CSE 421: Introduction to Algorithms

Complexity

Yin-Tat Lee

Definition for Efficiency in this course

Worst case complexity:

The worst case running time $T(n)$ of an algorithm is

max # steps algorithm takes on any input of size n.

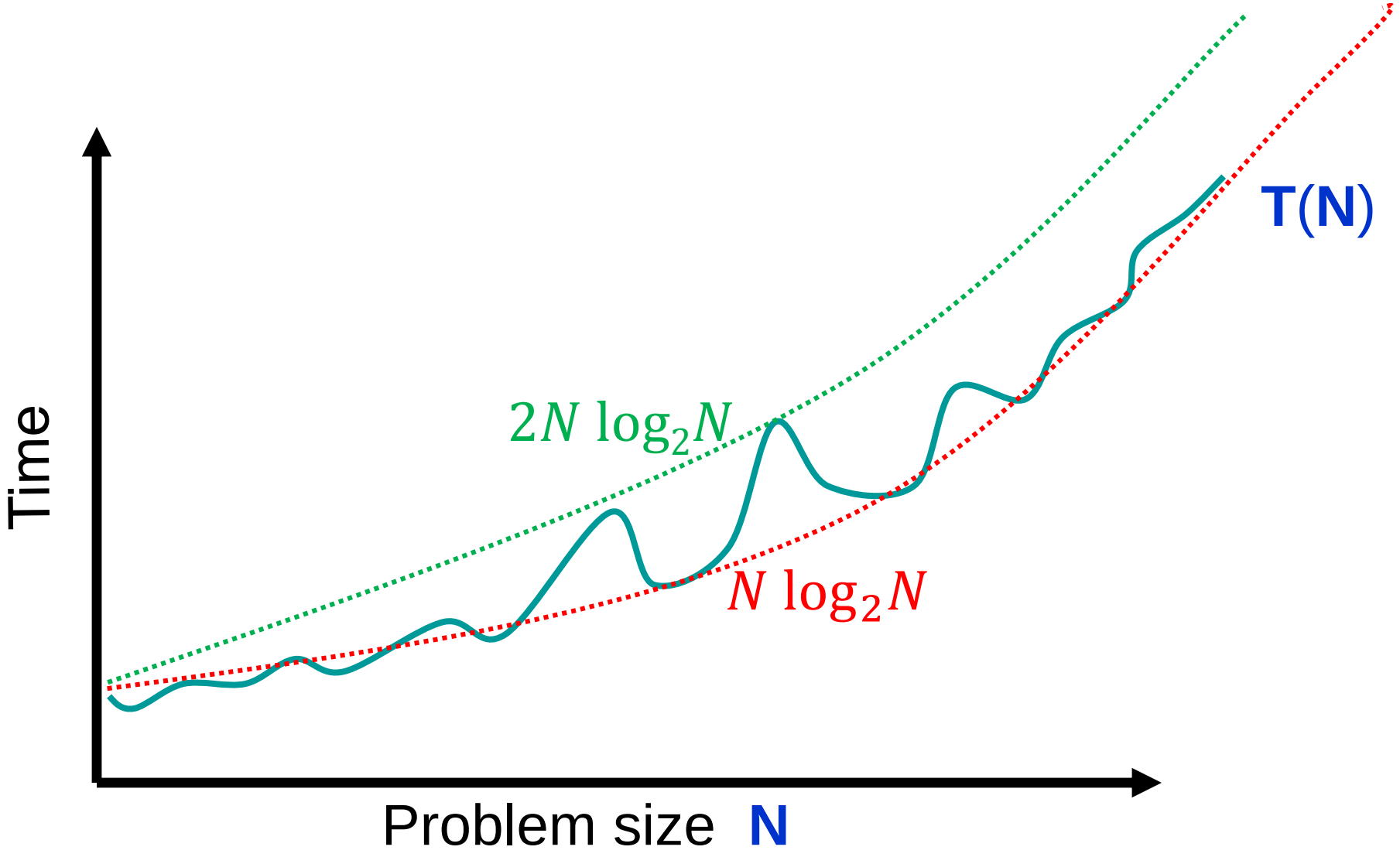
Definition of 1 step in this course:

- only simple operations (+,*,-,=,if,call,…).
- each operation takes one time step.
- each memory access takes one time step.
- no fancy stuff (add two matrices, copy long string,…).

Definition of efficiency in this course:

An algorithm is efficient if it has polynomial worst case runtime.

Time Complexity on Worst Case Inputs



O-Notation

Given two positive functions f and g

- $f(n) = O(g(n))$ if there is a constant $C > 0$ and N st
$$f(n) \leq Cg(n) \text{ for all } n > N$$

- $f(n) = \Omega(g(n))$ if there is a constant $C > 0$ and N st
$$f(n) \geq Cg(n) \text{ for all } n > N$$

- $f(n) = \Theta(g(n))$ if $f(n) = \mathrm{O}(g(n))$ and $f(n) = \Omega(g(n))$, namely

There is a constant $C_1, C_2 > 0$ and N st
$$C_1 g(n) \leq f(n) \leq C_2 g(n) \text{ for all } n > N$$

Common Asymptotic Bounds

- Polynomials:

  $a_0 + a_1 n + \cdots + a_d n^d$ is $O(n^d)$

- Logarithms:

  $\log_a n = O(\log_b n)$ for all constants $a, b > 0$

- Logarithms: log grows slower than every polynomial

  For all $x > 0$, $\log n = O(n^x)$

- $n \log n = O(n^{1.01})$

Running Time

An algorithm runs in polynomial time if $T(n) = n^{O(1)}$.

Equivalently, $T(n) = O(n^d)$ for some constant d.

Name	Complexity class	Running time ($T(n)$)	Examples of running times	Example algorithms
constant time		$O(1)$	10	Determining if an integer (represented in binary) is even or odd
inverse Ackermann time		$O(\alpha(n))$		Amortized time per operation using a disjoint set
iterated logarithmic time		$O(\log^* n)$		Distributed coloring of cycles
log-logarithmic		$O(\log \log n)$		Amortized time per operation using a bounded priority queue[2]
logarithmic time	DLOGTIME	$O(\log n)$	$\log n$, $\log(n^2)$	Binary search
polylogarithmic time		$\text{poly}(\log n)$	$(\log n)^2$	
fractional power		$O(n^c)$ where $0 < c < 1$	$n^{1/2}$, $n^{2/3}$	Searching in a kd-tree
linear time		$O(n)$	n	Finding the smallest or largest item in an unsorted array
"n log star n" time		$O(n \log^* n)$		Seidel's polygon triangulation algorithm.
quasilinear time		$O(n \log n)$	$n \log n$, $\log n!$	Fastest possible comparison sort; Fast Fourier transform.
quadratic time		$O(n^2)$	n^2	Bubble sort; Insertion sort; Direct convolution Stable matching!
cubic time		$O(n^3)$	n^3	Naive multiplication of two $n \times n$ matrices. Calculating partial correlation.
polynomial time	P	$2^{O(\log n)} = \text{poly}(n)$	n, $n \log n$, n^{10}	Karmarkar's algorithm for linear programming; AKS primality test
quasi-polynomial time	QP	$2^{\text{poly}(\log n)}$	$n^{\log \log n}$, $n^{\log n}$	Best-known $O(\log^2 n)$-approximation algorithm for the directed Steiner tree problem.
sub-exponential time (first definition)	SUBEXP	$O(2^{n^\varepsilon})$ for all $\varepsilon > 0$	$O(2^{\log n^{\log \log n}})$	Assuming complexity theoretic conjectures, BPP is contained in SUBEXP.[3]
sub-exponential time (second definition)		$2^{o(n)}$	$2^{n^{1/3}}$	Best-known algorithm for integer factorization and graph isomorphism
exponential time (with linear exponent)	E	$2^{O(n)}$	1.1^n, 10^n	Solving the traveling salesman problem using dynamic programming
exponential time	EXPTIME	$2^{\text{poly}(n)}$	2^n, 2^{n^2}	Solving matrix chain multiplication via brute-force search
factorial time		$O(n!)$	$n!$	Solving the traveling salesman problem via brute-force search
double exponential time	2-EXPTIME	$2^{2^{\text{poly}(n)}}$	2^{2^n}	Deciding the truth of a given statement in Presburger arithmetic

Why it matters?

Suppose we can do 1 million operations per second.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

not only get very big, but do so *abruptly*, which likely yields erratic performance on small instances

Update: RTX 2080 Ti do 13.4 Tflops/sec (1.3×10^{13})
It brings down the 31,710 years to 20 hour.
However, 2^{100} operations still takes billions of years.

Why “Polynomial”?

Point is not that n^{2000} is a practical bound, or that the differences among n and $2n$ and n^2 are negligible.

Rather, simple theoretical tools may not easily capture such differences, whereas exponentials are qualitatively different from polynomials, so more amenable to theoretical analysis.

- “My problem is in P” is a starting point for a more detailed analysis
- “My problem is not in P” may suggest that you need to shift to a more tractable variant

This symbols means out of syllabus.

Other Complexities

Average Case Complexity:

avg # steps algorithm takes

Communication Complexity:

max # communication algorithm send between servers

Space Complexity:

max # space algorithm needs

Parallel Complexity:

max length of the longest series of operations that have to be performed sequentially due to data dependencies

Energy Complexity: …

Landauer's principle: Take at least 10^{-21} J to erase a bit in room temp.

(Current specialized hardware takes around 10^{-15} J.)

In Class Exercise

Arrange in the increasing order of asymptotic growth:

- $n^{5/3}$.
- $n^{\sqrt{n}\log n}$.
- $n \log^{11} n$.
- $n\, 2^{\sqrt{\frac{\log n}{\log \log n}}}$.
- n^3.

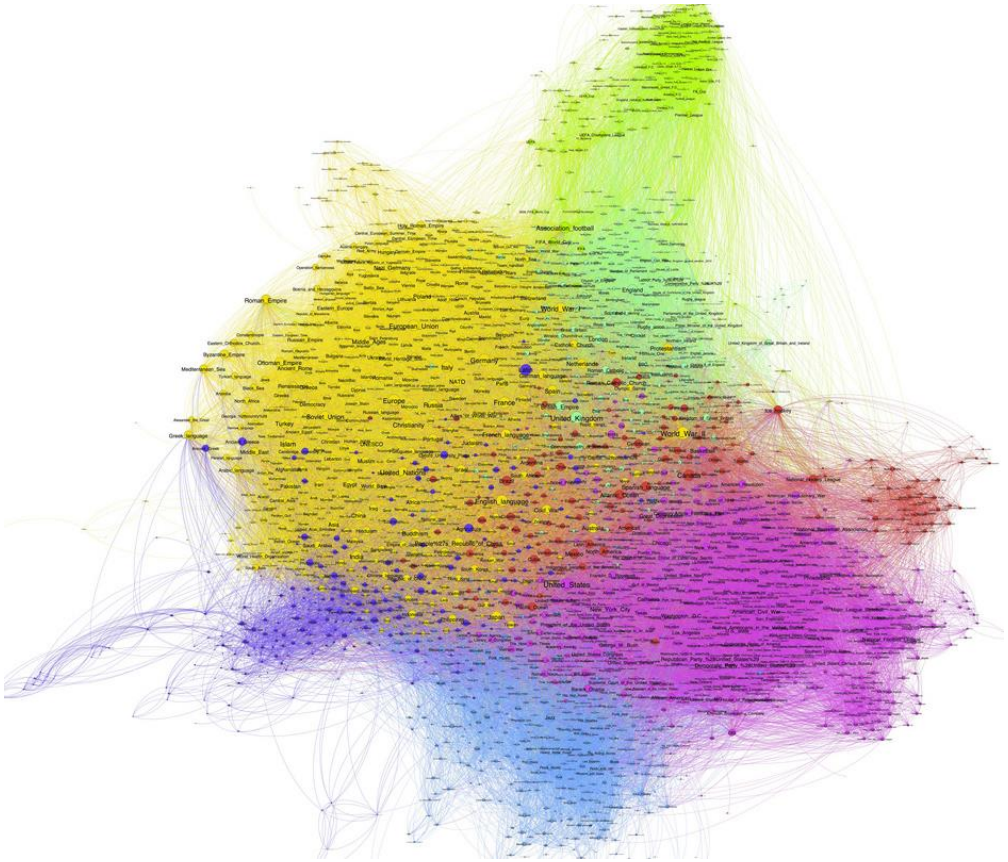
Different ways to solve maxflow.
Will explain one of them this quarter.

CSE 421: Introduction to Algorithms

Graph

Yin-Tat Lee

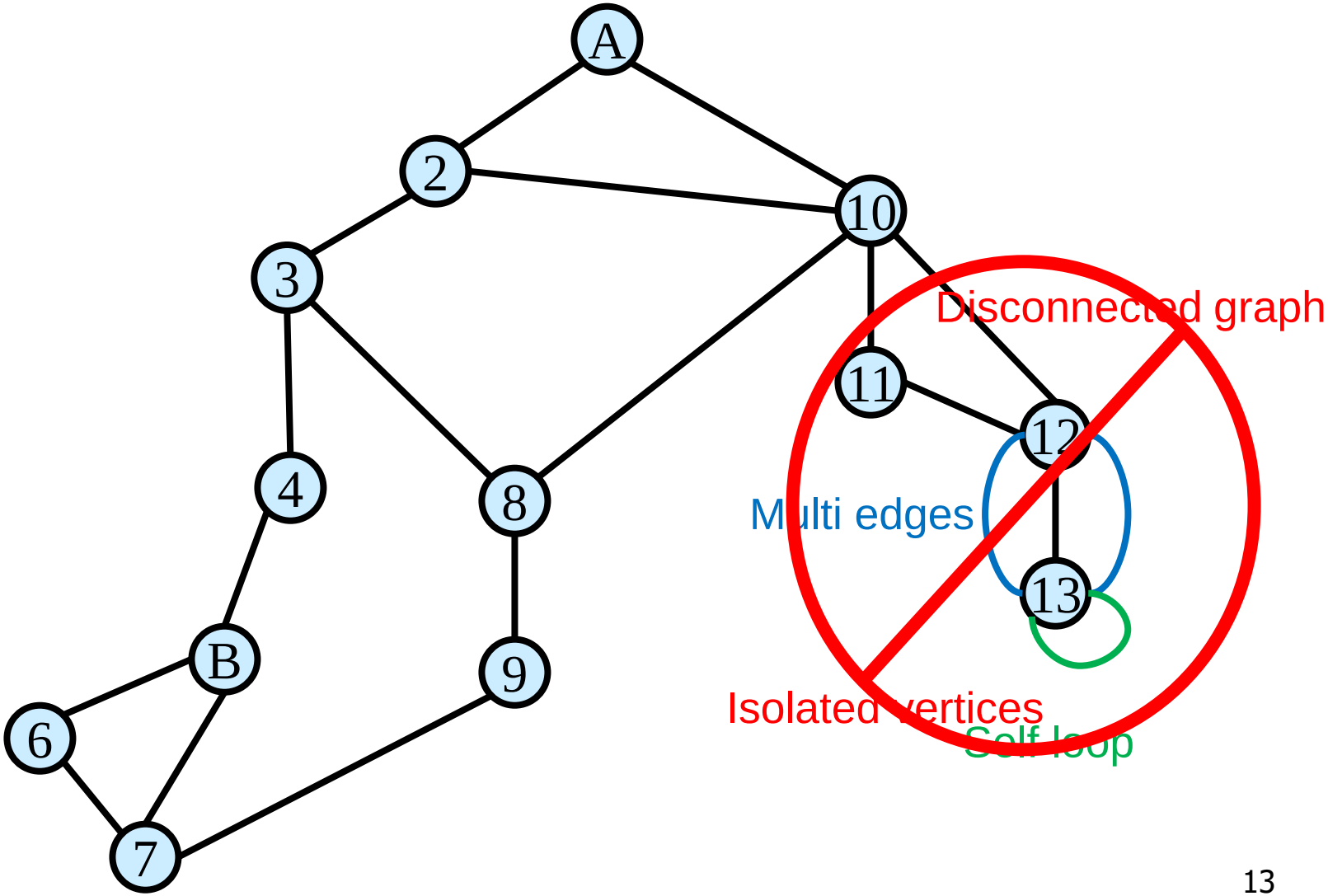
Graphs



Examples:

- Transportation networks
- Communication networks
- Internet
- Social networks
- Dependency networks

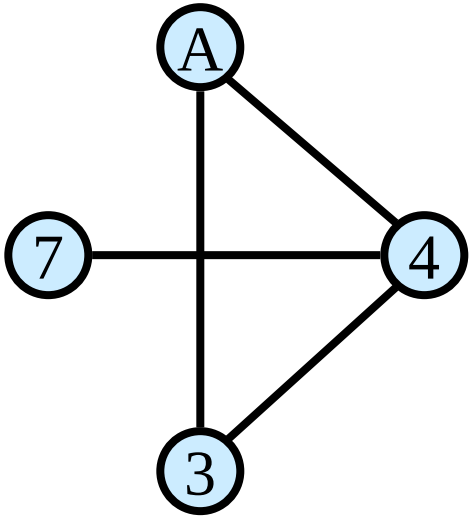
Undirected Graphs G=(V,E)

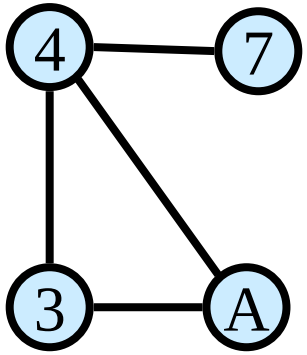


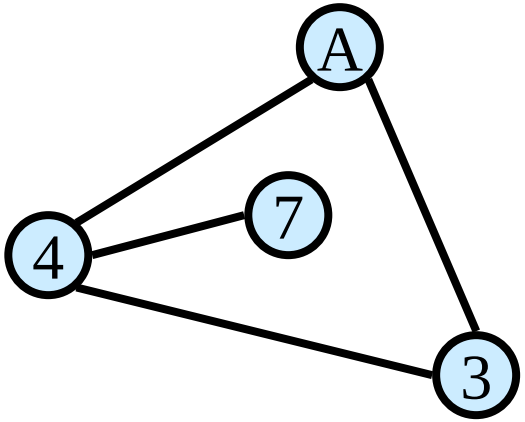
Graphs don’t Live in Flat Land

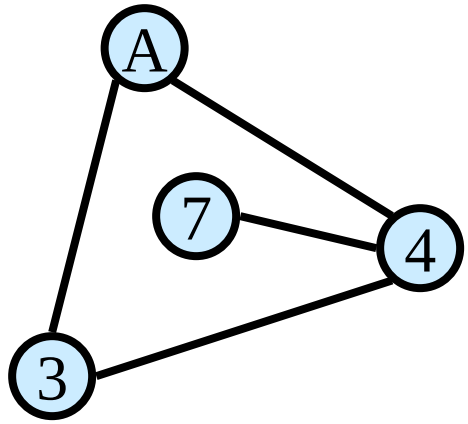
Geometrical drawing is mentally convenient, but mathematically irrelevant:

4 drawings of a single graph:

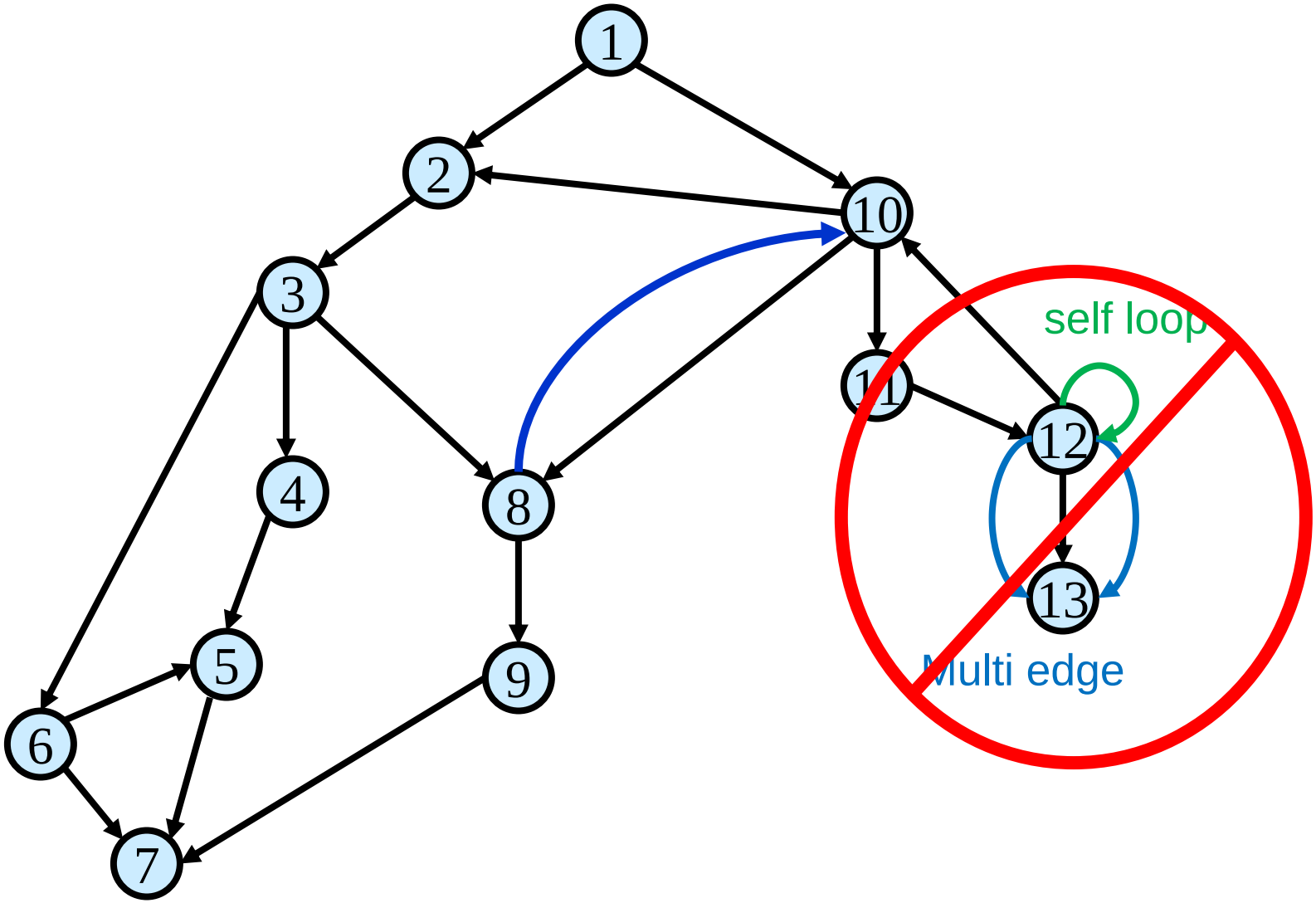








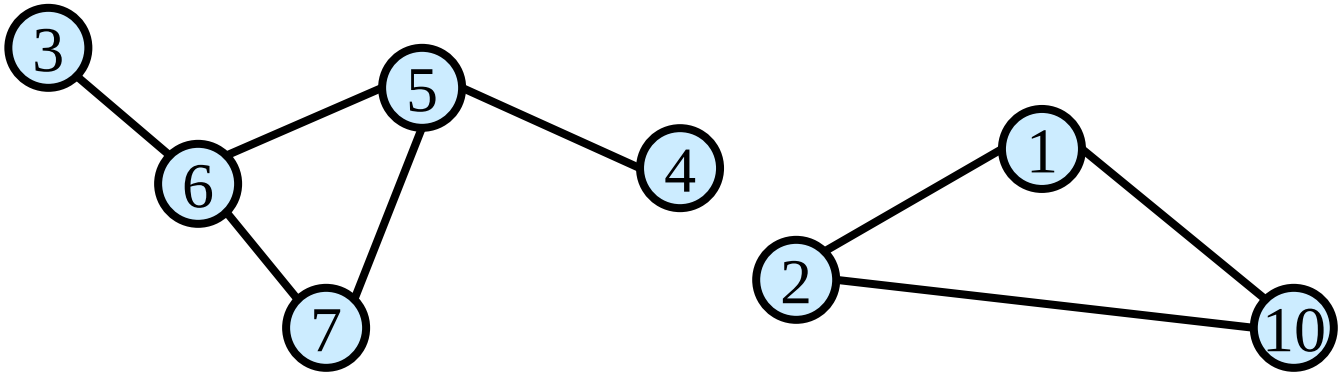
Directed Graphs



Terminology

- Degree of a vertex: # edges that touch that vertex

deg(6)=3



- Connected: Graph is connected if there is a path between every two vertices

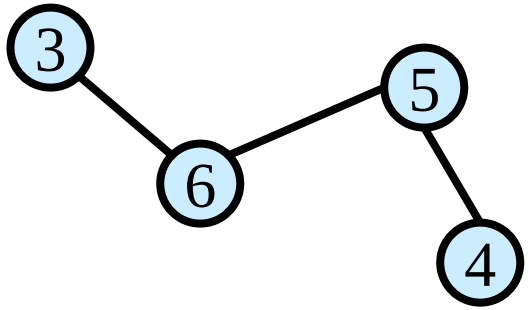
- Connected component: Maximal set of connected vertices

Terminology (cont'd)

- Path: A sequence of distinct vertices

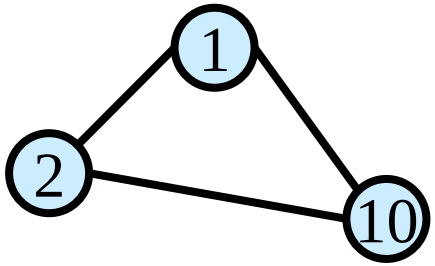
s.t. each vertex is connected

to the next vertex with an edge

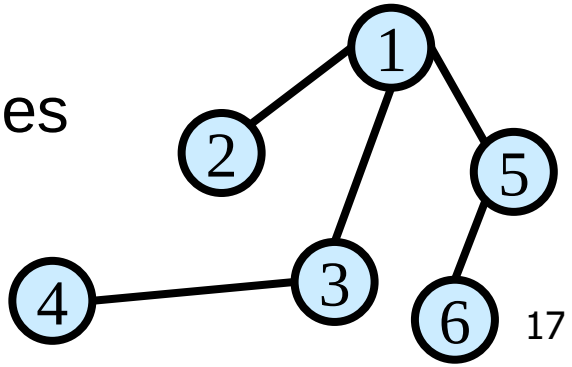


- Cycle: Path of length > 2 that has

the same start and end



- Tree: A connected graph with no cycles



Degree 1 vertices

Claim: If G has no cycle, then it has a vertex of degree ≤ 1

(Every tree has a leaf)

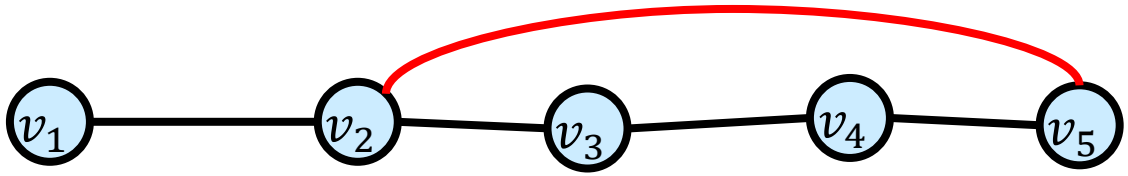
Proof: (By contradiction)

Suppose every vertex has degree ≥ 2.

Start from a vertex v_1 and follow a path, $v_1, \ldots, v_i$ when we are at v_i we choose the next vertex to be different from v_{i-1}. We can do so because $\deg(v_i) \geq 2$.

The first time that we see a repeated vertex ($v_j = v_i$) we get a cycle.

We always get a repeated vertex because G has finitely many vertices



Trees and Induction

Claim: Show that every tree with n vertices has $n - 1$ edges.

Proof: (Induction on n.)

Base Case: $n = 1$, the tree has no edge

Inductive Step: Let T be a tree with n vertices.

So, T has a vertex v of degree 1.

Remove v and the neighboring edge, and let T' be the new graph.

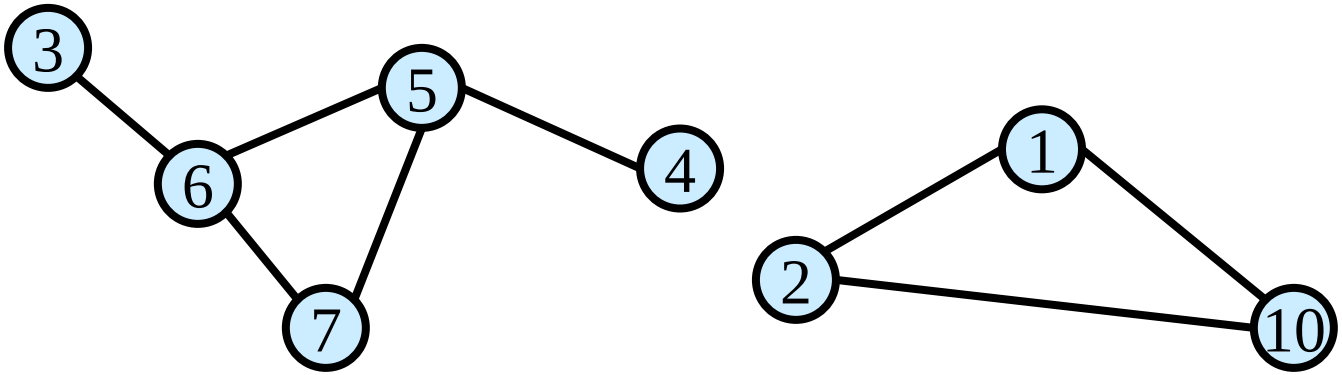
We claim T' is a tree: It has no cycle, and it must be connected.

So, T' has $n - 2$ edges and T has $n - 1$ edges.

Exercise: Degree Sum

Claim: In any undirected graph, the number of edges is equal to $(1/2)\sum_{\text{vertex } v} \deg(v)$

Pf: $\sum_{\text{vertex } v} \deg(v)$ counts every edge of the graph exactly twice; once from each end of the edge.



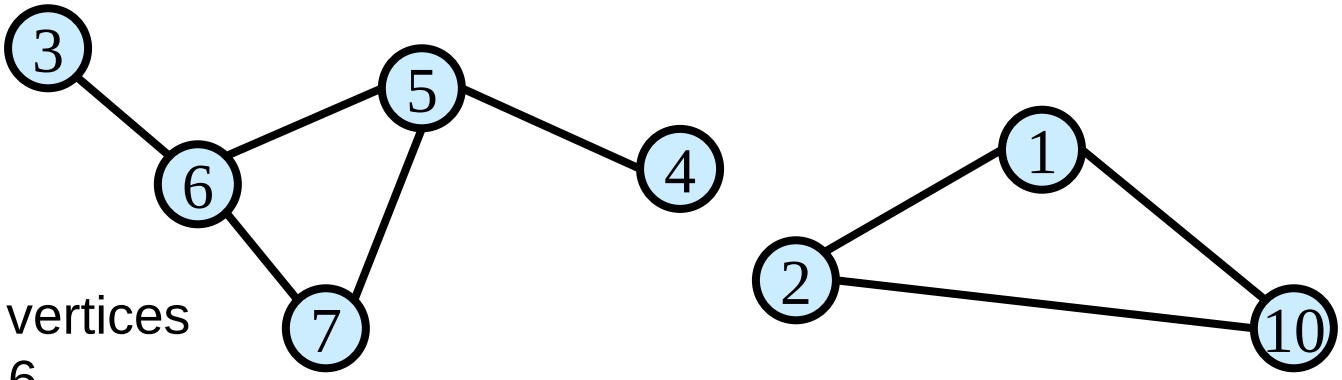
|E|=8

$$\sum_{\text{vertex } v} \deg(v) = 2 + 2 + 1 + 1 + 3 + 2 + 3 + 2 = 16$$

Exercise: Odd Degree Vertices

Claim: In any undirected graph, the number of odd degree vertices is even

Pf: In previous claim we showed sum of all vertex degrees is even. So there must be even number of odd degree vertices, because sum of odd number of odd numbers is odd.



4 odd degree vertices
3, 4, 5, 6

#edges

Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges.

Claim: $0 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$

Pf: Since every edge connects two distinct vertices (i.e., G has no loops)

and no two edges connect the same pair of vertices (i.e., G has no multi-edges)

It has at most $\binom{n}{2}$ edges.

Sparse Graphs

A graph is called sparse if $m \ll n^2$ and it is called dense otherwise.

Sparse graphs are very common in practice

- Friendships in social network
- Planar graphs
- Web graph

Q: Which is a better running time $O(n+m)$ vs $O(n^2)$?

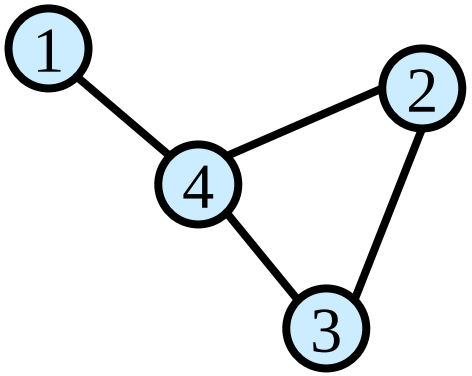
A: $O(n+m) = O(n^2)$, but $O(n+m)$ is usually much better.

Storing Graphs (Internally in ALG)

Vertex set $V = \{v_1, \ldots, v_n\}$.

Adjacency Matrix: A

- For all, $i, j, A[i,j] = 1$ iff $(v_i, v_j) \in E$
- Storage: n^2 bits



	1	2	3	4
1	0	0	0	1
2	0	0	1	1
3	0	1	0	1
4	1	1	1	0

Advantage:

- $O(1)$ test for presence or absence of edges

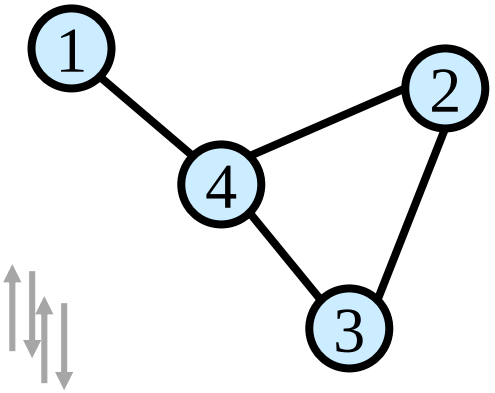
Disadvantage:

- Inefficient for sparse graphs both in storage and edge-access

Storing Graphs (Internally in ALG)

Adjacency List:

O(n+m) words

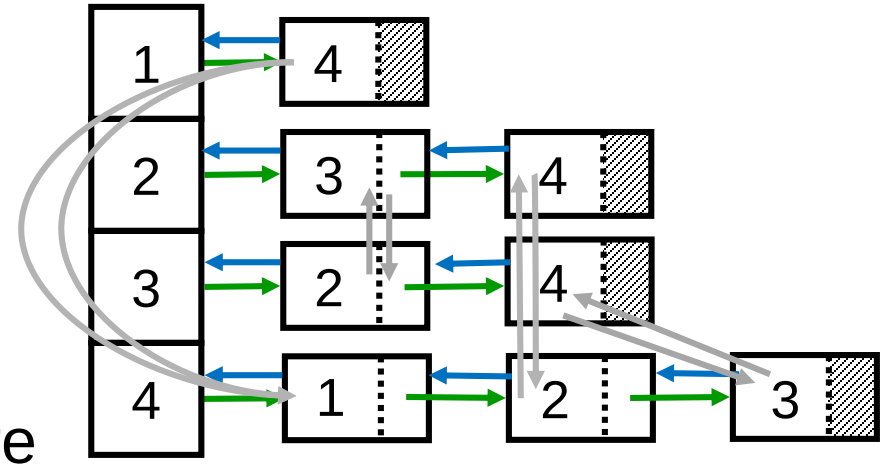


Advantage

- Compact for sparse
- Easily see all edges

Disadvantage

- No O(1) edge test
- More complex data structure



If $O(1)$ edge test is needed, one can use a hash table.