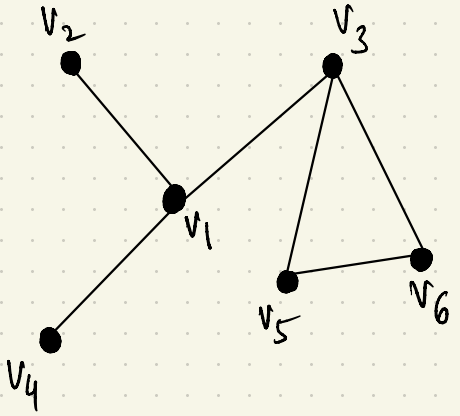
COL 351 : Analysis & Design of Algorithms

Lecture 9

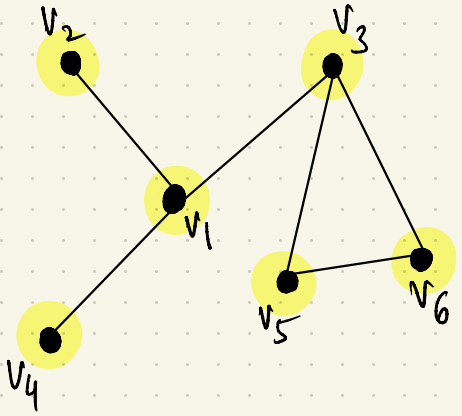
Graph Algorithms II : Representation and search

Aug 09, 2024 | Rohit Vaish

GRAPHS

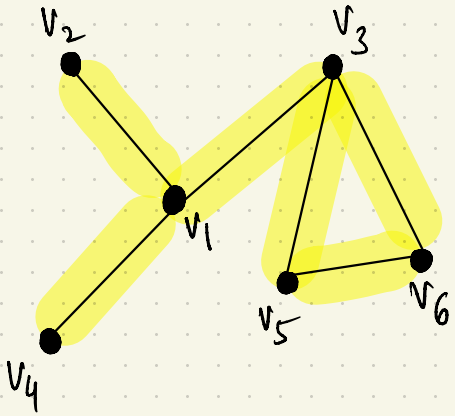


GRAPHS



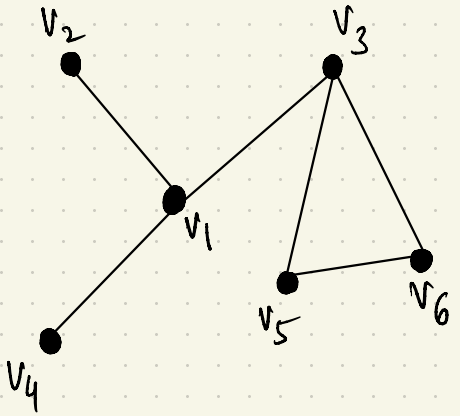
Vertices (or nodes)

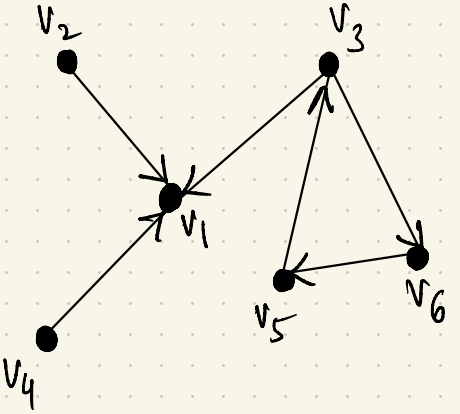
GRAPHS



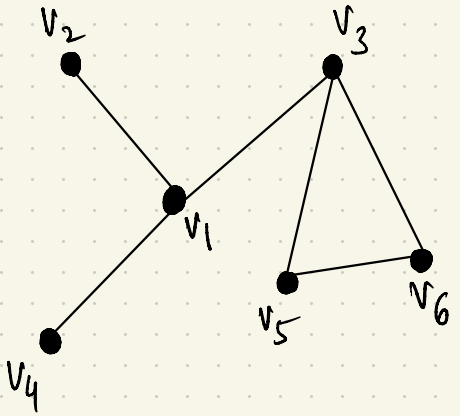
Edges

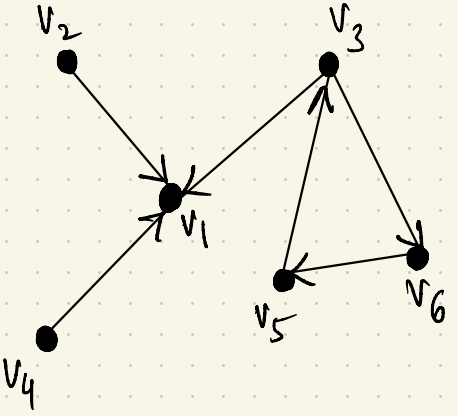
GRAPHS





GRAPHS

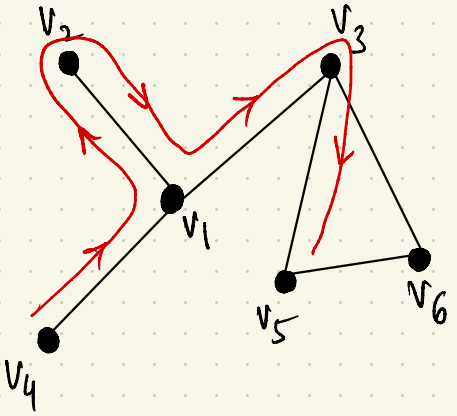


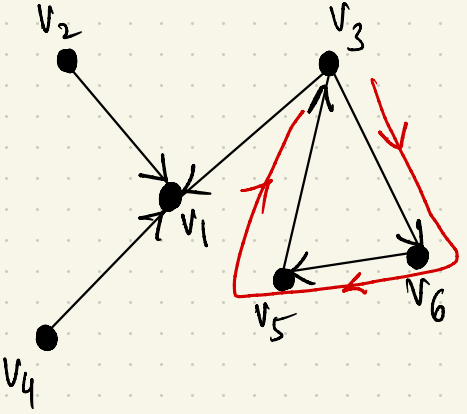


$(v_2, v_1) \in E$

$(v_1, v_2) \notin E$

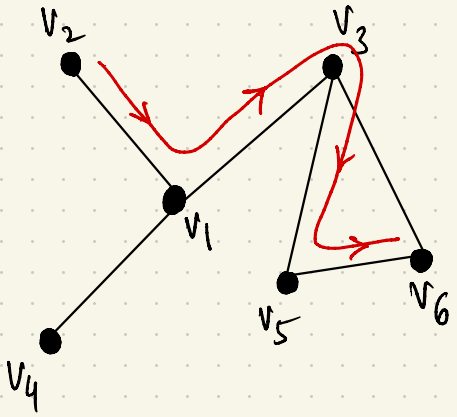
GRAPHS

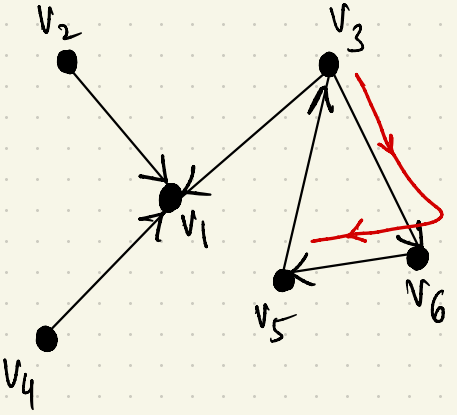




Walks

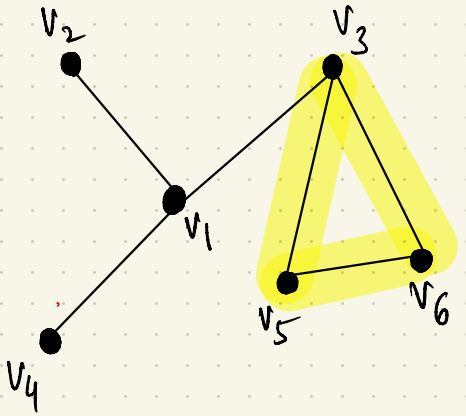
GRAPHS

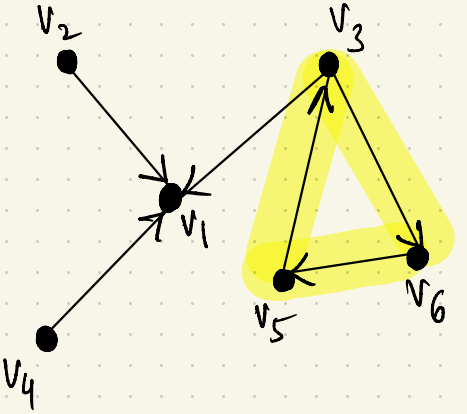




Paths

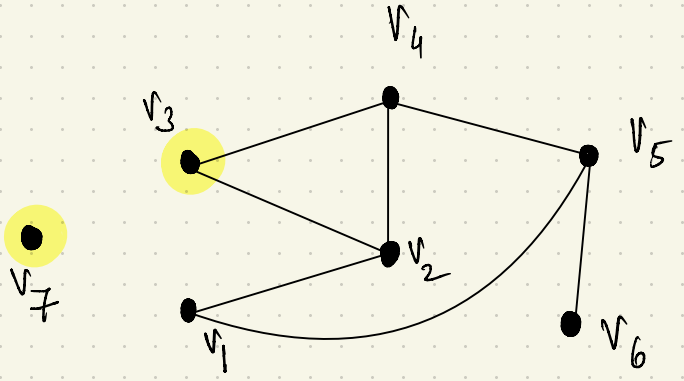
GRAPHS



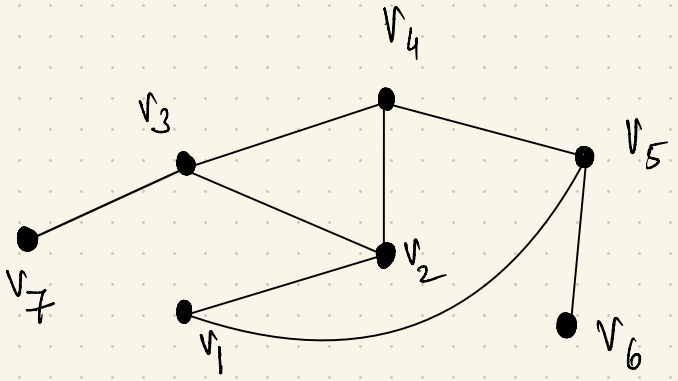


Closed walks and cycles

CONNECTIVITY



Not connected



Connected

SPARSE vs DENSE GRAPHS

SPARSE vs DENSE GRAPHS

n : number of vertices

m : number of edges

SPARSE vs DENSE GRAPHS

n : number of vertices $\qquad$ m : number of edges

Many applications : m is $\Omega(n)$ and $O(n^2)$.
(but not always)

SPARSE vs DENSE GRAPHS

n : number of vertices $\qquad$ m : number of edges

Many applications : m is $\Omega(n)$ and $O(n^2)$.
(but not always)

roughly,

sparse : $m = O(n \log n)$ close to linear

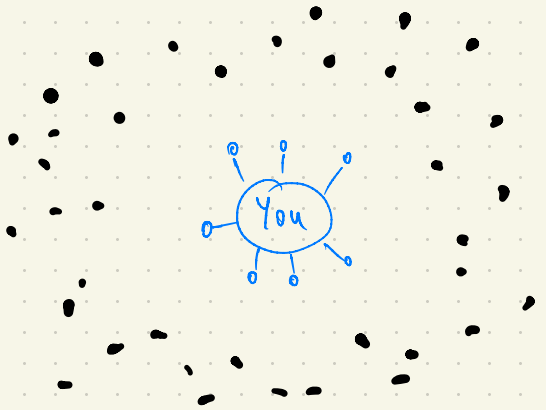
dense : $m = \Omega\left(\frac{n^2}{\log n}\right)$ close to quadratic

SPARSE vs DENSE GRAPHS

n : number of vertices $\qquad$ m : number of edges

Many applications : m is $\Omega(n)$ and $O(n^2)$.
(but not always)

Facebook graph is sparse ...



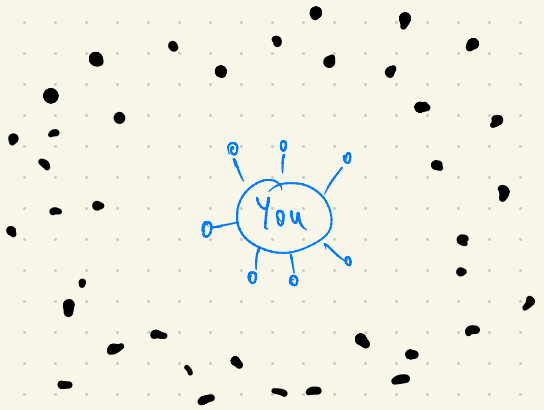
SPARSE vs DENSE GRAPHS

n : number of vertices

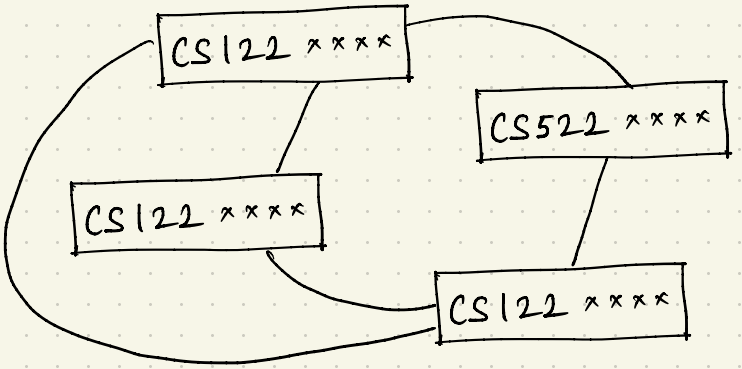
m : number of edges

Many applications : m is $\Omega(n)$ and $O(n^2)$.
(but not always)

Facebook graph is sparse ...



... but also dense



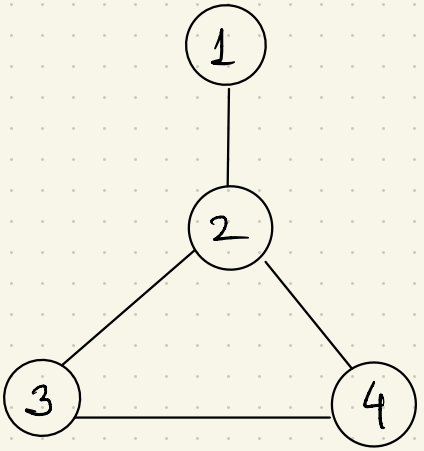
REPRESENTING GRAPHS

REPRESENTING GRAPHS

Adjacency matrix

REPRESENTING GRAPHS

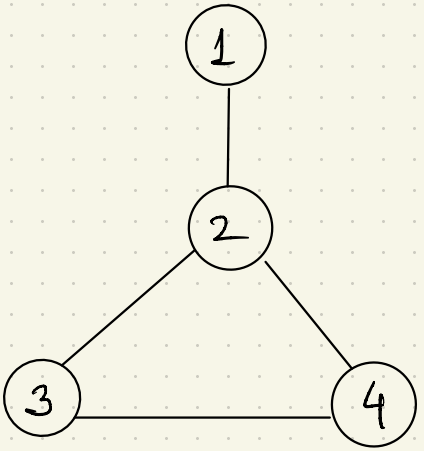
Adjacency matrix



REPRESENTING GRAPHS

Adjacency matrix

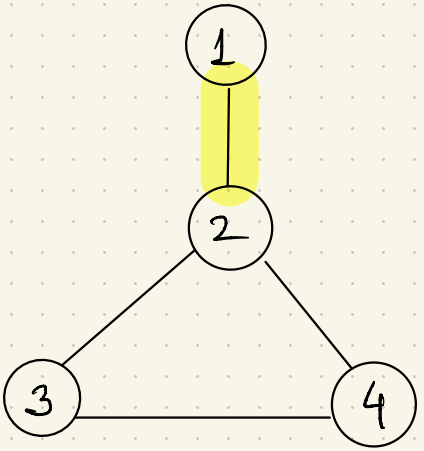
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \end{array}$$



REPRESENTING GRAPHS

Adjacency matrix

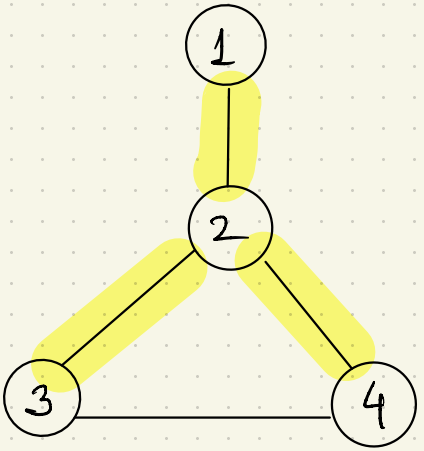
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ & & & \\ & & & \\ & & & \end{bmatrix} \end{array}$$



REPRESENTING GRAPHS

Adjacency matrix

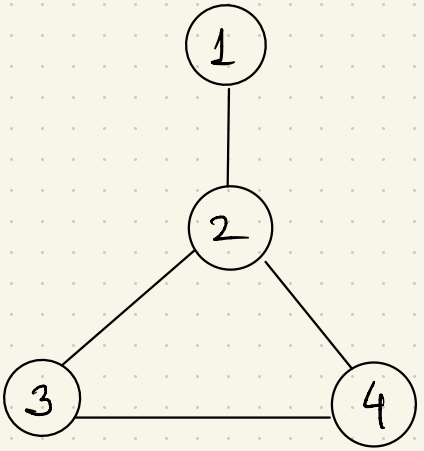
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ & & & \\ & & & \end{bmatrix} \end{array}$$



REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

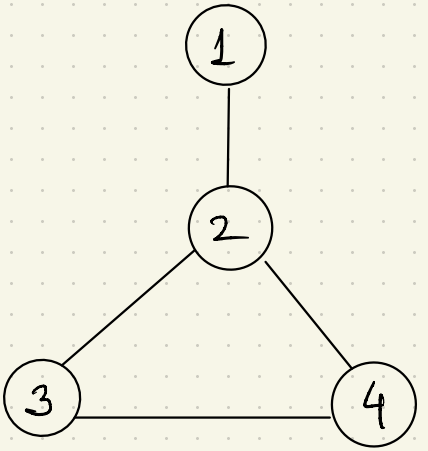


REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

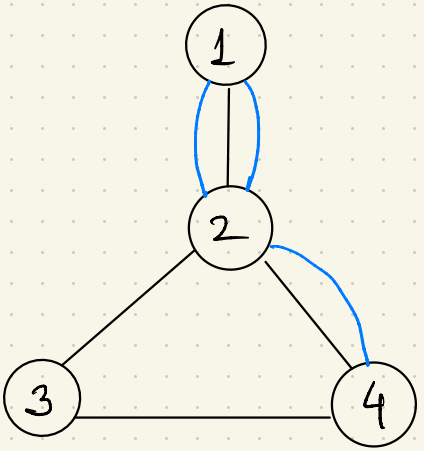
symmetric



REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 0 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix} \end{array}$$

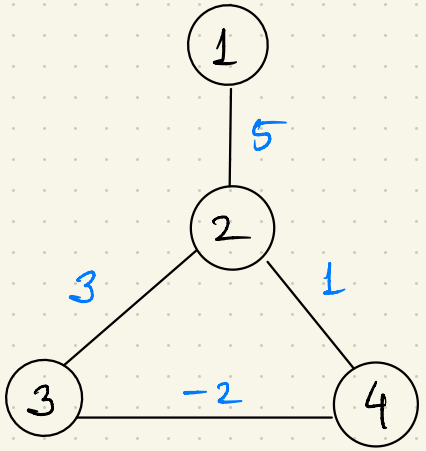


parallel edges

REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 5 & 0 & 0 \\ 5 & 0 & 3 & 1 \\ 0 & 3 & 0 & -2 \\ 0 & 1 & -2 & 0 \end{bmatrix} \end{array}$$



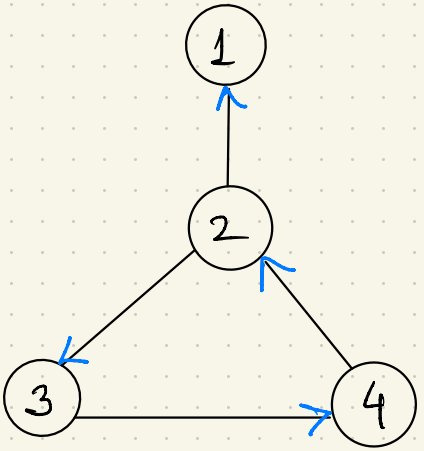
edge weights

REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & -1 & 0 & 0 \\ +1 & 0 & +1 & -1 \\ 0 & -1 & 0 & +1 \\ 0 & +1 & -1 & 0 \end{bmatrix} \end{array}$$

skew symmetric



Directed edges

REPRESENTING GRAPHS

Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \end{array}$$

$\Theta(n^2)$ space

OK for dense graphs

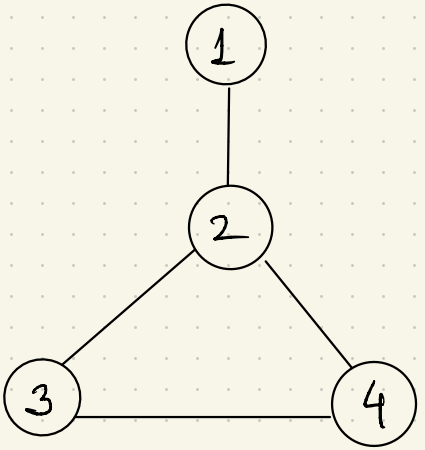
Not OK for sparse graphs

REPRESENTING GRAPHS

Adjacency list

REPRESENTING GRAPHS

Adjacency list

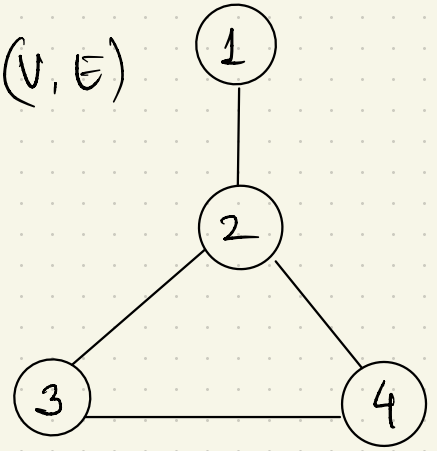


REPRESENTING GRAPHS

Adjacency list

array of $|V|$ linked lists

$G = (V, E)$



REPRESENTING GRAPHS

Adjacency list

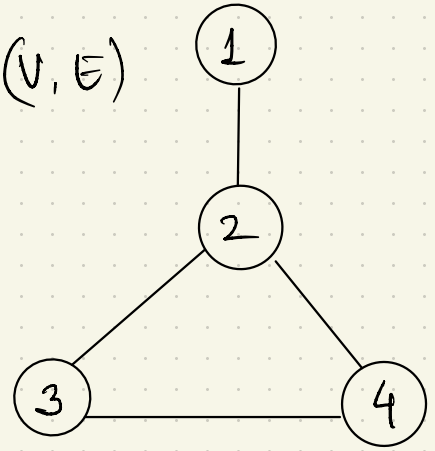
array of $|V|$ linked lists

for each vertex $u \in V$,

Adj[u] stores neighbors of u

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

$G = (V, E)$



REPRESENTING GRAPHS

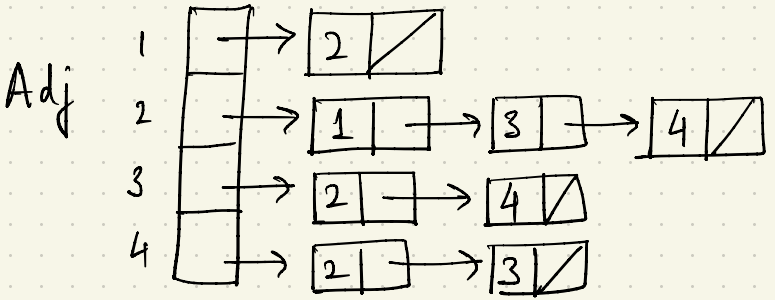
Adjacency list

array of $|V|$ linked lists

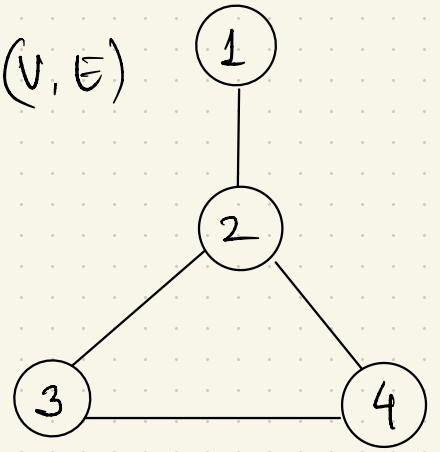
for each vertex $u \in V$,

Adj[u] stores neighbors of u

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



$G = (V, E)$



REPRESENTING GRAPHS

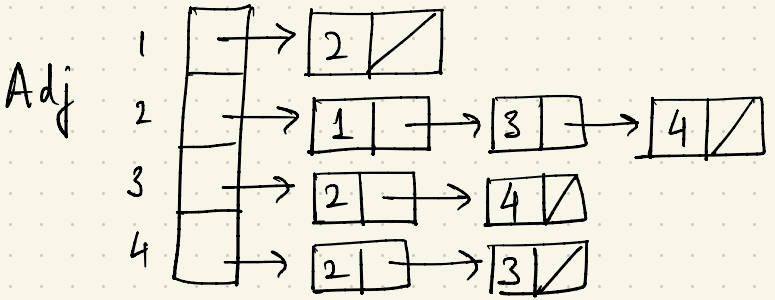
Adjacency list

array of $|V|$ linked lists

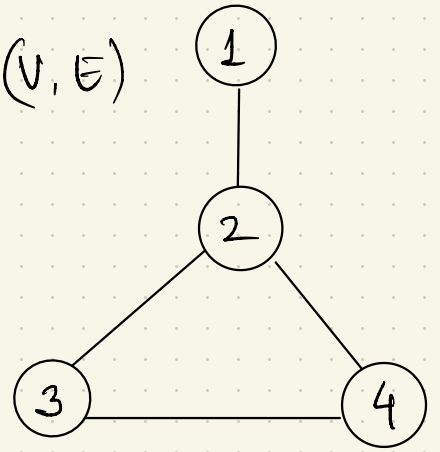
for each vertex $u \in V$,

Adj[u] stores neighbors of u

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



$G = (V, E)$



Space : ?

REPRESENTING GRAPHS

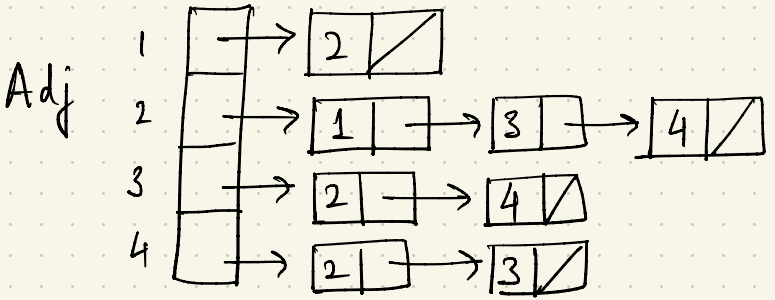
Adjacency list

array of $|V|$ linked lists

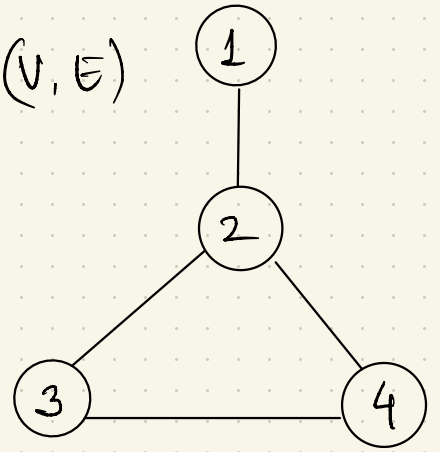
for each vertex $u \in V$,

Adj[u] stores neighbors of u

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



$G = (V, E)$



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

This course

Which is better : adjacency matrix or adjacency list ?

Depends on graph density and operations needed.

REPRESENTING GRAPHS

This course

Which is better : adjacency matrix or adjacency list ?

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

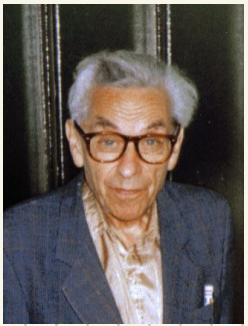
Checking connectivity

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

WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

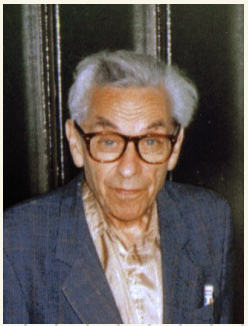
- Web crawling, "friend finder" on social networks, garbage collection, ...
- Is my Erdös number ≤ 3 ?



WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

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

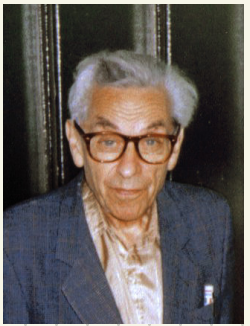


0

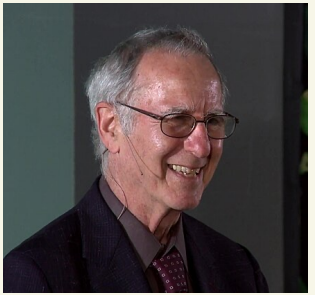
WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

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



0

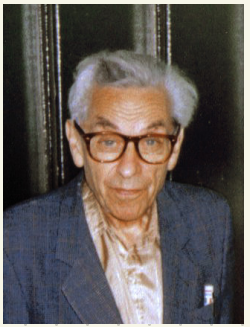


1

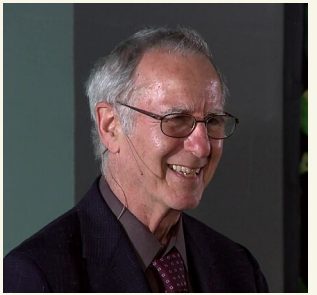
WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

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



0



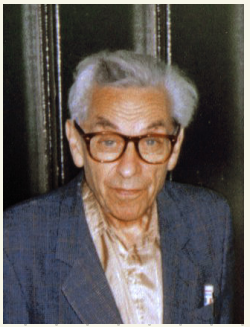
1

2

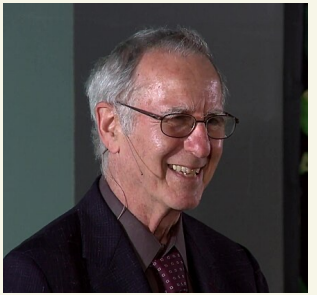
WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

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



0



1

2

3

WHY CARE ABOUT GRAPH SEARCH

Checking connectivity

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

Shortest paths

- fewest moves to solve Rubik's cube

Revealing structure

- finding "communities" in networks

GENERIC GRAPH SEARCH

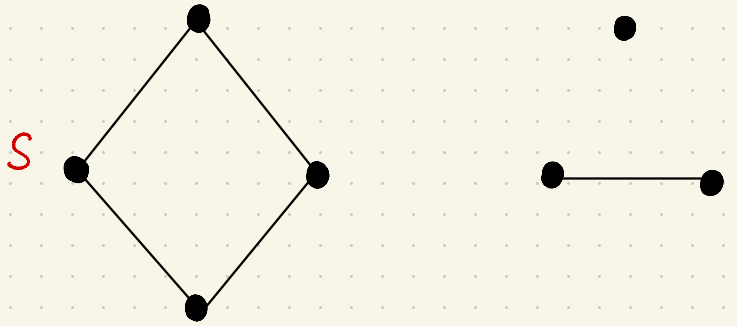
input: an undirected or directed graph $G=(V,E)$, starting vertex s

goal: identify the vertices of V reachable from s

GENERIC GRAPH SEARCH

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

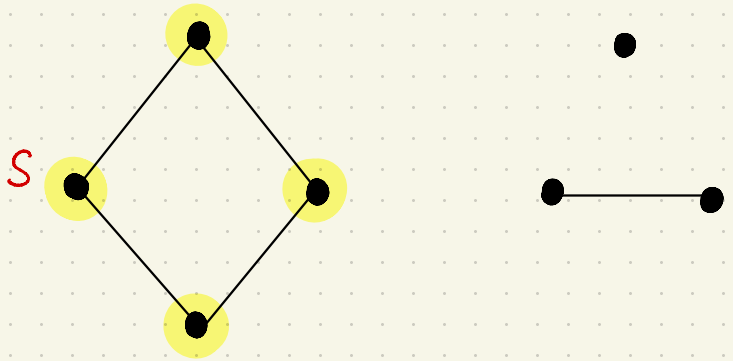
goal: identify the vertices of V reachable from s



Generic Graph Search

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

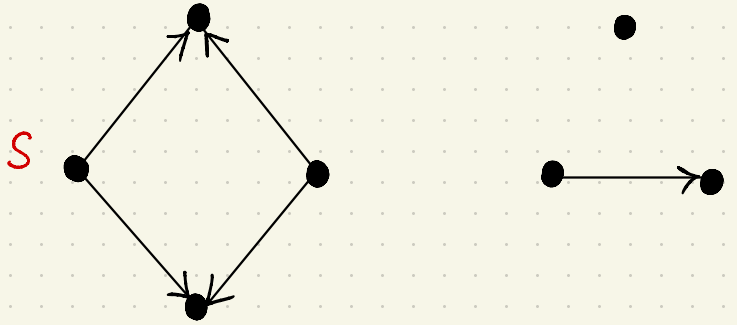
goal: identify the vertices of V reachable from s



Generic Graph Search

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

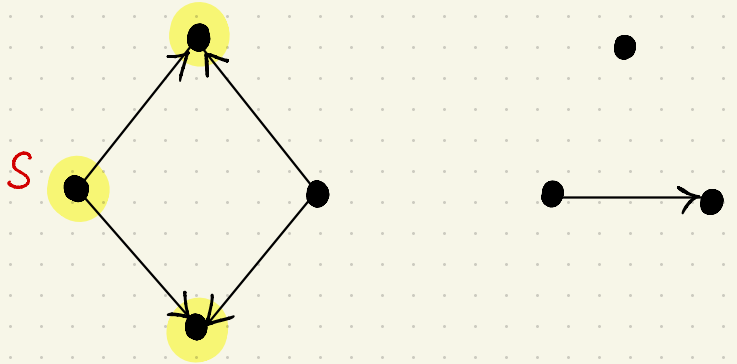
goal: identify the vertices of V reachable from s



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

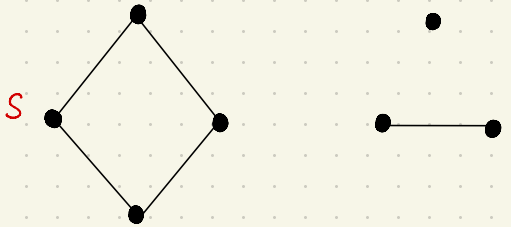
goal: identify the vertices of V reachable from s efficiently

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

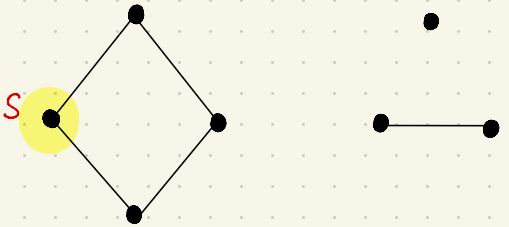
goal: identify the vertices of V reachable from s efficiently

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

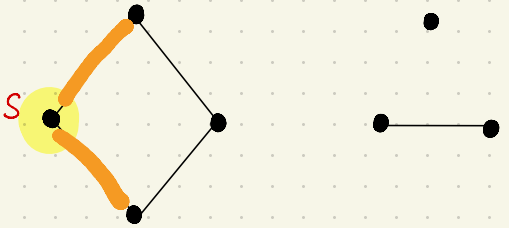
goal: identify the vertices of V reachable from s efficiently

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

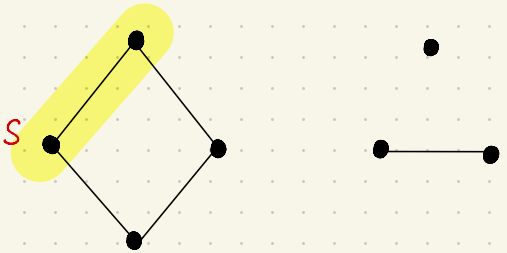
goal: identify the vertices of V reachable from s efficiently

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

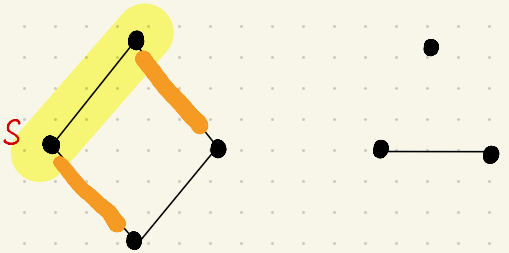
goal: identify the vertices of V reachable from s efficiently

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

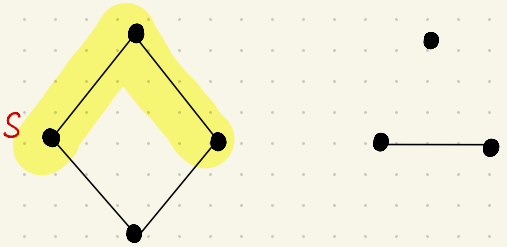
goal: identify the vertices of V reachable from s efficiently

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

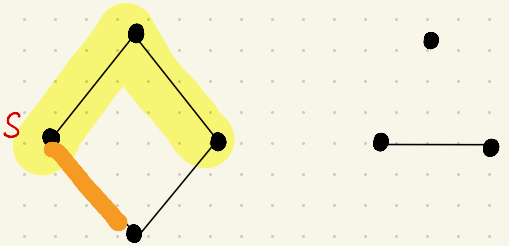
goal: identify the vertices of V reachable from s efficiently

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

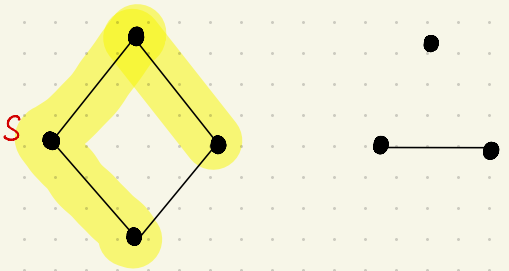
goal: identify the vertices of V reachable from s efficiently

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

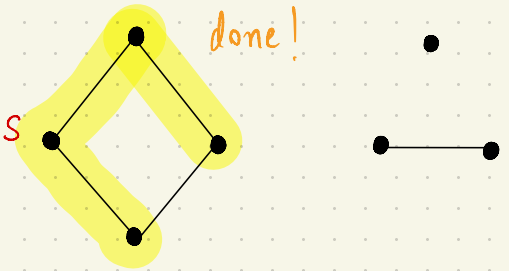
goal: identify the vertices of V reachable from s efficiently

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

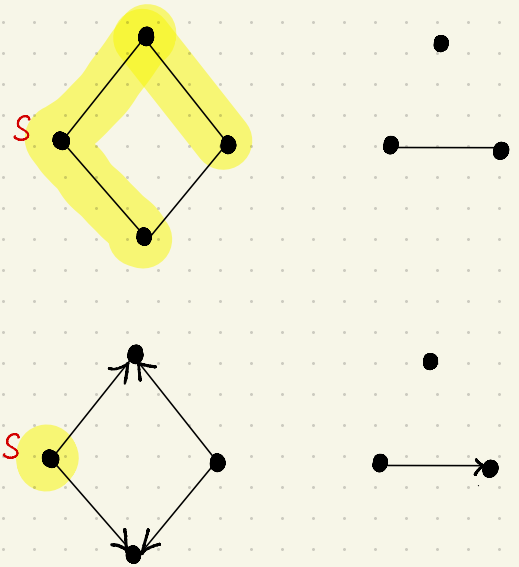
goal: identify the vertices of V reachable from s efficiently

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

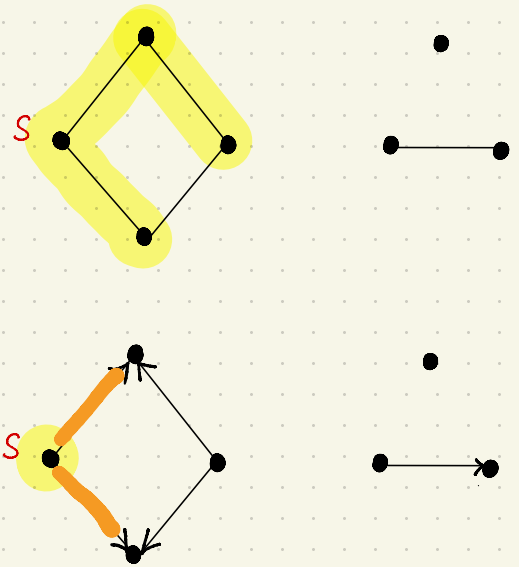
goal: identify the vertices of V reachable from s efficiently

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

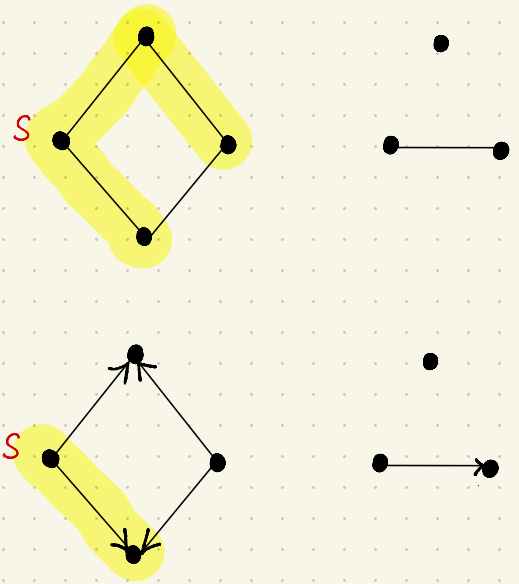
goal: identify the vertices of V reachable from s efficiently

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

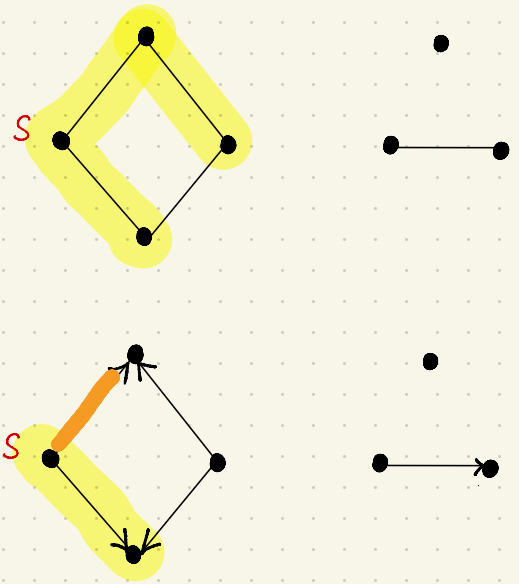
goal: identify the vertices of V reachable from s efficiently

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

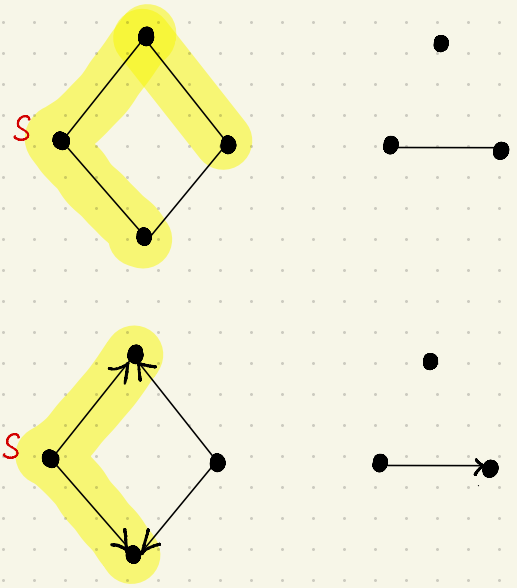
goal: identify the vertices of V reachable from s efficiently

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

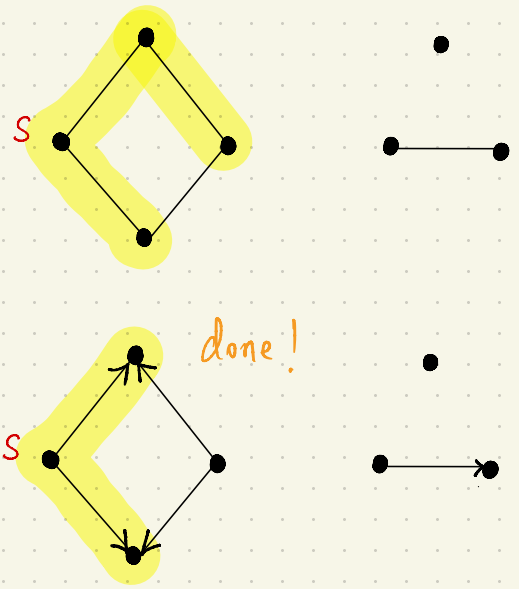
goal: identify the vertices of V reachable from s efficiently

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: ($\Rightarrow$) 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: ($\Rightarrow$) Only way to explore is via paths from s.

($\Leftarrow$)

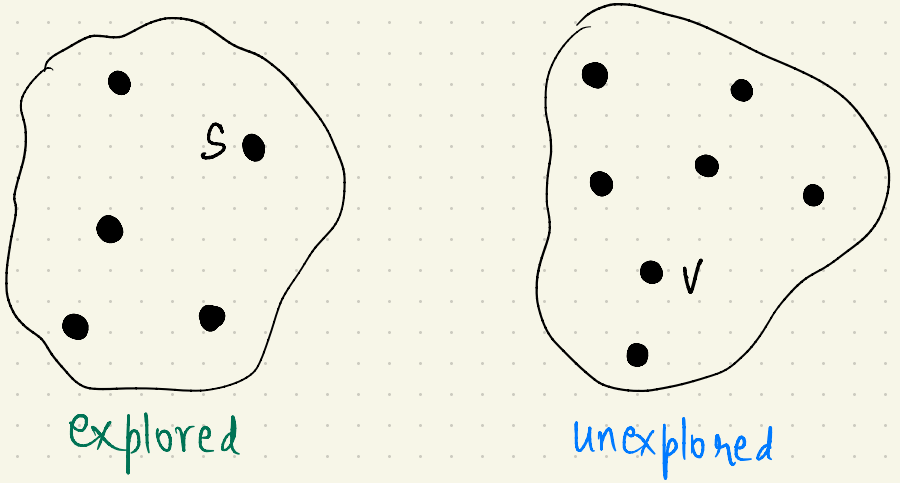
Generic Graph Search

Claim: At the conclusion of Generic algorithm,

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

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

($\Leftarrow$) by contradiction.



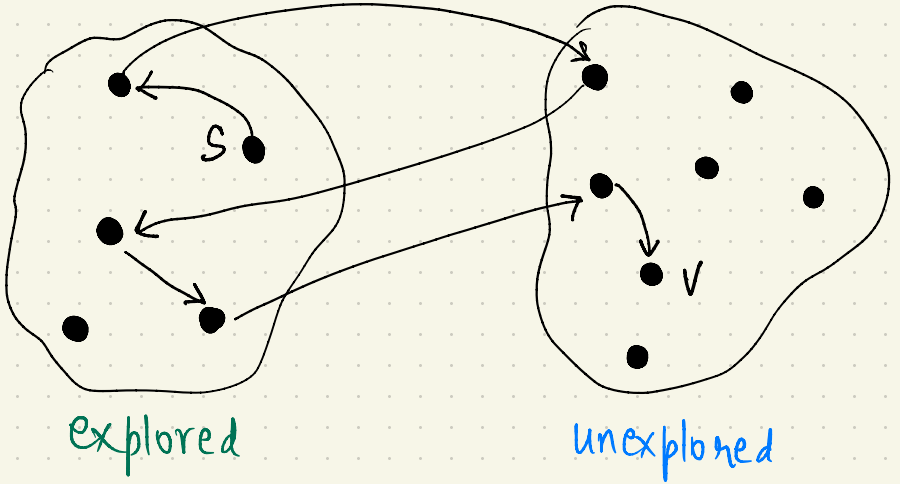
Generic Graph Search

Claim: At the conclusion of Generic algorithm,

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

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

($\Leftarrow$) by contradiction.



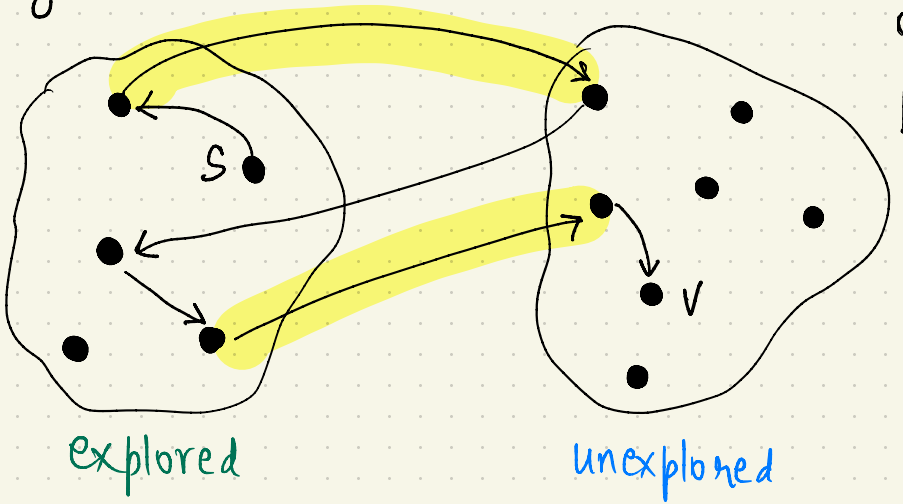
Generic Graph Search

Claim: At the conclusion of Generic algorithm,

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

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

($\Leftarrow$) by contradiction.



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

x : explored y : unexplored

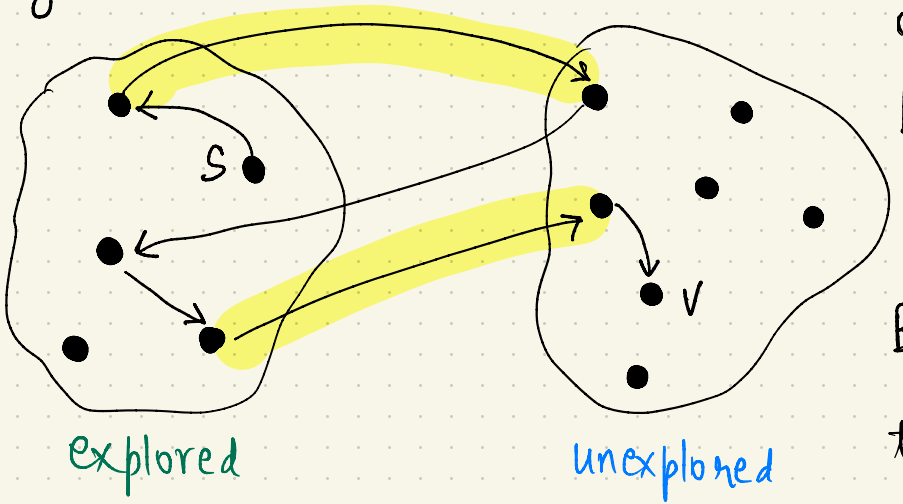
Generic Graph Search

Claim: At the conclusion of Generic algorithm,

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

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

($\Leftarrow$) by contradiction.



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

x : explored y : 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

  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

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., Erdös 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

  mark v explored

Breadth-First Search: explore nodes in "layers" (e.g., Erdös 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., Erdös 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., Erdös 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

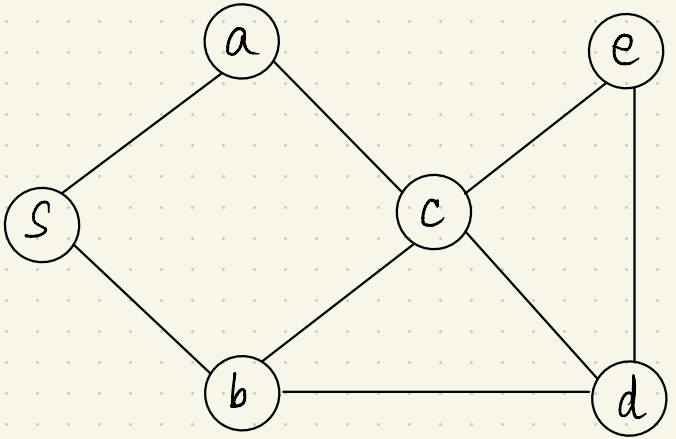
  mark v explored

Breadth-First Search

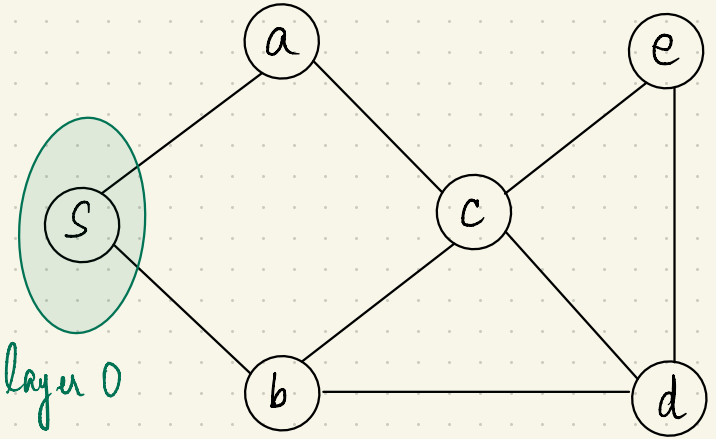
Depth-First Search

Linear time: $O(m+n)$

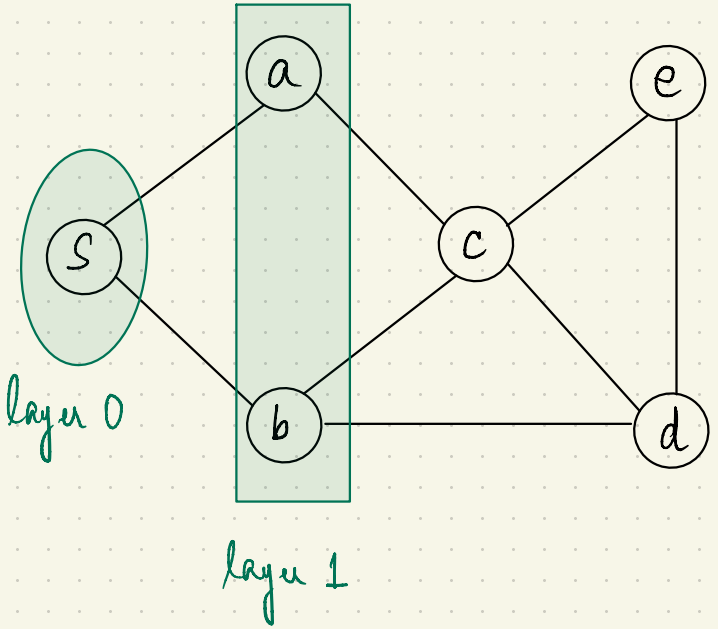
BREADTH - FIRST SEARCH



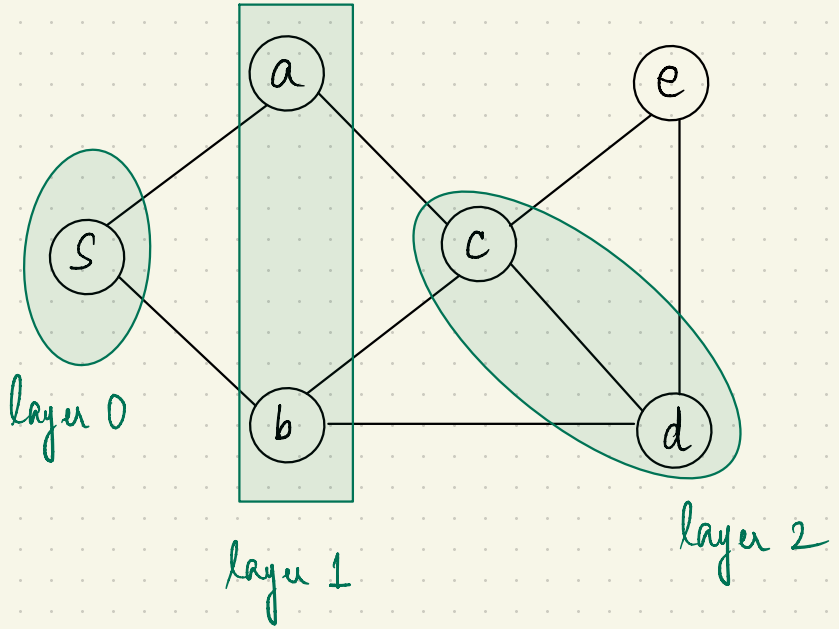
BREADTH - FIRST SEARCH



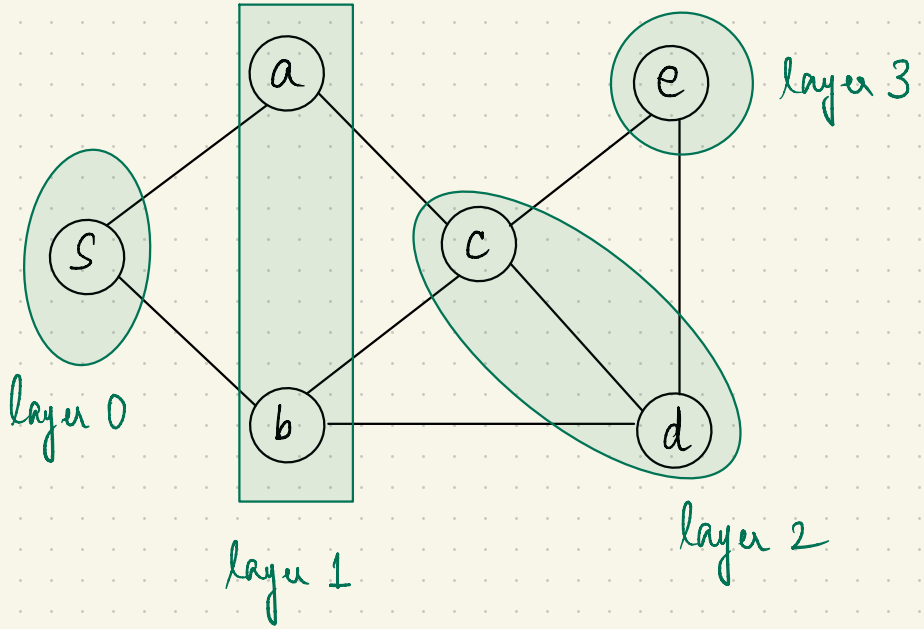
BREADTH - FIRST SEARCH



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 ≠ ∅
    remove the first node of Q, say v
    for each (v, w) ∈ 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

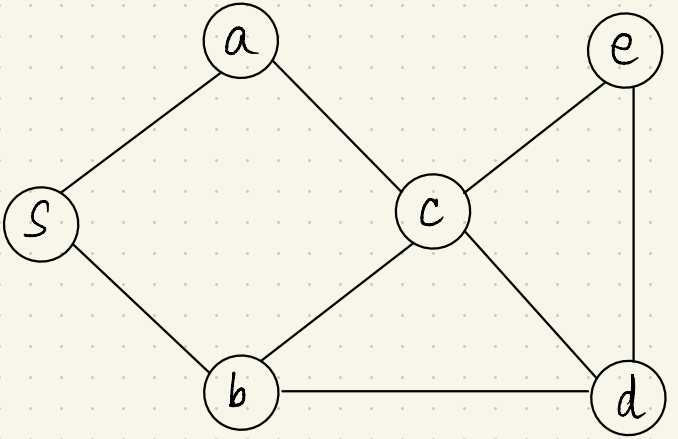
    for each $(v, w) \in E$

        if w is unexplored

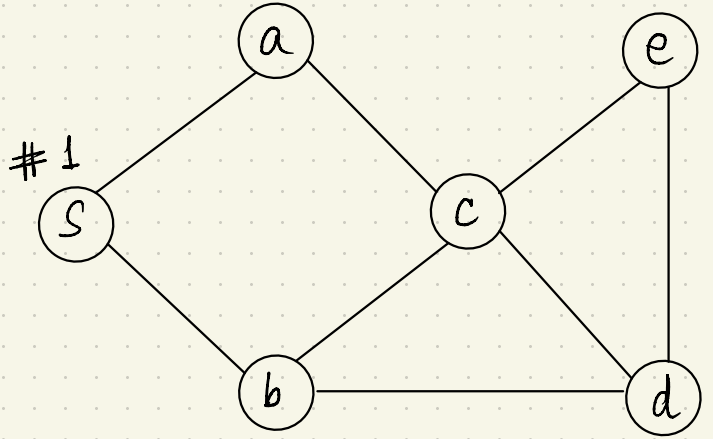
            mark w as explored

            add w to the end of Q

BREADTH - FIRST SEARCH

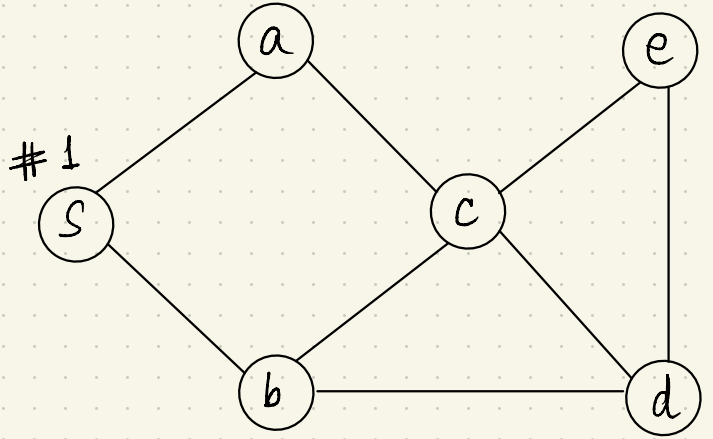


BREADTH - FIRST SEARCH



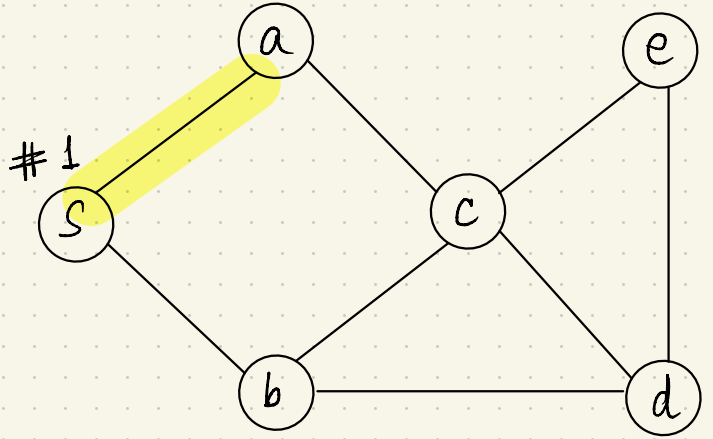
Q : S

BREADTH - FIRST SEARCH



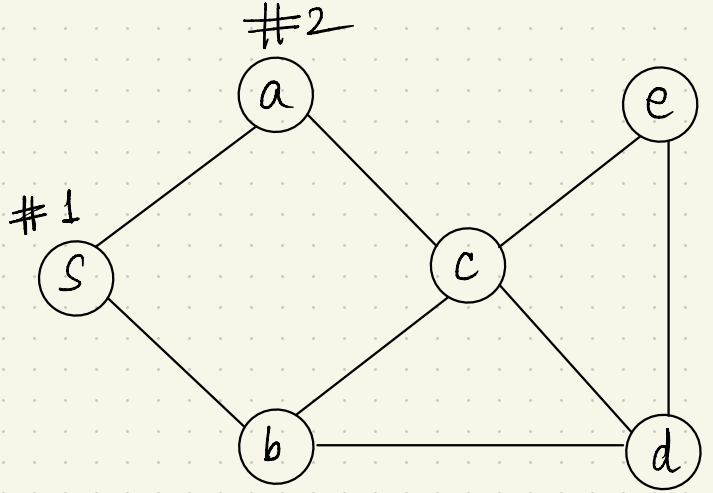
Q : ~~s~~

BREADTH - FIRST SEARCH



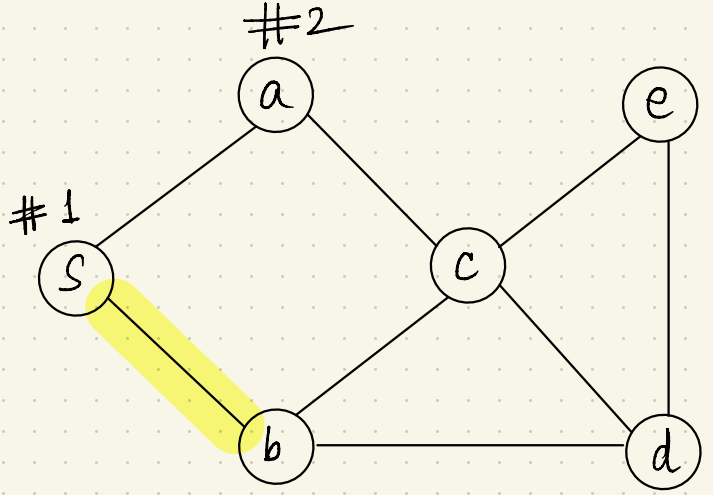
Q : ~~s~~

BREADTH - FIRST SEARCH



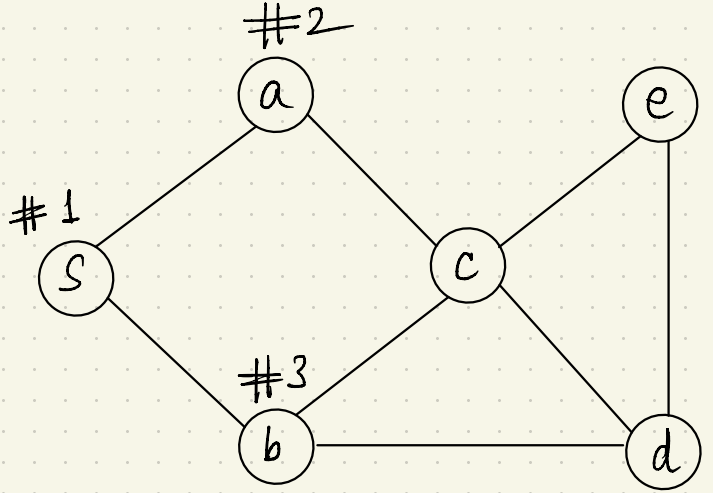
Q : ~~s~~ a

BREADTH - FIRST SEARCH



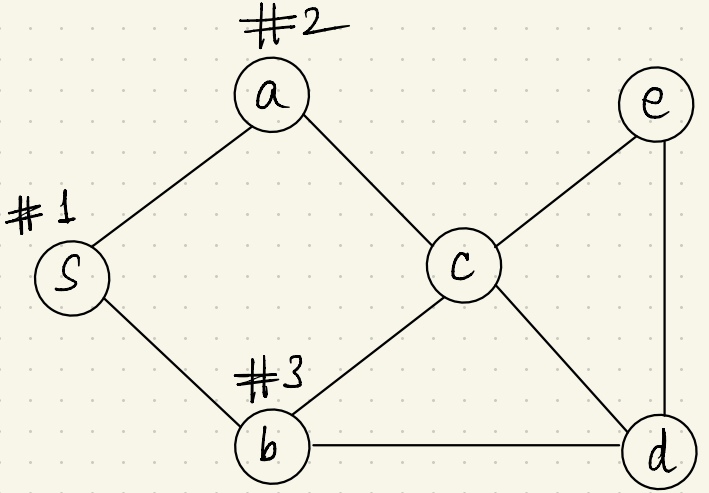
Q : ~~S~~ a

BREADTH - FIRST SEARCH



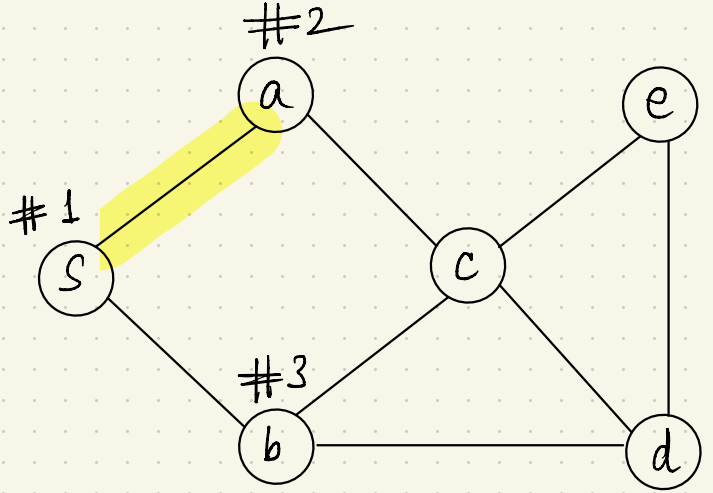
Q : ~~s~~ a b

BREADTH - FIRST SEARCH



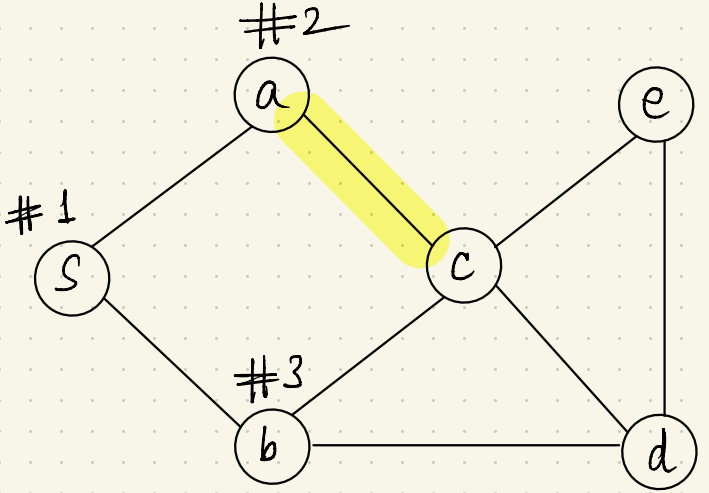
Q : ~~s~~ ~~a~~ b

BREADTH - FIRST SEARCH



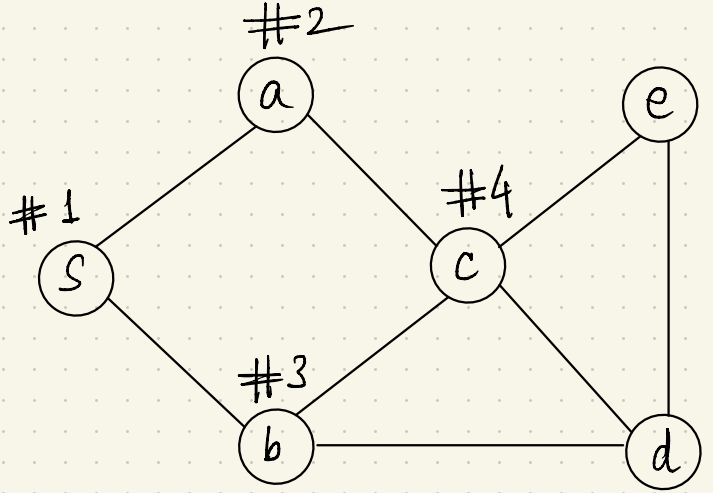
Q : ~~s~~ ~~a~~ b

BREADTH - FIRST SEARCH



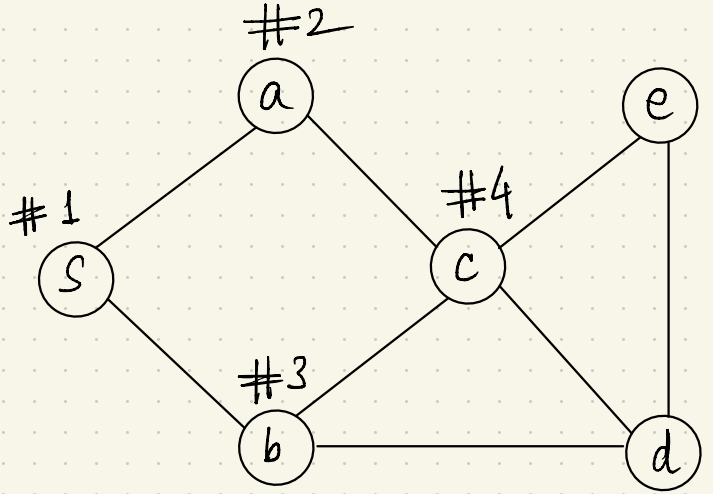
Q : ~~S~~ ~~a~~ b

BREADTH - FIRST SEARCH



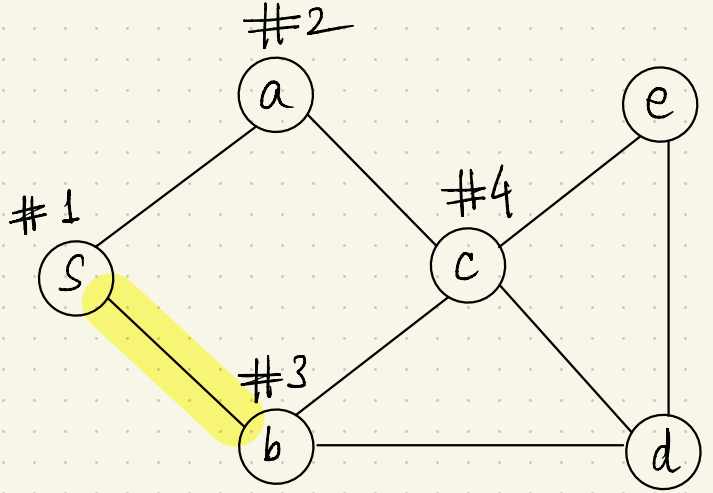
Q : ~~s~~ ~~a~~ b c

BREADTH - FIRST SEARCH



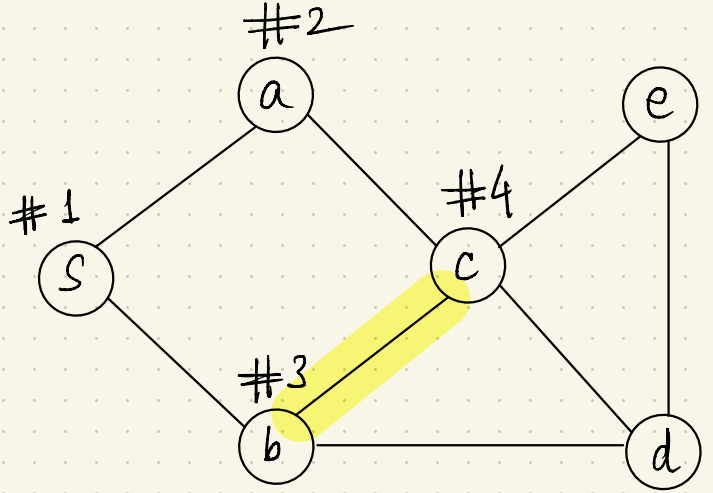
Q : ~~s~~ ~~a~~ ~~b~~ c

BREADTH - FIRST SEARCH



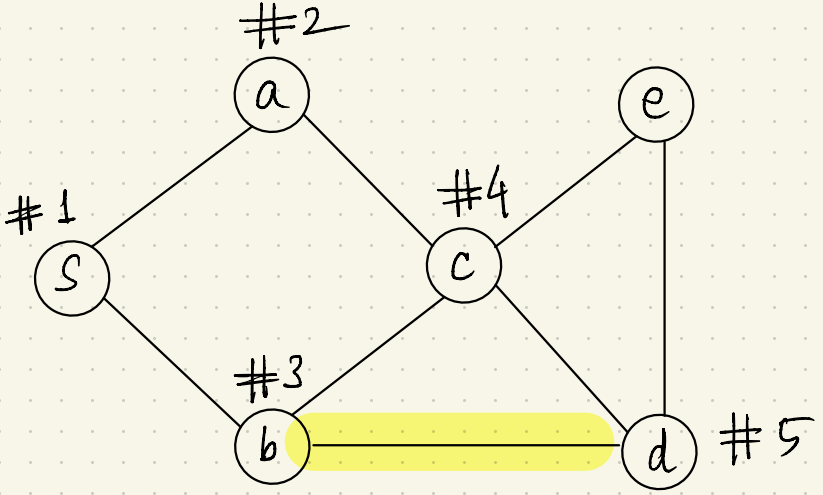
Q : ~~s~~ ~~a~~ ~~b~~ c

BREADTH - FIRST SEARCH



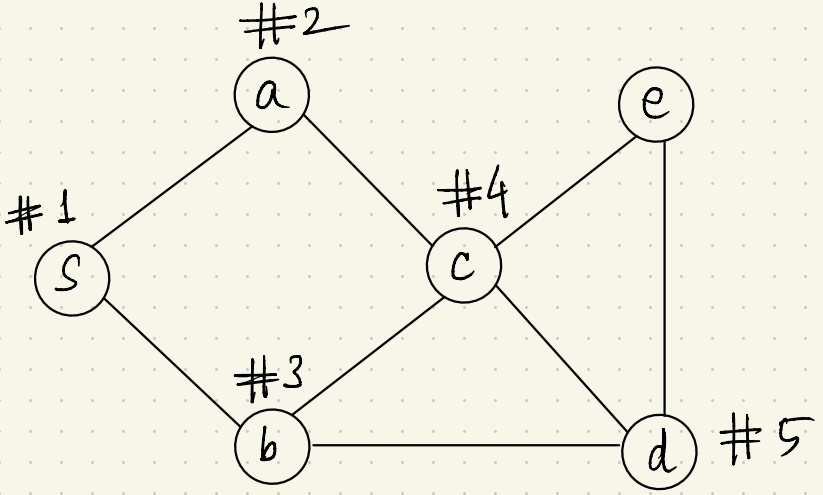
Q : ~~s~~ ~~a~~ ~~b~~ c

BREADTH - FIRST SEARCH



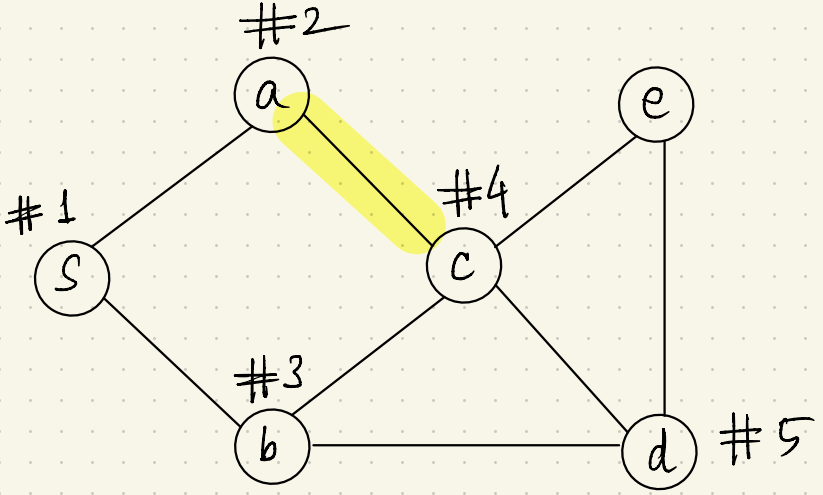
Q : ~~s~~ ~~a~~ ~~b~~ c d

BREADTH - FIRST SEARCH



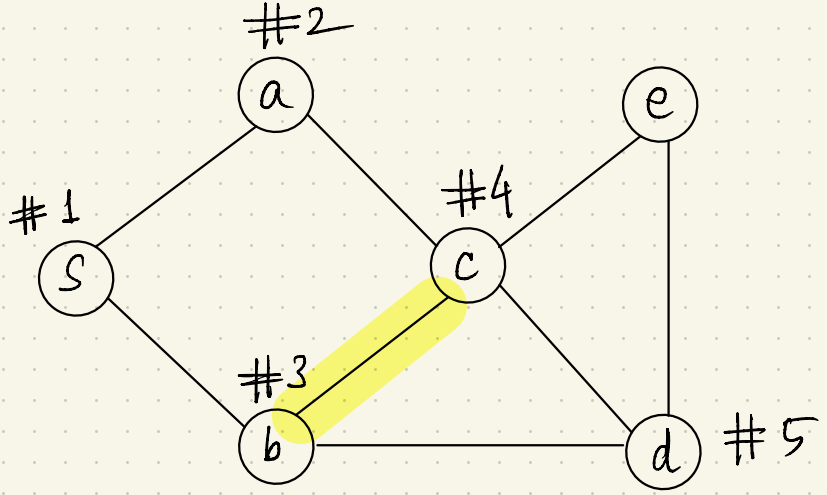
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



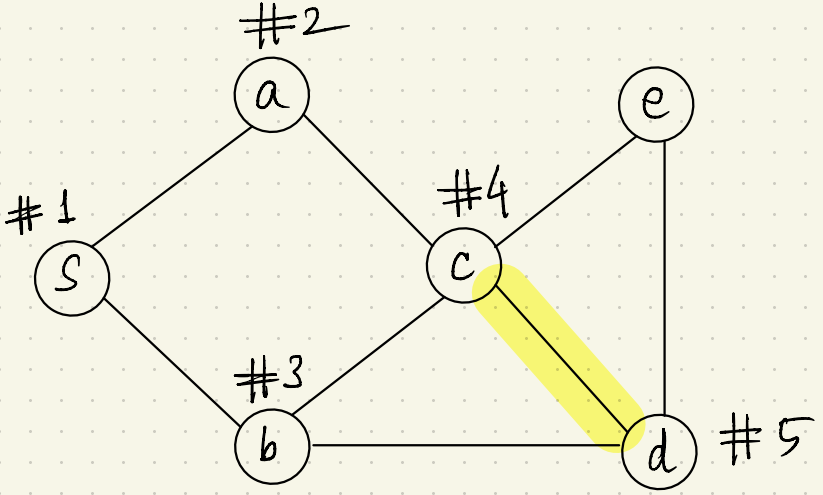
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



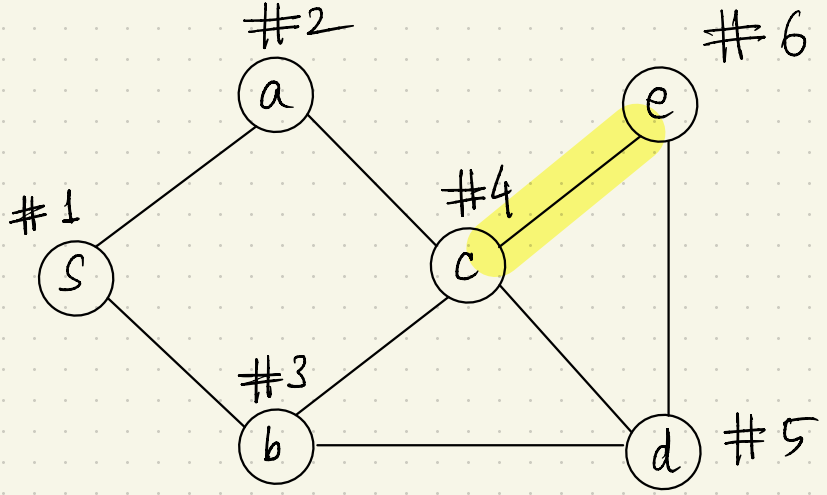
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



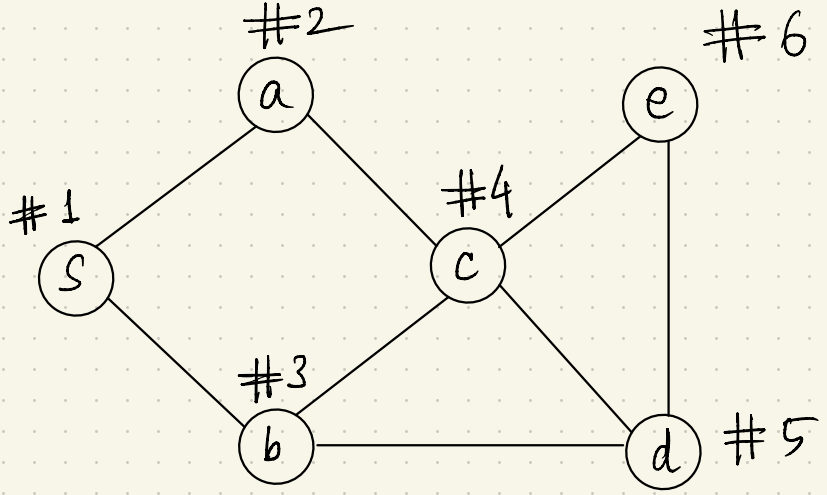
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d

BREADTH - FIRST SEARCH



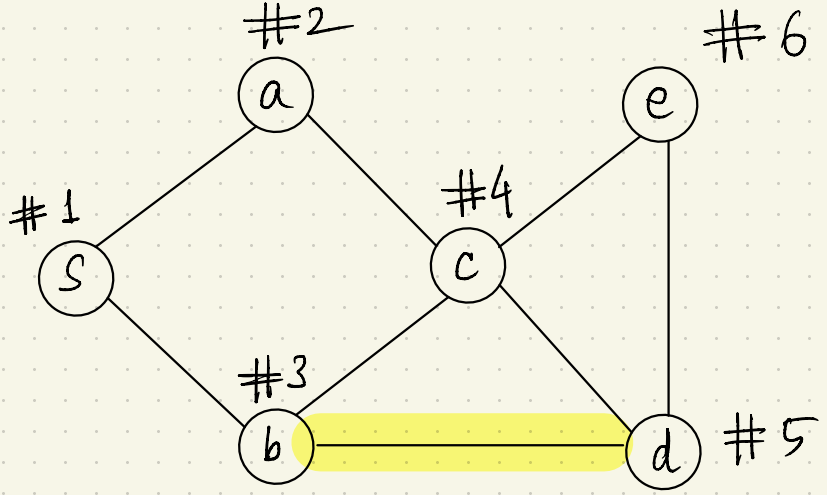
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ d e

BREADTH - FIRST SEARCH



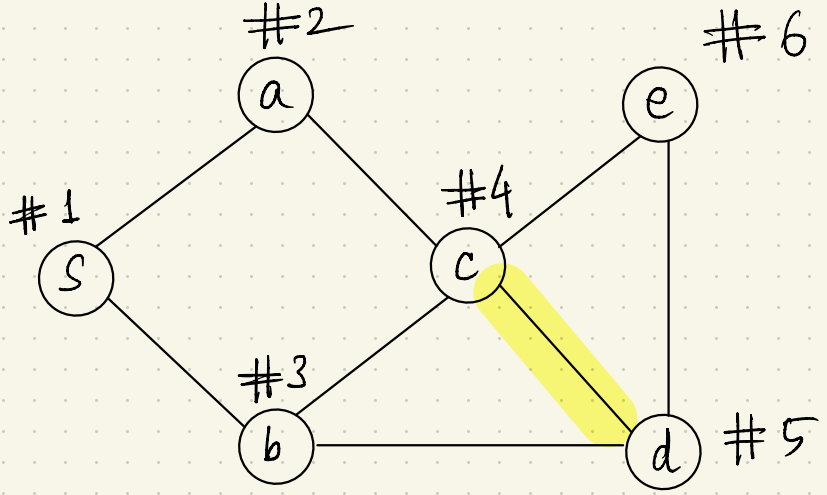
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



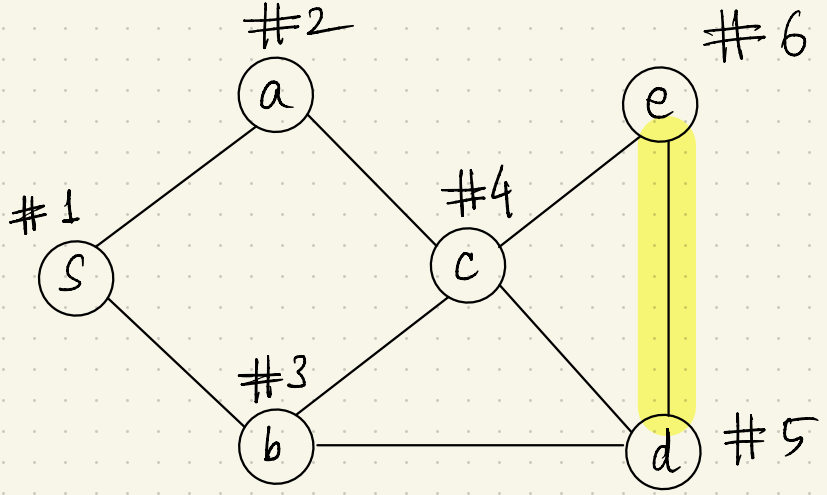
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



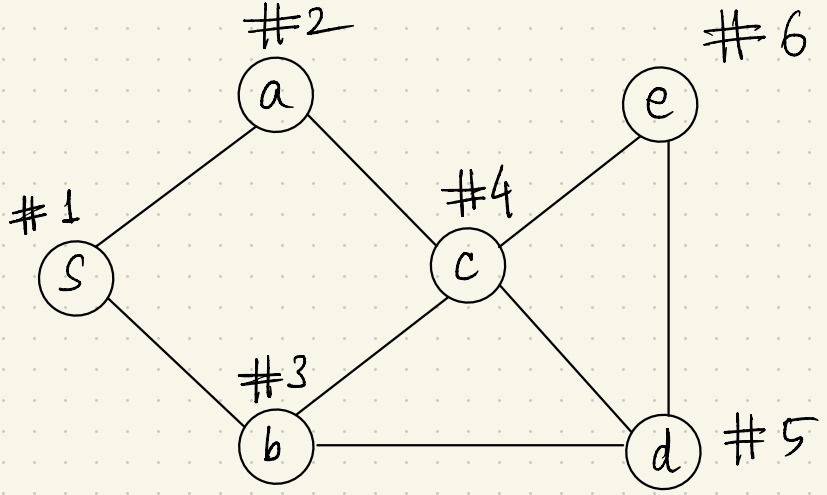
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



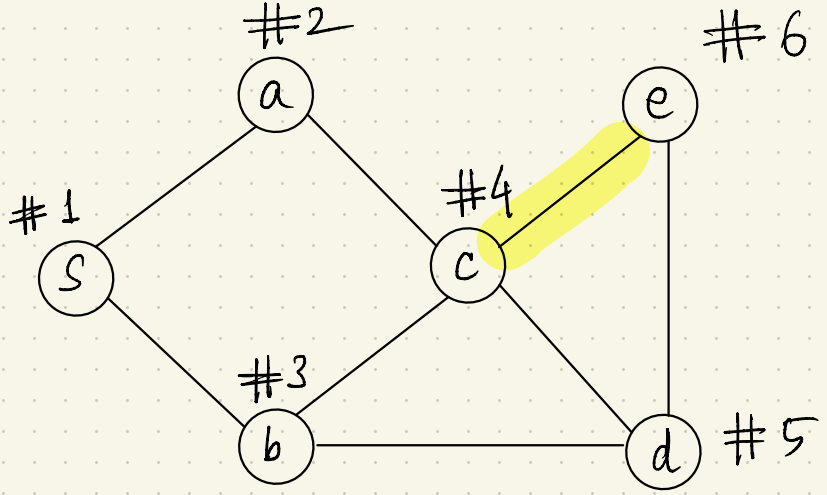
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

BREADTH - FIRST SEARCH



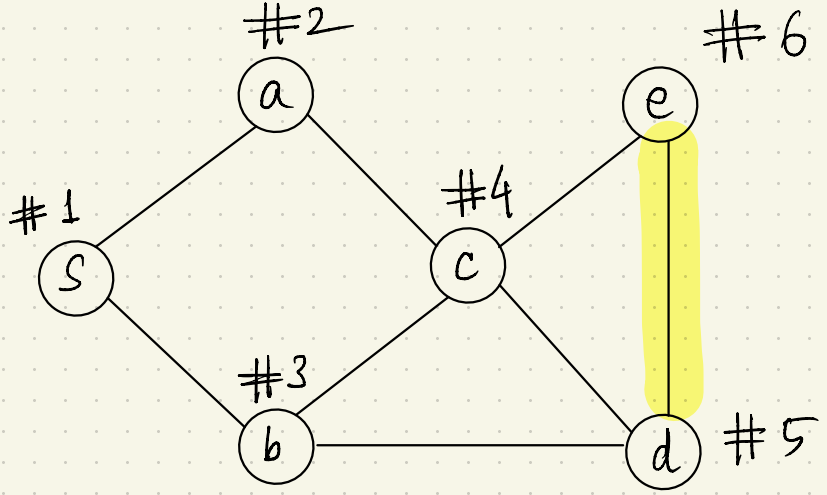
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~

BREADTH - FIRST SEARCH



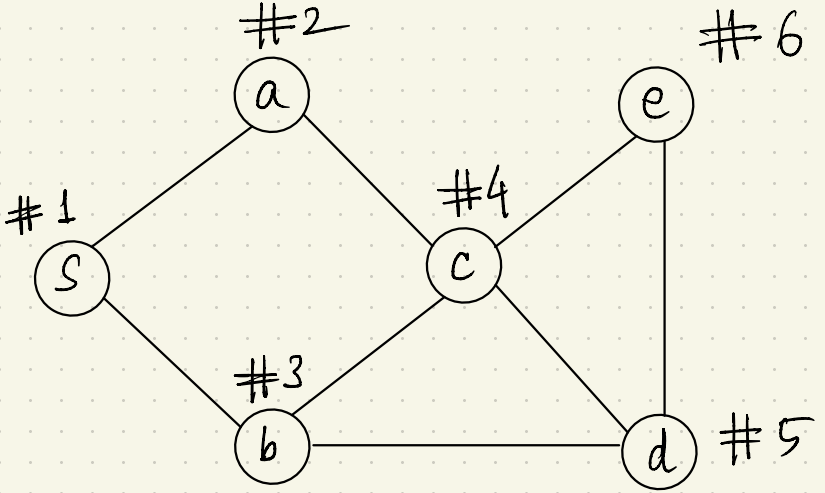
Q : ~~S~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~

BREADTH - FIRST SEARCH



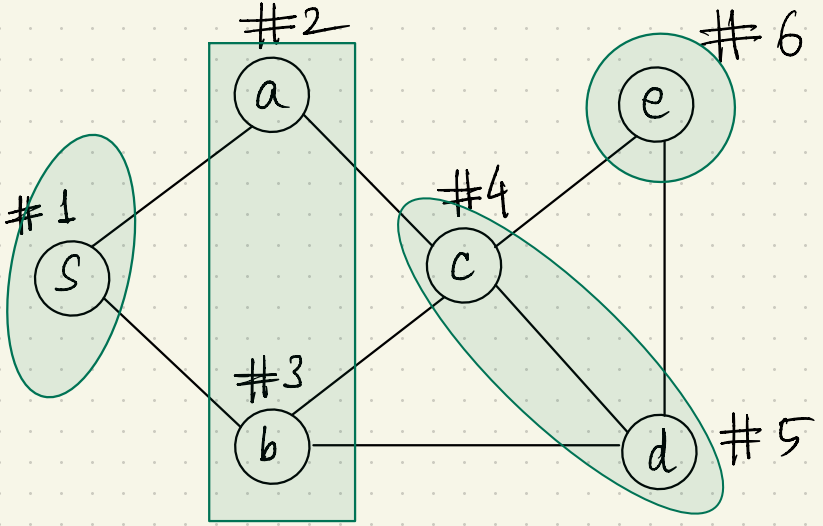
Q : ~~s~~ ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~

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 ≠ ∅
    remove the first node of Q, say v
    for each (v, w) ∈ E
        if w is unexplored
            mark w as explored
            add w to the end of Q
```