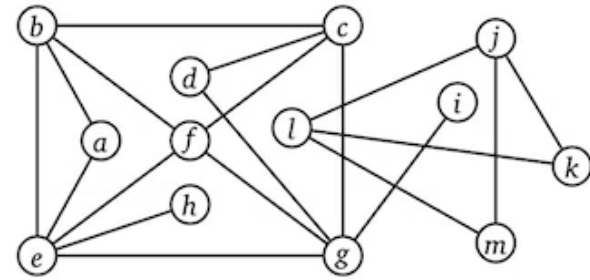
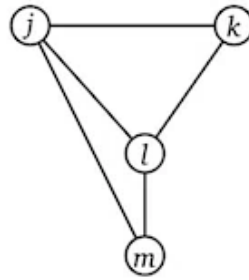
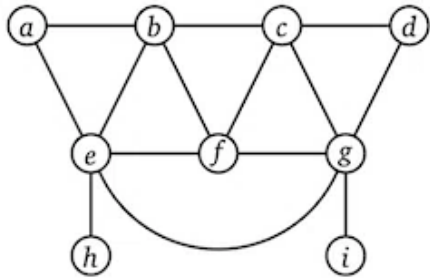


HW 7: today Oct 20
GPS 8: Mon Oct 26

Graphs



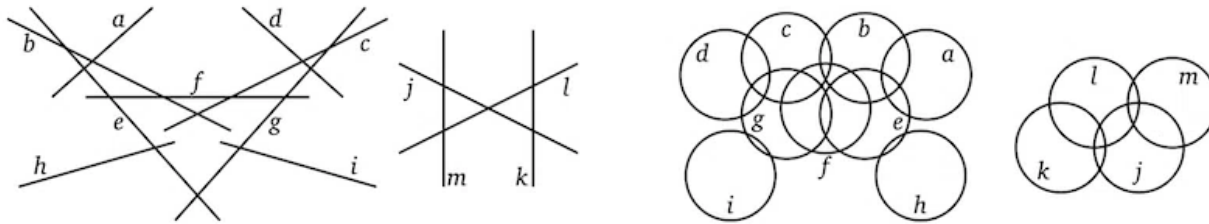
A graph $G = (V, E)$ consists of

V : an arbitrary non-empty finite set:
vertices or nodes

E : if G is directed; ordered pairs of vertices
if G is undirected: unordered sets of
two vertices

edges or arcs.

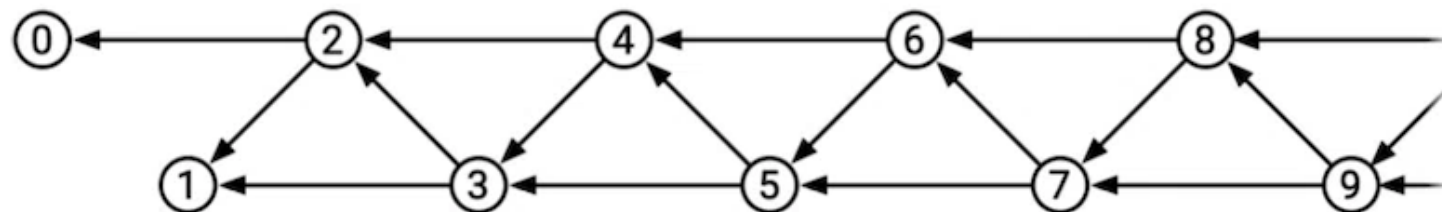
if directed: $u \rightarrow v$ really means some $(u, v) \in E$
 undirected: uv really means some $\{u, v\} \in E$



intersection graph

dependency graph

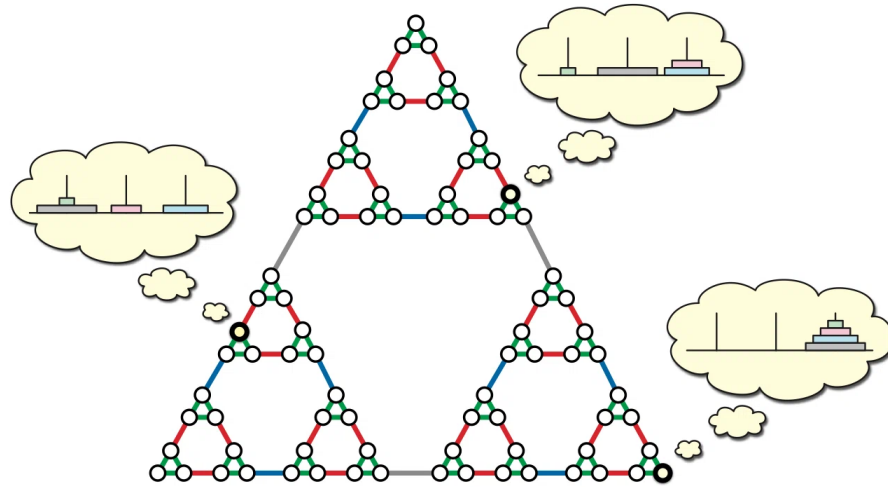
vertices: subproblems



$u \rightarrow v$:
 u calls v

Fibonacci computation

configuration graph



vertices: configurations

edge uv : can go

back & forth between

$u \neq v$

Data structures: Let $G=(V,E)$ be a graph (directed or undirected).

Assume G is simple (no parallel edges)

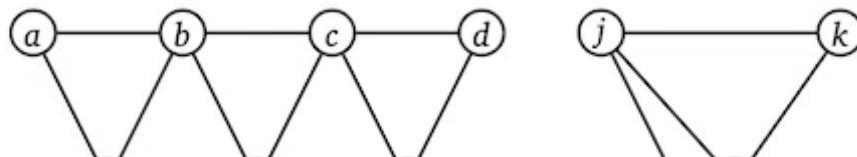
adjacency matrix: $|V| \times |V|$ matrix (2D array)

rows + columns indexed by vertex

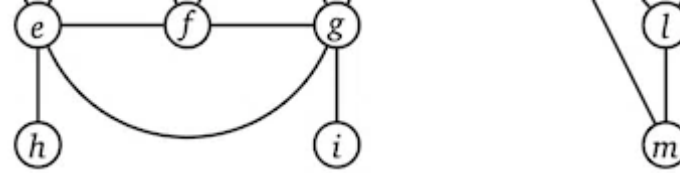
If G directed: entry $(u,v) = 1$ if $u \rightarrow v \in E$
 $= 0$ otherwise

undirected: entry $(u,v) = 1$ if $uv \in E$
 $= 0$ otherwise

	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0	1	0	0	1	0	0	0	0	0	0	0	0
b	1	0	1	0	1	1	0	0	0	0	0	0	0
c	0	1	0	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	1	0	0	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	0	0	0
f	0	1	1	0	1	0	1	0	0	0	0	0	0



j	0	1	1	0	1	0	1	0	0	0	0	0
g	0	0	1	1	1	1	0	0	1	0	0	0
h	0	0	0	0	1	0	0	0	0	0	0	0
i	0	0	0	0	0	0	1	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	1	1
k	0	0	0	0	0	0	0	0	0	1	0	1
l	0	0	0	0	0	0	0	0	0	1	1	0
m	0	0	0	0	0	0	0	0	0	1	0	1



Check if $u \rightarrow v \in E$ ($uv \in E$) in $O(1)$ time.

Find u 's neighbors (all $v \in V$ s.t. $u \rightarrow v \in E$
 $uv \in E$)

takes $O(|V|)$ time

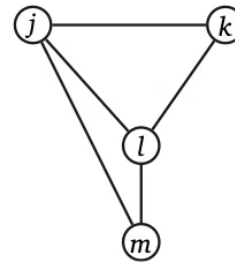
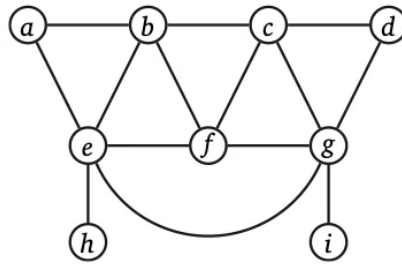
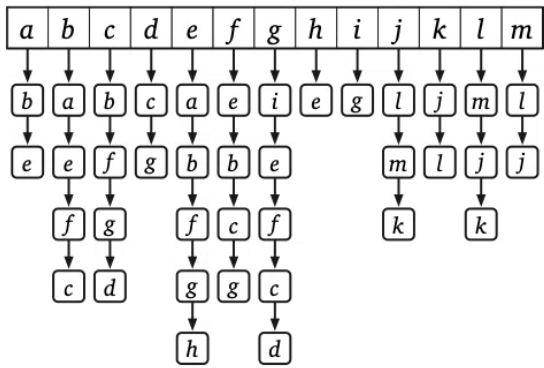
Always $\Theta(|V|^2)$ space

* Adjacency list: 1D array indexed by vertices

Entry u is a list of all vertices

v s.t. $u \rightarrow v \in E$.

($uv \in E$)



degree of u
 $= \#$ neighbors
of u

Check if $u \rightarrow v \in E$ takes $O(\deg(u))$ time.

$uv \in E$ takes $O(\min\{\deg(u), \deg(v)\})$

Find all $u \rightarrow v$ for a given u takes $O(1)$
per v .

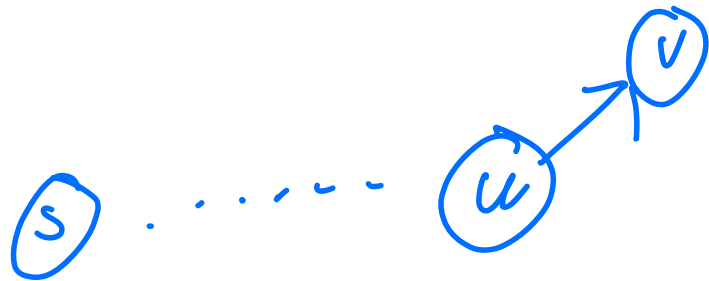
$O(|V| + |E|)$ space.

Reachability.

Given graph $G = (V, E)$ + a vertex $s \in V$.

Want to know all vertices $v \in V$ that s can reach: there is a path from s to v .

If G is undirected, we want the (connected) component of s .



if we find v that we can reach, remember to search out of v later

WhateverFirstSearch(s):

put s into the bag

while the bag is not empty

take u from the bag

if u is unmarked

mark u t.

for each edge $u \rightarrow v$

put v into the bag

to avoid loops, mark
 u when we're about
to explore its edges so
we never do so again

Bag: can insert a vertex
+ later remove a previously
inserted vertex
↑

which one depends
on which structure
is used for the bag

Claim: WFS(s) marks
exactly those vertices
that s can reach.

"Proof": If v is marked,
induction on order
of markings

If v is reachable, use induction on distance

from s to v .

Running time: T : time per insert or remove
from bag

$1 \leq |V|$ times (marks)

$\leq |E| + 1$ bag inserts

$\Rightarrow \leq |E| + 1$ bag removals

$O(|V| + |E| \cdot T)$ time

$O(|V| + |E|)$ if using a stack or queue

Return to the "bag" on Thursday.

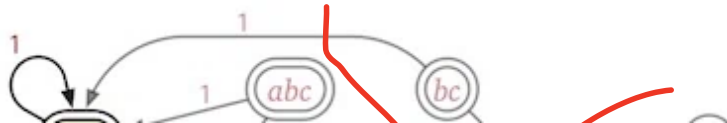
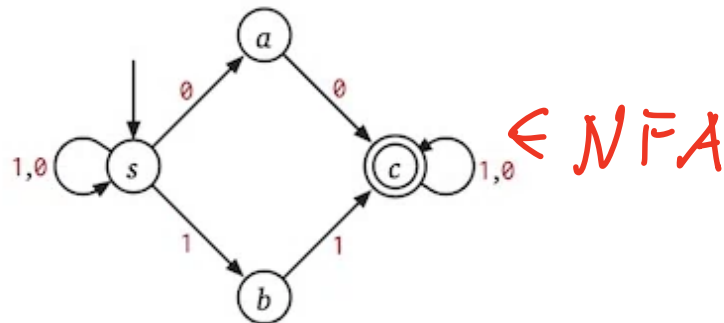
Reductions:

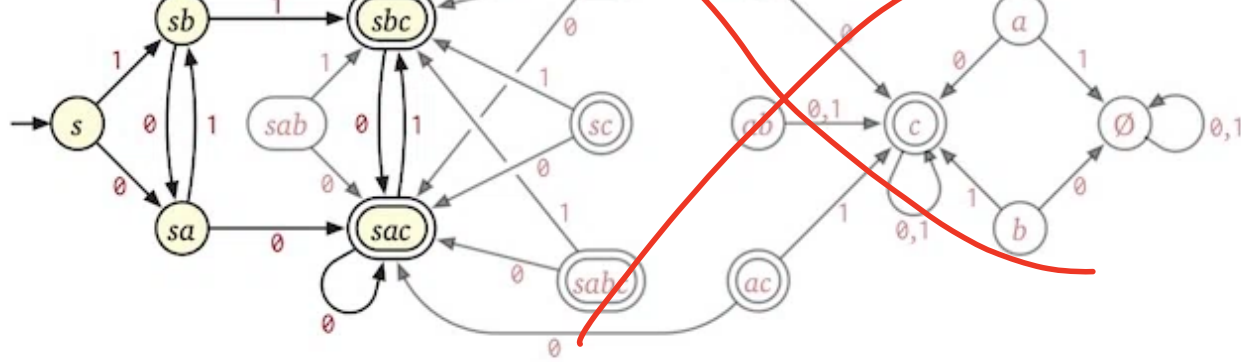
Given a DFA $M = (Q, \delta, s, A)$ with $n = |Q|$ states over $\{0, 1\}^*$.

State $q \in Q$ is useless if no input string brings us to q .

(i.e. no x s.t. $\delta^*(s, x) = q$.)

Want to find all useless states.





← DFA for same language

Build a graph $G = (V, E)$ where

$$V := Q$$

$$E := \{ p \rightarrow q \mid \delta(p, 0) = q \text{ or } \delta(p, 1) = q \}$$

A state q is useless iff s cannot reach q .

So call Whatever First Search (s) (with a queue)
 & return all states not marked.

↑
 assumed

Run time: $O(|V| + |E|)$ to build G .
 $O(|V| + |E|)$ for WFS.

$$\text{So } O(|V| + |E|) = \underline{O(n)}.$$

$$|V| = n$$

$$|E| = 2n$$