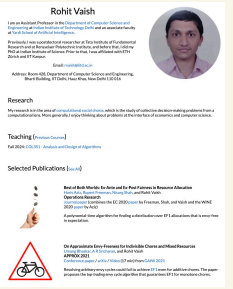
- **Computational Social Choice** – Rohit Vaish (Indian Institute of Technology Delhi)
  2023 · 2022-Fall · 2022-Spring



. . .

awesome collaborators!

The Web Graph



- **Computational Social Choice** – Rohit Vaish (Indian Institute of Technology Delhi)
  2023 · 2022-Fall · 2022-Spring

...

What does the web graph look like?

awesome collaborators!

The Web Graph

Computer Networks
Volume 33, Issues 1–6, June 2000, Pages 309-320

Graph structure in the Web

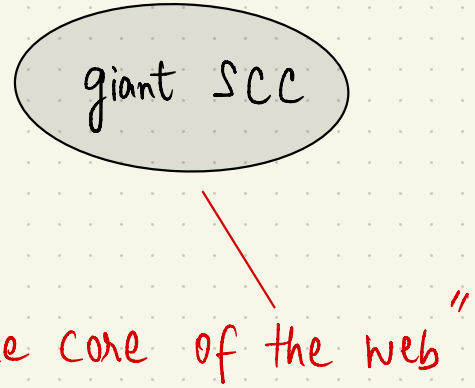
Andrei Broder a, Ravi Kumar b, Farzin Maghoul a, Prabhakar Raghavan b, Sridhar Rajagopalan b, Raymie Stata c, Andrew Tomkins b, Janet Wiener c

Size : ~ 200 million vertices

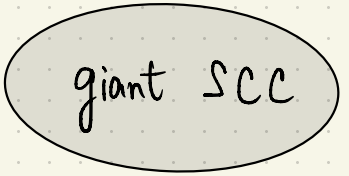
~ 1.5 billion edges

The Bow Tie

The Bow Tie



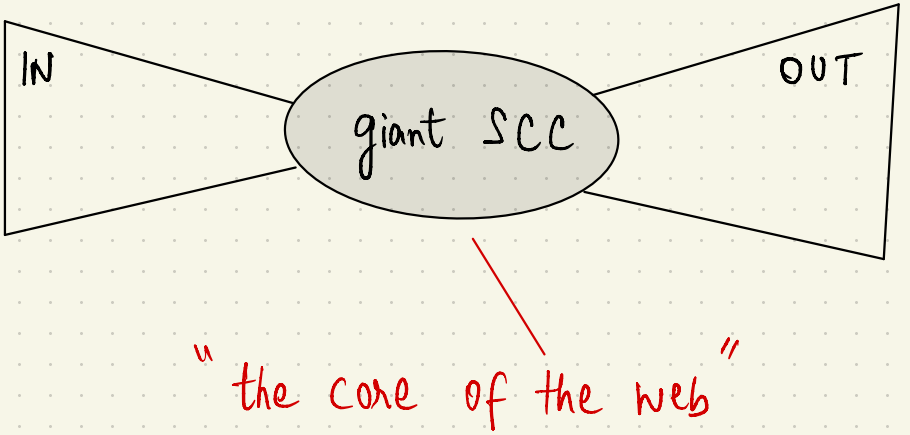
THE BOW TIE



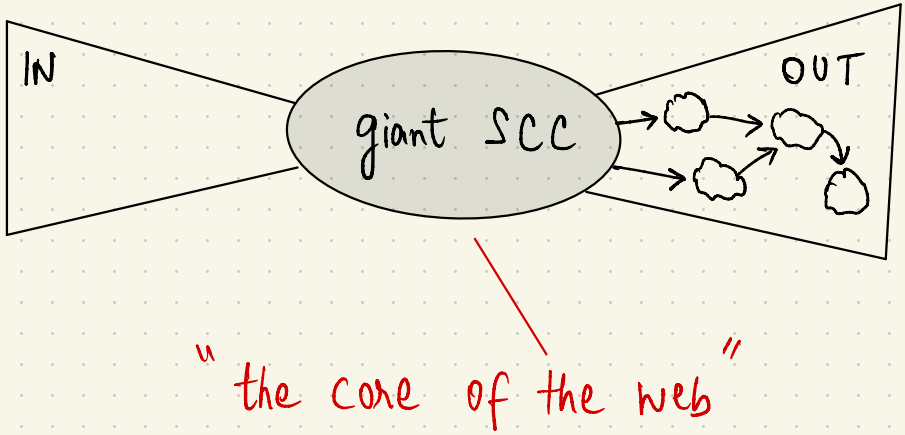
"the core of the web"

Intuitively, there should be only one giant SCC.

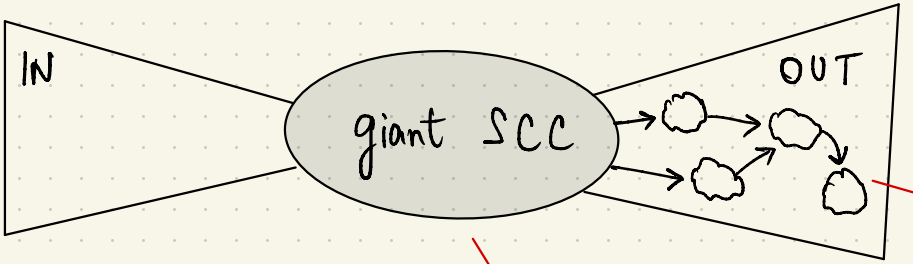
THE BOW TIE



THE BOW TIE



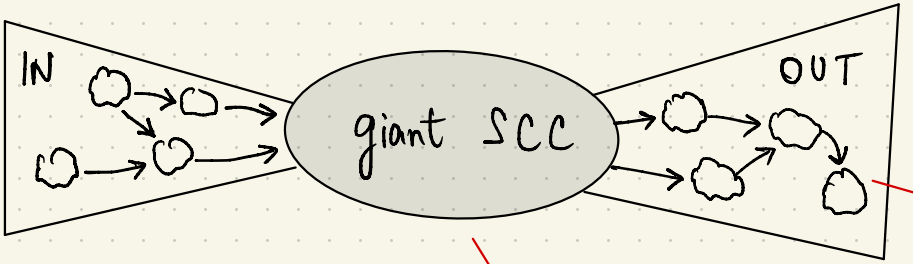
THE BOW TIE



e.g., corporate websites

"the core of the web"

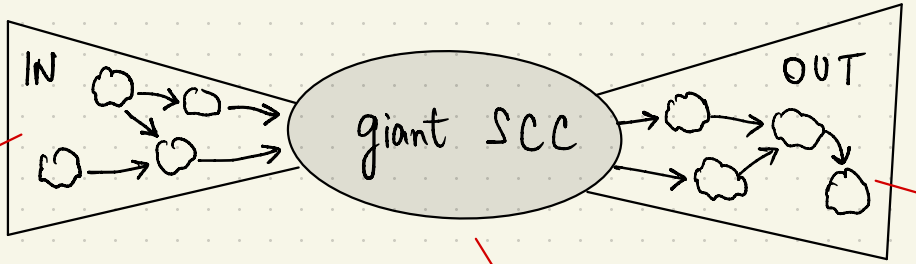
THE BOW TIE



e.g., corporate websites

"the core of the web"

THE BOW TIE

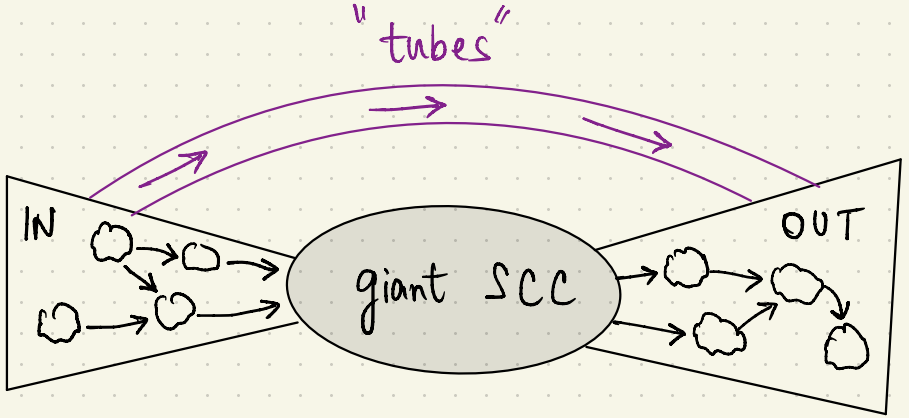


e.g., new webpages

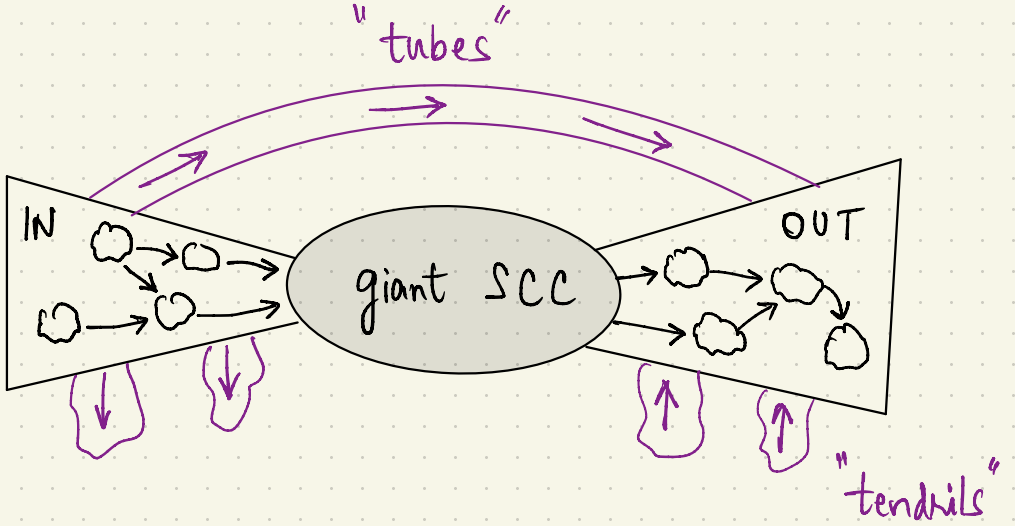
e.g., corporate websites

"the core of the web"

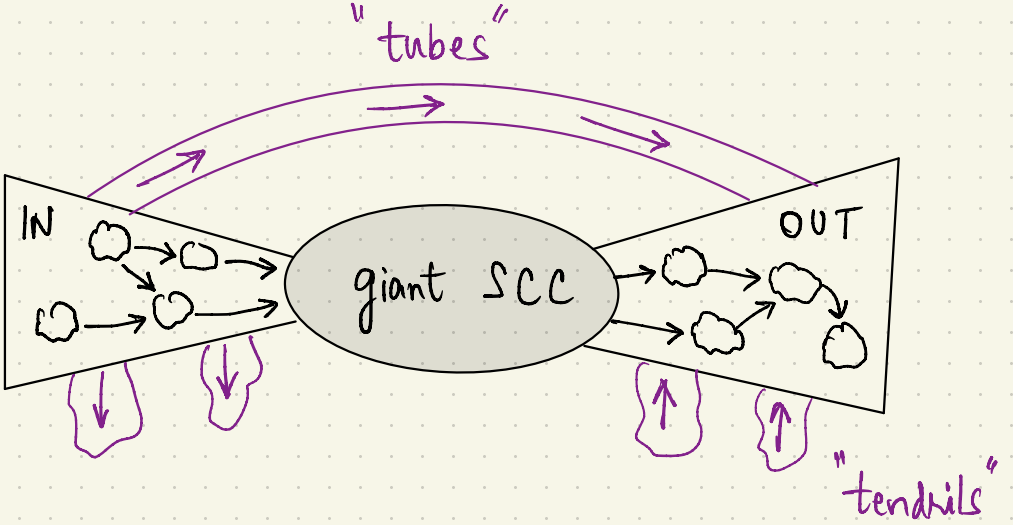
The Bow Tie



THE BOW TIE

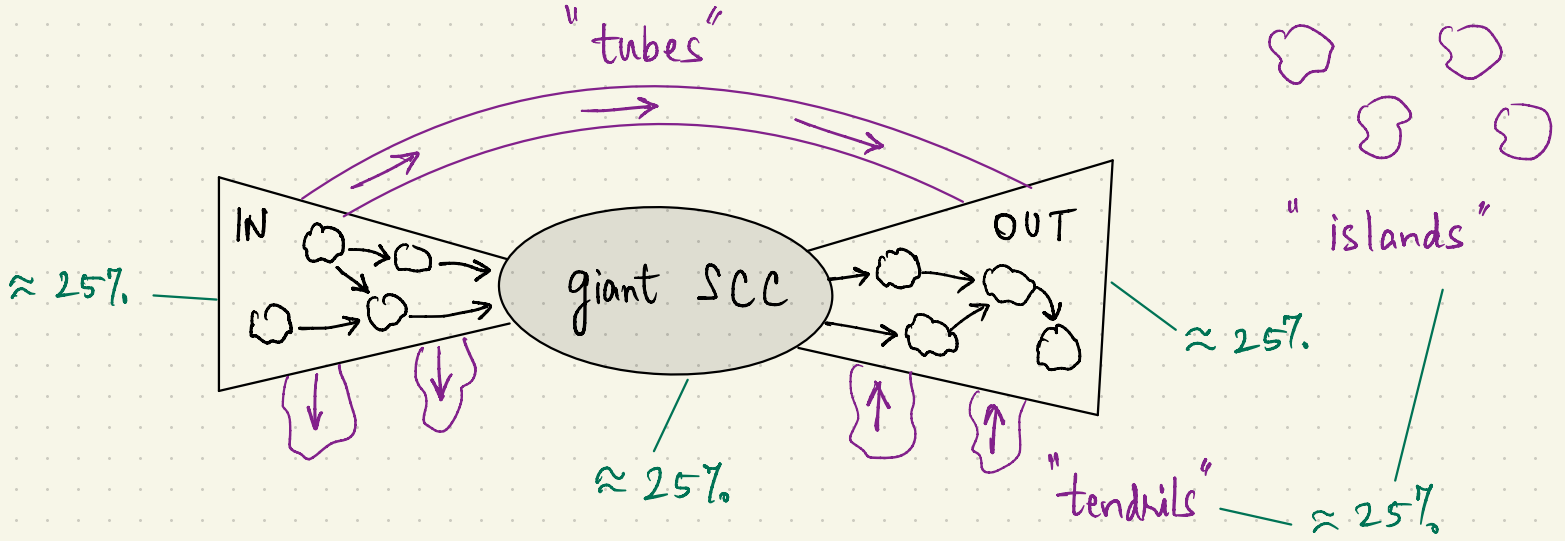


THE BOW TIE

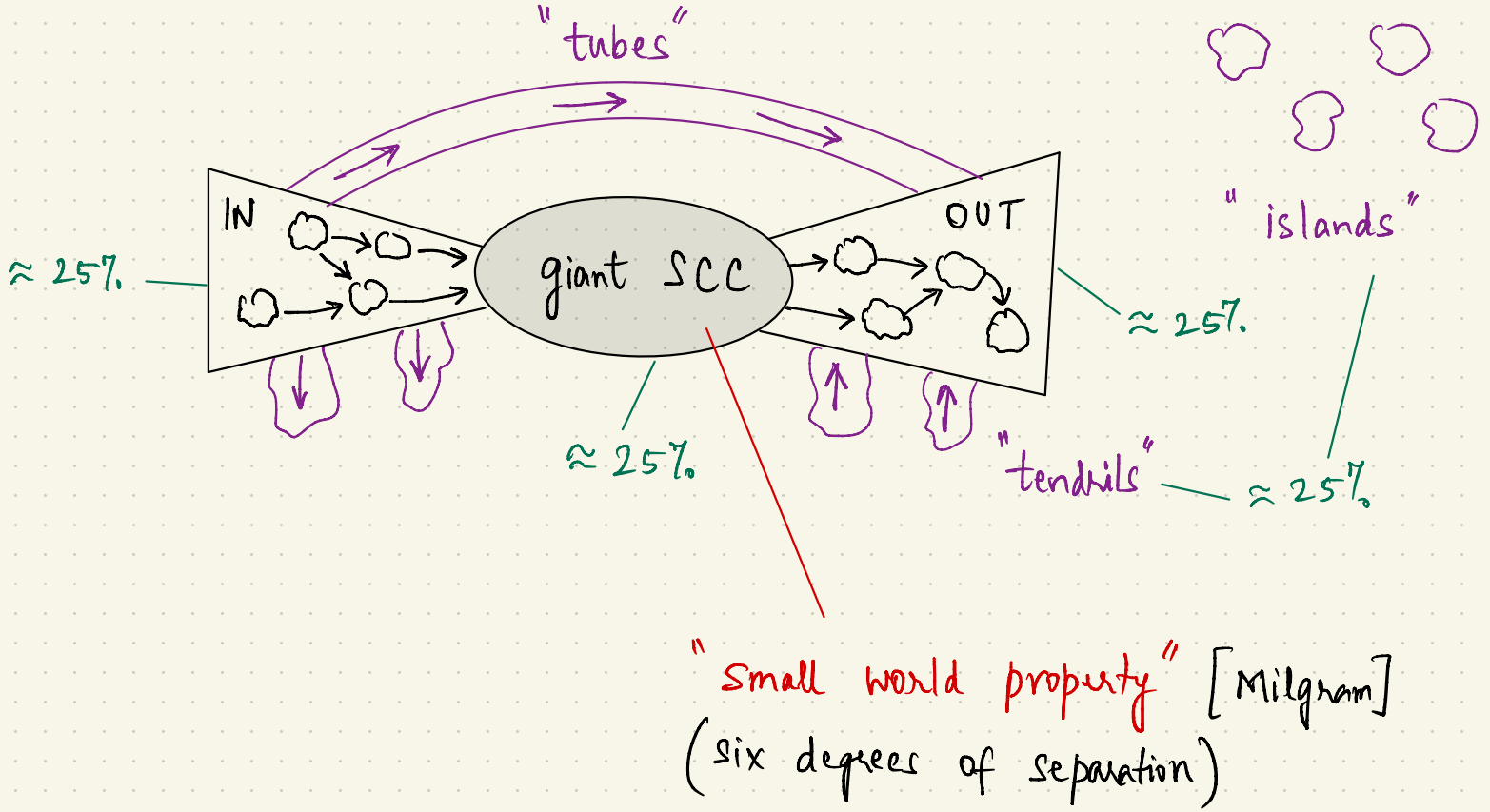


"islands"

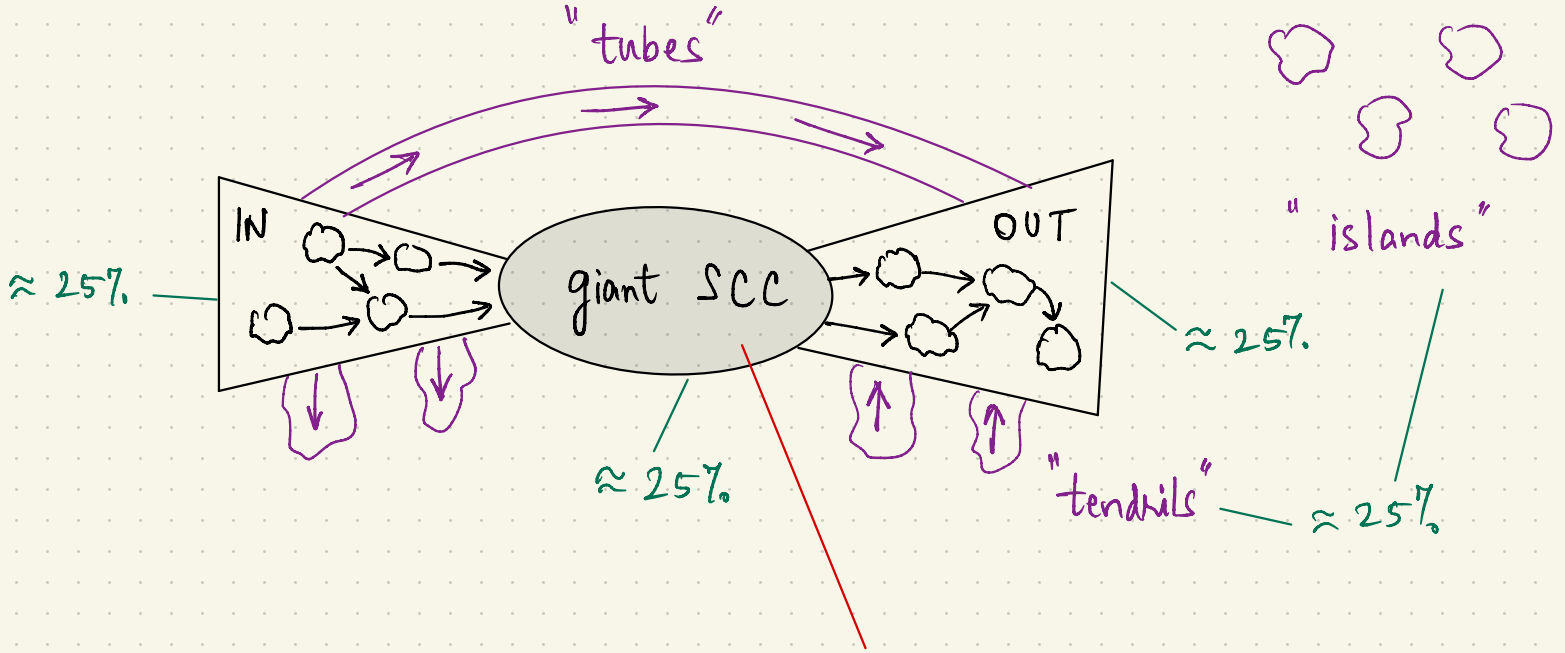
THE BOW TIE



THE BOW TIE


"tubes"
IN
giant SCC
OUT
"islands"
≈ 25%
≈ 25%
≈ 25%
"tendrils"
≈ 25%
"small world property" [Milgram]
(six degrees of separation)

THE BOW TIE



≈ 56 million vertices

typical short paths ≤ 20 hyperlinks

"small world property" [Milgram]

(six degrees of separation)

DIJKSTRA'S ALGORITHM

SINGLE-SOURCE SHORTEST PATH PROBLEM

SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.

SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.

length of the shortest path from s to v

SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.

length of the shortest path from s to v

sum of edge lengths

SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.

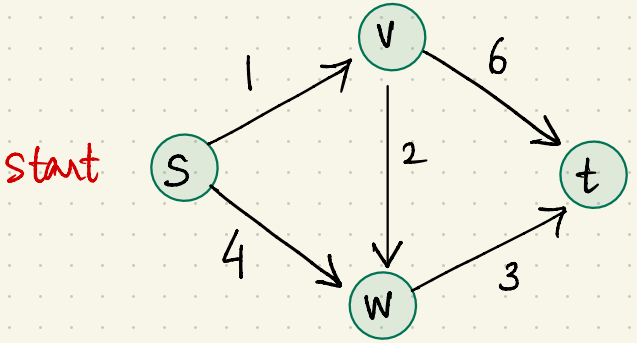
length of the shortest path from s to v if $s \rightsquigarrow v$ path exists

∞ otherwise

SINGLE-SOURCE SHORTEST PATH PROBLEM

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

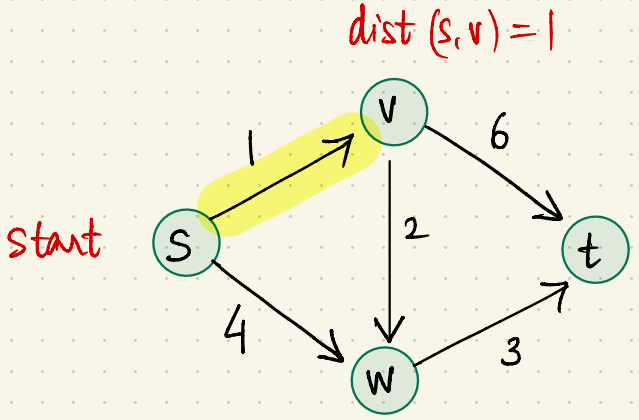
output: dist (s, v) for every vertex $v \in V$.



SINGLE-SOURCE SHORTEST PATH PROBLEM

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

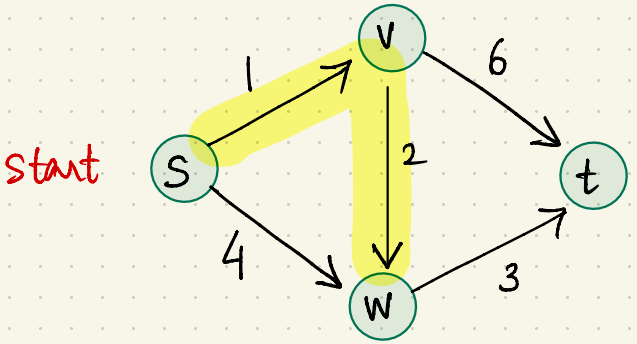
output: dist (s, v) for every vertex $v \in V$.



SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.

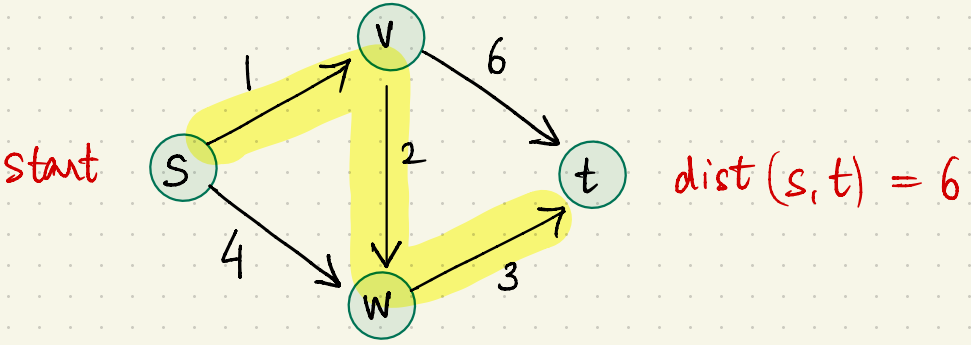


dist $(s, w) = 3$

SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.



SINGLE-SOURCE SHORTEST PATH PROBLEM

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

output: dist (s, v) for every vertex $v \in V$.

For negative lengths, use Bellman-Ford algorithm.

SINGLE-SOURCE SHORTEST PATH PROBLEM

Doesn't BFS already compute shortest paths?

SINGLE-SOURCE SHORTEST PATH PROBLEM

Doesn't BFS already compute shortest paths?

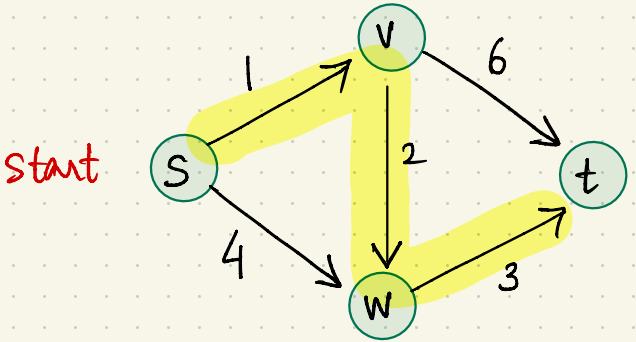
Yes, if length of a path = number of edges in it

($\ell_e = 1$ for every edge e)

SINGLE-SOURCE SHORTEST PATH PROBLEM

Doesn't BFS already compute shortest paths?

Yes, if length of a path = number of edges in it
($\ell_e = 1$ for every edge e)



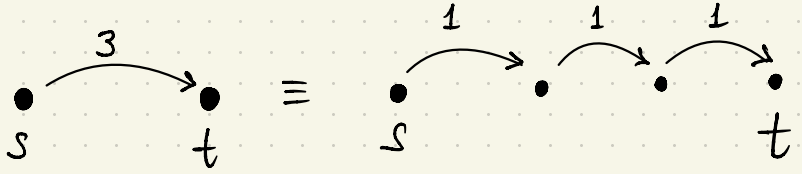
BFS fails

SINGLE-SOURCE SHORTEST PATH PROBLEM

Isn't an edge with length l_e the same as l_e unit length edges?

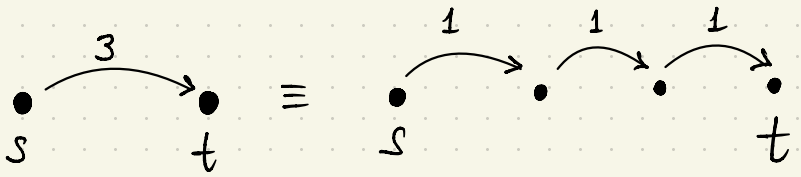
SINGLE-SOURCE SHORTEST PATH PROBLEM

🤔 Isn't an edge with length l_e the same as l_e unit length edges?



SINGLE-SOURCE SHORTEST PATH PROBLEM

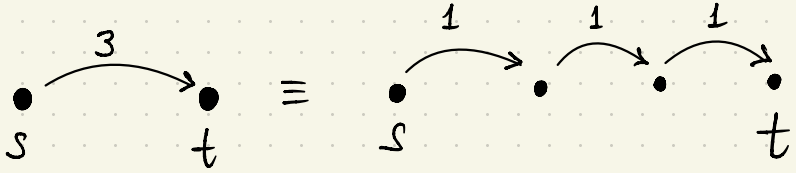
Isn't an edge with length l_e the same as l_e unit length edges?



This representation is faithful but wasteful.

SINGLE-SOURCE SHORTEST PATH PROBLEM

Isn't an edge with length l_e the same as l_e unit length edges?

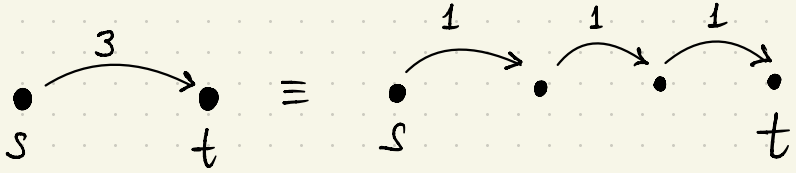


This representation is faithful but wasteful.

BFS time is linear in the size of the blown up graph

SINGLE-SOURCE SHORTEST PATH PROBLEM

Isn't an edge with length l_e the same as l_e unit length edges?



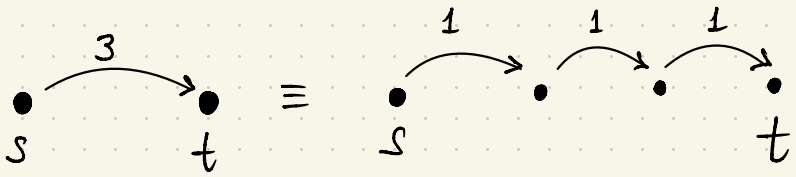
This representation is faithful but wasteful.

BFS time is linear in the size of the blown up graph

$$\sum_e l_e >> n + m$$

SINGLE-SOURCE SHORTEST PATH PROBLEM

Isn't an edge with length l_e the same as l_e unit length edges?



This representation is faithful but wasteful.

BFS time is linear in the size of the blown up graph

$$\sum_e l_e >> n + m$$

Solution: Dijkstra's algorithm

DIJKSTRA'S ALGORITHM

When $\ell_e = 1$ for all edges e, Dijkstra's algorithm becomes breadth-first search.

DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$A[v] = +\infty$ for all $v \neq s$

DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

only to help with explanation

DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

// main loop

while $X \neq V$ need to grow X

DIJKSTRA'S ALGORITHM

Initialize

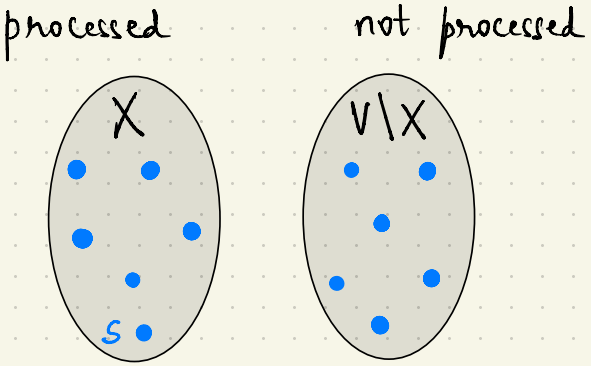
$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

// main loop

while $X \neq V$ need to grow X



DIJKSTRA'S ALGORITHM

Initialize

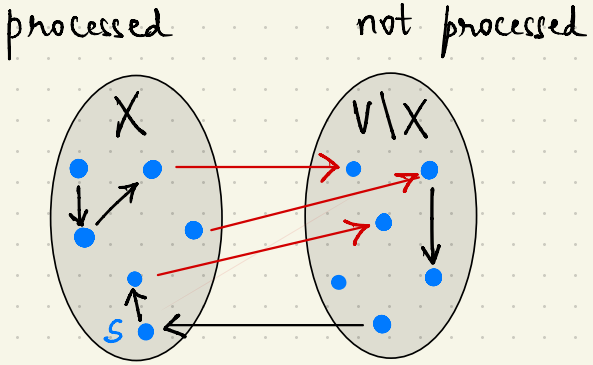
$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

// main loop

while $X \neq V$ need to grow X



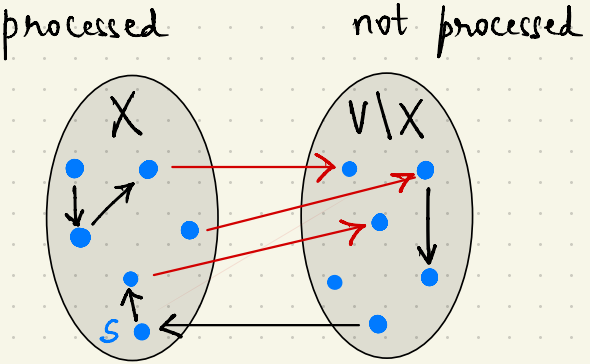
DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths



// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$

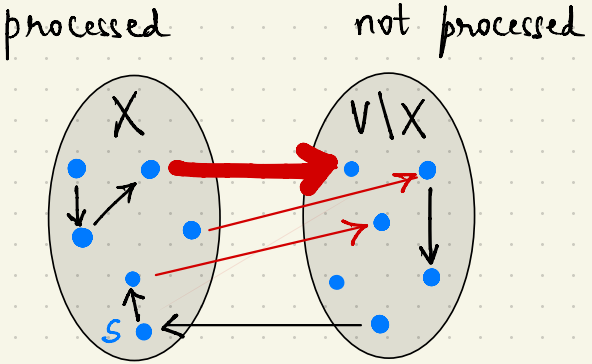
DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths



// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$

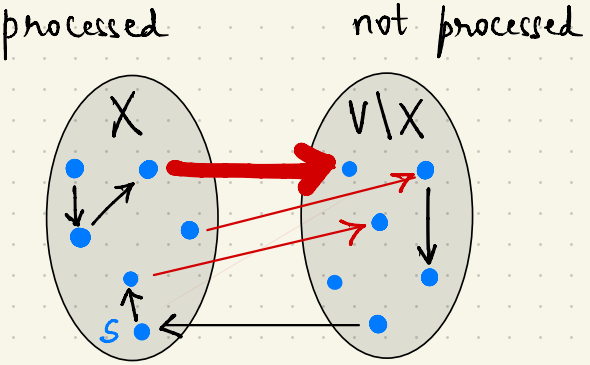
DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths



// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$A[v] + \ell_{vw}$ Dijkstra score

DIJKSTRA'S ALGORITHM

Initialize

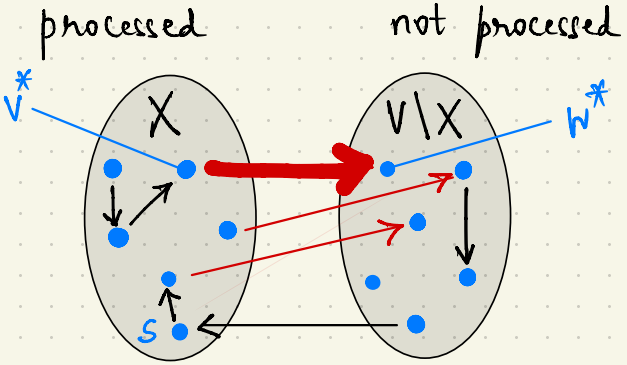
$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

// main loop

while $X \neq V$ need to grow X



among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

DIJKSTRA'S ALGORITHM

Initialize

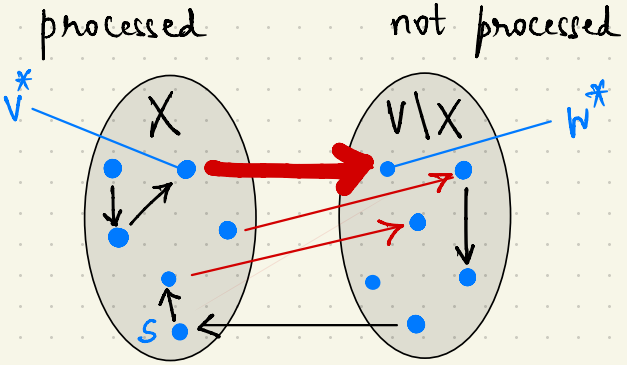
$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

// main loop

while $X \neq V$ need to grow X



among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

Add w^* to X.

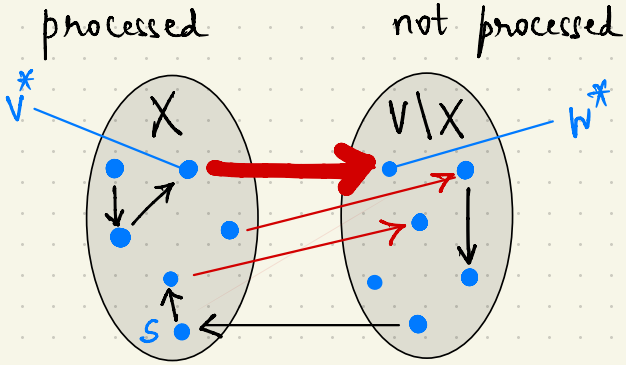
DIJKSTRA'S ALGORITHM

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths



// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

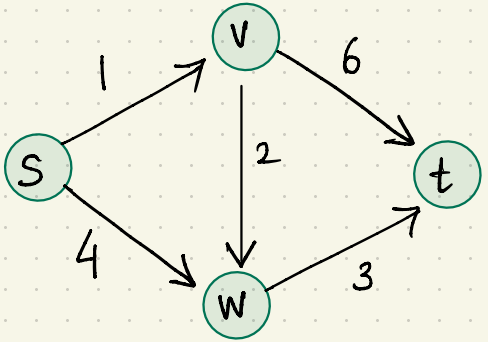
$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

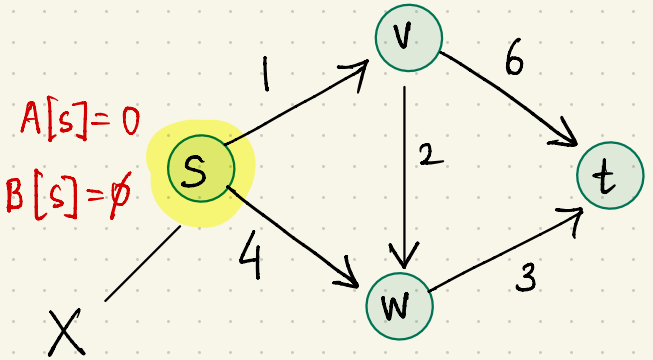
Add w^* to X.

Set $A[w^*] := A[v^*] + \ell_{v^*w^*}$ and $B[w^*] := B[v^*] \cup (v^*, w^*)$.

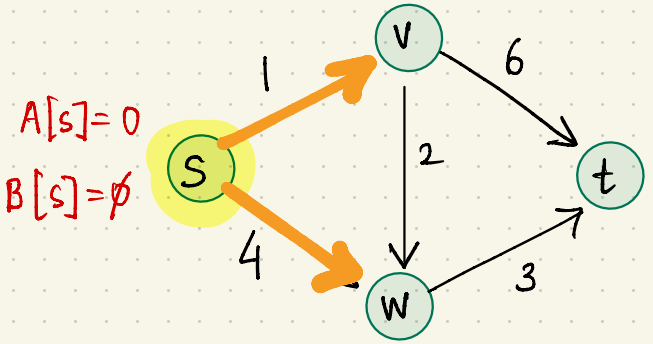
DIJKSTRA'S ALGORITHM



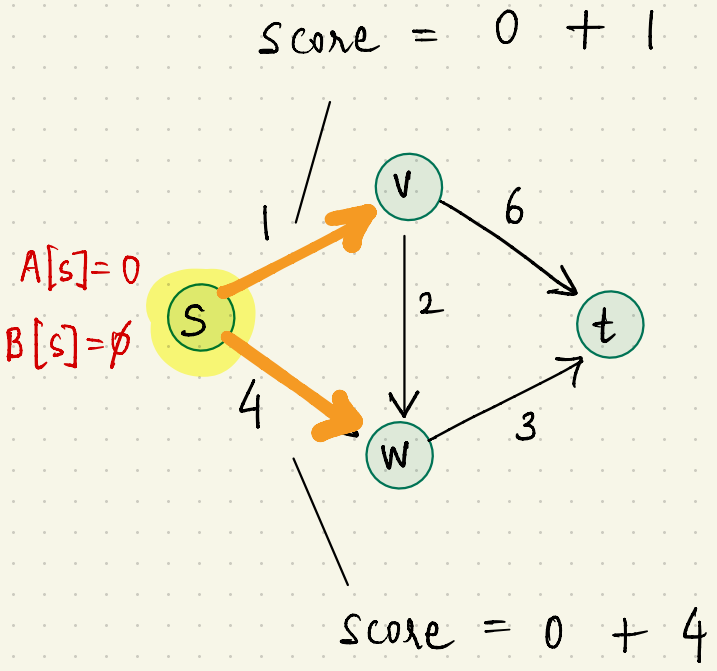
DIJKSTRA'S ALGORITHM



DIJKSTRA'S ALGORITHM

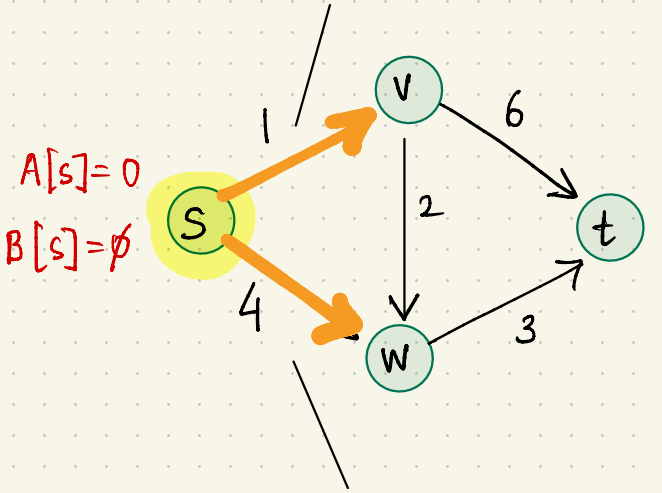


DIJKSTRA'S ALGORITHM



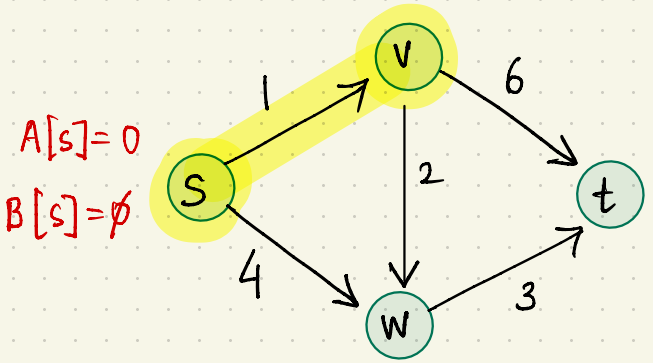
DIJKSTRA'S ALGORITHM

score = 0 + 1 (minimum)

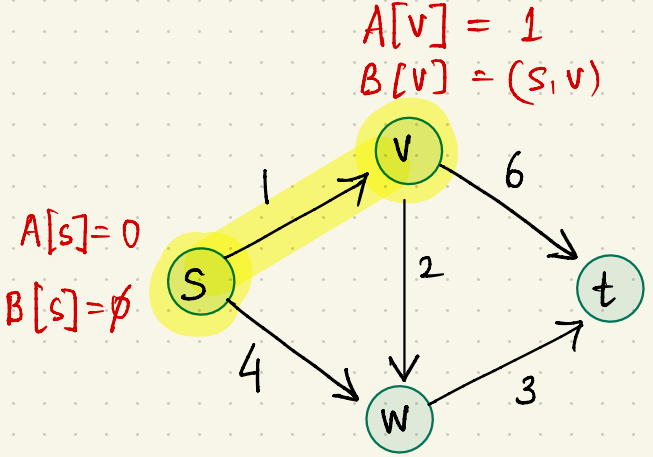


score = 0 + 4

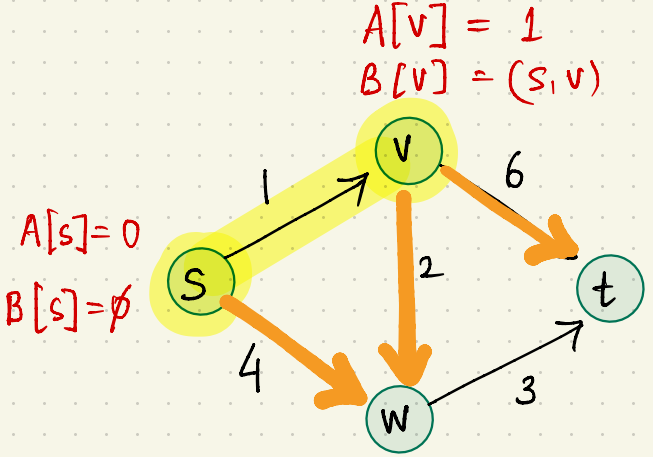
DIJKSTRA'S ALGORITHM



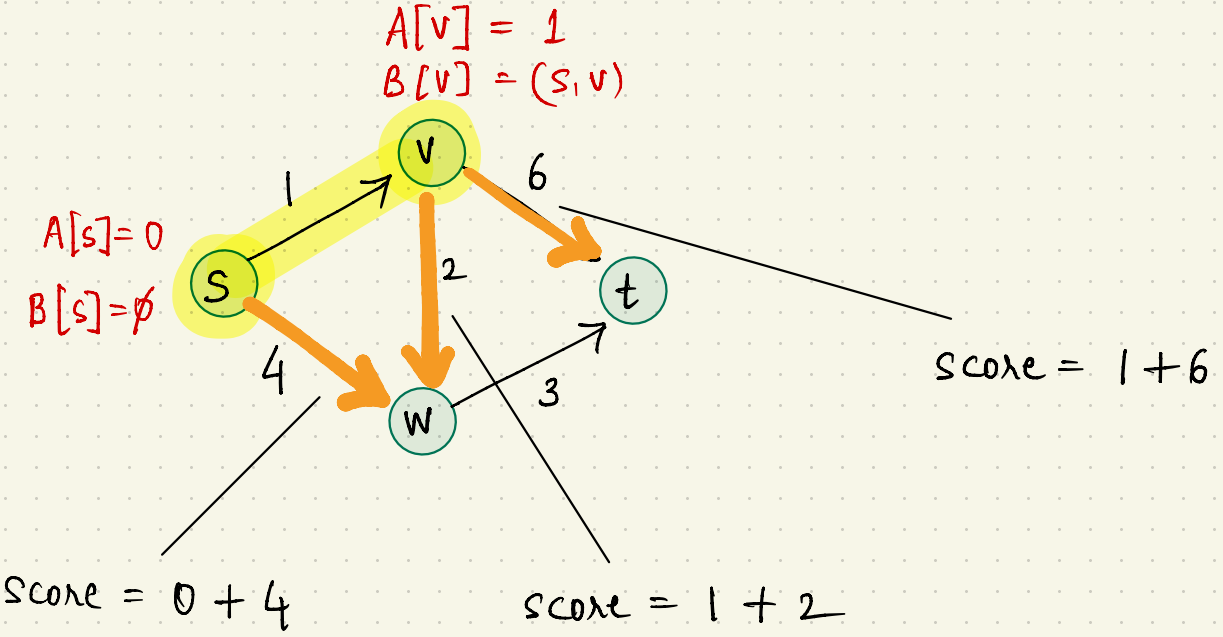
DIJKSTRA'S ALGORITHM



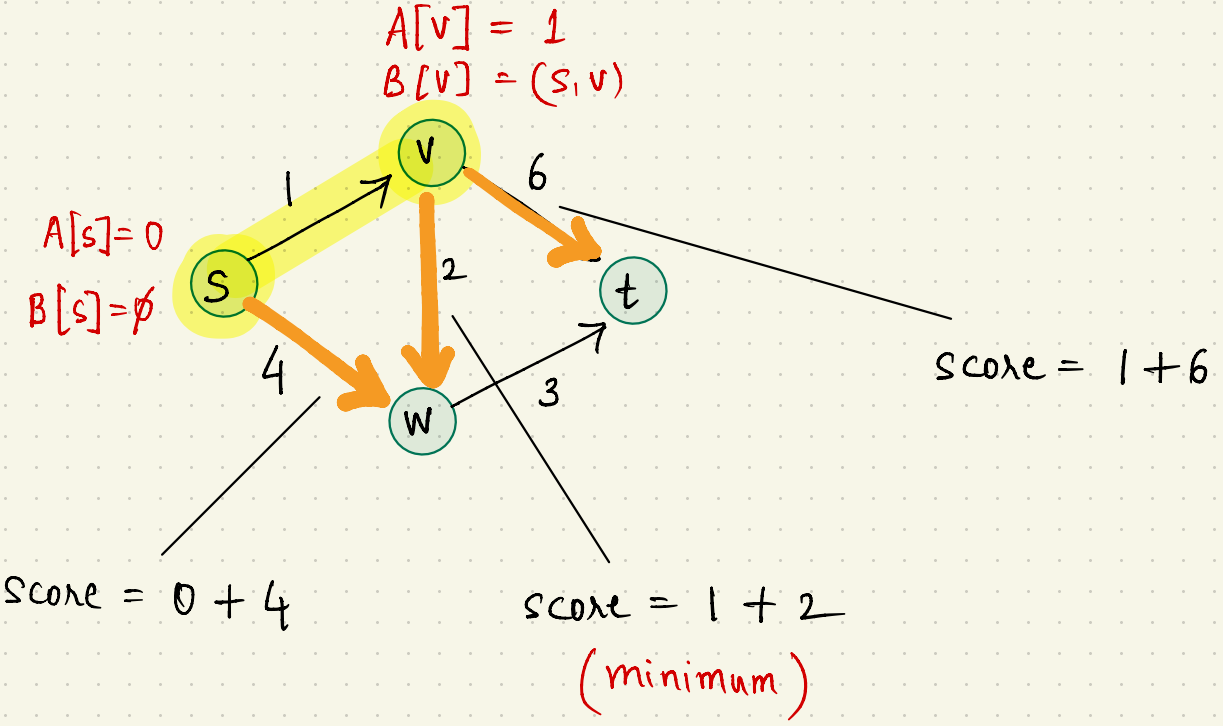
DIJKSTRA'S ALGORITHM



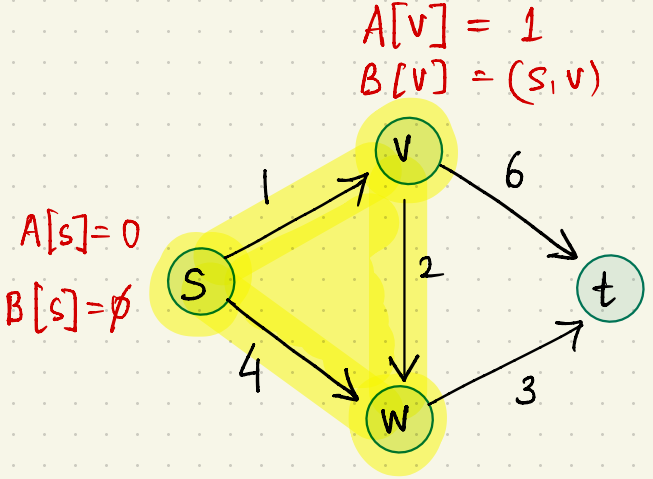
DIJKSTRA'S ALGORITHM



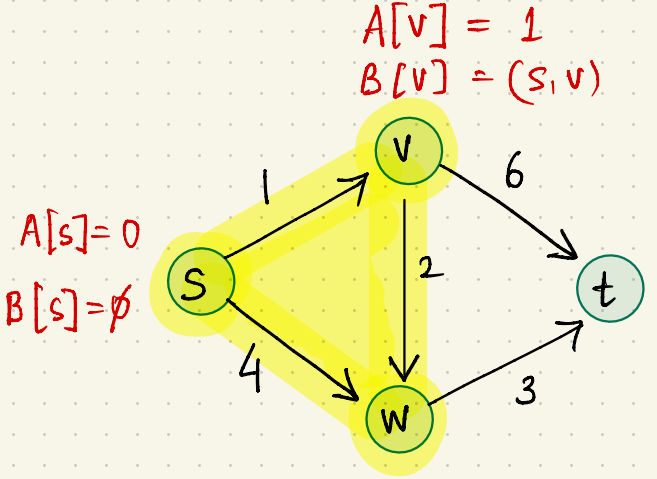
DIJKSTRA'S ALGORITHM



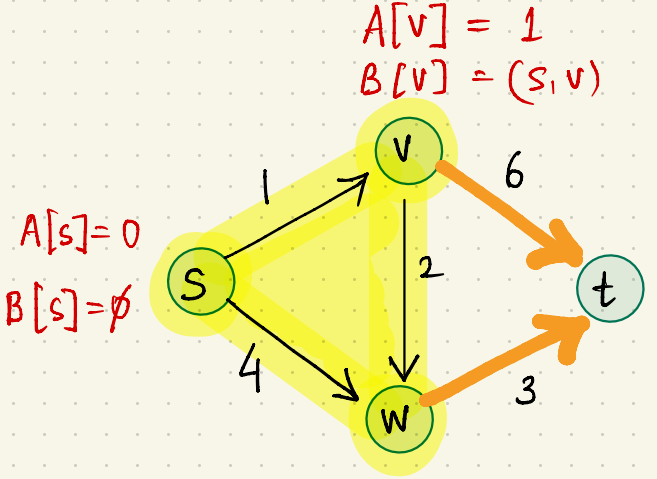
DIJKSTRA'S ALGORITHM



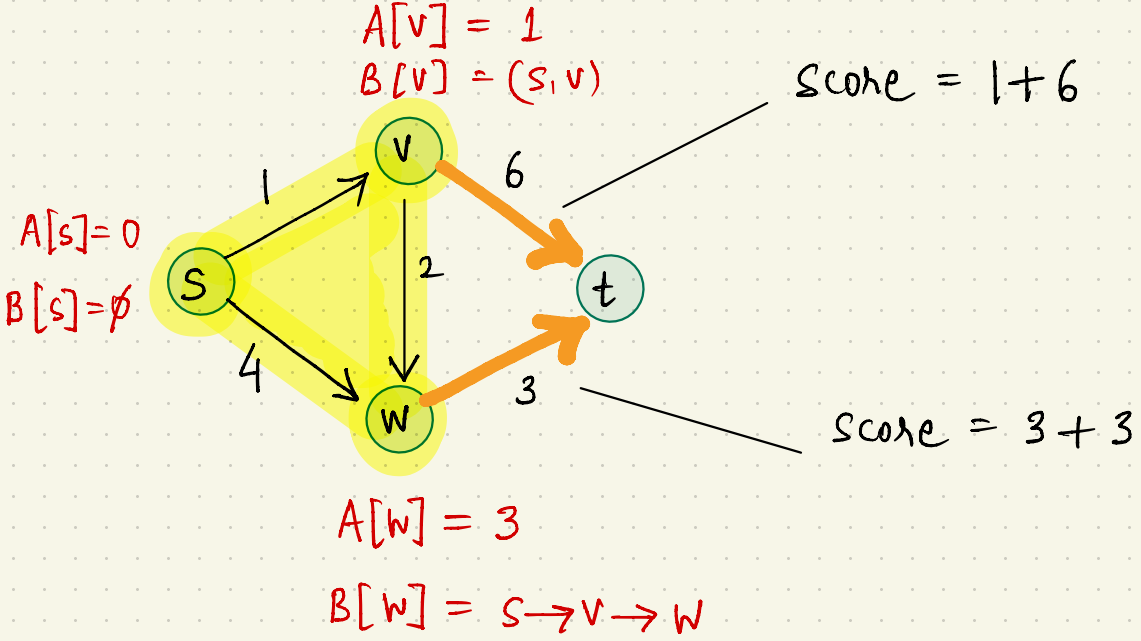
DIJKSTRA'S ALGORITHM



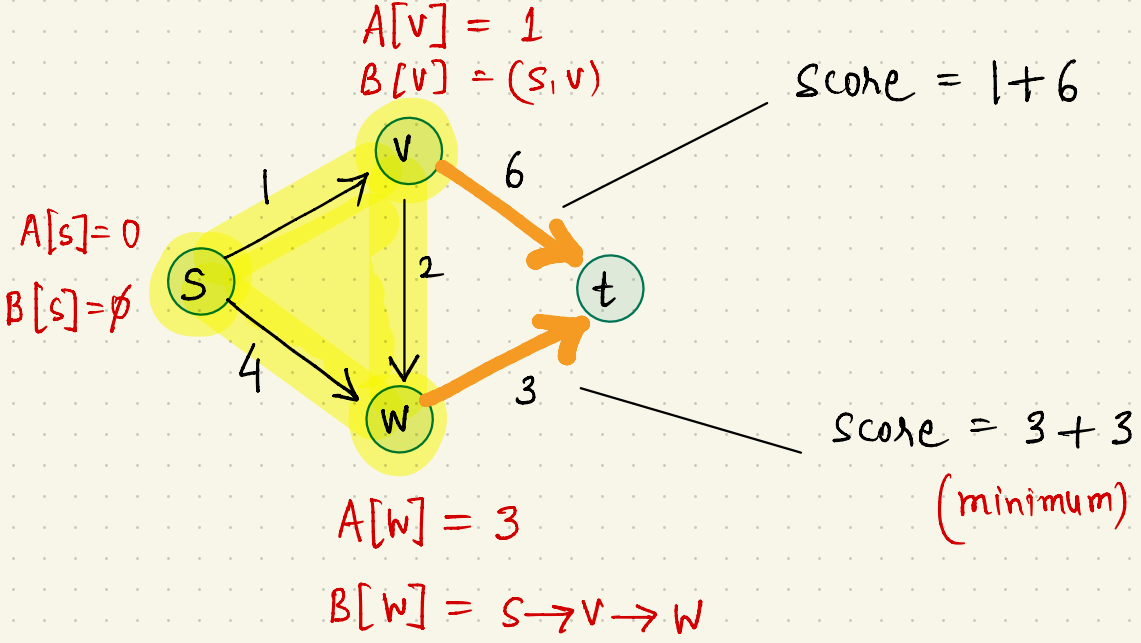
DIJKSTRA'S ALGORITHM



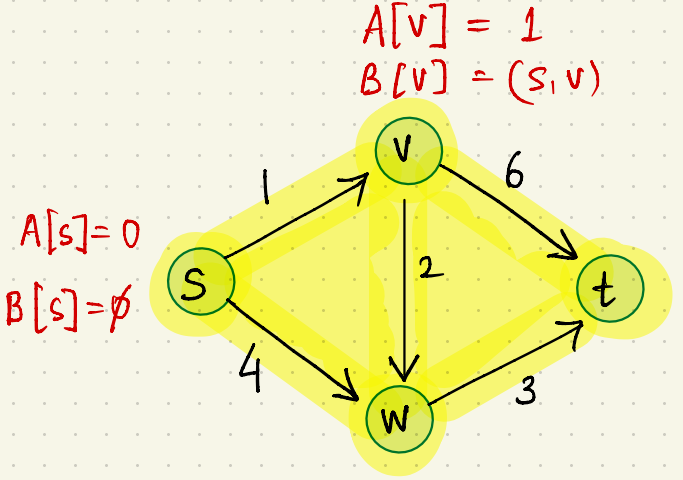
DIJKSTRA'S ALGORITHM



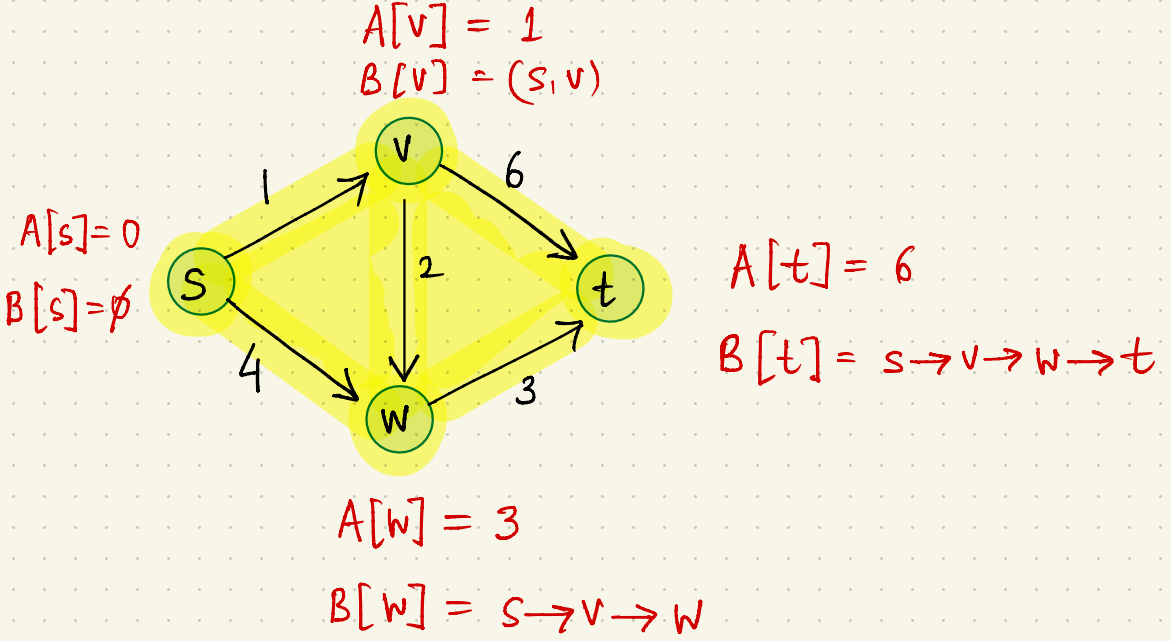
DIJKSTRA'S ALGORITHM



DIJKSTRA'S ALGORITHM

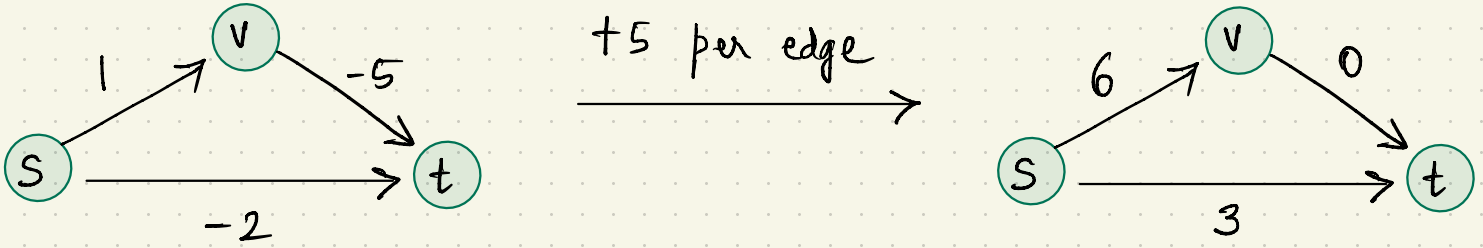


DIJKSTRA'S ALGORITHM

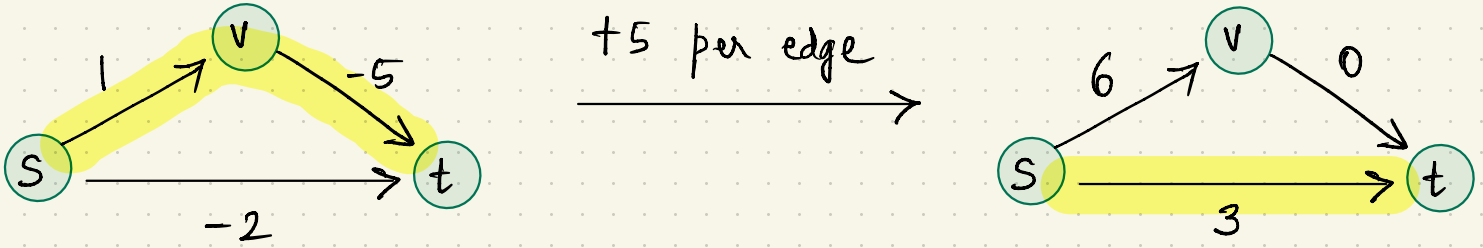


Negative edge lengths: what's the big deal?

🤔 Negative edge lengths: what's the big deal?

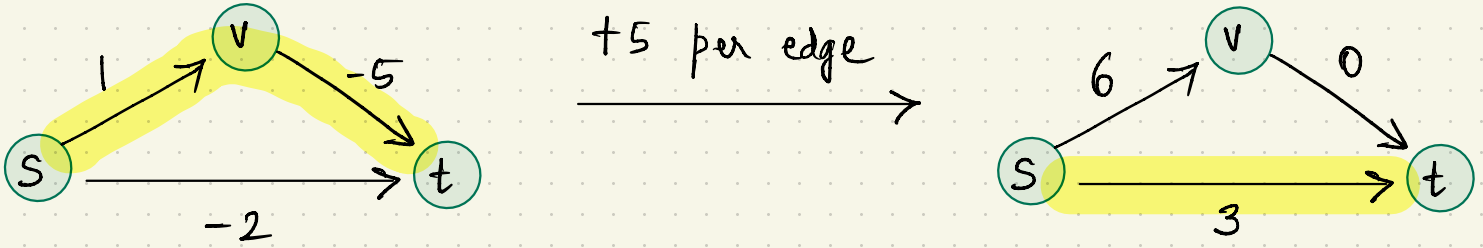


🤔 Negative edge lengths: what's the big deal?



Doesn't preserve shortest paths!

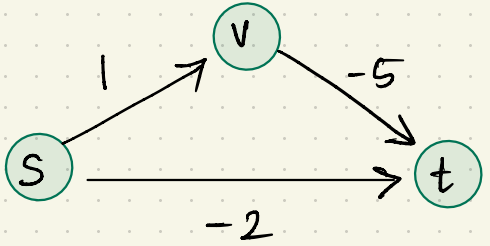
🤔 Negative edge lengths: what's the big deal?



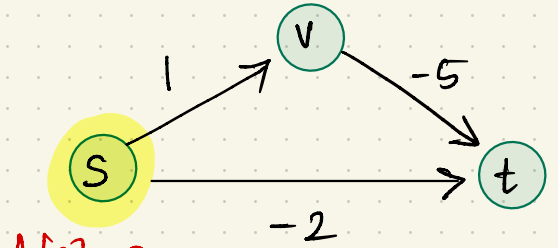
Doesn't preserve shortest paths!

Different paths have different number of edges.

🤔 Never mind. Why don't we run Dijkstra anyway?



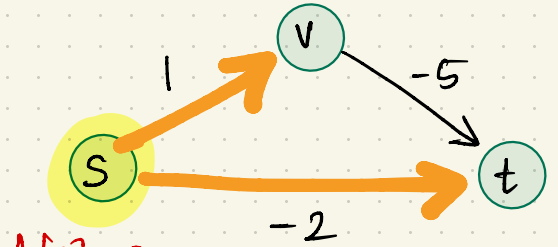
🤔 Never mind. Why don't we run Dijkstra anyway?



A[s] = 0

B[s] = ∅

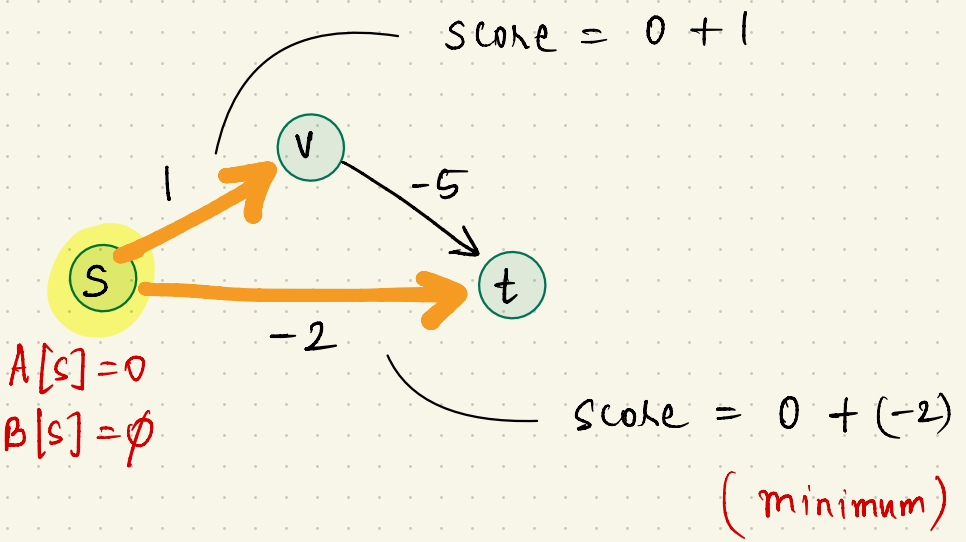
🤔 Never mind. Why don't we run Dijkstra anyway?



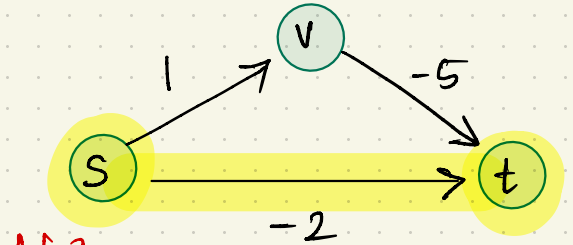
A[s] = 0

B[s] = ∅

🤔 Never mind. Why don't we run Dijkstra anyway?



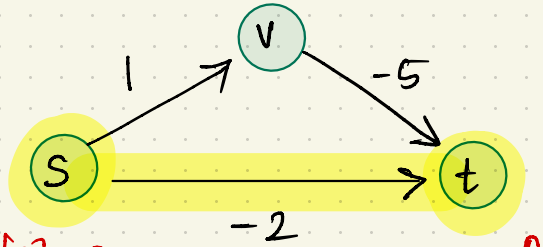
🤔 Never mind. Why don't we run Dijkstra anyway?



A[s] = 0

B[s] = ∅

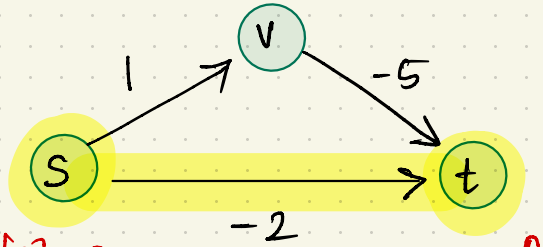
🤔 Never mind. Why don't we run Dijkstra anyway?



$A[s] = 0$
$B[s] = \emptyset$

$A[t] = -2$
$B[t] = s \to t$

🤔 Never mind. Why don't we run Dijkstra anyway?



A[s] = 0
B[s] = ∅

A[t] = -2
B[t] = s → t

INCORRECT!

s → v → t is shorter

DIJKSTRA'S ALGORITHM: CORRECTNESS

DIJKSTRA'S ALGORITHM: CORRECTNESS

Theorem: For every directed graph $G = (V, E)$ with non negative edge lengths and for every starting vertex s, at the conclusion of Dijkstra's algorithm,

$$A[v] = dist(s, v).$$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Theorem: For every directed graph $G = (V, E)$ with non negative edge lengths and for every starting vertex s, at the conclusion of Dijkstra's algorithm,

$$A[v] = dist(s, v).$$

length of shortest path from s to v

DIJKSTRA'S ALGORITHM: CORRECTNESS

Theorem: For every directed graph $G = (V, E)$ with non negative edge lengths and for every starting vertex s, at the conclusion of Dijkstra's algorithm,

$$A[v] = \text{dist}(s, v).$$

$$\text{length of } B[v] = \text{dist}(s, v).$$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Theorem: For every directed graph $G = (V, E)$ with non negative edge lengths and for every starting vertex s, at the conclusion of Dijkstra's algorithm,

$$A[v] = \text{dist}(s, v).$$

$$\text{length of } B[v] = \text{dist}(s, v).$$

Proof by strong induction.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

(and length of $B[v] = dist(s, v)$)

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Base case $k=1$: $A[s] = 0 = dist(s,s)$ ✓

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Base case $k=1$: $A[s] = 0 = dist(s,s)$ ✓

requires no negative length path from s to itself

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Base case $k=1$: $A[s] = 0 = dist(s,s)$ ✓

$B[s] = \emptyset$ ✓

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Assume for all previous iterations $1, 2, \dots, k-1$

$A[v] = dist(s,v)$ and length of $B[v] = dist(s,v)$.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Assume for all previous iterations $1, 2, \dots, k-1$

$A[v] = dist(s,v)$ and length of $B[v] = dist(s,v)$.

Let w^* be the k^{th} vertex added to X

say, through the edge (v^*, w^*) where $v^* \in X$, $w^* \notin X$.

Want: $A[w^*] = dist(s, w^*)$

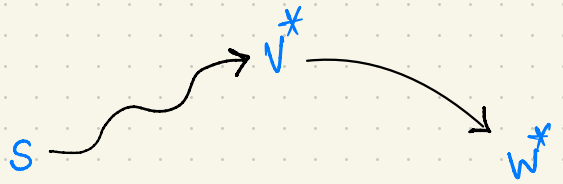
length of $B[w^*] = dist(s, w^*)$.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step:

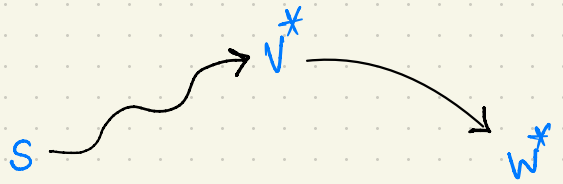


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step:

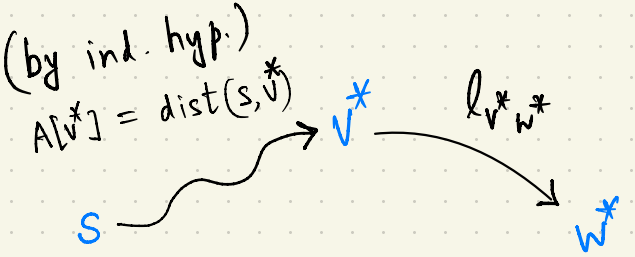


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step:

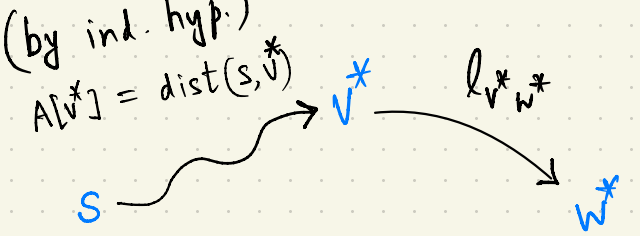


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step:



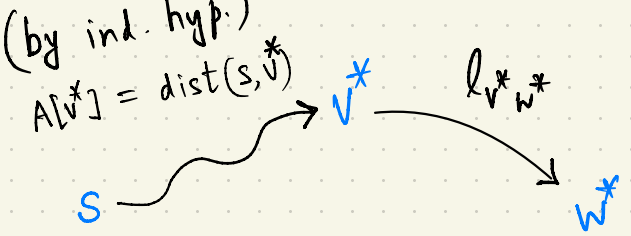
We set $A[w^*] = A[v^*] + l_{v^*w^*}$ and $B[w^*] = B[v^*] \cup (v^*, w^*)$.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step:



We set $A[w^*] = A[v^*] + \ell_{v^*w^*}$ and $B[w^*] = B[v^*] \cup (v^*, w^*)$.

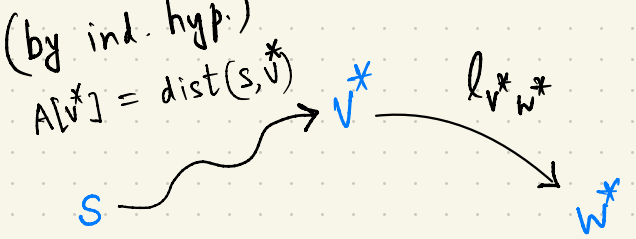
$$\Rightarrow \quad dist(s,w^*) \leq A[v^*] + \ell_{v^*w^*}$$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step:



We set $A[w^*] = A[v^*] + \ell_{v^*w^*}$ and $B[w^*] = B[v^*] \cup (v^*, w^*)$.

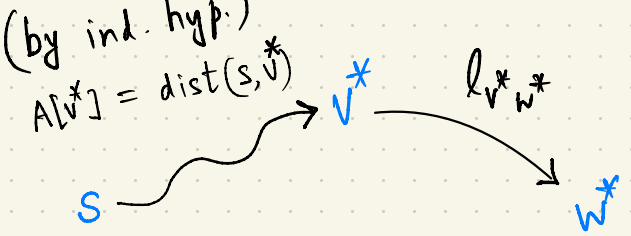
$\Rightarrow$ $\boxed{dist(s,w^*) \leq A[v^*] + \ell_{v^*w^*}}$ — ☺

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step:



We set $A[w^*] = A[v^*] + \ell_{v^*w^*}$ and $B[w^*] = B[v^*] \cup (v^*, w^*)$.

$\Rightarrow$ $\boxed{dist(s,w^*) \leq A[v^*] + \ell_{v^*w^*}}$ —— ☺

Want: " " $\geq$ " " "

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Suffices to show that every $s-w^*$ path has length at least $A[v^*] + \ell_{v^*w^*}$.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Suffices to show that every $s-w^*$ path has length at least $A[v^*] + \ell_{v^*w^*}$.

Let P be any $s-w^*$ path.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Suffices to show that every $s - w^*$ path has length at least $A[v^*] + \ell_{v^* w^*}$.

Let P be any $s - w^*$ path.

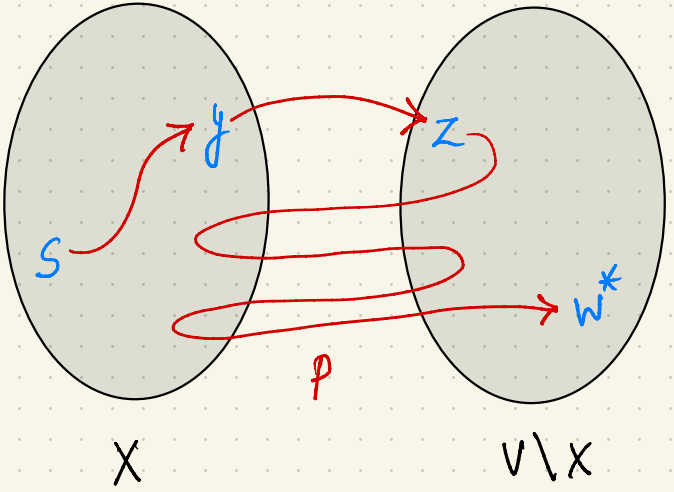
Any such path must cross the frontier between X and $V \setminus X$.

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step:

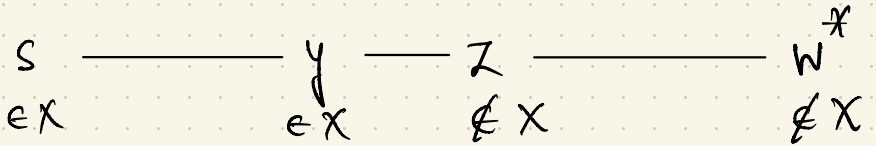


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Any $s - w^*$ path P has the form:

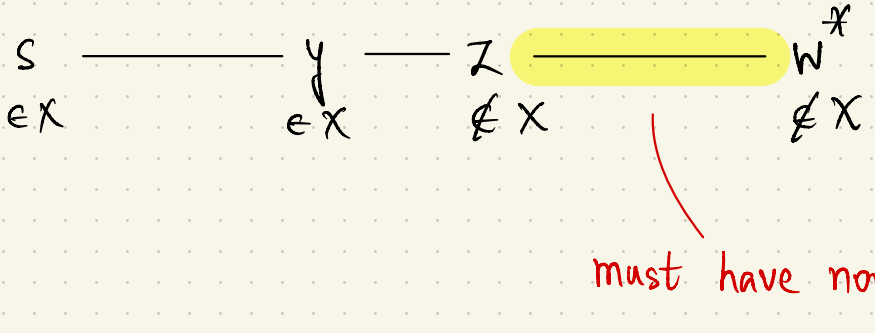


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Any $s - w^*$ path P has the form:

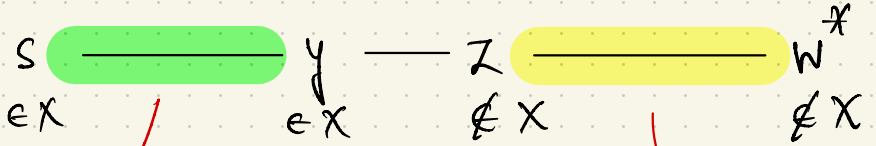


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Any $s-w^*$ path P has the form:



lies entirely in X

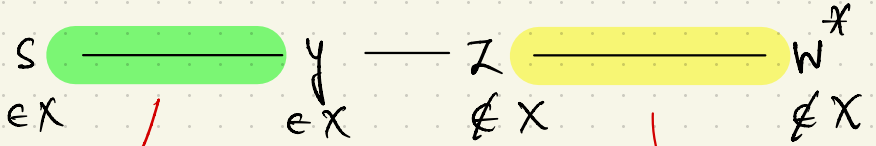
must have nonnegative length

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Any $s - w^*$ path P has the form:



lies entirely in X

length $\geq$ dist (s, y)

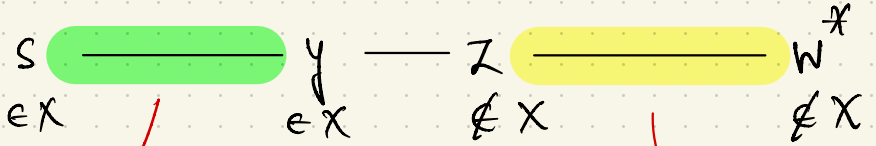
must have nonnegative length

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Any $s - w^*$ path P has the form:



lies entirely in X

length $\geq$ $dist(s, y)$

$= A[y]$ (by ind. hyp.)

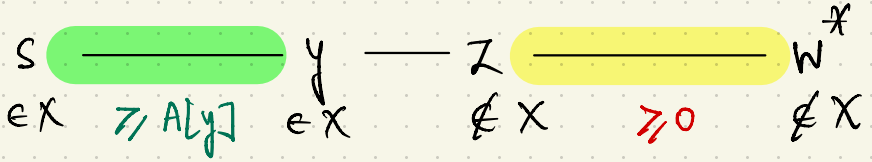
must have nonnegative length

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: Any $s - w^*$ path P has the form:

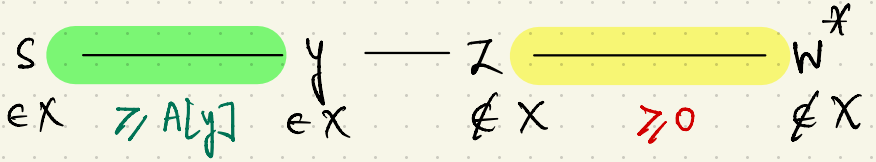


DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Any $s-w^*$ path P has the form:



length of path P = length $s \rightsquigarrow y$ + length $y \rightsquigarrow z$ + length $z \rightsquigarrow w^*$

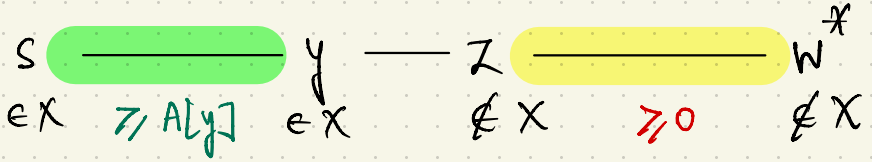
$\geq A[y] + l_{yz}$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Any $s \to w^*$ path P has the form:



length of path P = length $s \rightsquigarrow y$ + length $y \rightsquigarrow z$ + length $z \rightsquigarrow w^*$

$\geq A[y] + \ell_{yz}$

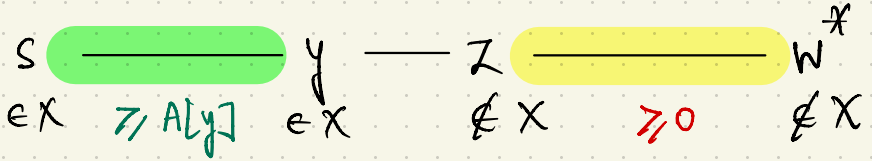
By **Dijkstra score criterion**: $A[v^*] + \ell_{v^*w^*} \leq A[y] + \ell_{yz}$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Any $s \rightsquigarrow w^*$ path P has the form:



length of path P = length $s \rightsquigarrow y$ + length $y \rightsquigarrow z$ + length $z \rightsquigarrow w^*$

$\geq A[y] + l_{yz}$

By **Dijkstra score criterion**: $A[v^*] + l_{v^*w^*} \leq A[y] + l_{yz}$

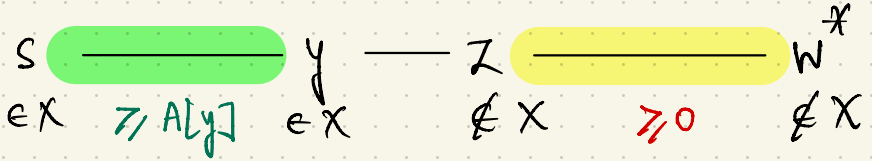
(y,z) was available to algorithm, but it chose (v^*, w^*).

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Any $s \to w^*$ path P has the form:



length of path P = length $s \rightsquigarrow y$ + length $y \rightsquigarrow z$ + length $z \rightsquigarrow w^*$

$\geq A[y] + \ell_{yz}$

By Dijkstra score criterion: $A[v^*] + \ell_{v^*w^*} \leq A[y] + \ell_{yz}$

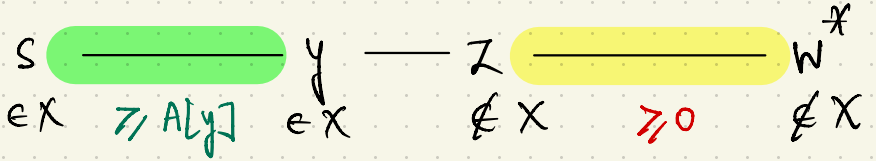
$\leq$ length of path P

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s,v)$.

Ind. step: Any $s-w^*$ path P has the form:



length of path P = length $s \rightsquigarrow y$ + length $y \rightsquigarrow z$ + length $z \rightsquigarrow w^*$

$\geq A[y] + \ell_{yz}$

By Dijkstra score criterion: $A[v^*] + \ell_{v^*w^*} \leq A[y] + \ell_{yz}$

$\leq$ length of path P ☺☺

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: By ☺☺

$$dist(s, w^*) \geq A[v^*] + l_{v^*w^*}$$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: By ☺☺

$$dist(s, w^*) \geq A[v^*] + \ell_{v^* w^*}$$

By ☺

$$dist(s, w^*) \leq A[v^*] + \ell_{v^* w^*}$$

DIJKSTRA'S ALGORITHM: CORRECTNESS

Proof: (by strong induction on the number of iterations)

$P(k)$: for the k^{th} vertex v added to X, $A[v] = dist(s, v)$.

Ind. step: By ☺☺

$$dist(s, w^*) \geq A[v^*] + \ell_{v^* w^*}$$

By ☺

$$dist(s, w^*) \leq A[v^*] + \ell_{v^* w^*}$$

$\Rightarrow$ $s \rightarrow v^* \rightarrow w^*$ is the shortest path to w^*.

DIJKSTRA'S ALGORITHM: RUNNING TIME

DIJKSTRA'S ALGORITHM: RUNNING TIME

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

$B[s] = \emptyset$ computed shortest paths

// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

Add w^* to X.

Set $A[w^*] := A[v^*] + \ell_{v^*w^*}$ and $B[w^*] := B[v^*] \cup (v^*, w^*)$.

DIJKSTRA'S ALGORITHM: RUNNING TIME

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

Add w^* to X.

Set $A[w^*] := A[v^*] + \ell_{v^*w^*}$

DIJKSTRA'S ALGORITHM: RUNNING TIME

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

iterations $= n-1$ (n = # vertices)

// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

Add w^* to X.

Set $A[w^*] := A[v^*] + \ell_{v^*w^*}$

DIJKSTRA'S ALGORITHM: RUNNING TIME

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

iterations = $n-1$ (n = # vertices)

// main loop

while $X \neq V$ need to grow X

checks = m (m = # edges)

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$A[v] + l_{vw}$ Dijkstra score

Call it (v^*, w^*).

Add w^* to X.

Set $A[w^*] := A[v^*] + l_{v^*w^*}$

DIJKSTRA'S ALGORITHM: RUNNING TIME

Naive implementation: $O(mn)$

DIJKSTRA'S ALGORITHM: RUNNING TIME

Naive implementation: $O(mn)$

Can we do better?

DIJKSTRA'S ALGORITHM: RUNNING TIME

Initialize

$X := \{s\}$ vertices processed so far

$A[s] = 0$ computed shortest-path distances

How to speed up computation of "minimum"?

// main loop

while $X \neq V$ need to grow X

among all edges $(v,w) \in E$ with $v \in X$ and $w \notin X$, pick one that minimizes

$$A[v] + \ell_{vw}$$ Dijkstra score

Call it (v^*, w^*).

Add w^* to X.

Set $A[w^*] := A[v^*] + \ell_{v^*w^*}$

DIJKSTRA'S ALGORITHM: RUNNING TIME

Naive implementation: $O(mn)$

Can we do better?

Yes! $O((m+n) \cdot \log n)$ with heaps.

(see "Additional Reading").