

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 9

GRAPH ALGORITHMS II : REPRESENTATION (CONTD.) AND SEARCH

AUG 12, 2026

|

ROHIT VAISH

REPRESENTING GRAPHS

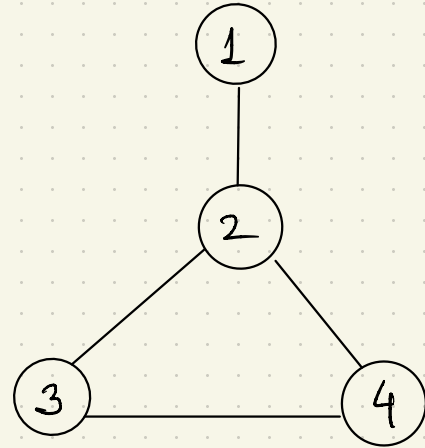
REPRESENTING GRAPHS

Adjacency matrix

REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$



REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \end{matrix}$$

$\Theta(n^2)$ space

OK for dense graphs

Not OK for sparse graphs

REPRESENTING GRAPHS

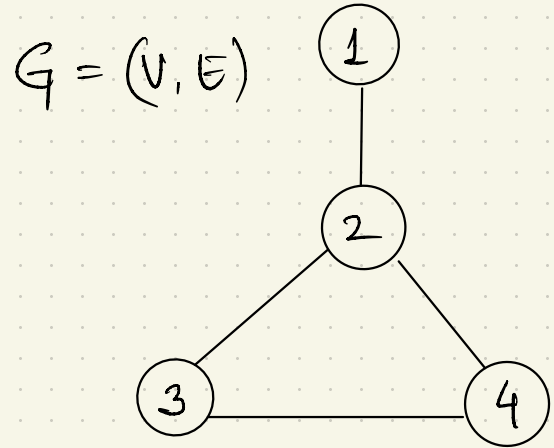
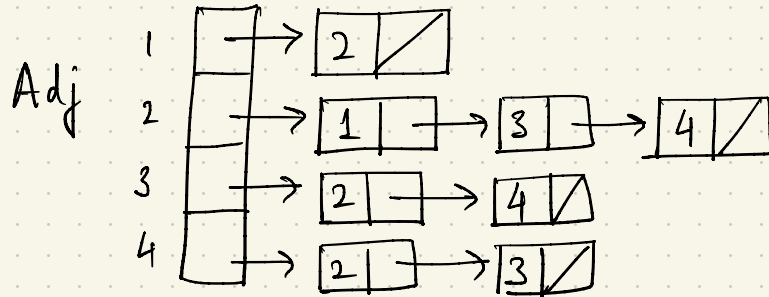
Adjacency list

array of $|V|$ linked lists

for each vertex $u \in V$,

$\text{Adj}[u]$ stores neighbors of u

$$\{ v \in V : \{u, v\} \in E \}.$$



Space : $\Theta(|V| + |E|)$
or $\Theta(m + n)$

REPRESENTING GRAPHS

Which is better : adjacency matrix or adjacency list ?

REPRESENTING GRAPHS

Which is better : adjacency matrix or adjacency list ?

Depends on graph density and operations needed.

REPRESENTING GRAPHS

Which is better : adjacency matrix or adjacency list ?

This course

Depends on graph density and operations needed.

REPRESENTING GRAPHS

Which is better : adjacency matrix or adjacency list? (This course

Depends on graph density and operations needed.

Adjacency lists

- great for "follow your nose" operations
- better for massive sparse graphs
(e.g., the web graph)

GRAPH SEARCH

GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$ and a starting vertex s

goal: identify the vertices of V reachable from s

WHY CARE ABOUT GRAPH SEARCH

WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...

WHY CARE ABOUT GRAPH SEARCH

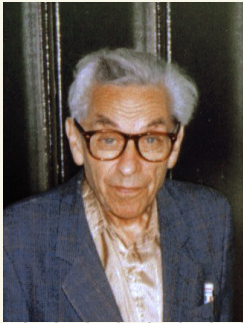
Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?

WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

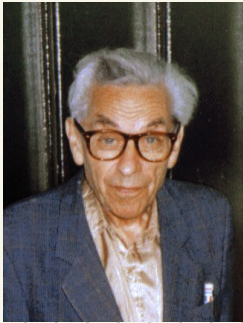
- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?



WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?

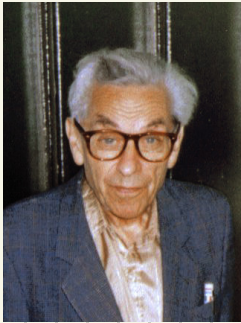


0

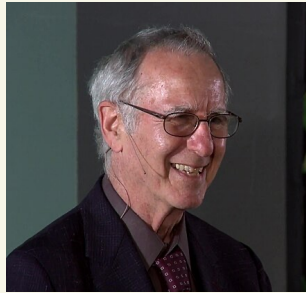
WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?



0

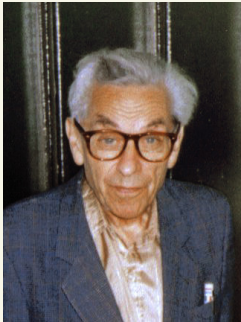


1

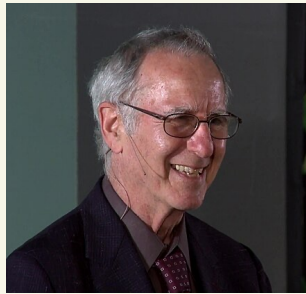
WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?



0



1

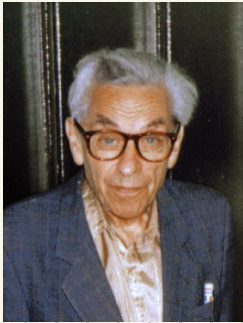


2

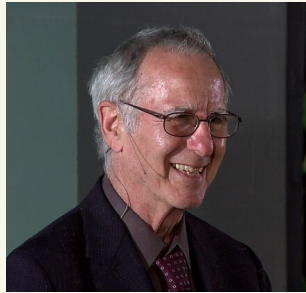
WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Endős number ≤ 3 ?



0



1



2



3

WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?

Shortest paths

- fewest moves to solve Rubik's cube

WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdős number ≤ 3 ?

Shortest paths

- fewest moves to solve Rubik's cube

Revealing structure

- finding "communities" in networks

GENERIC GRAPH SEARCH

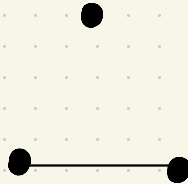
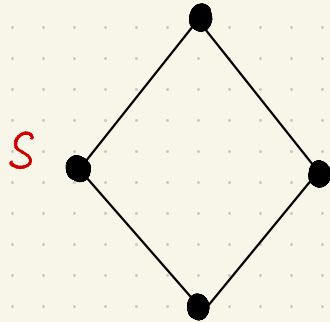
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s

GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

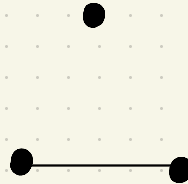
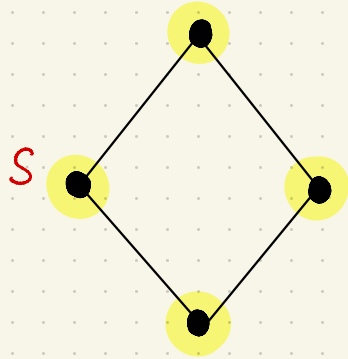
goal: identify the vertices of V reachable from s



GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

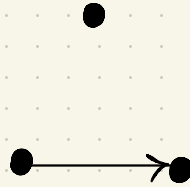
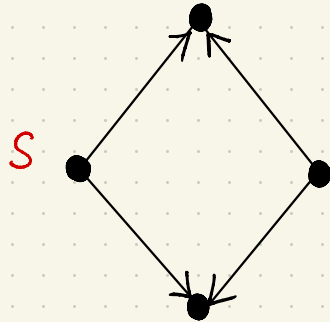
goal: identify the vertices of V reachable from s



GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

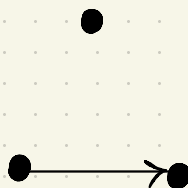
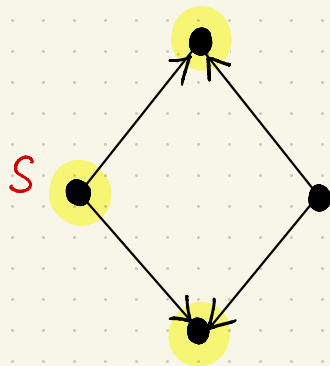
goal: identify the vertices of V reachable from s



GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s



GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

$O(|V|+|E|)$ time

GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is explored, other vertices unexplored

GENERIC GRAPH SEARCH

input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u,v) \in E$ with u explored
and v unexplored
| mark v explored

GENERIC GRAPH SEARCH

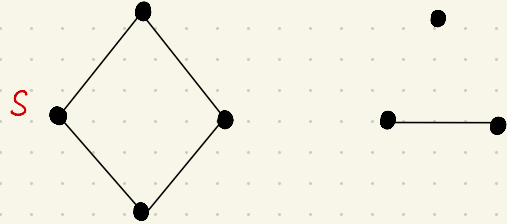
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

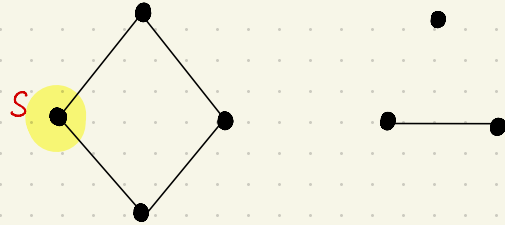
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u,v) \in E$ with u explored
and v unexplored
| mark v explored



GENERIC GRAPH SEARCH

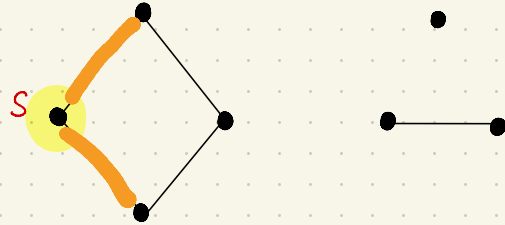
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

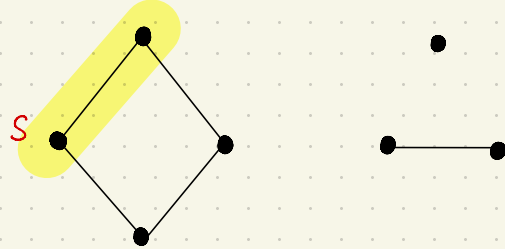
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u,v) \in E$ with u explored
and v unexplored
| mark v explored



GENERIC GRAPH SEARCH

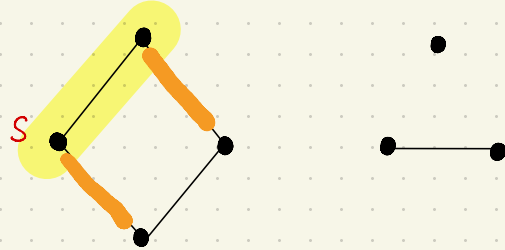
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

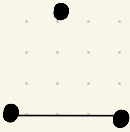
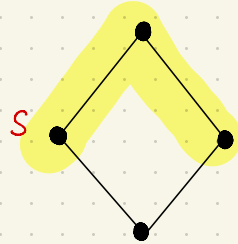
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

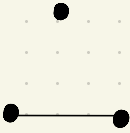
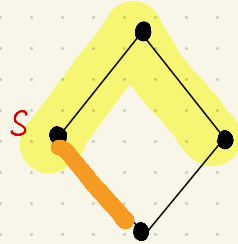
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

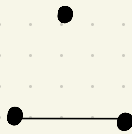
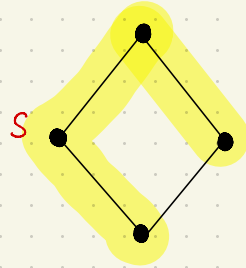
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u,v) \in E$ with u explored
and v unexplored
└ mark v explored



GENERIC GRAPH SEARCH

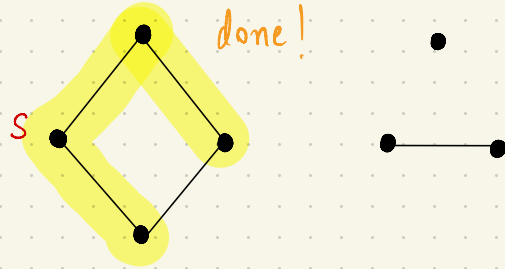
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

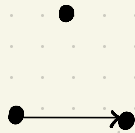
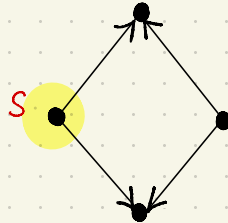
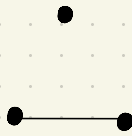
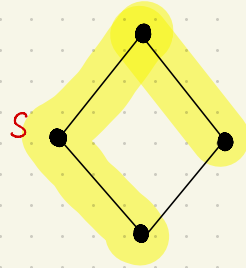
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

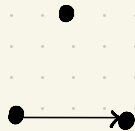
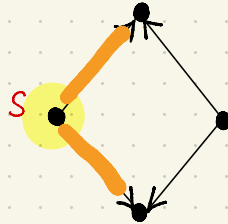
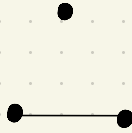
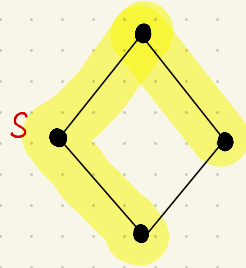
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

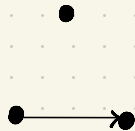
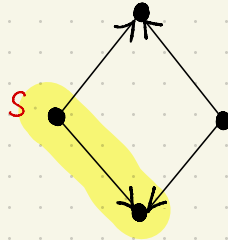
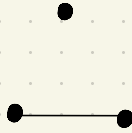
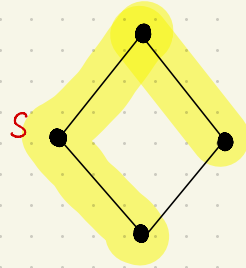
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

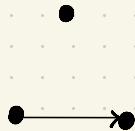
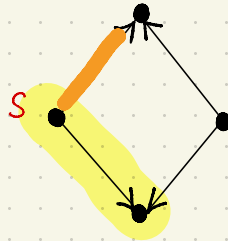
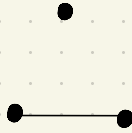
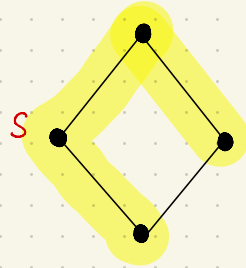
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

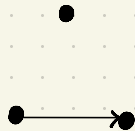
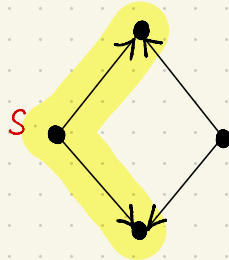
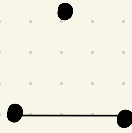
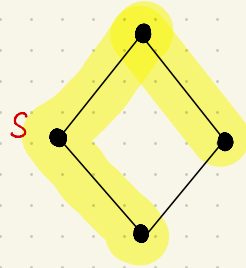
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

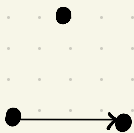
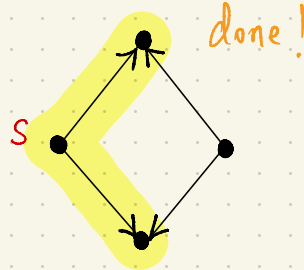
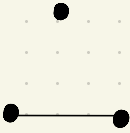
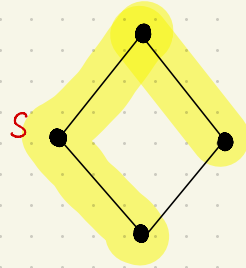
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s efficiently

Generic Algorithm

initially s is **explored**, other vertices **unexplored**

while some edge $(u,v) \in E$ with u **explored**
and v **unexplored**
└ mark v **explored**



GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

v is marked explored $\iff G$ has a path from s to v

GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

v is marked explored $\iff G$ has a path from s to v

Proof sketch: ($\implies$) Only way to explore is via paths from s .

GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

v is marked explored $\iff G$ has a path from s to v

Proof sketch: $(\implies)$ Only way to explore is via paths from s .
 $(\impliedby)$

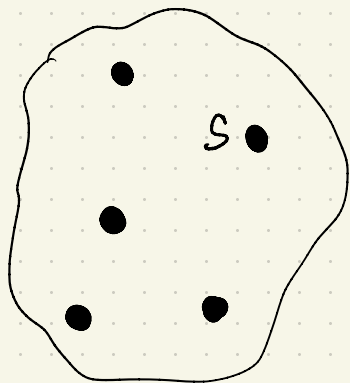
GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

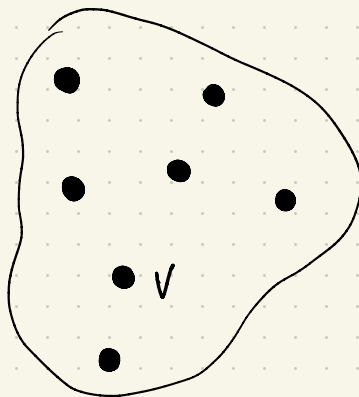
v is marked explored $\iff G$ has a path from s to v

Proof sketch: ($\implies$) Only way to explore is via paths from s .

($\impliedby$) by contradiction.



explored



unexplored

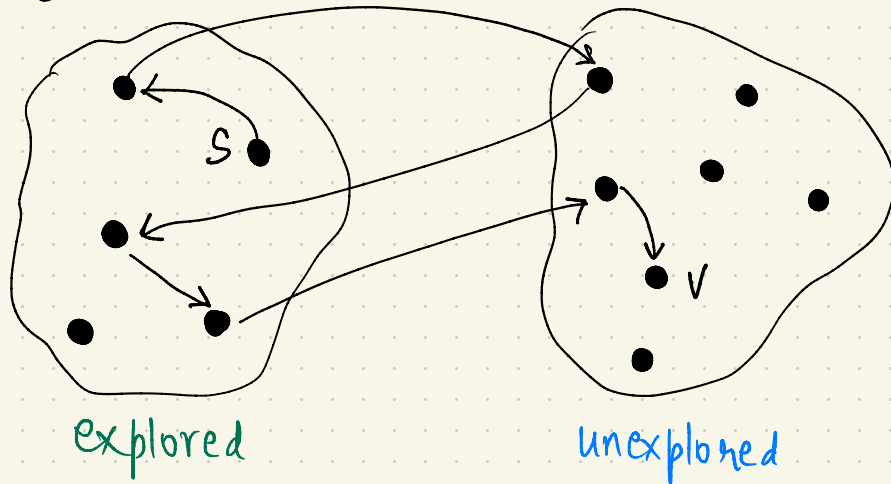
GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

v is marked explored $\iff G$ has a path from s to v

Proof sketch: ($\implies$) Only way to explore is via paths from s .

($\impliedby$) by contradiction.



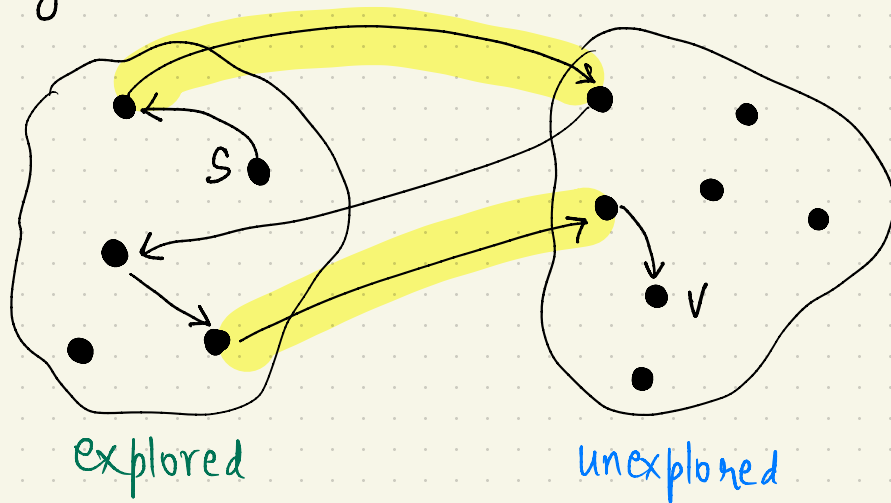
GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

v is marked explored $\iff G$ has a path from s to v

Proof sketch: $(\implies)$ Only way to explore is via paths from s .

$(\impliedby)$ by contradiction.



on $s \rightsquigarrow v$ path, there must
be some $(x, y) \in E$ such that
 x : explored y : unexplored

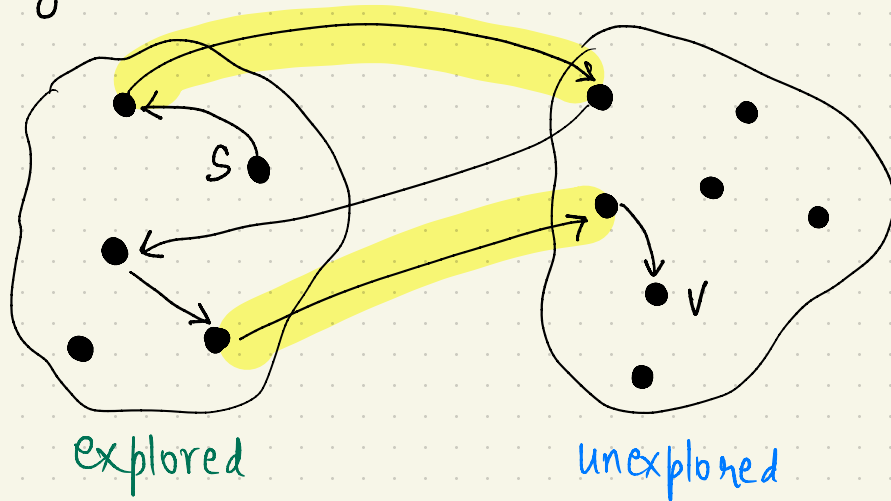
GENERIC GRAPH SEARCH

Claim: At the conclusion of Generic algorithm,

v is marked explored $\iff G$ has a path from s to v

Proof sketch: $(\implies)$ Only way to explore is via paths from s .

$(\impliedby)$ by contradiction.



on $s \rightsquigarrow v$ path, there must be some $(u, v) \in E$ such that
 u : explored v : unexplored

But then, algo. wouldn't have terminated.



Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u, v) \in E$ with u explored
and v unexplored
 $\downarrow$
 mark v explored

How to choose among
the "crossing" edges?

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u, v) \in E$ with u explored
and v unexplored
 $\downarrow$
 mark v explored

How to choose among
the "crossing" edges?

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u, v) \in E$ with u explored
and v unexplored
 mark v explored

Breadth-First Search: explore nodes in "layers" (e.g., Endő's number)

How to choose among
the "crossing" edges?

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u, v) \in E$ with u explored
and v unexplored
 $\downarrow$
 mark v explored

Breadth-First Search: explore nodes in "layers" (e.g., Endő's number)
- shortest paths and connected components

How to choose among
the "crossing" edges?

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u,v) \in E$ with u explored
and v unexplored
 mark v explored

Breadth-First Search: explore nodes in "layers" (e.g., Erdős number)
- shortest paths and connected components

Depth-First Search: explores aggressively like a maze

How to choose among
the "crossing" edges?

Generic Algorithm

initially s is explored, other vertices unexplored

while some edge $(u, v) \in E$ with u explored
and v unexplored
 mark v explored

Breadth-First Search: explore nodes in "layers" (e.g., Endő's number)

- shortest paths and connected components

Depth-First Search: explores aggressively like a maze

- topological ordering in DAGs
- strongly connected components in directed graphs

How to choose among
the "crossing" edges?

Generic Algorithm

initially s is explored, other vertices unexplored

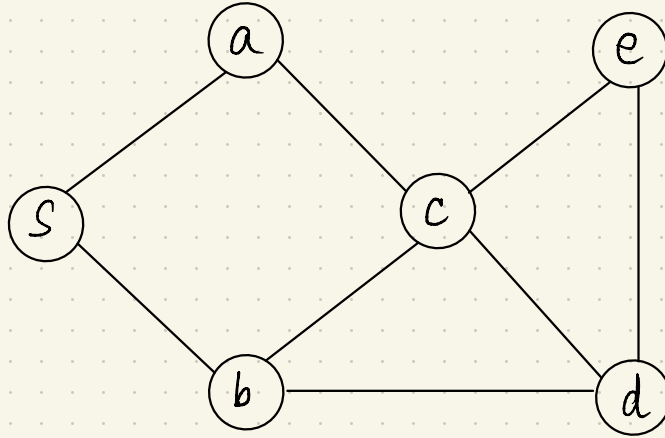
while some edge $(u, v) \in E$ with u explored
and v unexplored
 mark v explored

Breadth-First Search

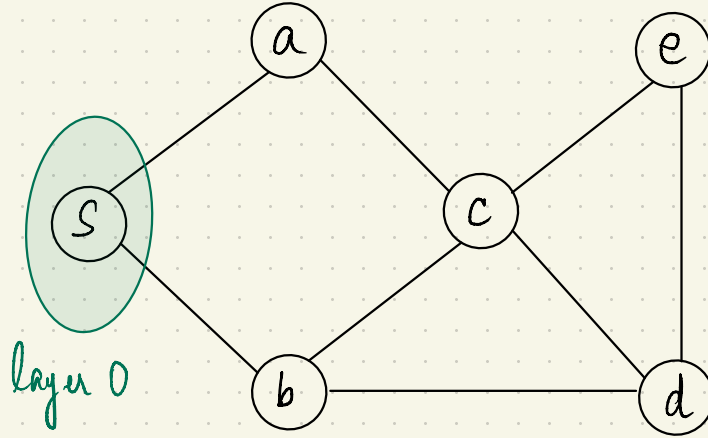
Depth-First Search

Linear time : $O(m + n)$

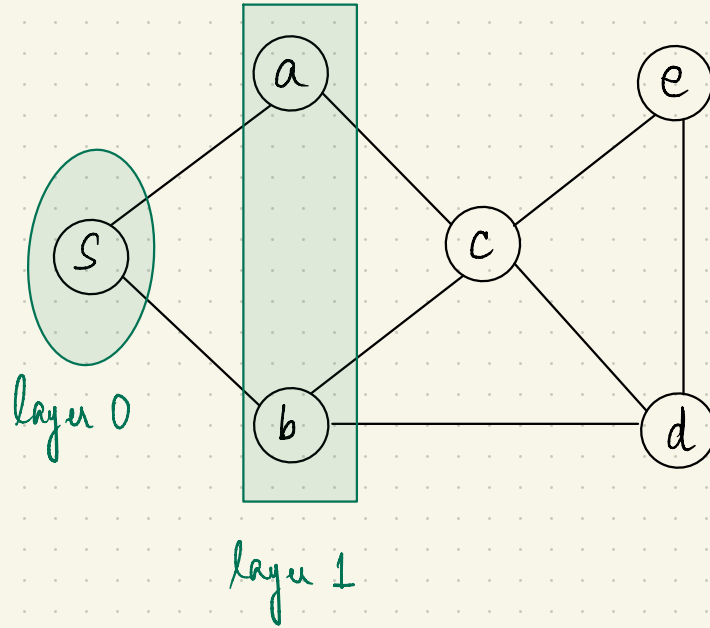
BREADTH - FIRST SEARCH



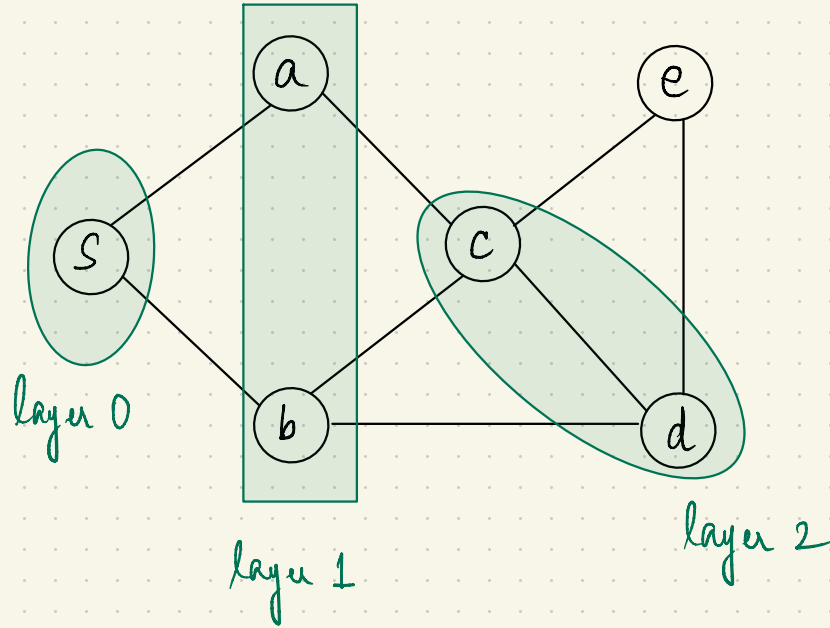
BREADTH - FIRST SEARCH



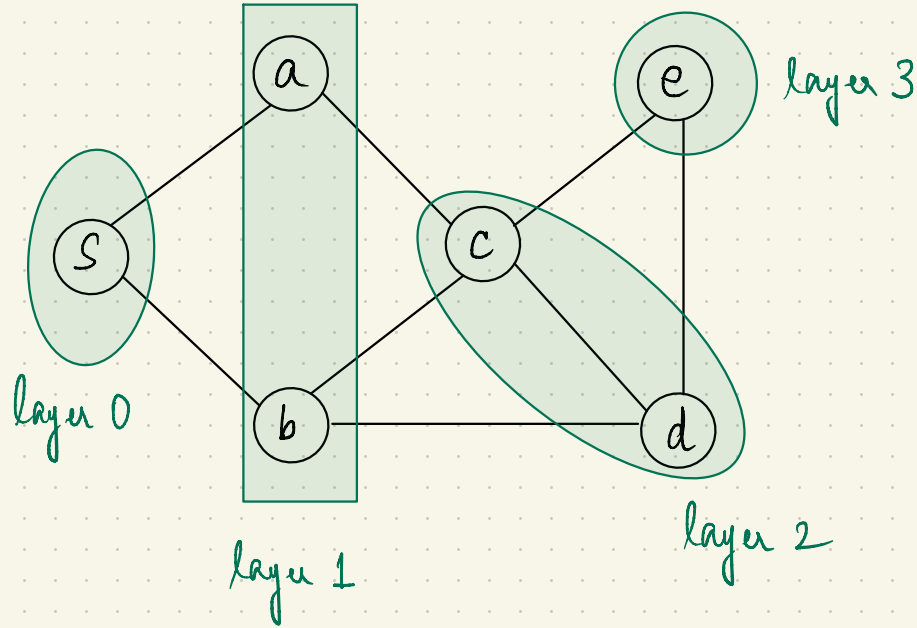
BREADTH - FIRST SEARCH



BREADTH - FIRST SEARCH



BREADTH - FIRST SEARCH



BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked explored.

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked explored.

mark s as explored, all other vertices unexplored

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

|

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

 |

|

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

 if w is **unexplored**

 mark w as **explored**

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

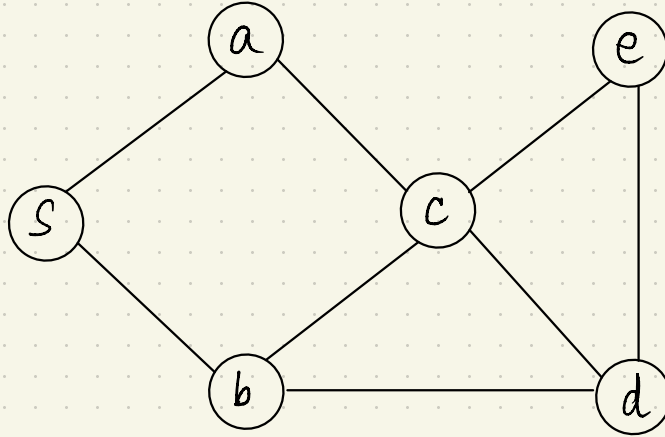
 for each $(v, w) \in E$

 if w is **unexplored**

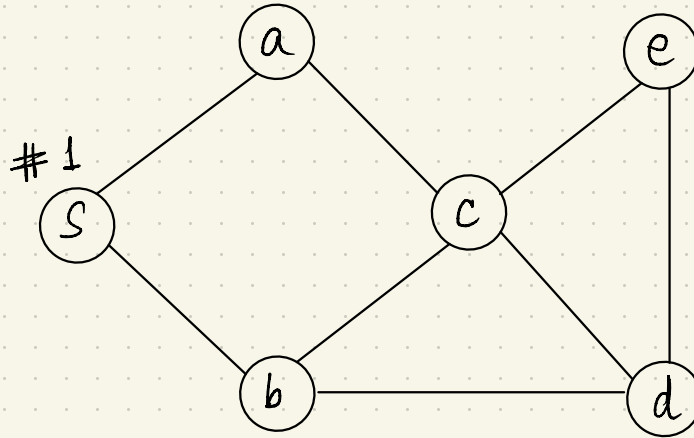
 mark w as **explored**

 add w to the end of Q

BREADTH - FIRST SEARCH

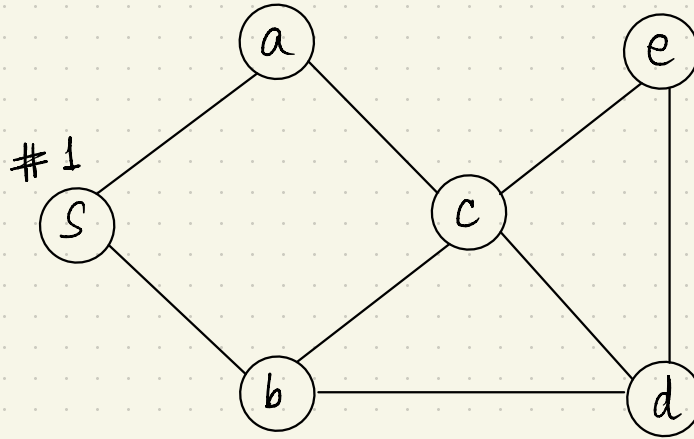


BREADTH - FIRST SEARCH



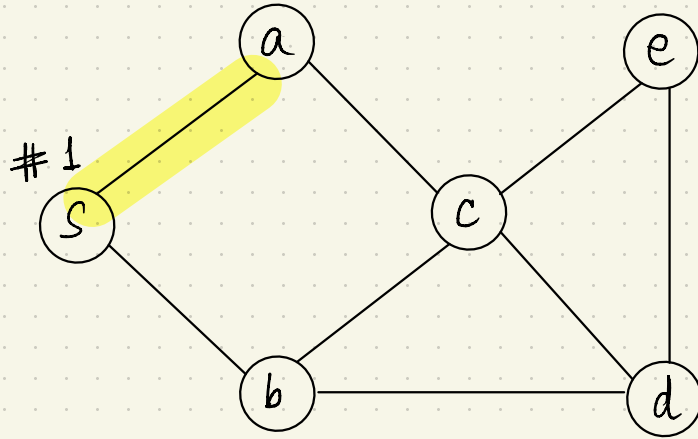
Q : s

BREADTH - FIRST SEARCH



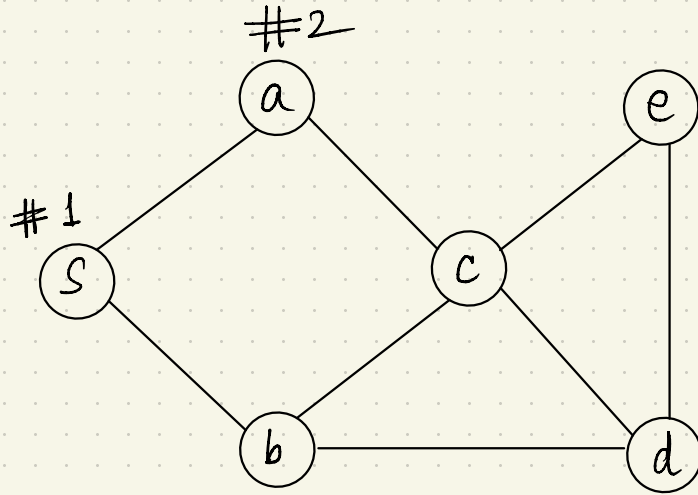
Q : ~~S~~

BREADTH - FIRST SEARCH



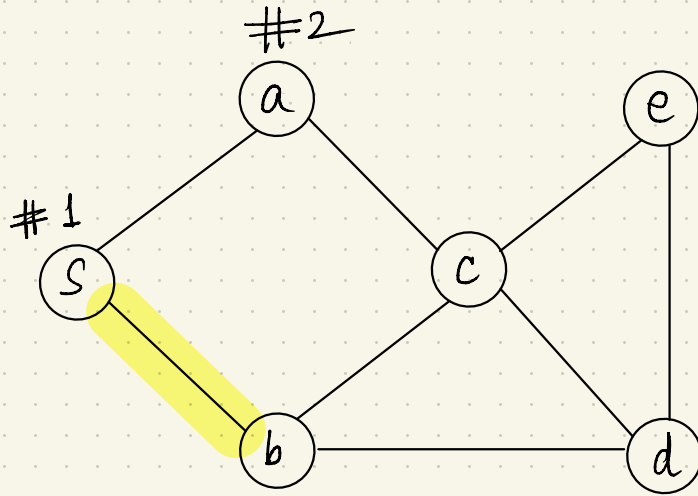
Q : ~~X~~

BREADTH - FIRST SEARCH



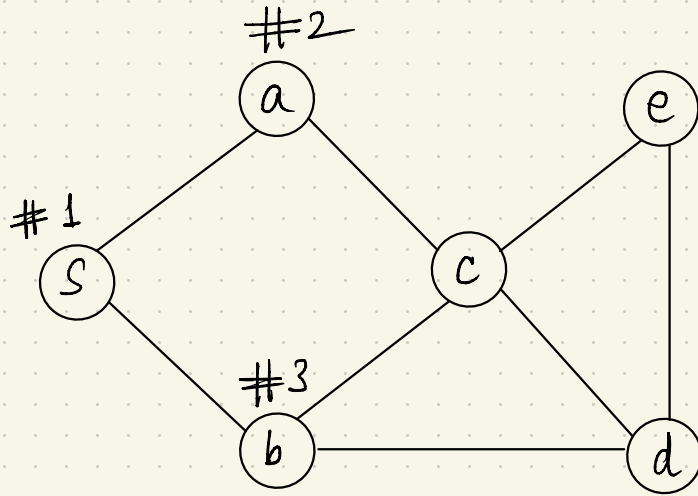
Q : ~~s~~ a

BREADTH - FIRST SEARCH



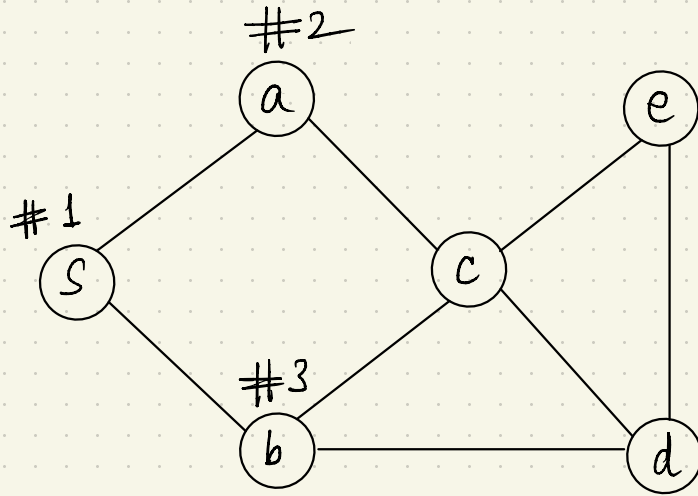
Q : ~~s~~ a

BREADTH - FIRST SEARCH



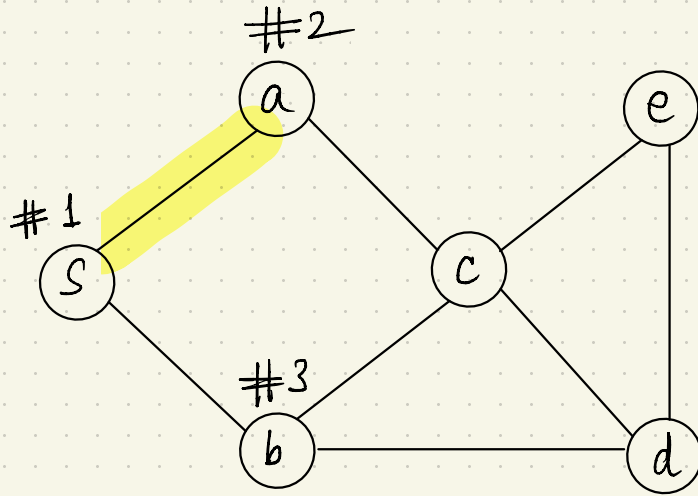
Q : ~~s~~ a b

BREADTH - FIRST SEARCH



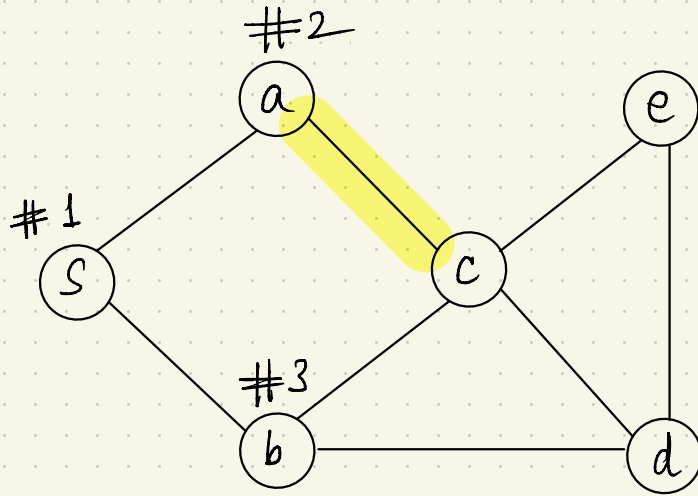
Q : ~~s~~ ~~a~~ b

BREADTH - FIRST SEARCH



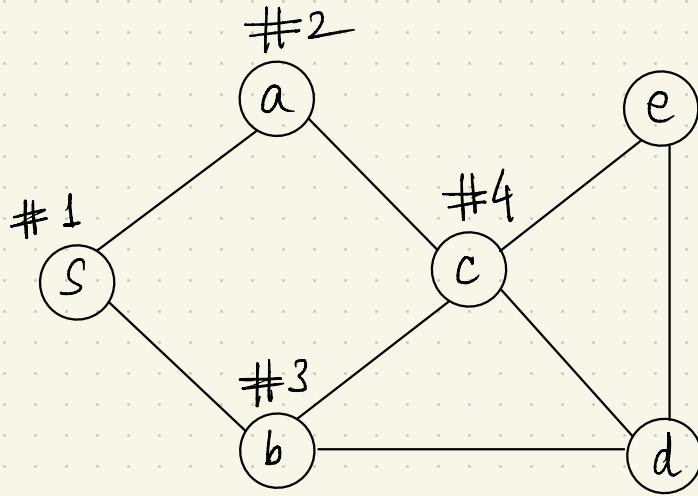
Q : ~~s~~ ~~a~~ b

BREADTH - FIRST SEARCH



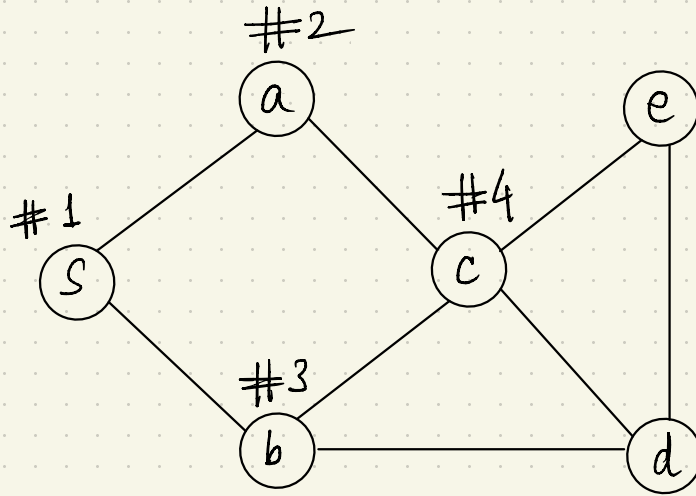
Q : ~~s~~ ~~a~~ b

BREADTH - FIRST SEARCH



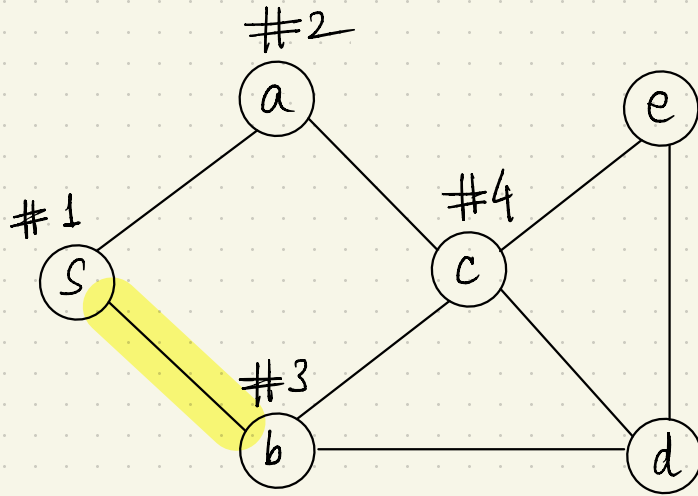
Q : ~~s~~ ~~a~~ b c

BREADTH - FIRST SEARCH



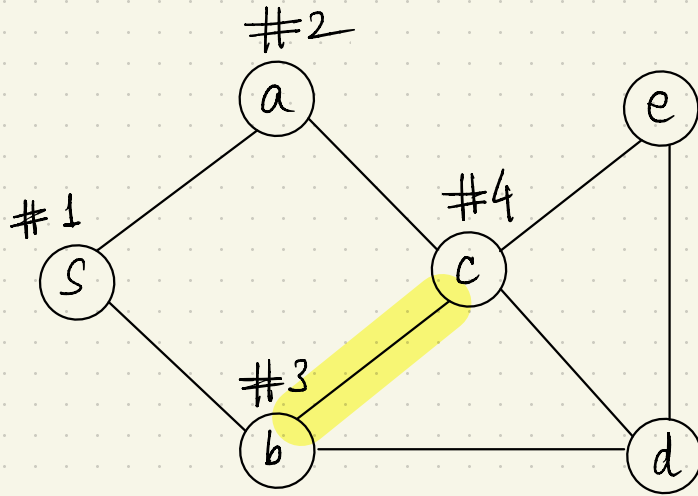
Q : ~~s~~ ~~a~~ ~~b~~ c

BREADTH - FIRST SEARCH



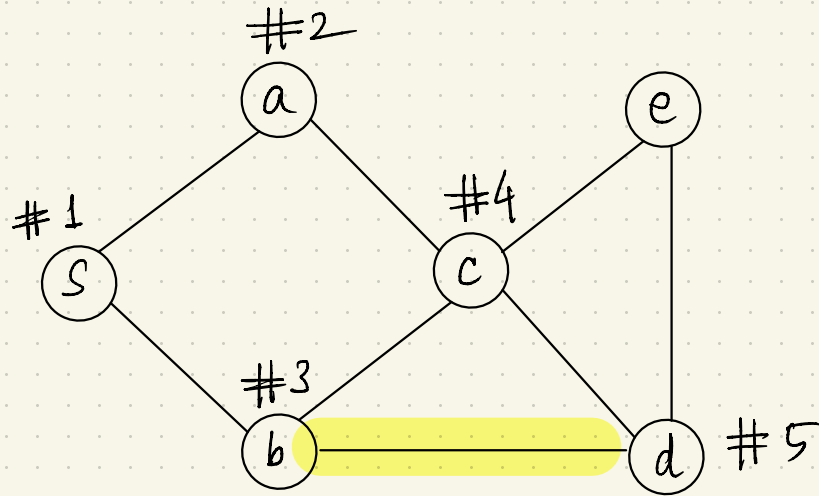
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~

BREADTH - FIRST SEARCH



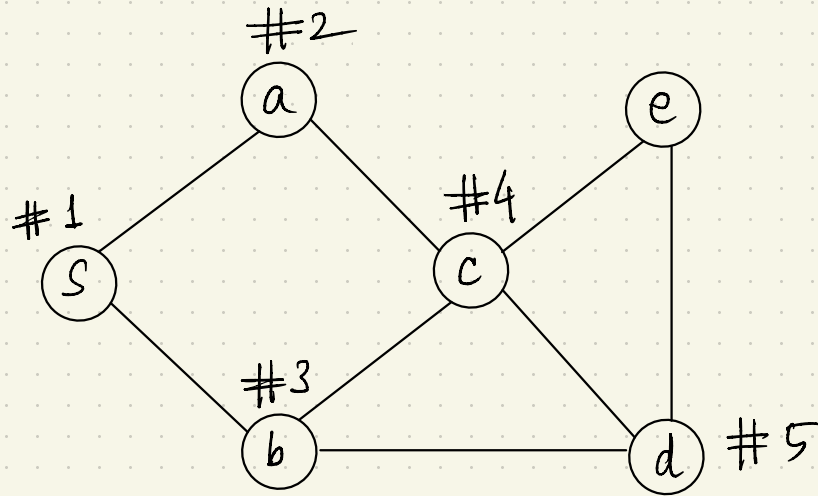
Q : ~~s~~ ~~a~~ ~~b~~ c

BREADTH - FIRST SEARCH



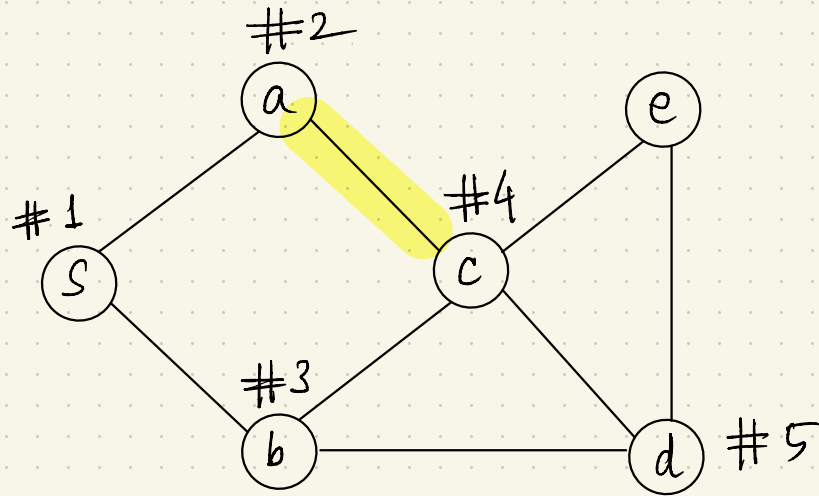
Q : ~~s~~ ~~a~~ ~~c~~ d

BREADTH - FIRST SEARCH



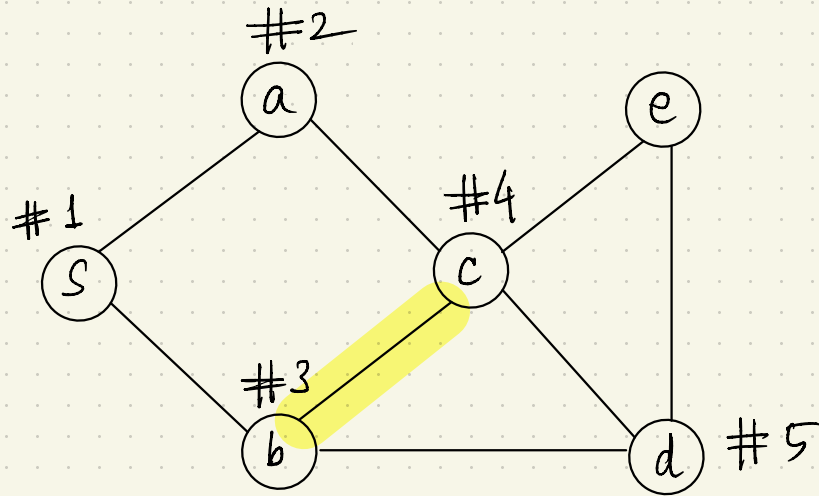
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



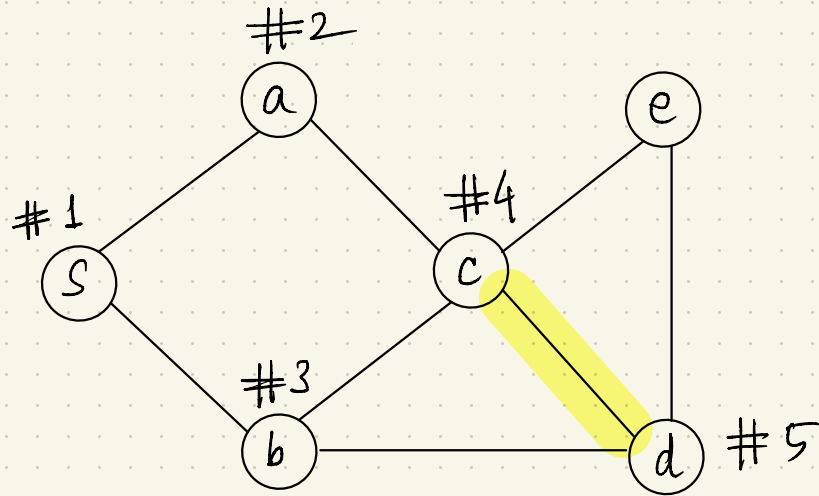
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



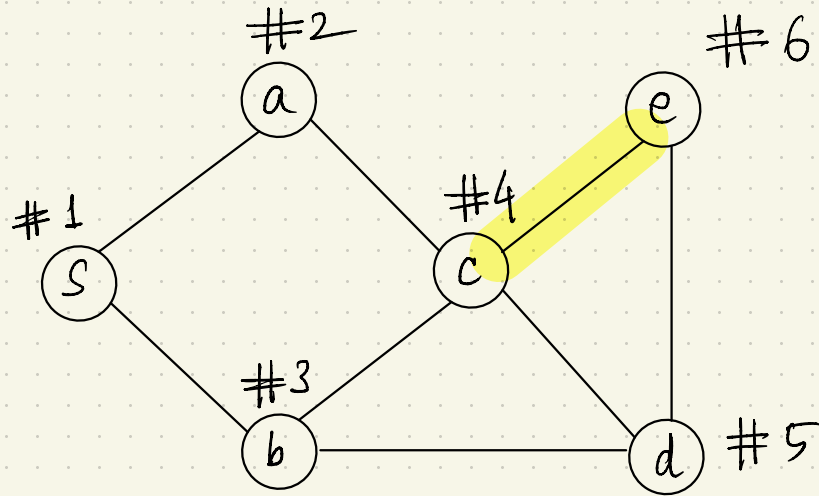
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



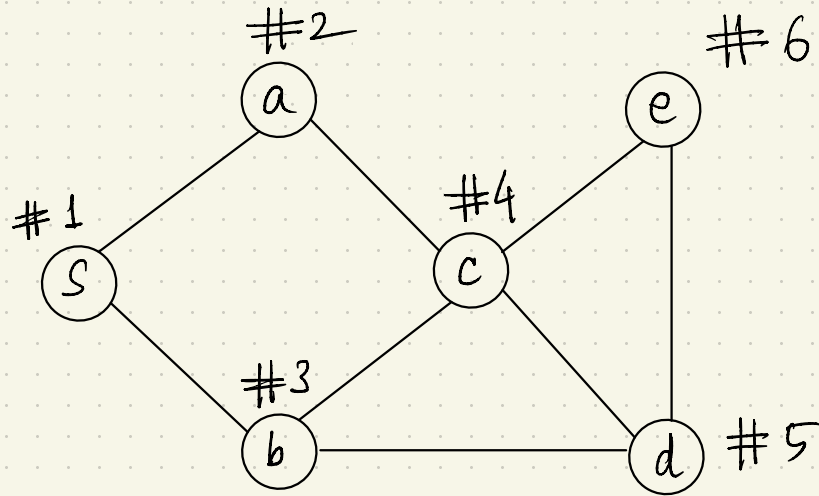
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



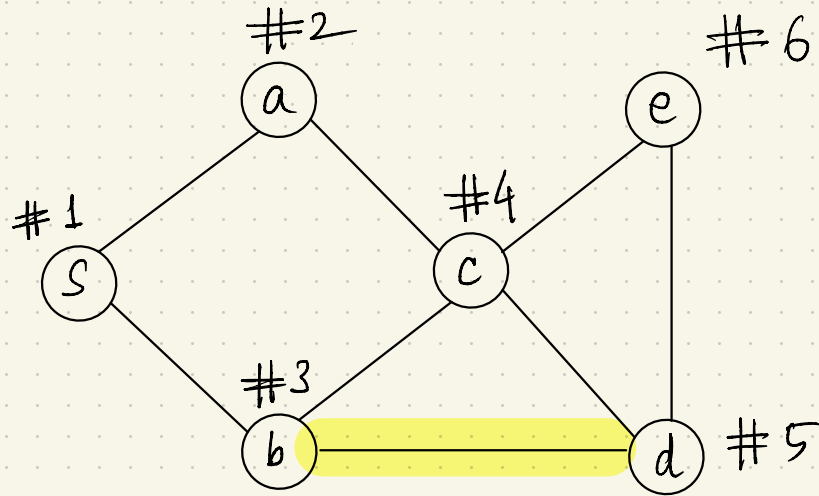
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d e

BREADTH - FIRST SEARCH



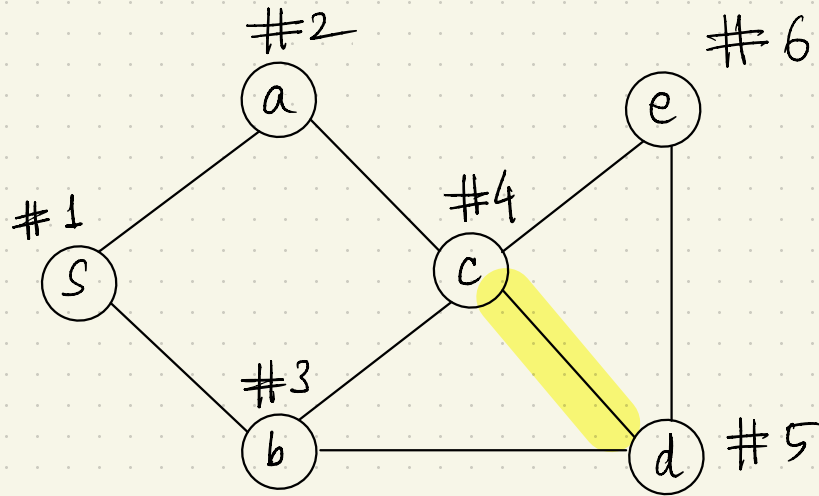
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



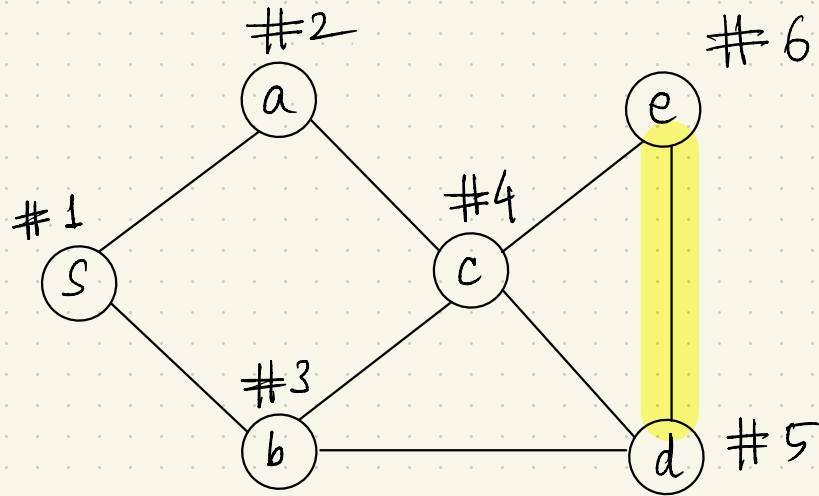
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



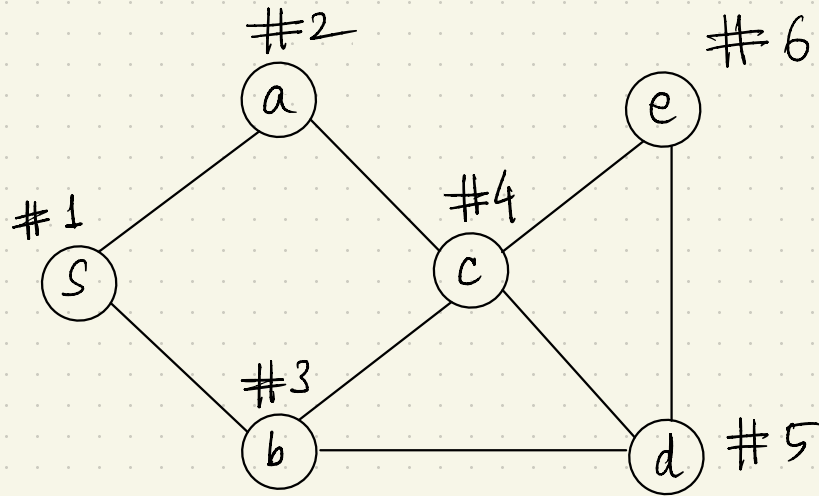
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



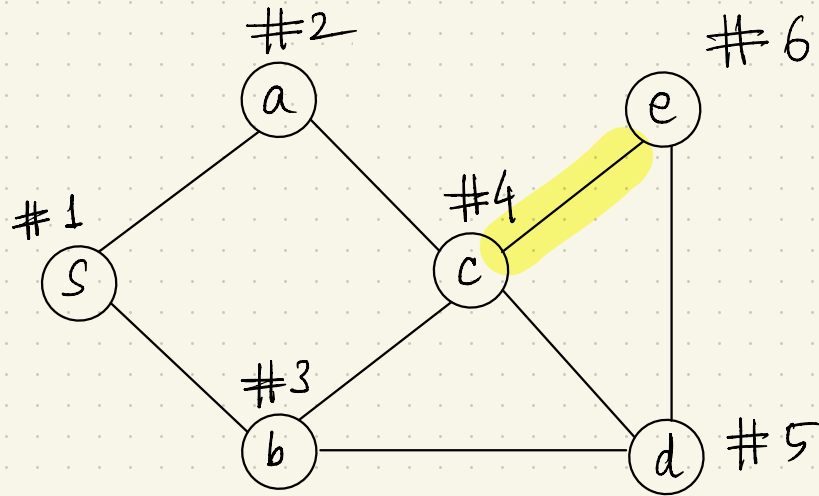
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



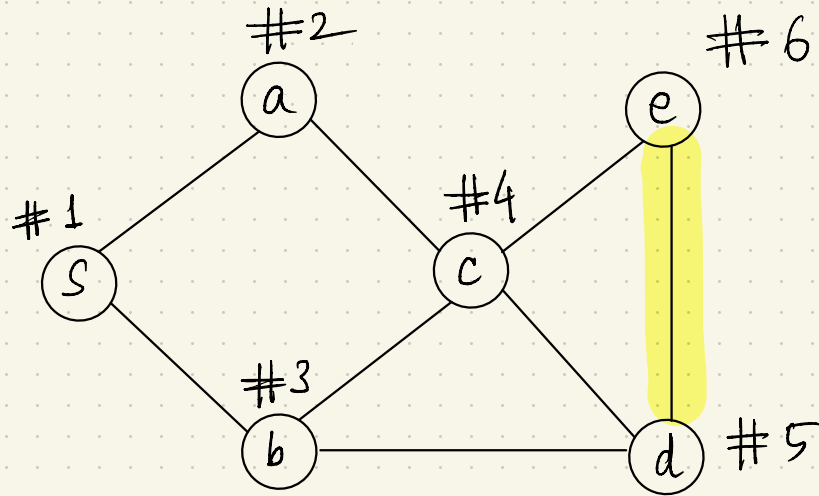
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~

BREADTH - FIRST SEARCH



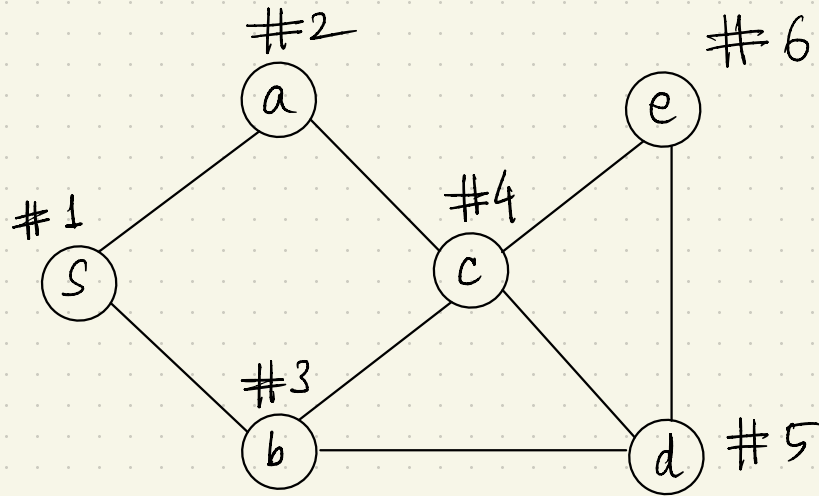
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~

BREADTH - FIRST SEARCH



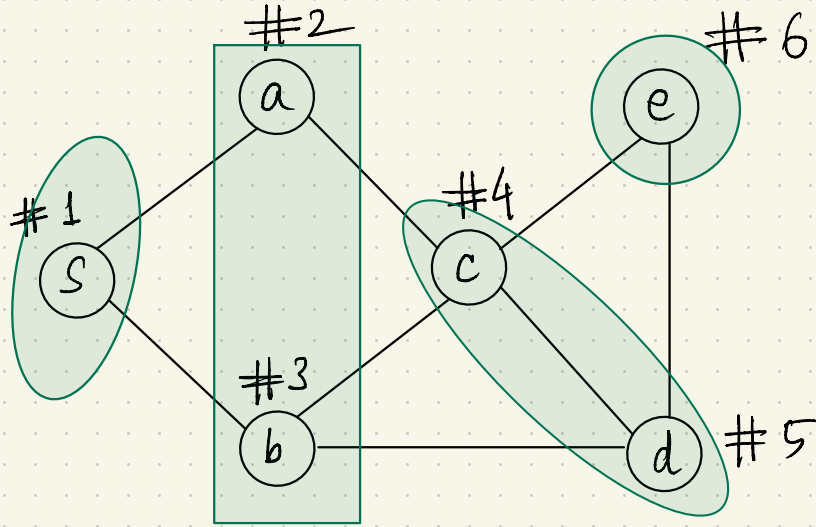
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~

BREADTH - FIRST SEARCH



Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~ stop!

BREADTH - FIRST SEARCH



Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~ stop!

BREADTH - FIRST SEARCH

Claim 1: At the end of BFS

v explored $\iff$ there is a path from s to v

BREADTH - FIRST SEARCH

Claim 1: At the end of BFS

v explored $\iff$ there is a path from s to v

Reason: BFS is a special case of Generic Algorithm.

BREADTH - FIRST SEARCH

Claim 1: At the end of BFS

v explored $\iff$ there is a path from s to v

Reason: BFS is a special case of Generic Algorithm.

Claim 2: Running time of while-loop of BFS is

$$O(n_s + m_s)$$

where $n_s = \#$ vertices reachable from s

$m_s = \#$ edges " " "

BREADTH - FIRST SEARCH

input: graph $G = (V, E)$, start vertex s

output: a vertex is reachable from s iff it is marked **explored**.

mark s as **explored**, all other vertices **unexplored**

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

 if w is **unexplored**

 mark w as **explored**

 add w to the end of Q

APPLICATIONS OF BFS

Shortest paths

Connected Components

APPLICATIONS OF BFS

Shortest paths

Connected Components

SHORTEST PATHS

input: an undirected/directed graph $G=(V,E)$, a starting vertex s

output: $\text{dist}(s,v)$ for every vertex $v \in V$

SHORTEST PATHS

input: an undirected/directed graph $G=(V,E)$, a starting vertex s

output: $\text{dist}(s,v)$ for every vertex $v \in V$

Number of edges

$\text{dist}(s,v) =$ length of shortest path from s to v if one exists,
 ∞ otherwise

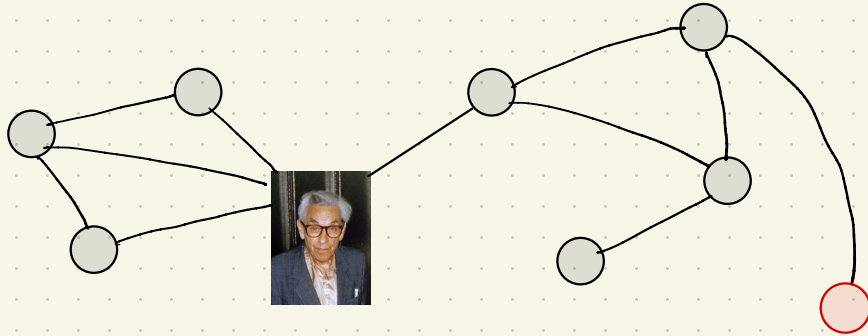
SHORTEST PATHS

input: an undirected/directed graph $G=(V,E)$, a starting vertex s

output: $\text{dist}(s,v)$ for every vertex $v \in V$

number of edges

$\text{dist}(s,v) =$ length of shortest path from s to v if one exists,
 ∞ otherwise



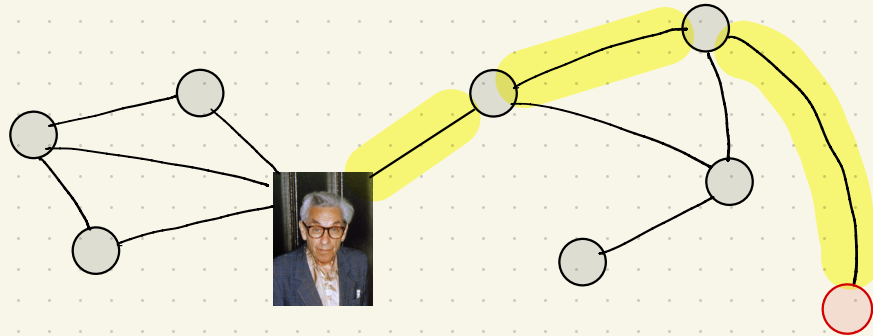
SHORTEST PATHS

input: an undirected/directed graph $G=(V,E)$, a starting vertex s

output: $\text{dist}(s,v)$ for every vertex $v \in V$

number of edges

$\text{dist}(s,v) =$ length of shortest path from s to v if one exists,
 ∞ otherwise



SHORTEST PATHS

BFS pseudo code 

mark s as explored, all other vertices unexplored

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

| remove the first node of Q , say v
| for each $(v, w) \in E$
| | if w is unexplored
| | | mark w as explored
| | | add w to the end of Q

SHORTEST PATHS

mark s as explored, all other vertices unexplored

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

 if w is unexplored

 mark w as explored

 add w to the end of Q

SHORTEST PATHS

mark s as **explored**, all other vertices **unexplored**

$\text{dist}(s) := 0$, $\text{dist}(v) = \infty$ for all $v \neq s$.

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

 if w is **unexplored**

 mark w as **explored**

 add w to the end of Q

SHORTEST PATHS

mark s as explored, all other vertices unexplored

$\text{dist}(s) := 0$, $\text{dist}(v) = \infty$ for all $v \neq s$.

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

 if w is unexplored

 mark w as explored

 add w to the end of Q

SHORTEST PATHS

mark s as explored, all other vertices unexplored

$\text{dist}(s) := 0$, $\text{dist}(v) = \infty$ for all $v \neq s$.

$Q :=$ a queue data structure (FIFO), initialized with s

while $Q \neq \emptyset$

 remove the first node of Q , say v

 for each $(v, w) \in E$

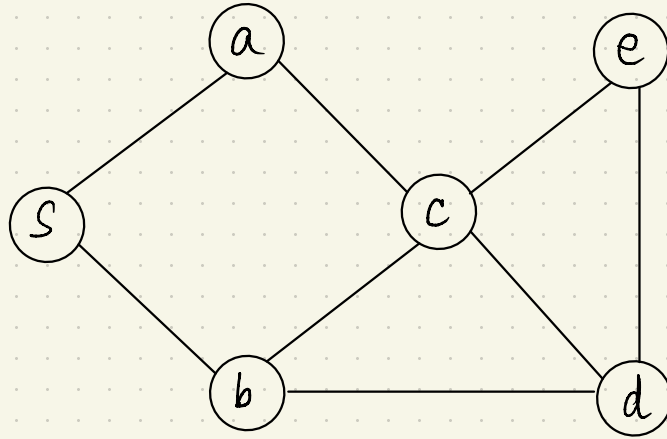
 if w is unexplored

 mark w as explored

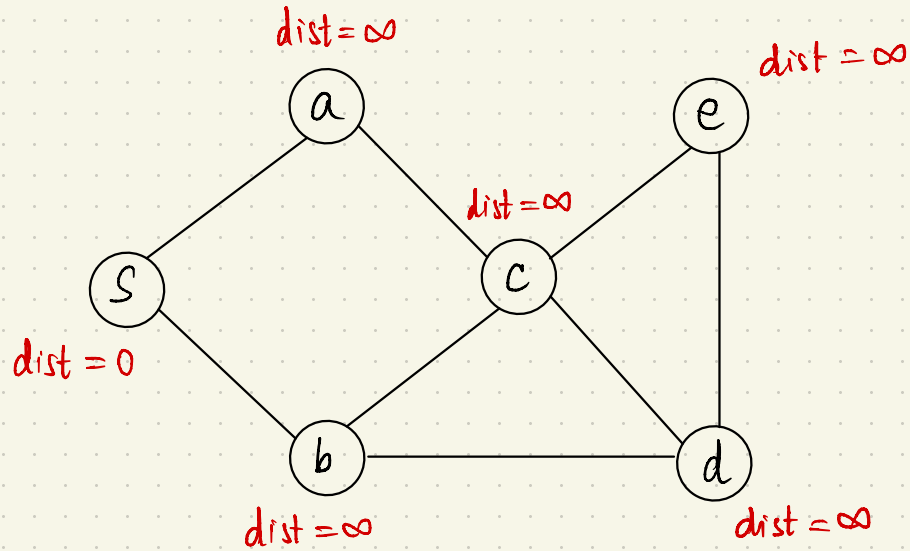
$\text{dist}(w) := \text{dist}(v) + 1$

 add w to the end of Q

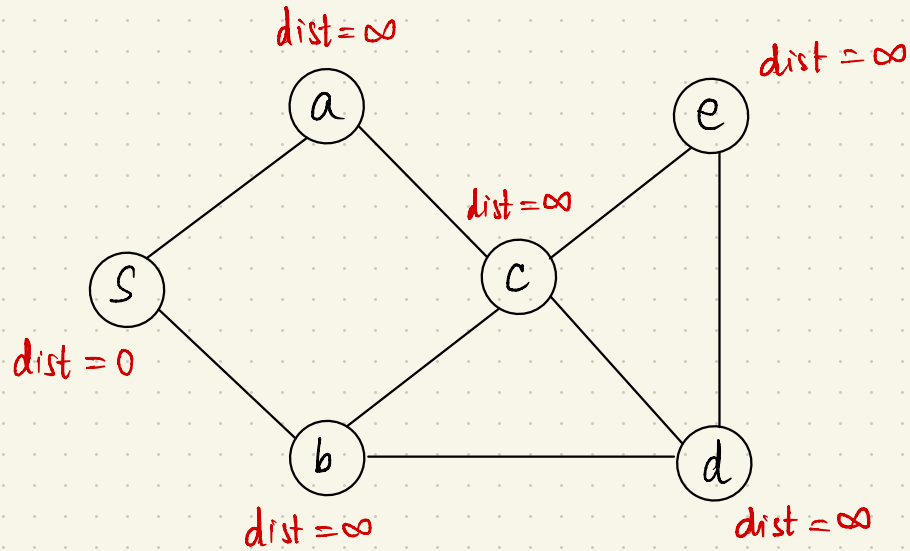
SHORTEST PATHS



SHORTEST PATHS

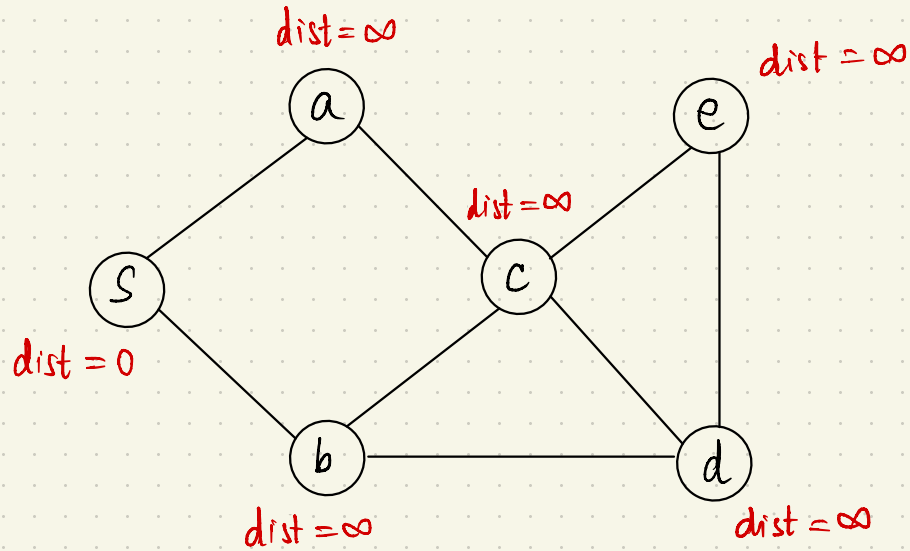


SHORTEST PATHS



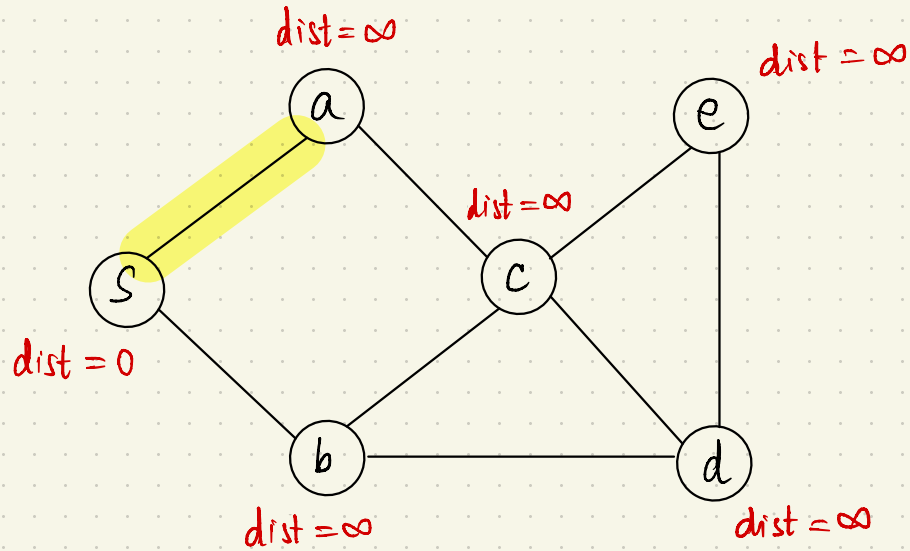
$$Q = s$$

SHORTEST PATHS



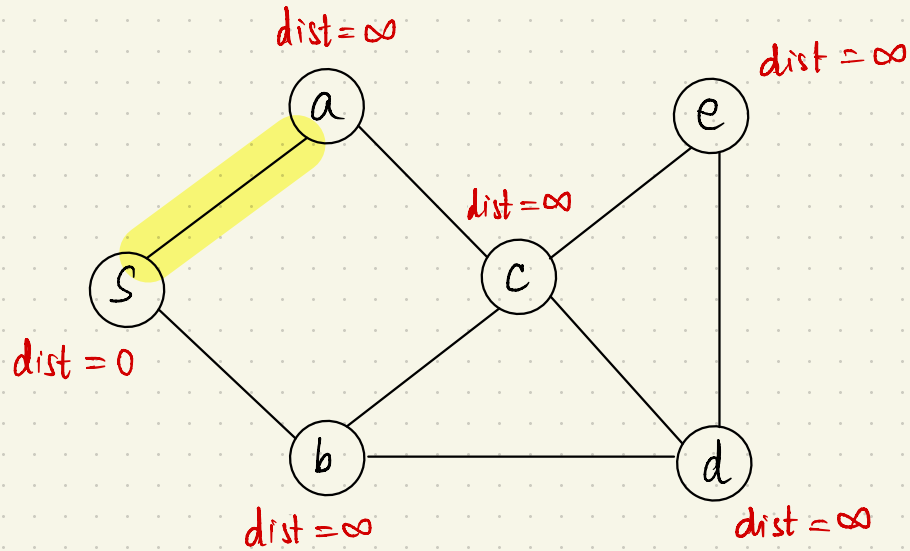
$Q = \cancel{s}$

SHORTEST PATHS



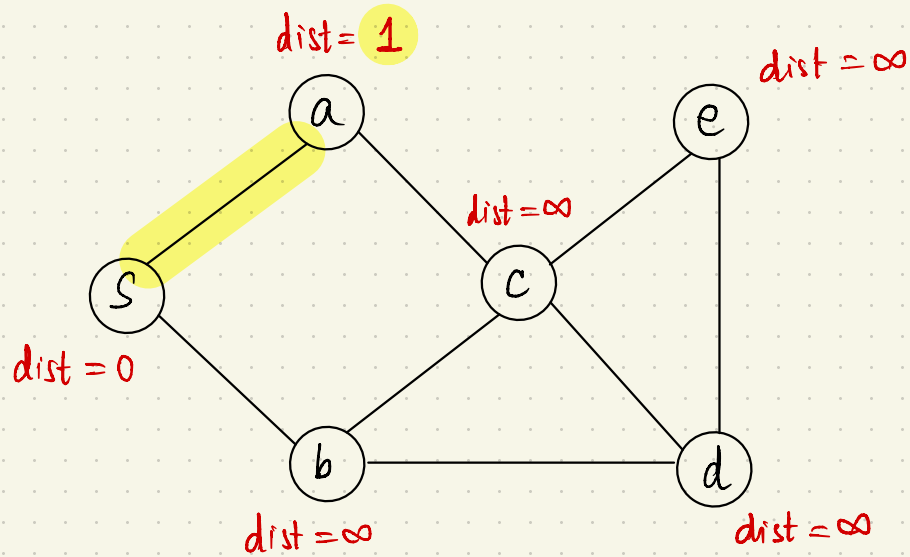
$$Q = \emptyset$$

SHORTEST PATHS



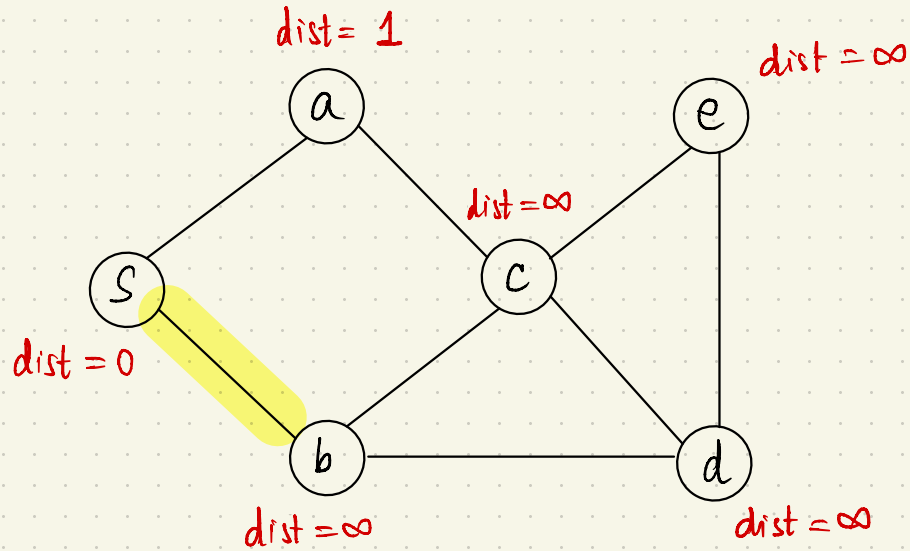
$$Q = \cancel{s} \text{ } a$$

SHORTEST PATHS



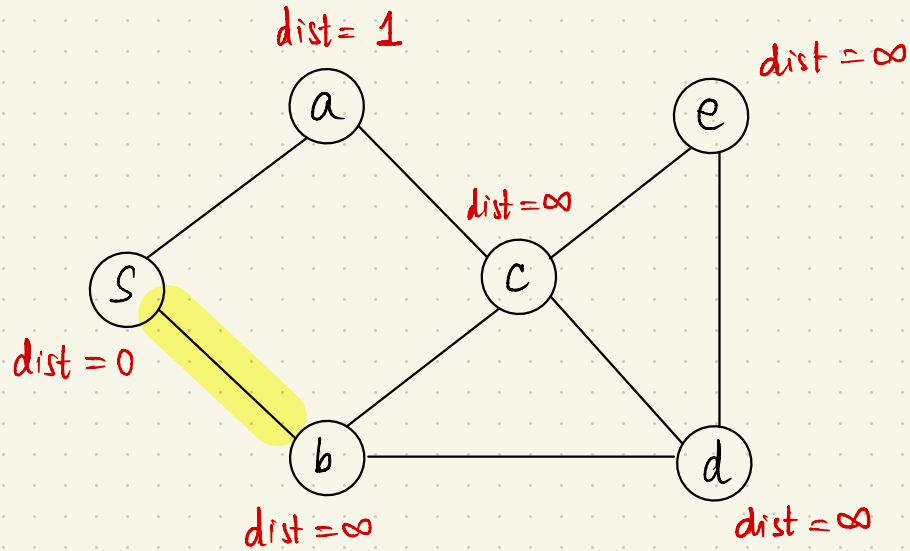
$$Q = \cancel{s} a$$

SHORTEST PATHS



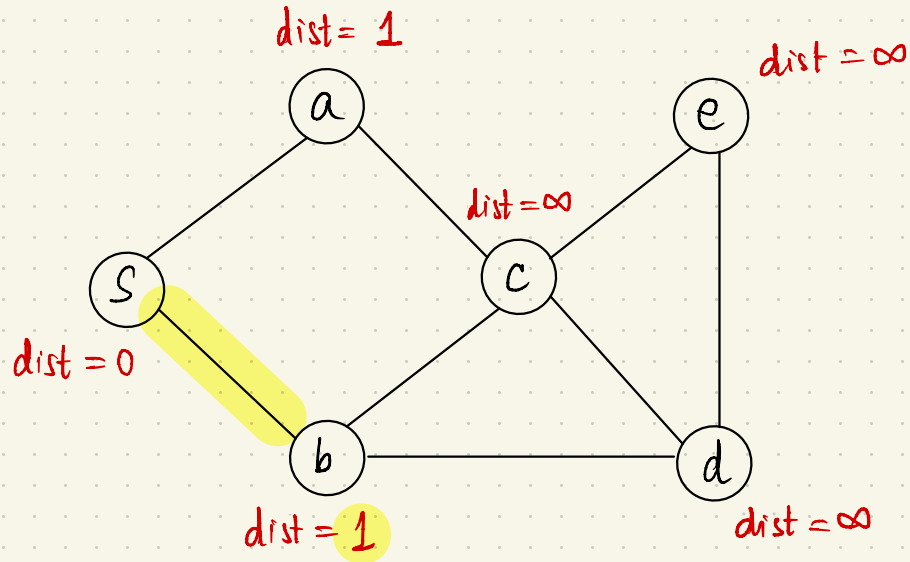
$$Q = \cancel{s} a$$

SHORTEST PATHS



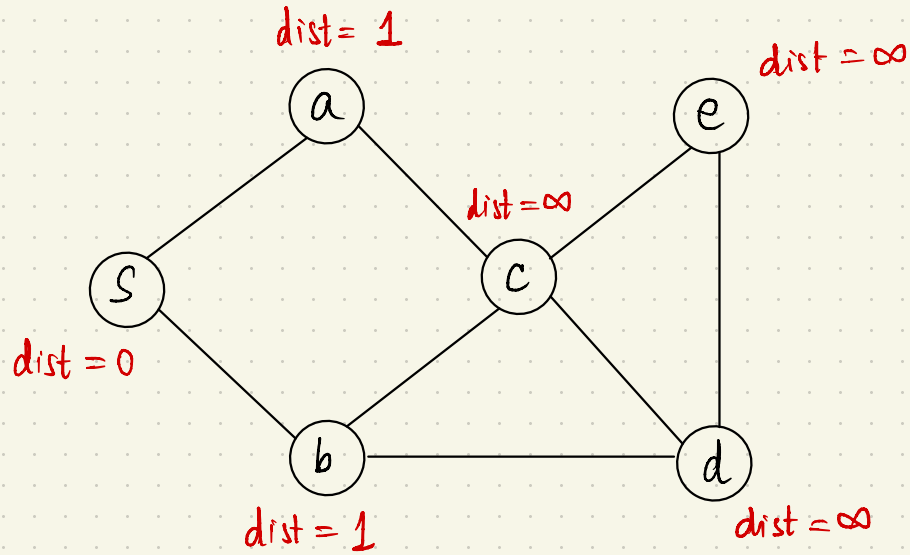
$$Q = \cancel{s} a b$$

SHORTEST PATHS



$Q = \cancel{s} a b$

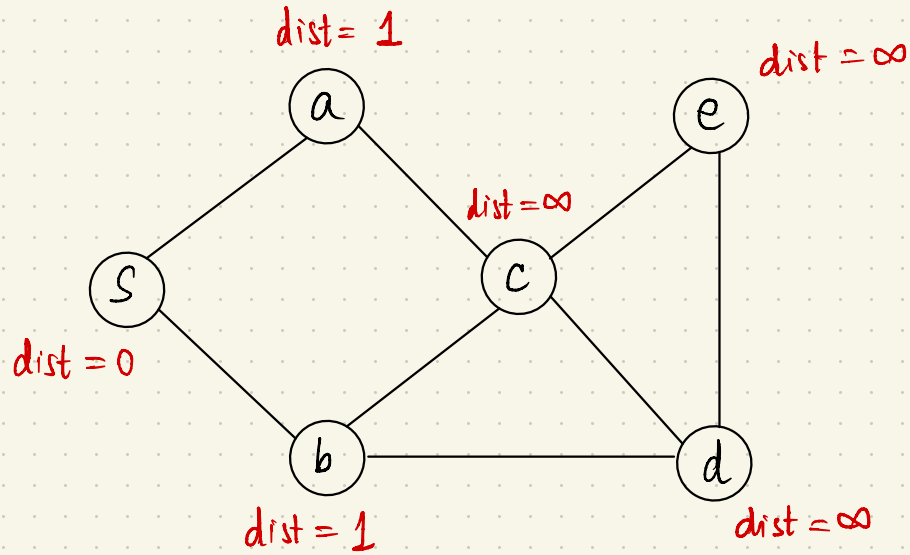
SHORTEST PATHS



$Q = \cancel{s} a b$

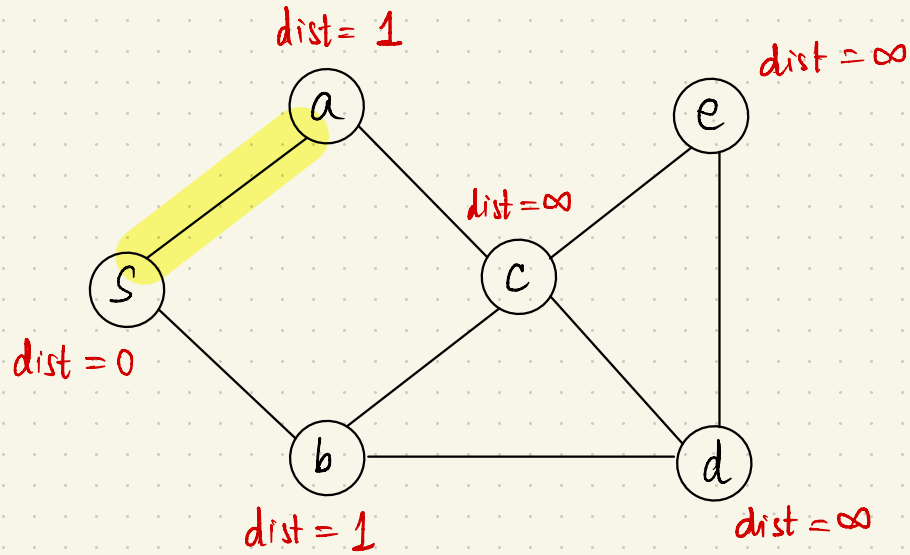
fully explored

SHORTEST PATHS



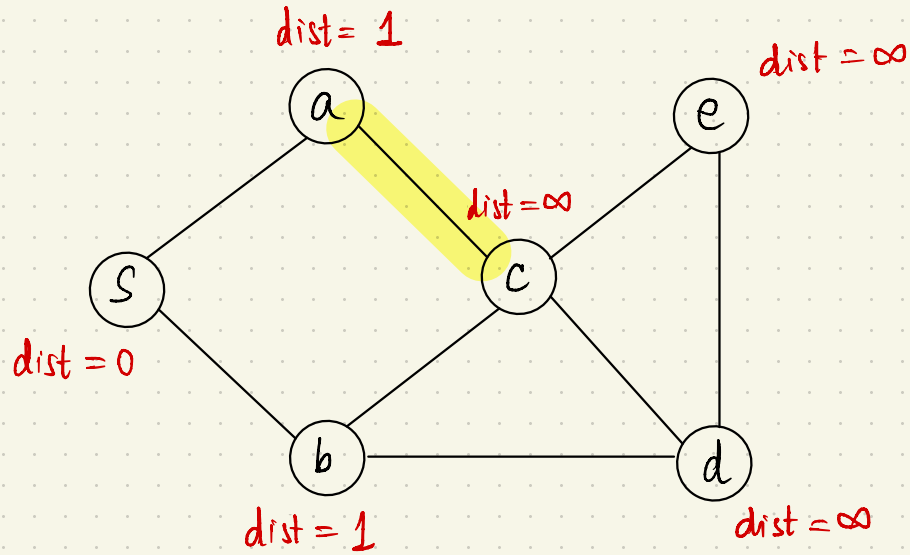
$$Q = \cancel{s} \cancel{a} \cancel{b}$$

SHORTEST PATHS



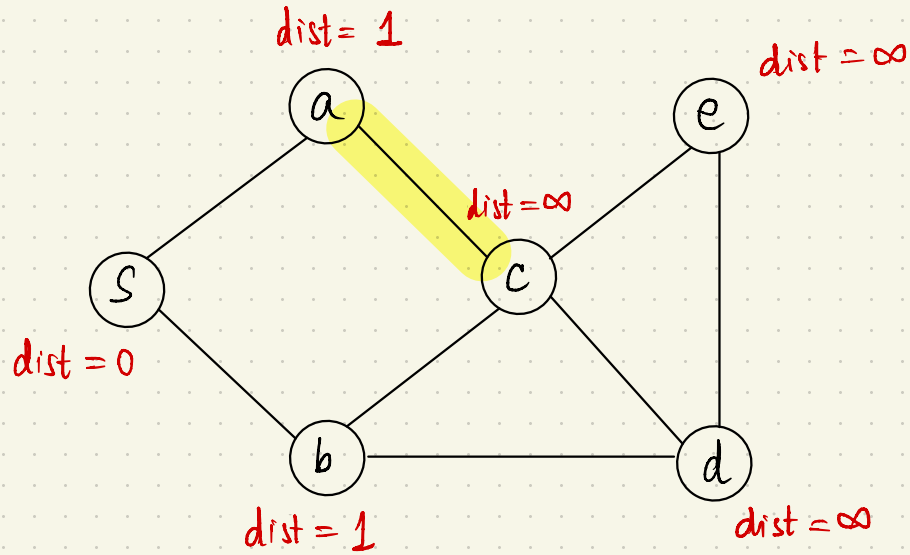
$$Q = \cancel{s} \cancel{a} b$$

SHORTEST PATHS



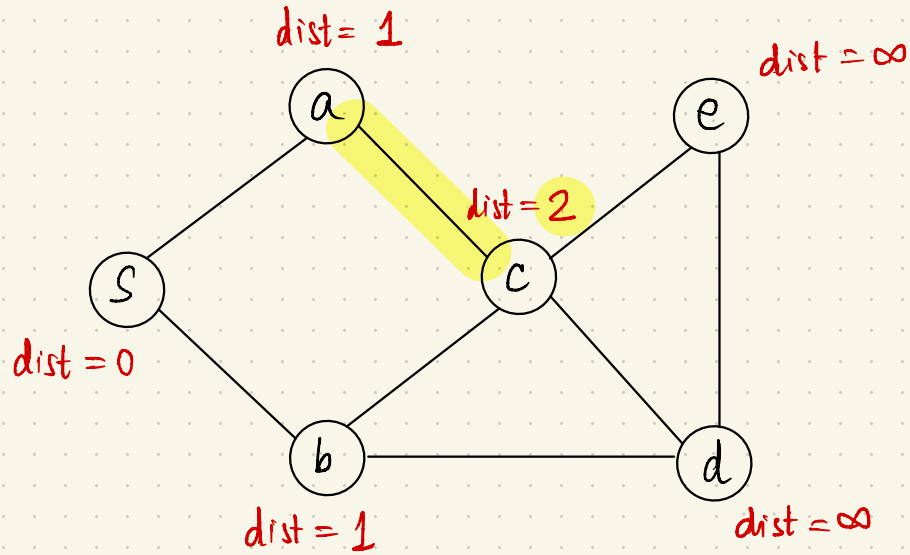
$$Q = \cancel{s} \cancel{a} b$$

SHORTEST PATHS



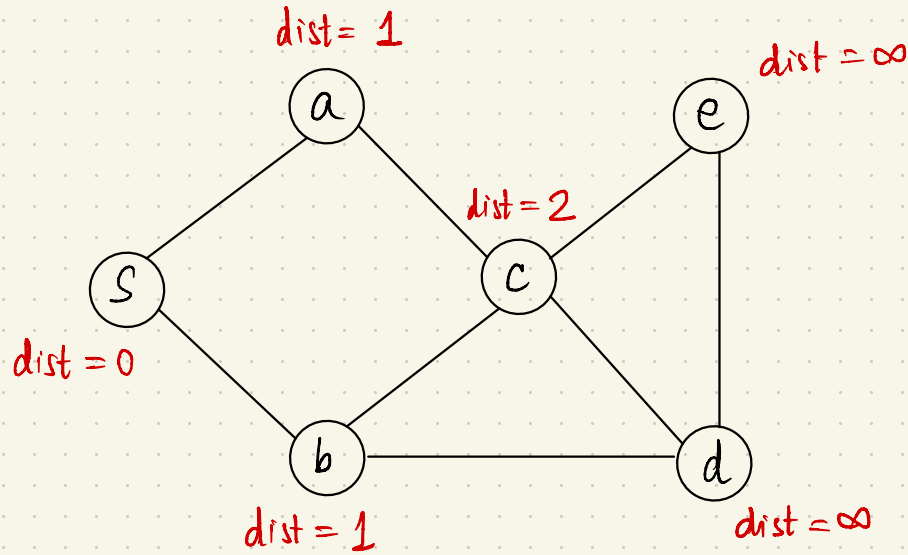
$Q = \cancel{s} \cancel{a} \cancel{b} c$

SHORTEST PATHS



$Q = \cancel{s} \cancel{a} b c$

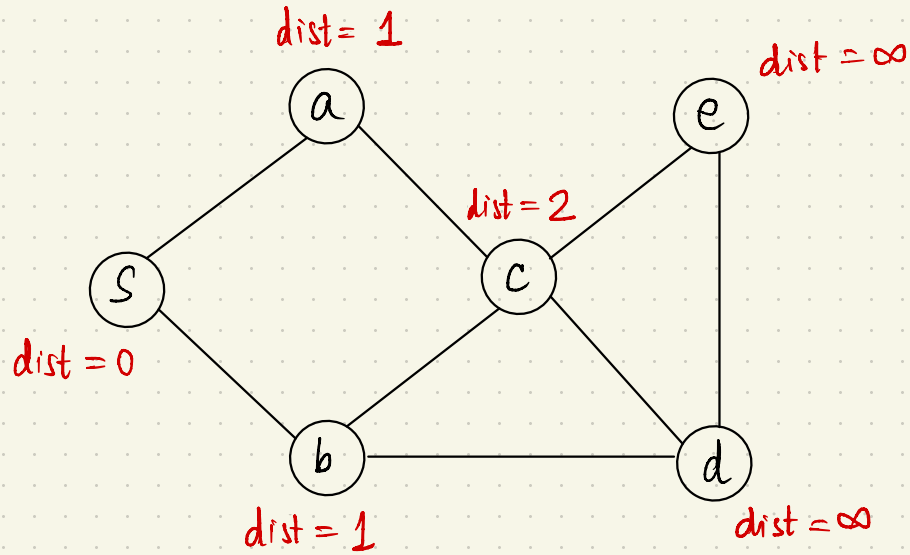
SHORTEST PATHS



$Q = \cancel{s} \cancel{a} b c$

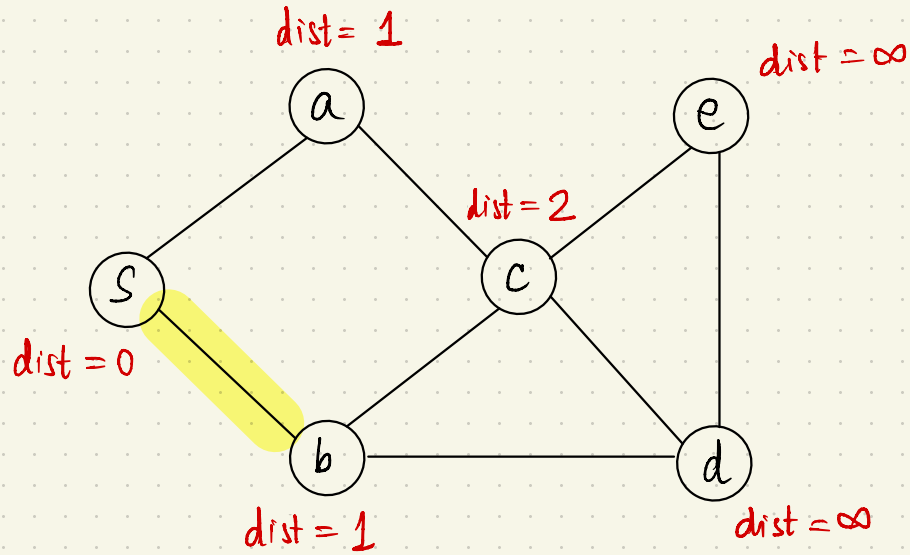
fully explored

SHORTEST PATHS



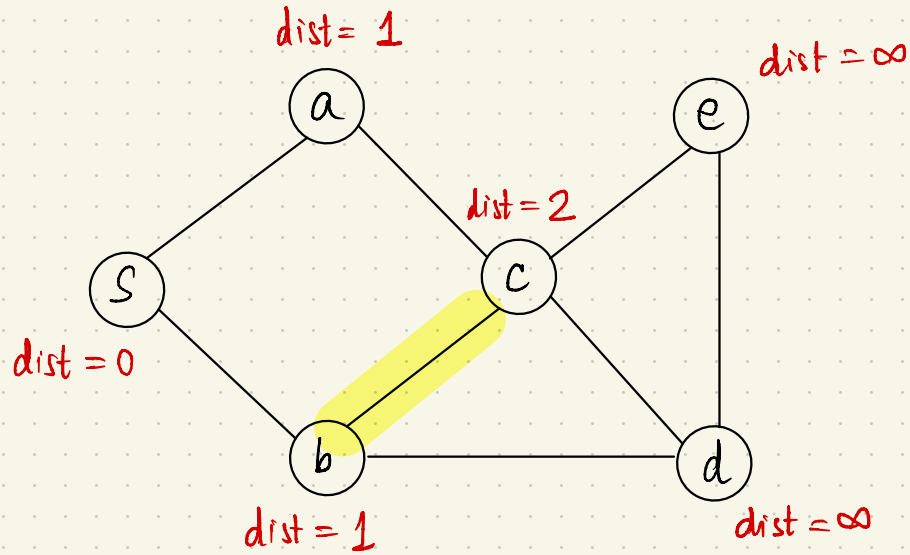
$Q = \cancel{s} \cancel{a} \cancel{b} c$

SHORTEST PATHS



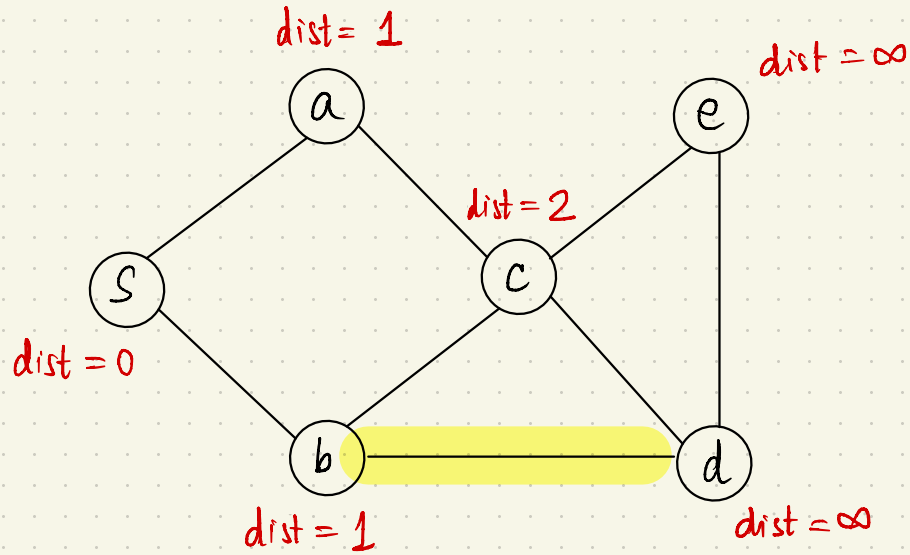
$$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c}$$

SHORTEST PATHS



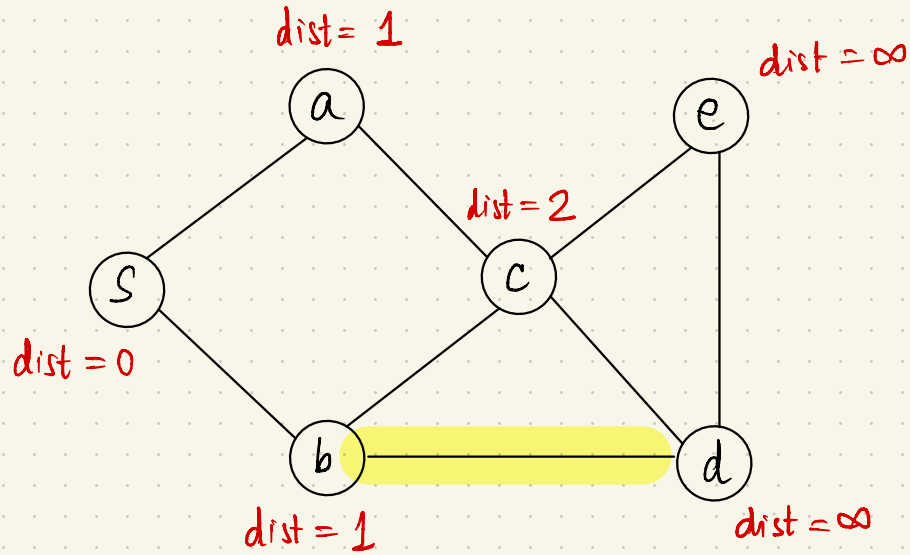
$$Q = \cancel{s} \cancel{a} \cancel{b} c$$

SHORTEST PATHS



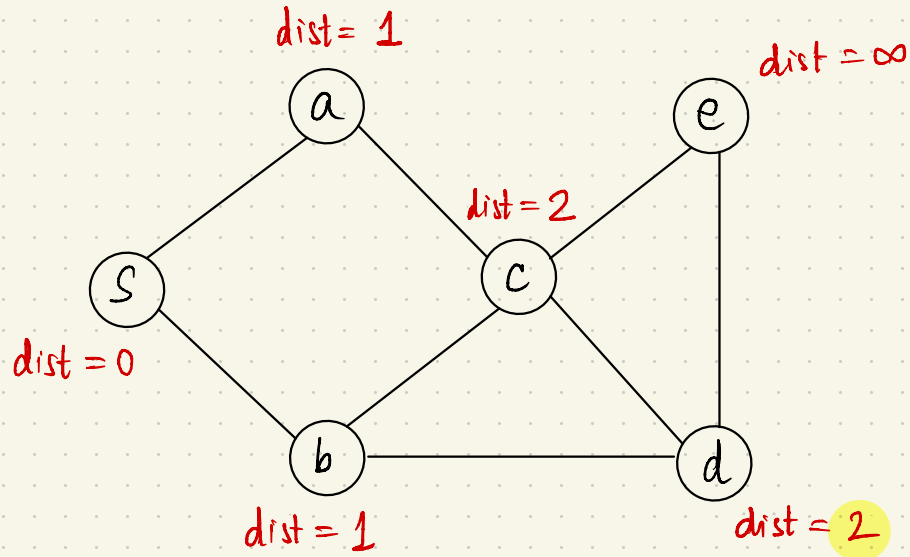
$$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c}$$

SHORTEST PATHS



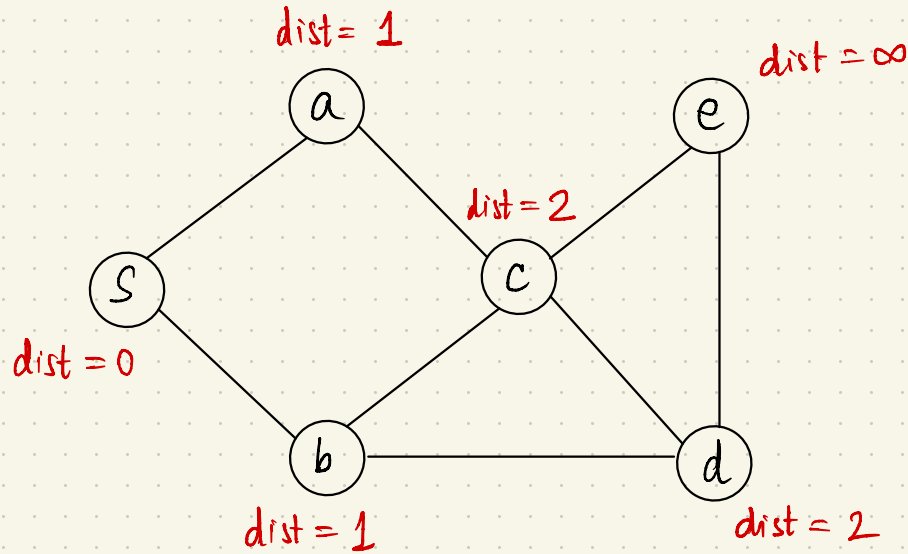
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} d$

SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} c d$

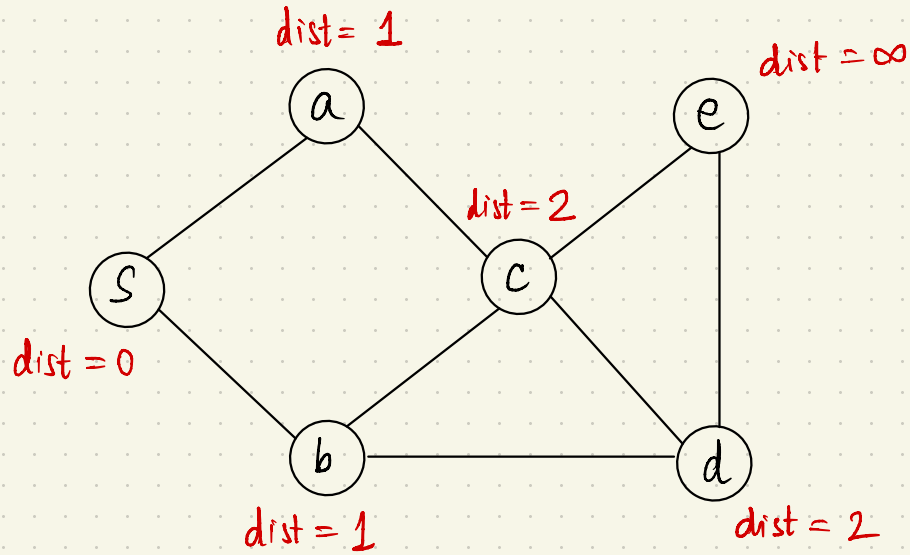
SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} c d$

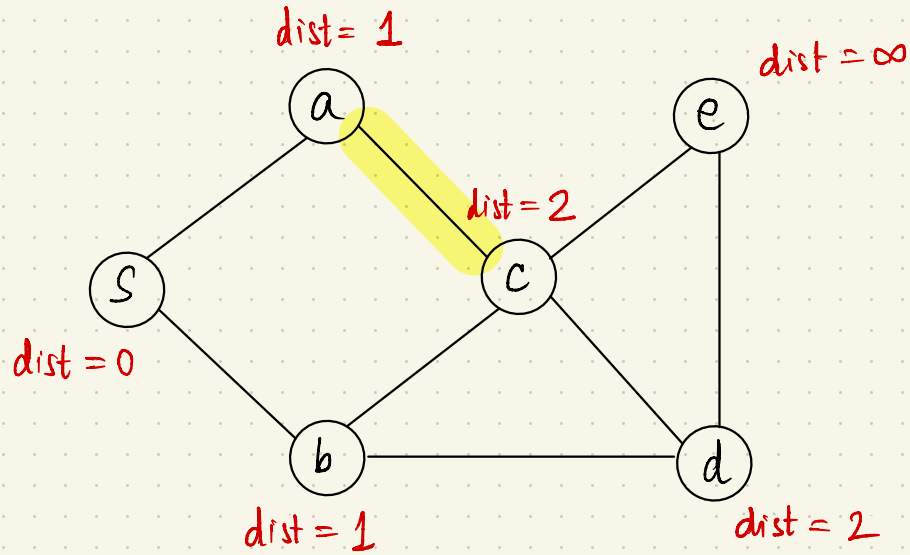
— fully explored

SHORTEST PATHS



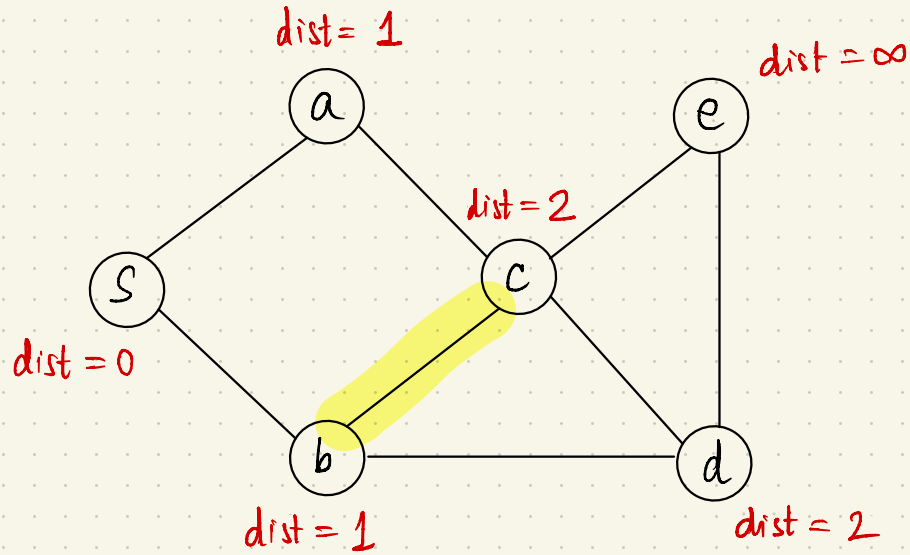
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} d$

SHORTEST PATHS



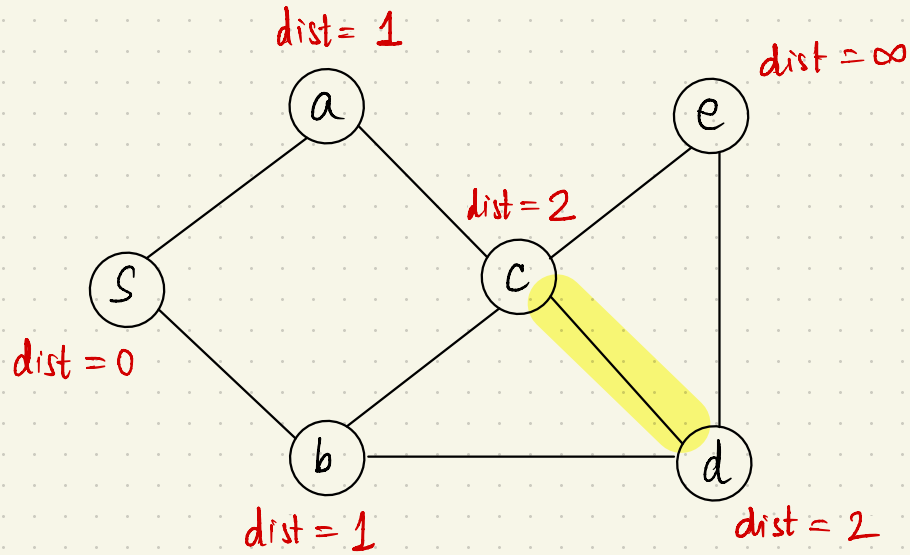
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d}$

SHORTEST PATHS



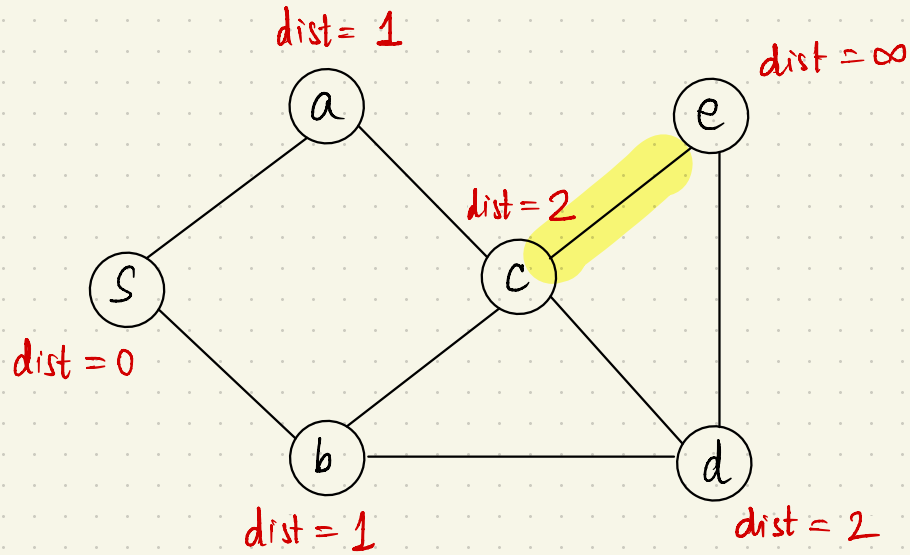
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d}$

SHORTEST PATHS



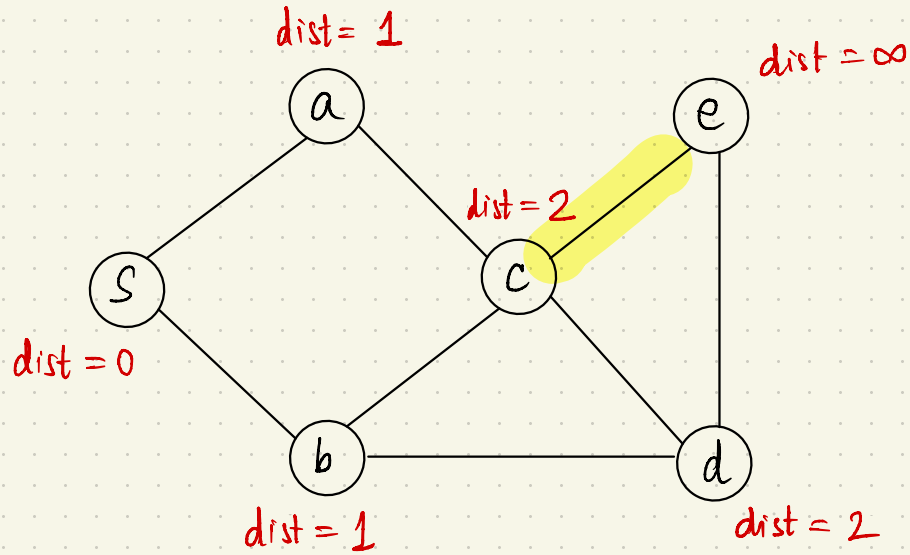
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d}$

SHORTEST PATHS



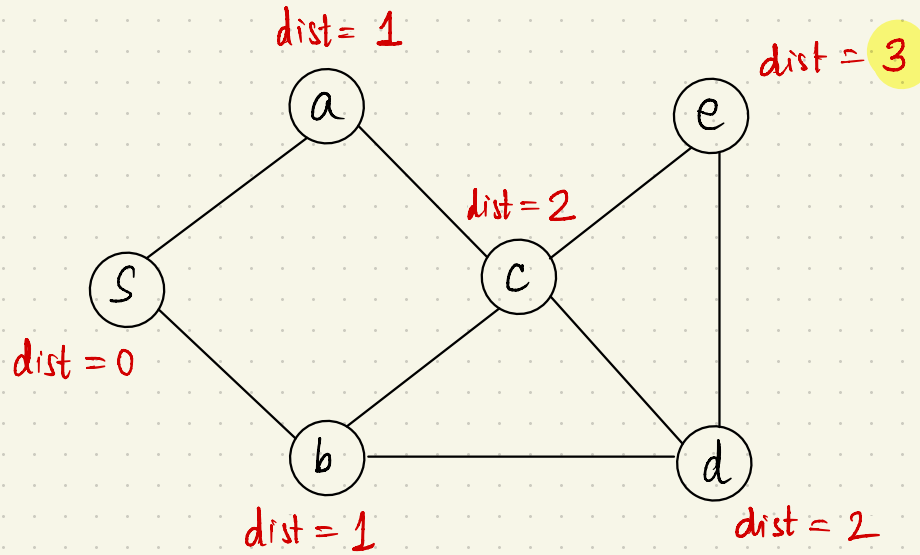
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d}$

SHORTEST PATHS



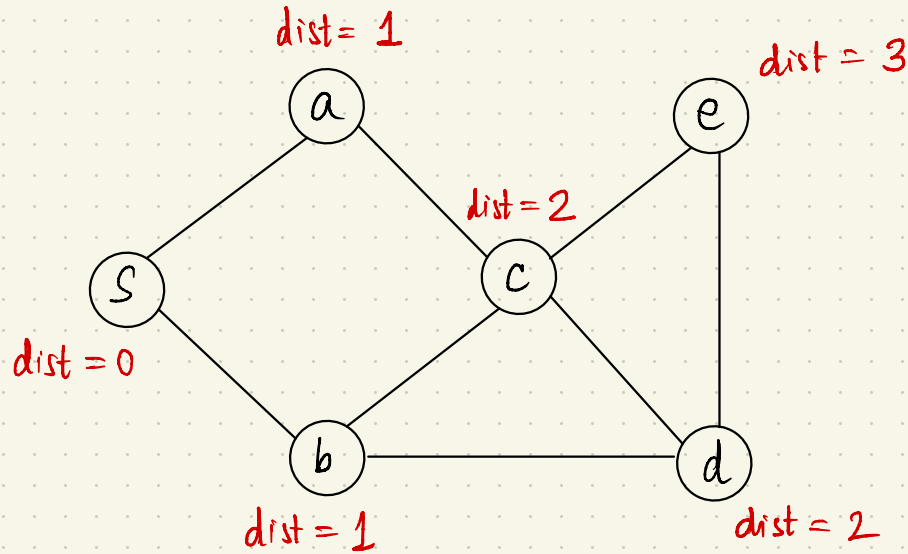
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} e$

SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} d e$

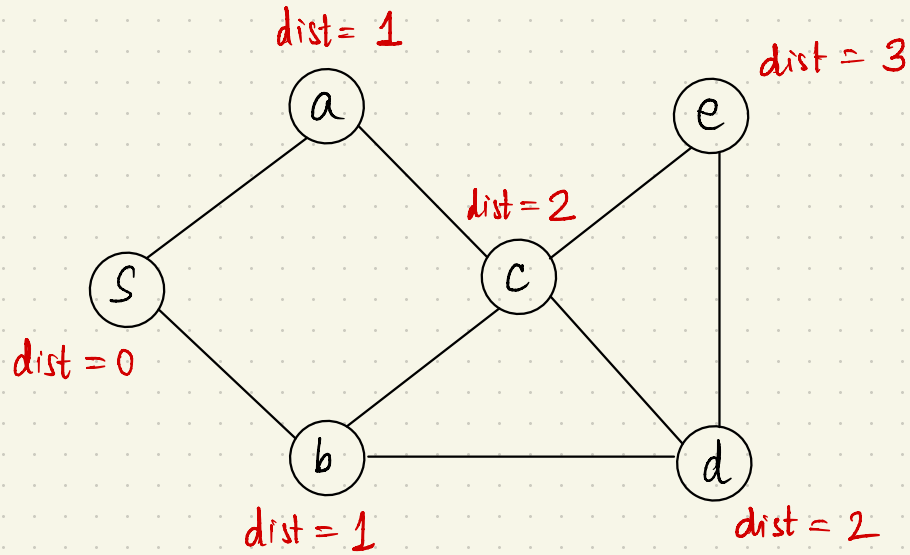
SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} d e$

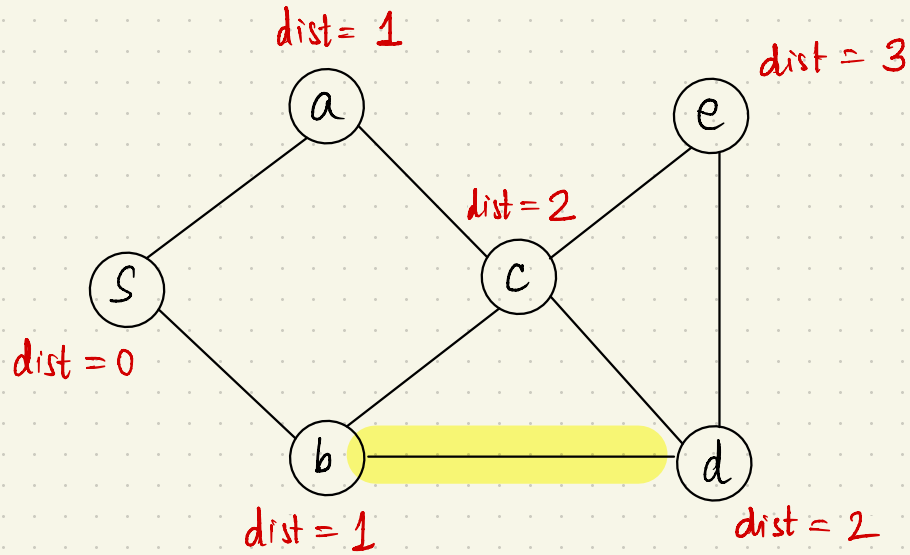
fully explored

SHORTEST PATHS



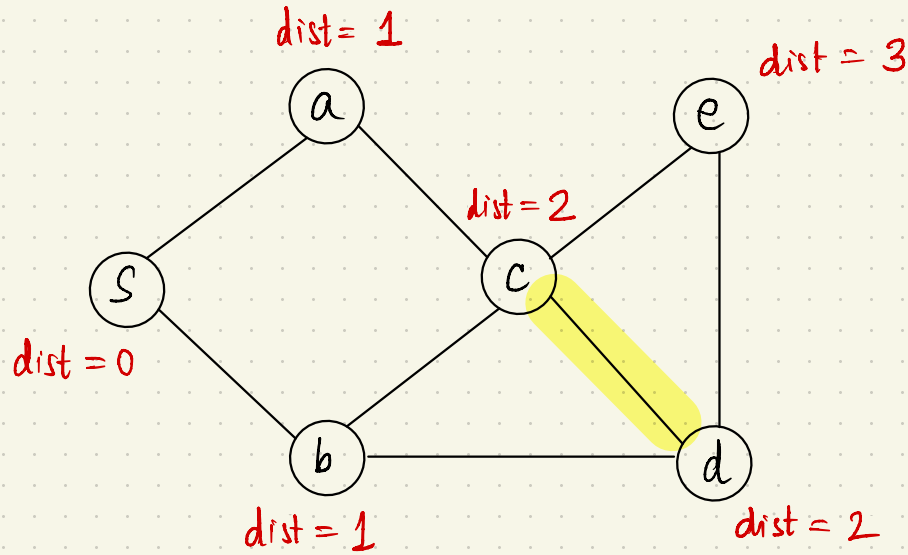
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} e$

SHORTEST PATHS



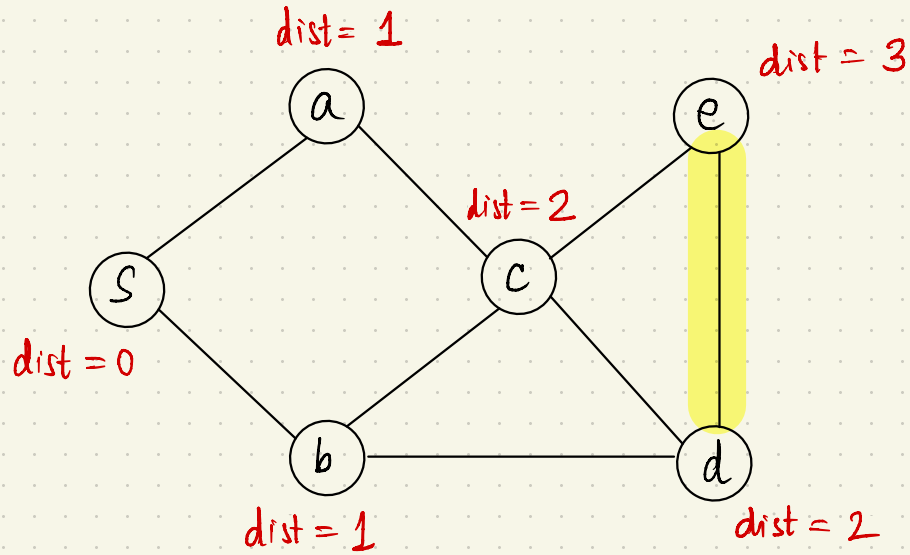
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} e$

SHORTEST PATHS



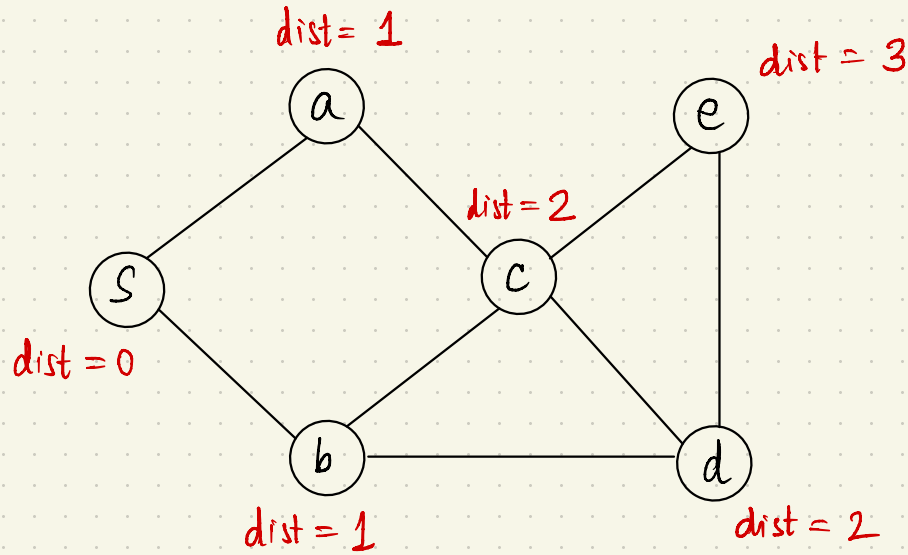
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} \cancel{e}$

SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} e$

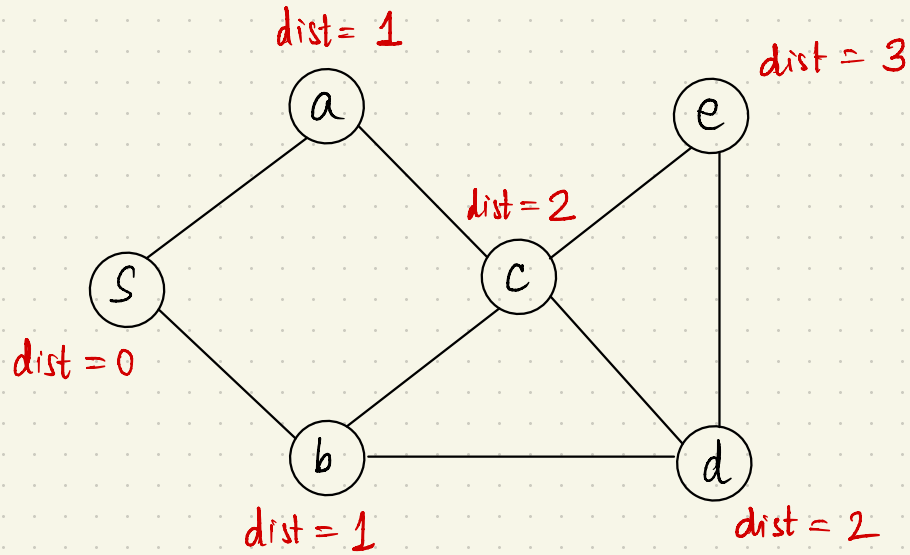
SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} \cancel{e}$

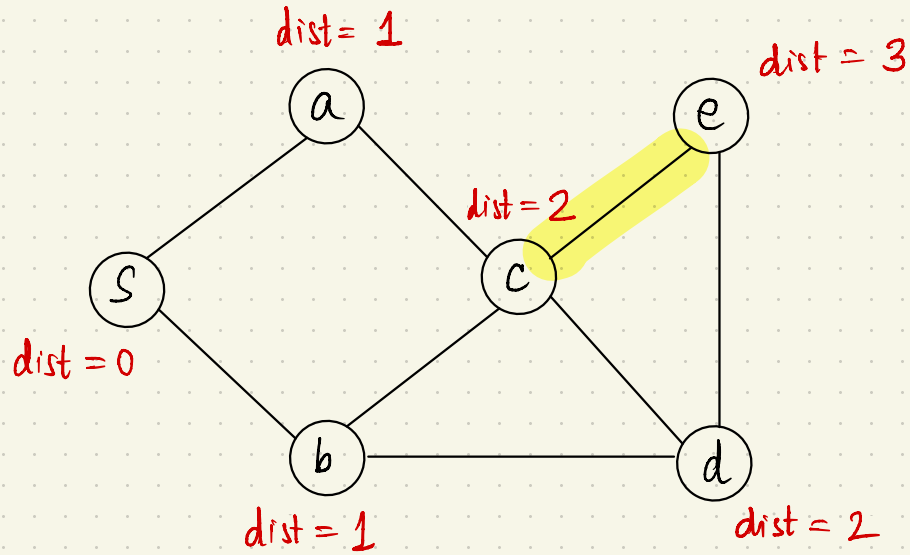
fully explored

SHORTEST PATHS



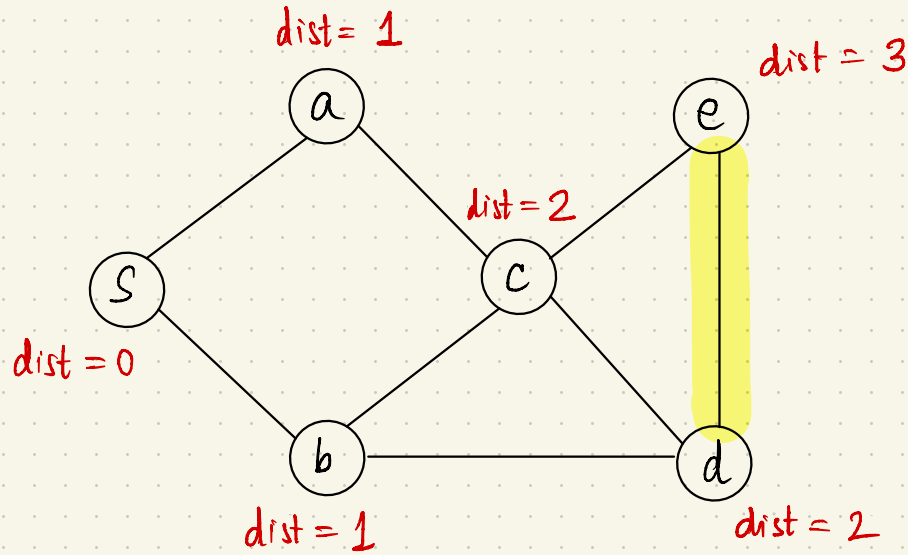
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} \cancel{e}$

SHORTEST PATHS



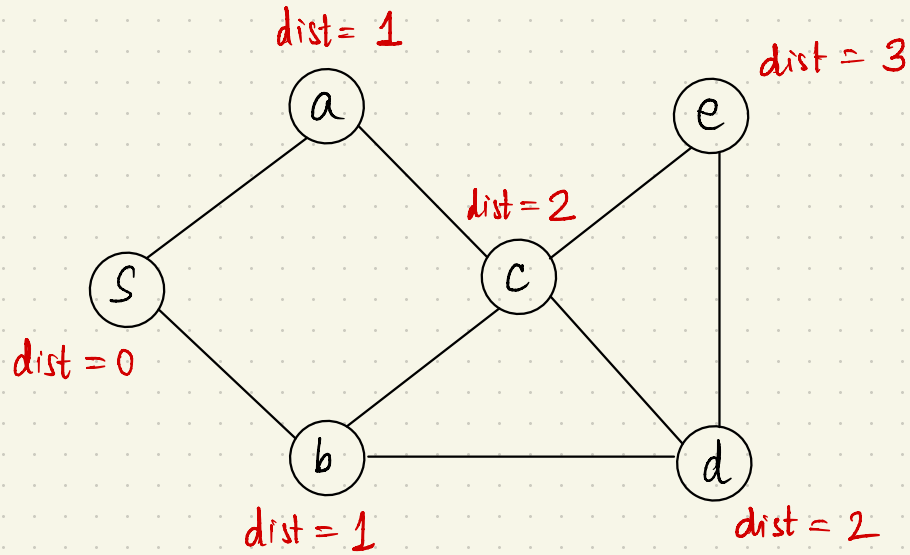
$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} \cancel{e}$

SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} \cancel{e}$

SHORTEST PATHS



$Q = \cancel{s} \cancel{a} \cancel{b} \cancel{c} \cancel{d} \cancel{e}$ done!