CSE 421: Introduction to Algorithms

Induction - Graphs

Shayan Oveis Gharan

O-Notation

Given two positive functions **f** and **g**

$\leq$

- **f(N)** is **O(g(N))** iff there is a constant **c**>**0** s.t.,

  f(N) is eventually always $\leq$ **c g(N)**

$\geq$

- **f(N)** is **Ω(g(N))** iff there is a constant **ε**>**0** s.t.,

  f(N) is $\geq$ **ε g(N)** for infinitely

  if $f(N) = \frac{1}{1000} N^2 \Rightarrow \Omega(N^2)$

$\approx$

- **f(N)** is **Θ(g(N))** iff there are constants c_1, c_2>0 so that

  eventually always $c_1 g(N) \leq f(N) \leq c_2 g(N)$

Asymptotic Bounds for common fns

- Polynomials:

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

- Logarithms:

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

  $= \frac{\lg_2 n}{\lg_2 a}$

- Logarithms: log grows slower than every polynomial

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

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

  $= n \cdot n^{0.01}$

Efficient = Polynomial Time

An algorithm runs in polynomial time if $T(n)=O(n^d)$ for some constant d independent of the input size n.

Why Polynomial time?

If problem size grows by at most a constant factor then so does the running time

- E.g. $\mathbf{T(2N) \leq c(2N)^k \leq 2^k(cN^k)}$
- Polynomial-time is exactly the set of running times that have this property

Typical running times are small degree polynomials, mostly less than $\mathbf{N^3}$, at worst $\mathbf{N^6}$, not $\mathbf{N^{100}}$

Why it matters?

ALG runs 2^n

- #atoms in universe < 2^{240}
- Life of the universe < 2^{54} seconds
- A CPU does < 2^{30} operations a second

$2^{240} \cdot 2^{54} \cdot 2^{30} \simeq 2^{320}$

If every atom is a CPU, a 2^n time ALG cannot solve n=350 if we start at Big-Bang.

	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

Why “Polynomial”?

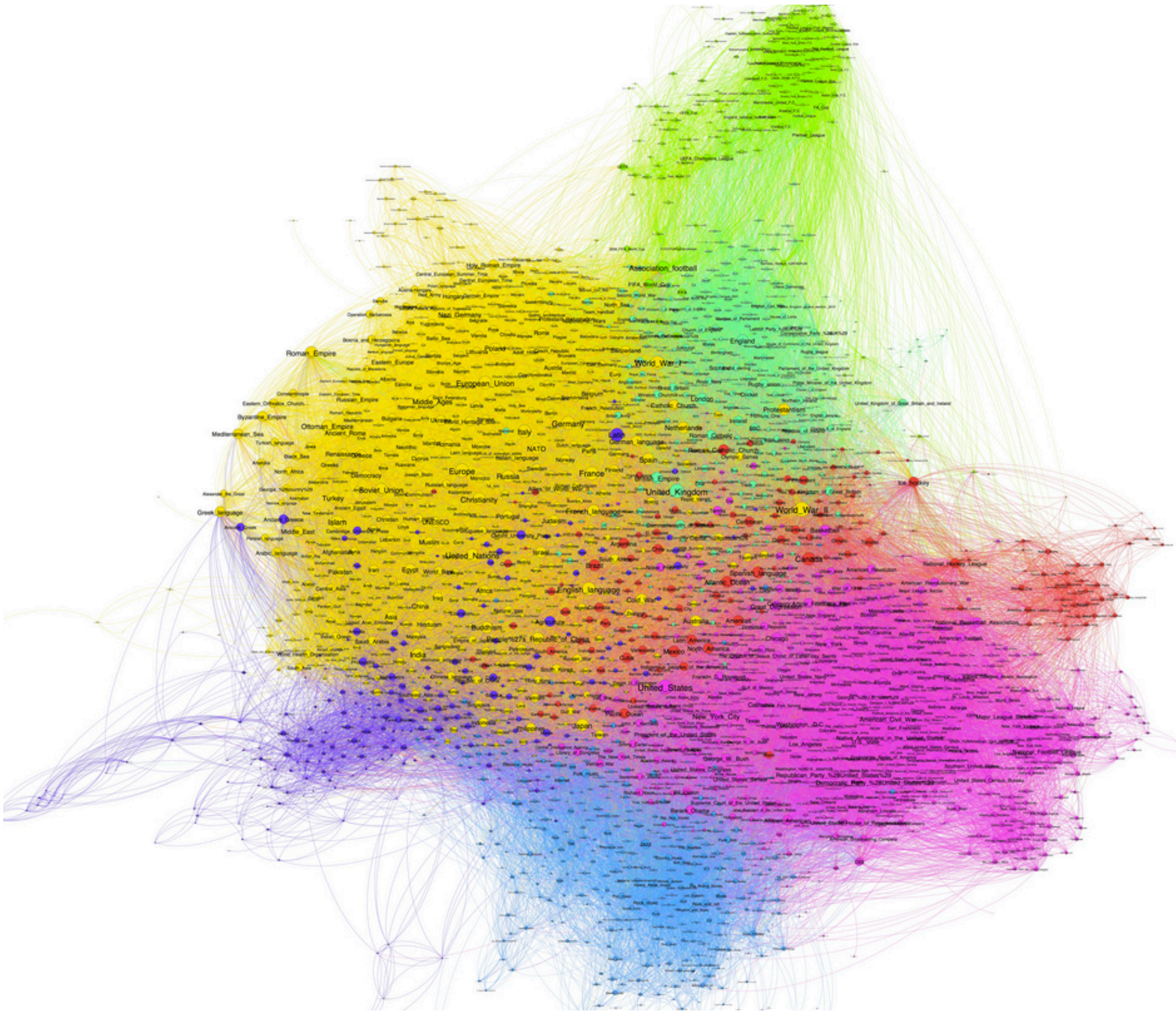
Point is not that n^{2000} is a practical bound, or that the differences among n and 2n and n^2 are negligible.

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

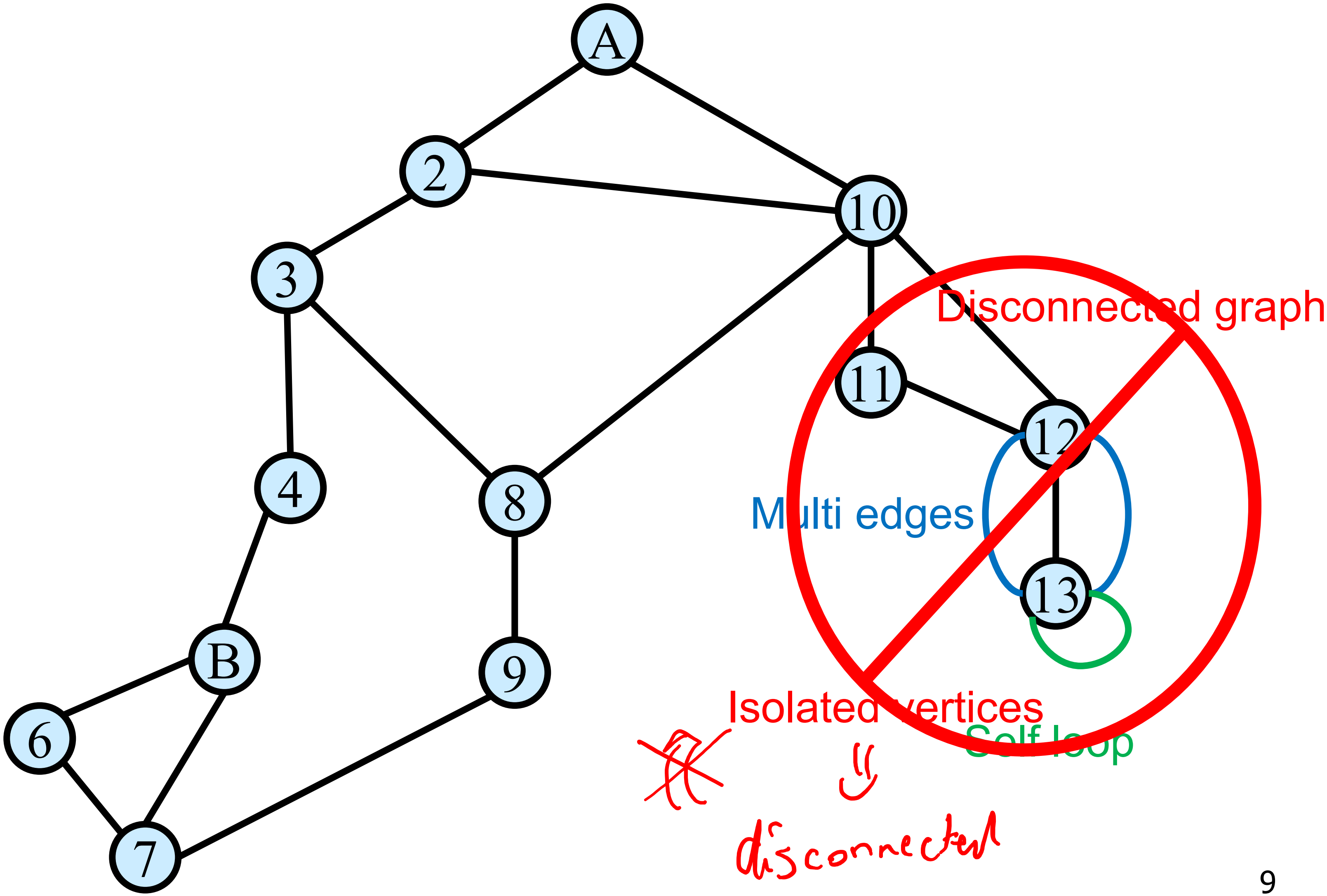
- “My problem is in P” is a starting point for a more detailed analysis
- “My problem is not in P” may suggest that you need to shift to a more tractable variant

Graphs

Graphs

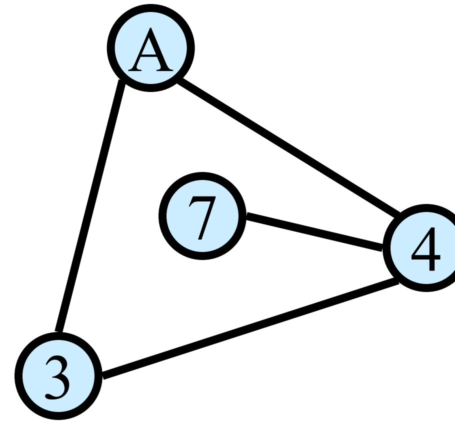

Association_football
Roman_Empire
European_Union
World_War_I
Germany
United_Kingdom
World_War_II
Canada
English_language
United_Nations
United_States

Undirected Graphs G=(V,E)

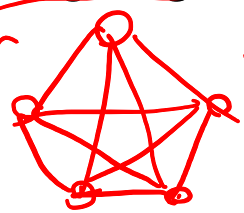


Graphs don't Live in Flat Land

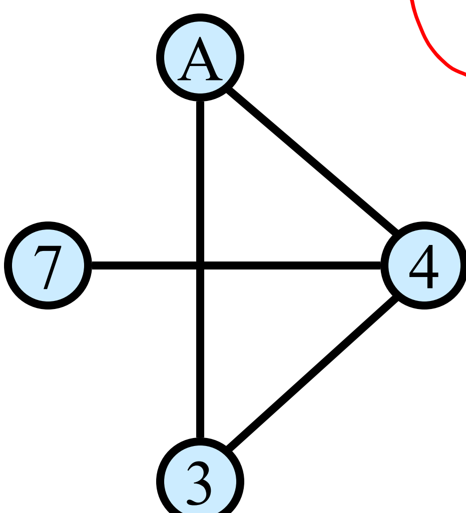
Geometrical drawing is mentally convenient, but mathematically irrelevant:

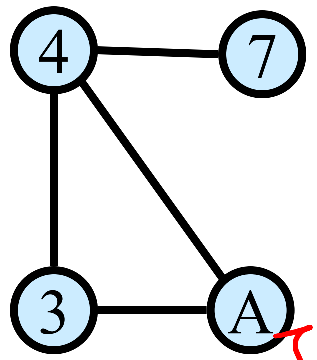


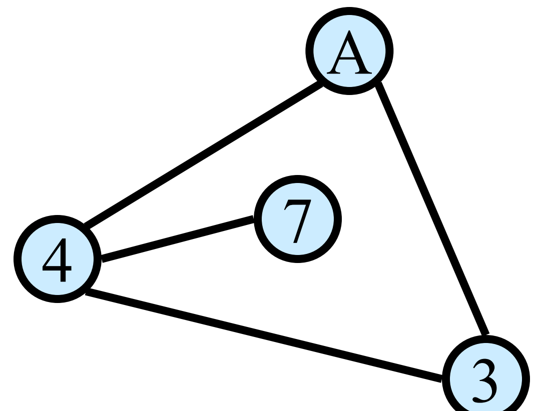
4 drawings of a single graph:



not planar

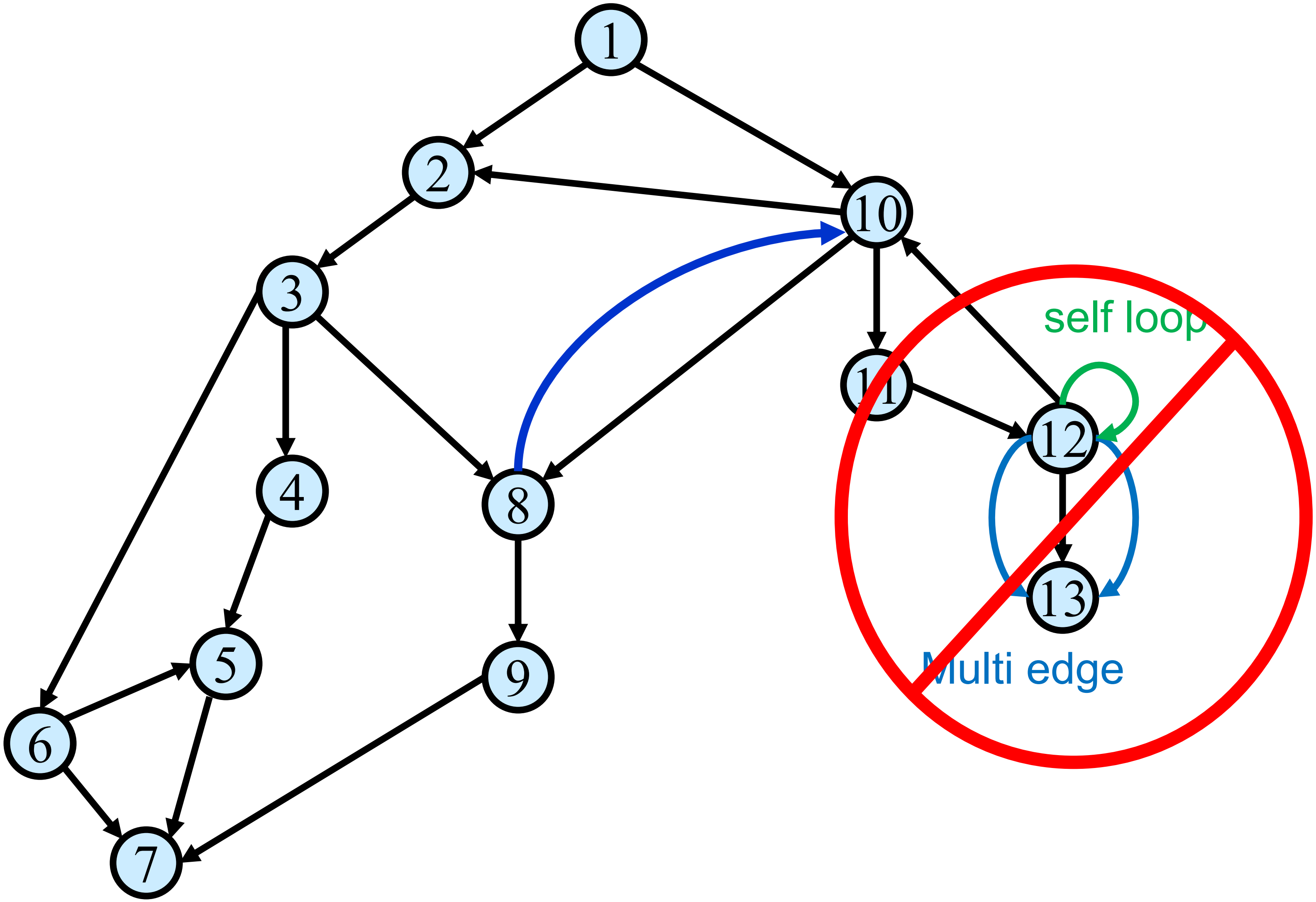






planar = $\exists$ drawing that no two edges cross

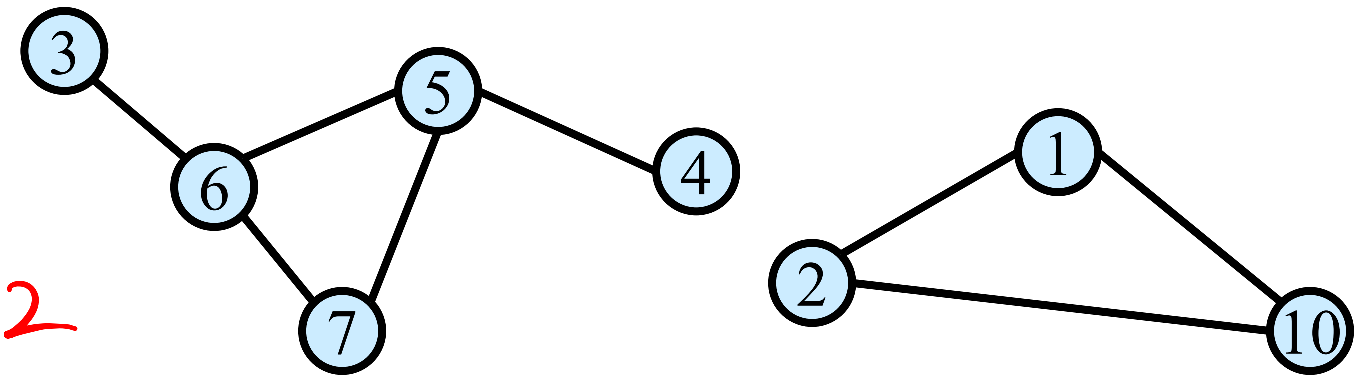
Directed Graphs



Terminology

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

incident

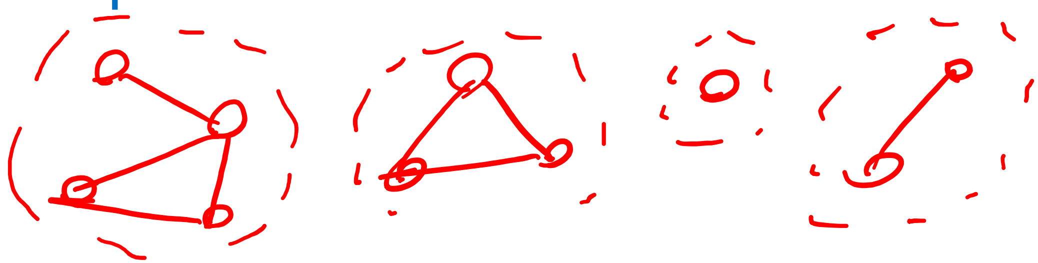


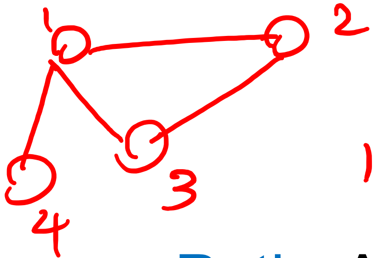
deg(6)=3

deg(10)=2

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

- Connected component: Maximal set of connected vertices

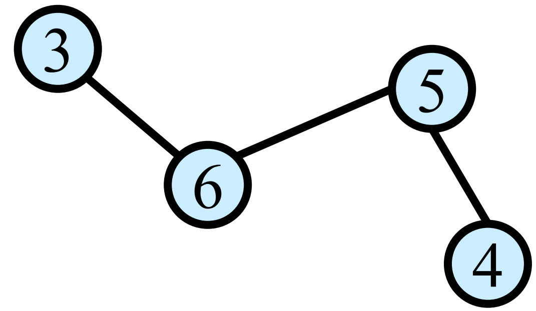




Terminology (cont'd)

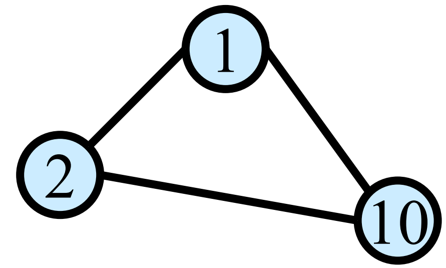
1, 2, 3, 1, 4 not a path

- Path: A sequence of distinct vertices s.t. each vertex is connected to the next vertex with an edge



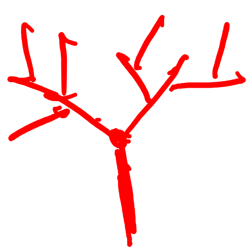
3, 6, 5, 4

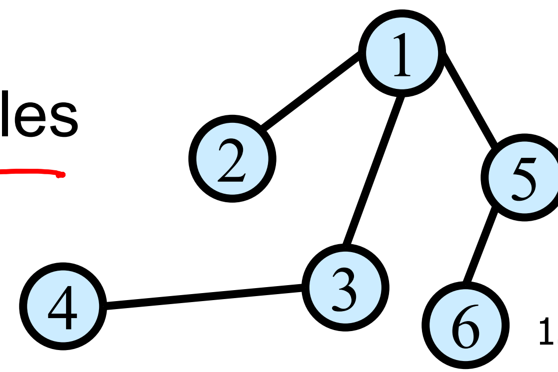
- Cycle: Path of length > 2 that has the same start and end



1, 2, 10, 1

- Tree: A connected graph with no cycles

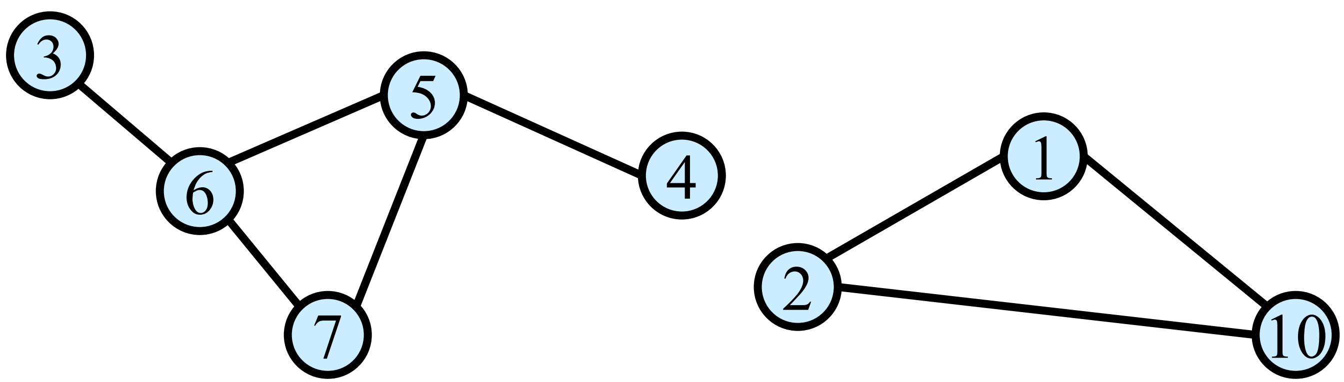




Degree Sum

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

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

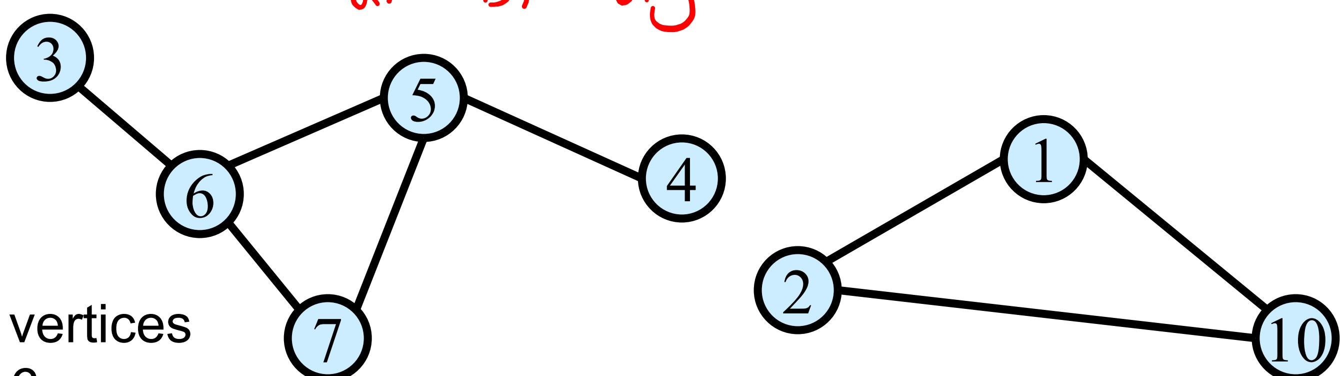
Odd Degree Vertices

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.

sum of odd number of odd number = odd

sum of deg = even



4 odd degree vertices
3, 4, 5, 6

Degree 1 vertices

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

(So, every tree has a leaf) = vertex of deg exactly 1

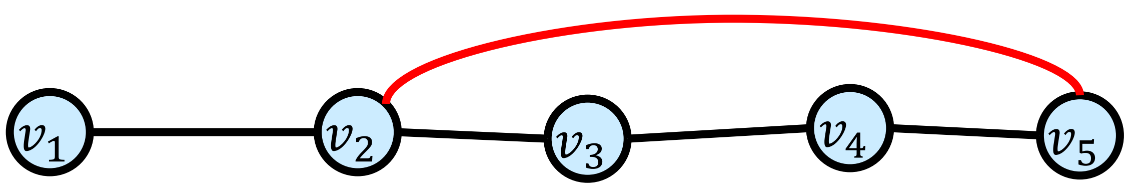
Pf: (By contradiction)

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.

Pf: By induction.

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

IH: Suppose every tree with n-1 vertices has n-2 edges

IS: 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.