# CSE 421: Introduction to Algorithms

## Induction - Graphs

Shayan Oveis Gharan

# Maximum Consecutive Subsequence

Problem: Given a sequence $x_1, \ldots, x_n$ of integers (not necessarily positive),

Goal: Find a subsequence of consecutive elements s.t., the sum of its numbers is maximum.

$$1 \quad -3 \quad \boxed{7 \quad -2 \quad -3 \quad 8} \quad -10 \quad 1 \quad -7$$

# Second Attempt (Strengthing Ind Hyp)

Stronger Ind Hypothesis: Given $x_1, \ldots, x_{n-1}$ we can compute the maximum-sum subsequence, and the maximum-sum suffix subsequence.

$$-3,\ \boxed{7,\ -2,\ 1,}\ -8,\ \boxed{6,\ -2}$$

$$x_i \qquad x_j \qquad x_k \qquad x_{n-1}$$

Say $x_i, \ldots, x_j$ is the maximum-sum and $x_k, \ldots, x_{n-1}$ is the maximum-sum suffix subsequences.

- If $x_k + \cdots + x_{n-1} + x_n > x_i + \cdots + x_j$ then $x_k, \ldots, x_n$ will be the new maximum-sum subsequence

# Updating Max Suffix Subsequence

-3, 7, -2, 1, -8, $\boxed{6, \ -2,}$ 4

$x_k$ $x_{n-1}$ $x_n$

Say $x_k, \ldots, x_{n-1}$ is the maximum-sum suffix subsequences of $x_1, \ldots, x_{n-1}$.

- If $x_k + \cdots + x_n \geq 0$ then,
  $x_k, \ldots, x_n$ is the new maximum-sum suffix subsequence

- Otherwise,
  The new maximum-sum suffix is the empty string.

# Maximum Sum Subsequence ALG

```
Initialize S=0 (Sum of numbers in Maximum Subseq)
Initialize U=0 (Sum of numbers in Maximum Suffix)
for  (i=1 to n) {
    if (x[i] + U > S)
        S = x[i] + U

    if (x[i] + U > 0)
        U = x[i] + U
    else
        U = 0
}
Output S.
```

# Pf of Correct: Maximum Sum Subseq

Ind Hypo: Suppose

- $x_i, \ldots, x_j$ is the max-sum-subseq of $x_1, \ldots, x_{n-1}$
- $x_k, \ldots, x_{n-1}$ is the max-suffix-sum-sub of $x_1, \ldots, x_{n-1}$

Ind Step: Suppose $x_a, \ldots, x_b$ is the max-sum-subseq of $x_1, \ldots, x_n$

Case 1 ($b < n$): $x_a, \ldots, x_b$ is also the max-sum-subseq of $x_1, \ldots, x_{n-1}$
So, $a = i, b = j$ and the algorithm correctly outputs OPT

Case 2 ($b = n$): We must have $x_a, \ldots, x_{b-1}$ is the max-suff-sum of $x_1, \ldots, x_{n-1}$.
If not, then

$$x_k + \cdots x_{n-1} > x_a + \cdots + x_{n-1}$$

So, $x_k + \cdots + x_n > x_a + \cdots + x_b$ which is a contradiction.
Therefore, $a = k$ and the algorithm correctly outputs OPT

Special Cases (You don't need to mention if follows from above):

- The max-suffix-sum is empty string
- There are multiple maximum sum subsequences.

# Pf of Correct: Max-Sum Suff Subseq

Ind Hypo: Suppose

- $x_k, \dots, x_{n-1}$ is the max-suffix-sum-sub of $x_1, \dots, x_{n-1}$

Ind Step: Suppose $x_a, \dots, x_n$ is the max-sum-subseq of $x_1, \dots, x_n$
Note that we may also have an empty sequence

Case 1 (OPT is empty): Then, we must have $x_k + \cdots + x_n < 0$. So the algorithm correctly finds max-suffix-sum subsequence.

Case 2 ($x_a, \dots, x_n$ is nonempty): We must have $x_a + \cdots + x_n \geq 0$.
Also, $x_a, \dots, x_{n-1}$ must be the max-suffix-sum of $x_1, \dots, x_{n-1}$. If not,

$$x_a + \cdots + x_{n-1} < x_k + \cdots + x_{n-1}$$

which implies $x_a + \cdots + x_n < x_k + \cdots + x\_n$ which is a contradiction.

Therefore, $a = k$. So, the algorithm correctly finds max-suffix-sum subseqence.

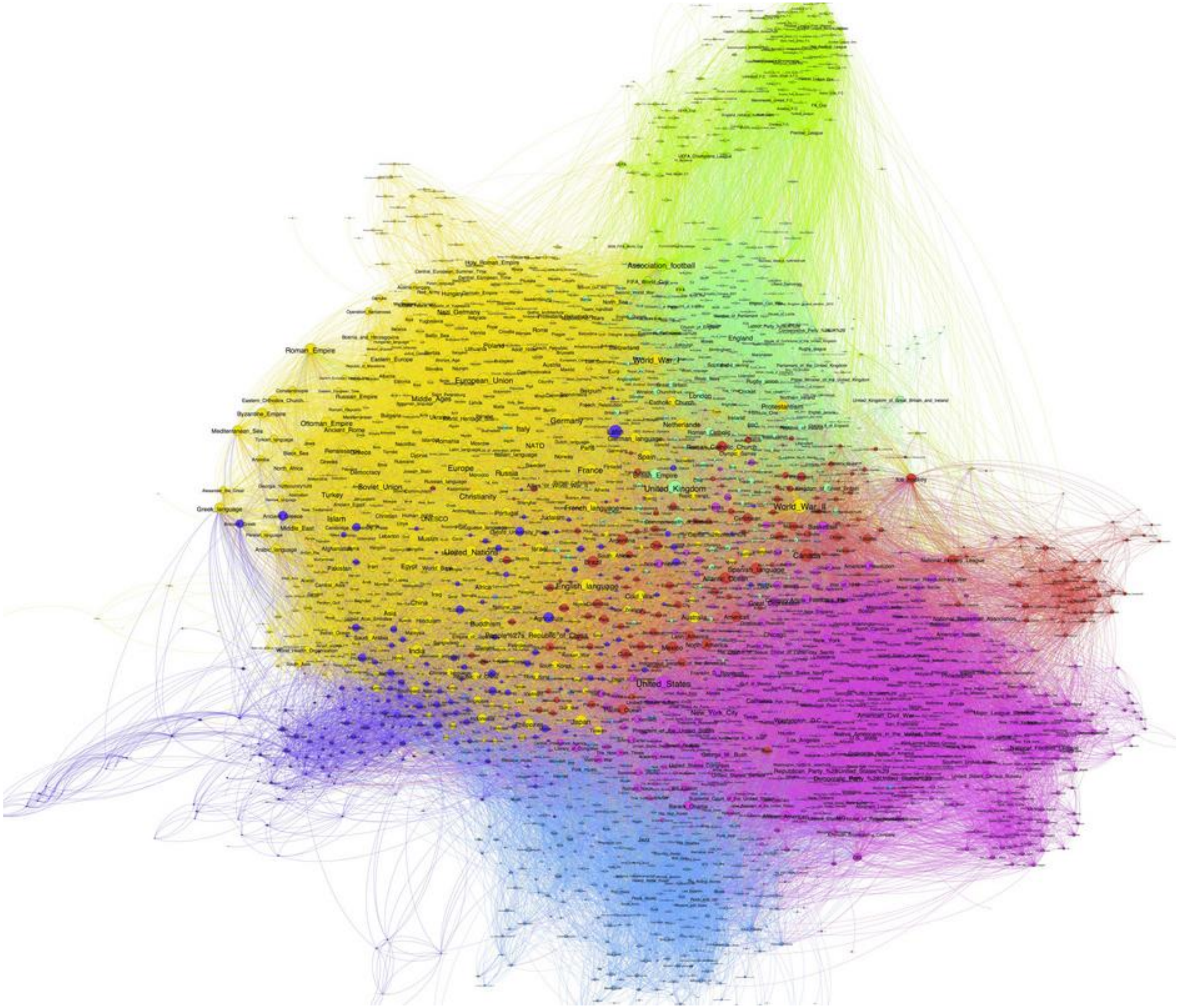
# Summary

- Try to reduce an instance of size n to smaller instances
  - Never solve a problem twice

- Before designing an algorithm study properties of optimum solution

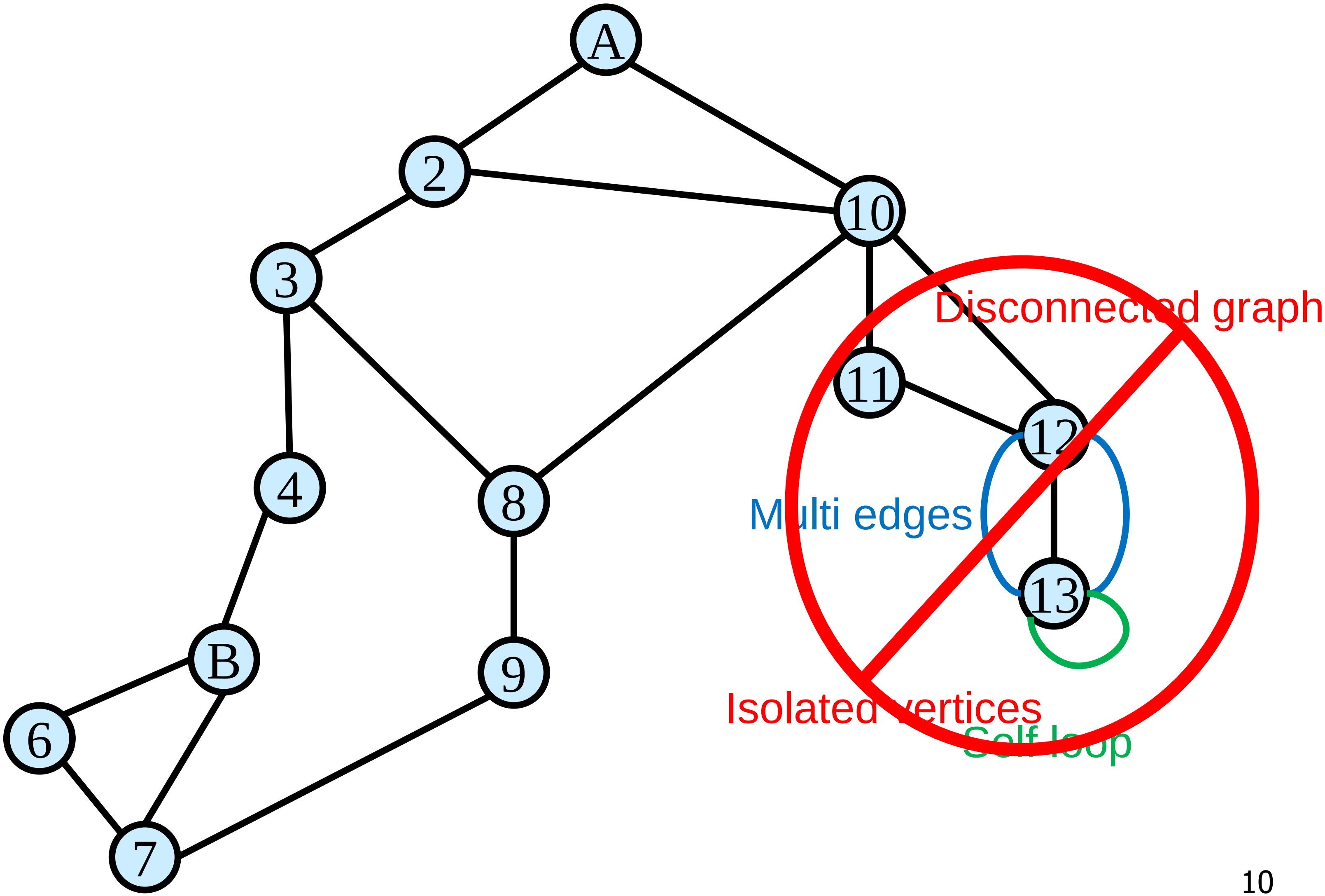
- If ordinary induction fails, you may need to strengthen the induction hypothesis

# Graphs



Association_football
Roman_Empire
European_Union
World_War_I
Germany
Europe
Russia
France
United_Kingdom
World_War_II
English_language
Canada
United_States
New_York_City

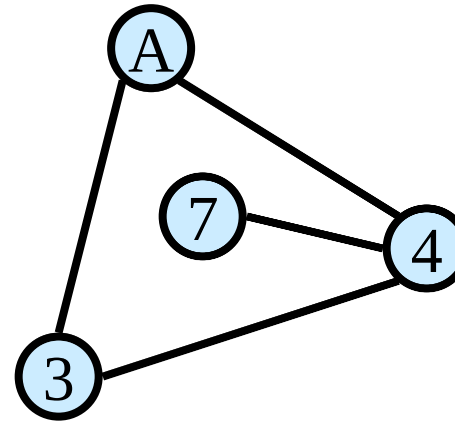
# Undirected Graphs G=(V,E)

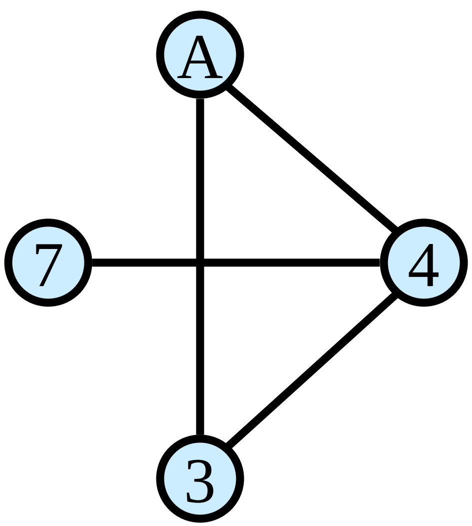


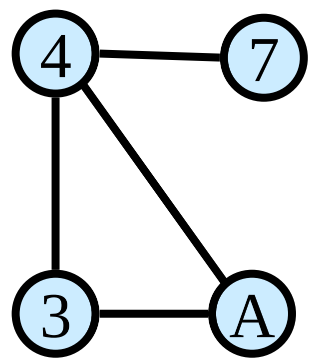
# Graphs don’t Live in Flat Land

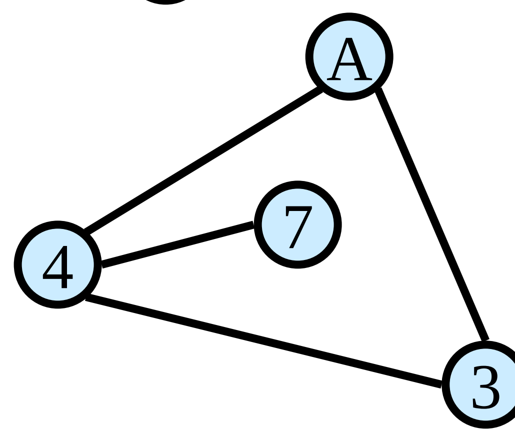
Geometrical drawing is mentally convenient, but mathematically irrelevant:

4 drawings of a single graph:

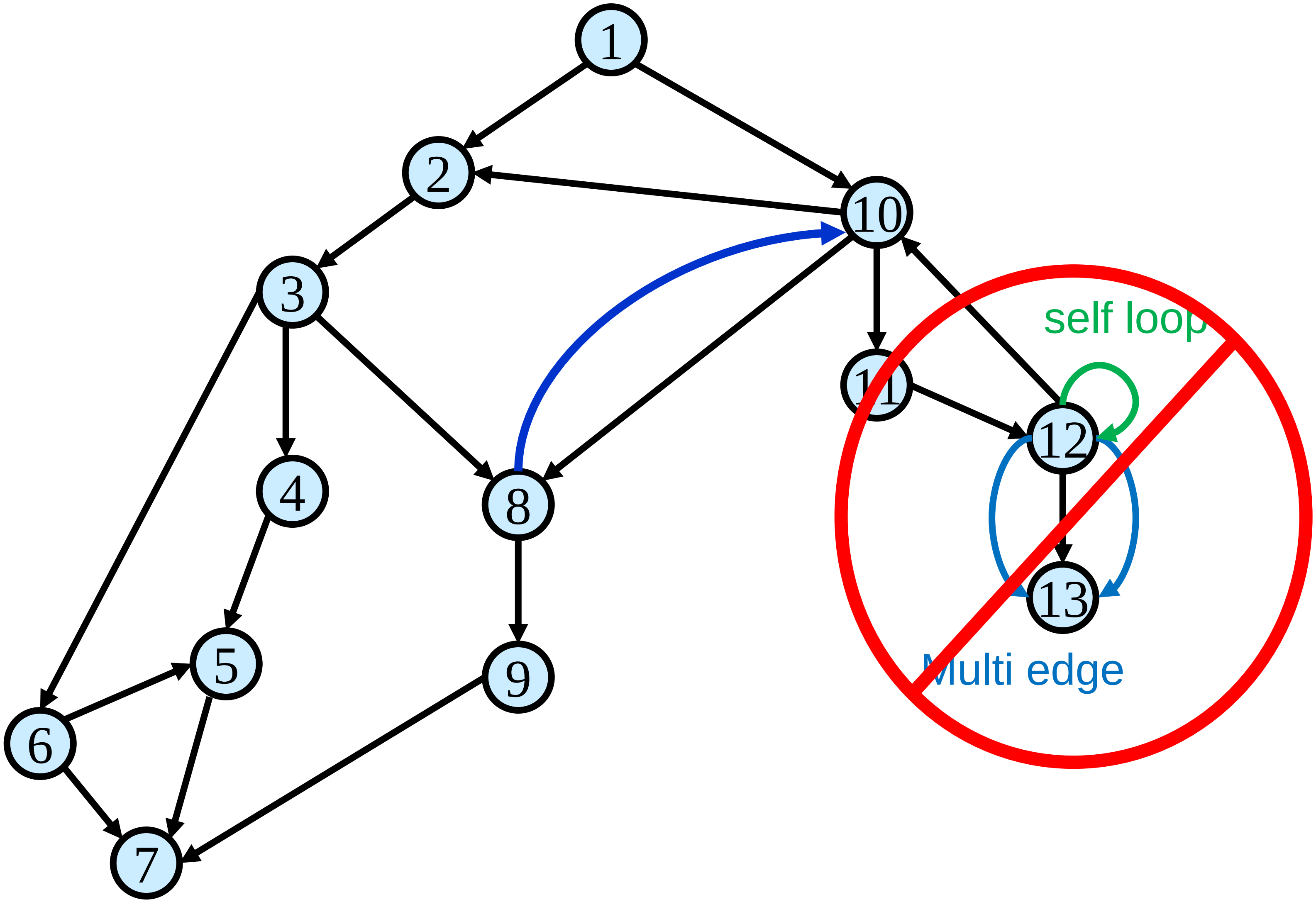


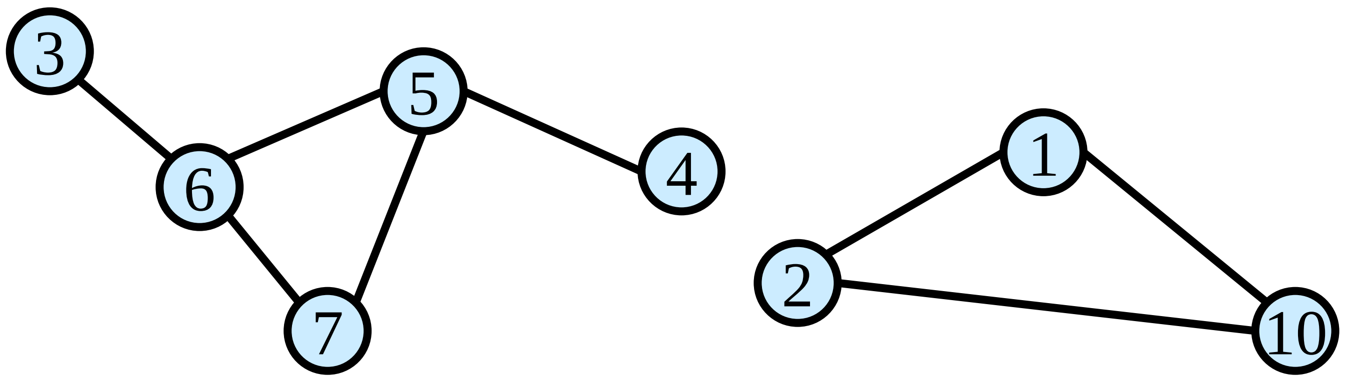



# Directed Graphs



# Terminology

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



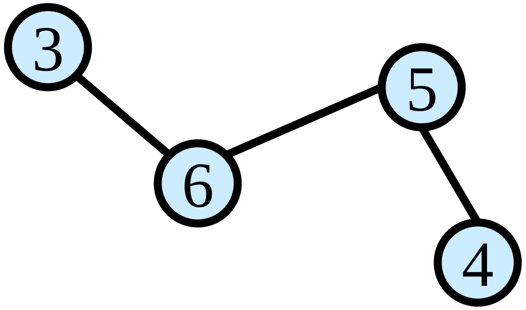

deg(6)=3

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

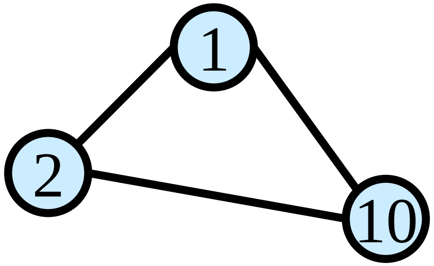
- Connected component: Maximal set of connected vertices

# Terminology (cont'd)

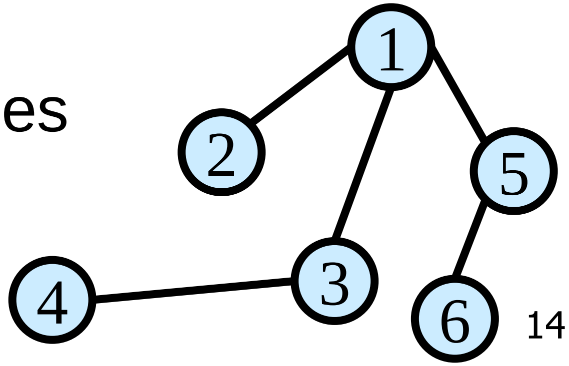
- Path: A sequence of distinct vertices s.t. each vertex is connected to the next vertex with an edge




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



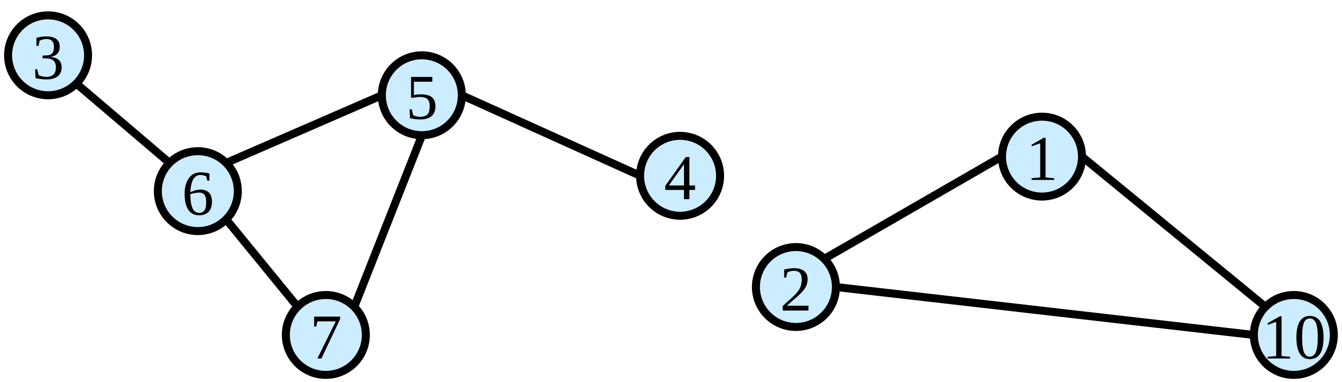

- Tree: A connected graph with no cycles



# Degree 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.



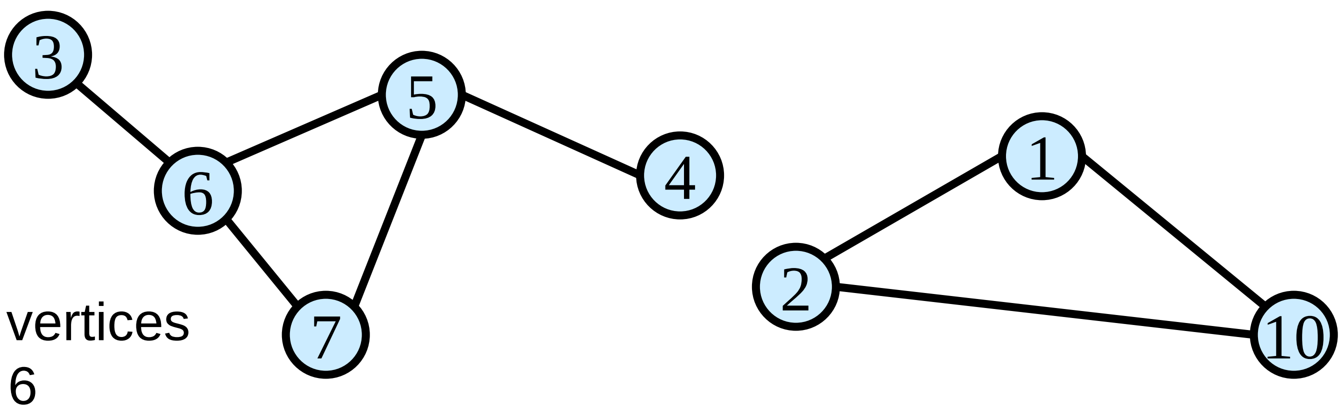

|E|=8

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

# 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.




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 braph

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

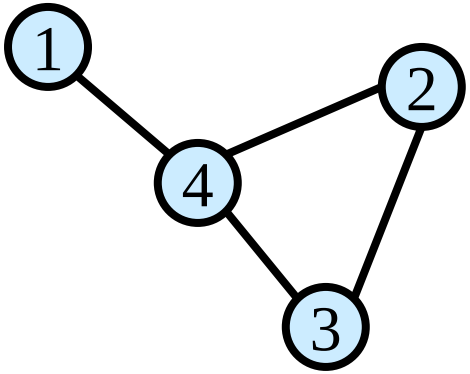
A: $O(n+m) = O(n^2)$, but $O(n+m)$ is usually much better.

# 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 $\left(v_i, v_j\right) \in E$
- 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 |

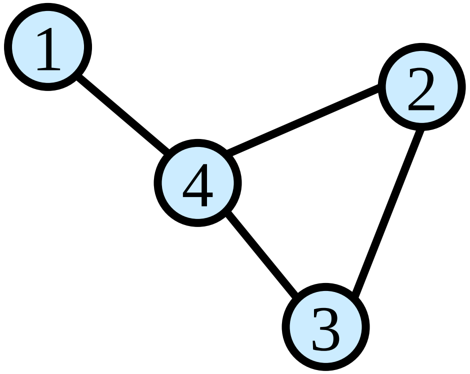
Advantage:

- $O(1)$ 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