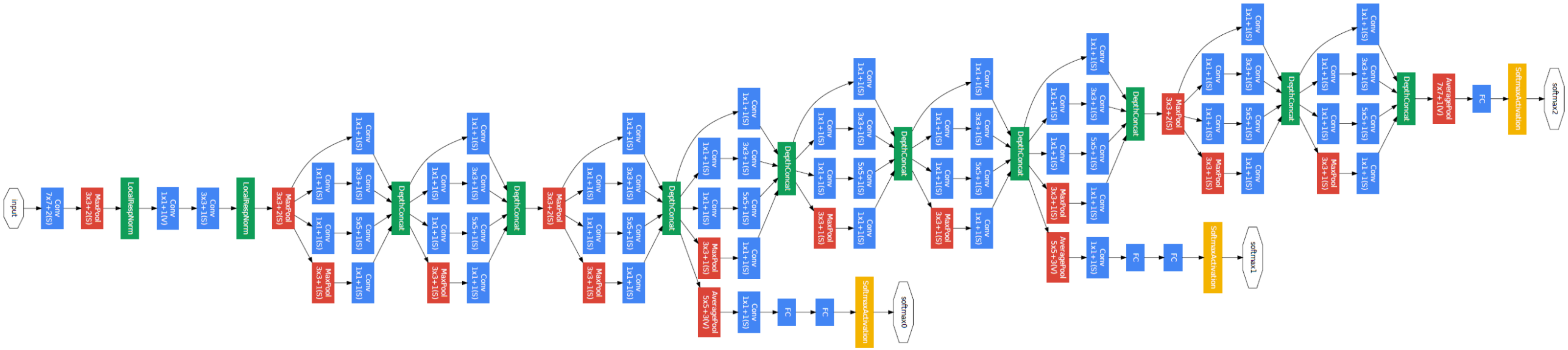


CSCI 241

Lecture N-1
Graph Planarity
Topological Sort

Announcements

- Quiz 6 grades are out.
  - video coming soon
- A4 is due Wednesday
- No lab deliverable this week.
  - TAs will be available in lab sections for questions.
- Material from today onward will not be on the exam
  - this week: a mix of fun bonus topics and review

Tentative Goals - This week

- Analyze the runtime of Dijkstra's algorithm.
- Know the definition of graph planarity
- Know how to use Topological Sort to determine whether a graph is acyclic.
- Know the definition of a spanning tree.
- Know how to build spanning trees using:
  - Prim's algorithm
  - Kruskal's algorithm
- Coding trees?
- Tries?

Exercise: Analyze the runtime of Dijkstra's Algorithm.

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
    f = node in F with min d value;
    Remove f from F, add it to S;
    for each neighbor w of f {
        if (w not in S or F) {
            w.d =  f.d + weight(f, w);
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
        }
    }
}
```

Let $e = |E|$, $v = |V|$
Assume hash table lookups are O(1).
For all else, assume worst-case.

One group member: submit your group's answer via Socrative.

Pseudocode is available at:

https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_20s/lectures/L21/algorithm.pdf

Course webpage > Schedule > 5/27 > algorithm

Exercise: Analyze the runtime of Dijkstra's Algorithm.

```
S = { }; F = {v};  v.d = 0;
while  (F ≠ {})  {
    f = node in F with min d value;
    Remove f from F, add it to S;
    for each neighbor w of f {
        if (w not in S or F) {
            w.d =  f.d + weight(f, w);
            add w to F;
        } else if (f.d+weight(f,w) < w.d) {
            w.d = f.d+weight(f,w);
        }
    }
}
```

Let $e = |E|$, $v = |V|$
Assume hash table lookups are O(1).
For all else, assume worst-case.

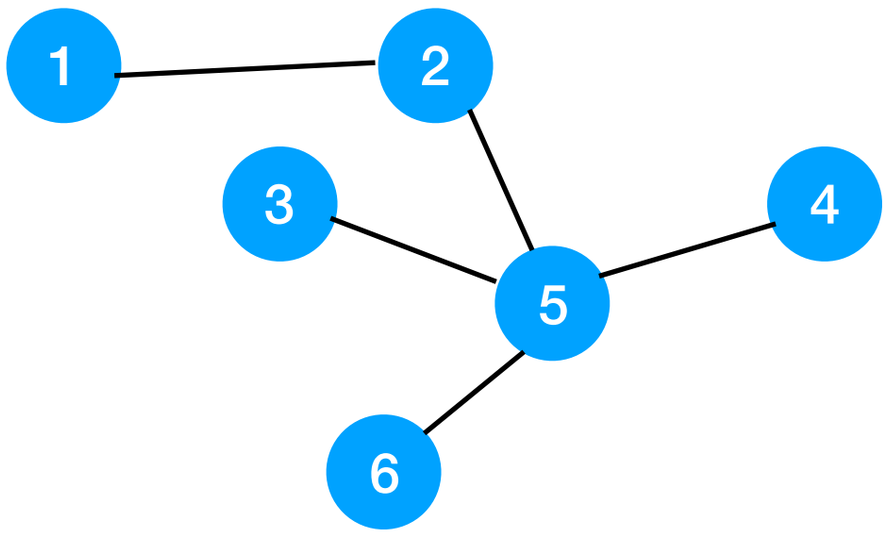
https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_20s/lectures/L21/algorithm.pdf
Course webpage > Schedule > 5/27 > algorithm

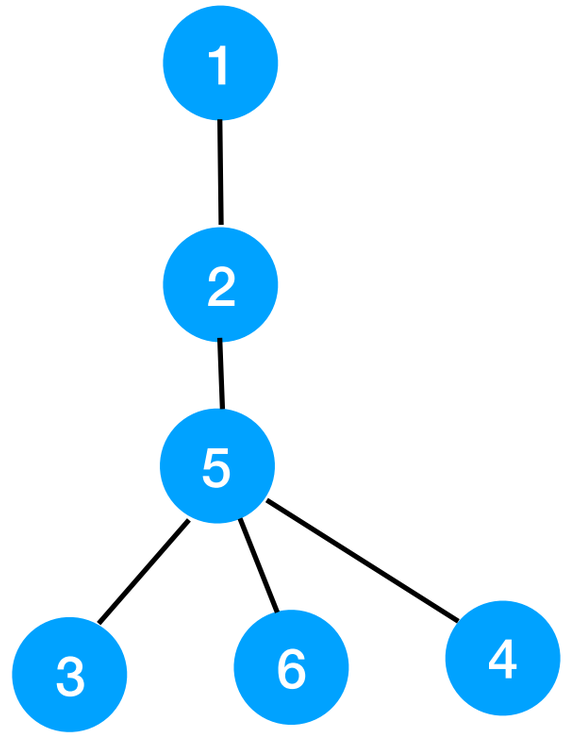
Drawing Graphs

- The same graph can be drawn (infinitely!) many different ways.

$$V = \{1,2,3,4,5,6\}$$

$$E = \{(1,2), (2,5), (3,5) (4,5), (5,6)\}$$



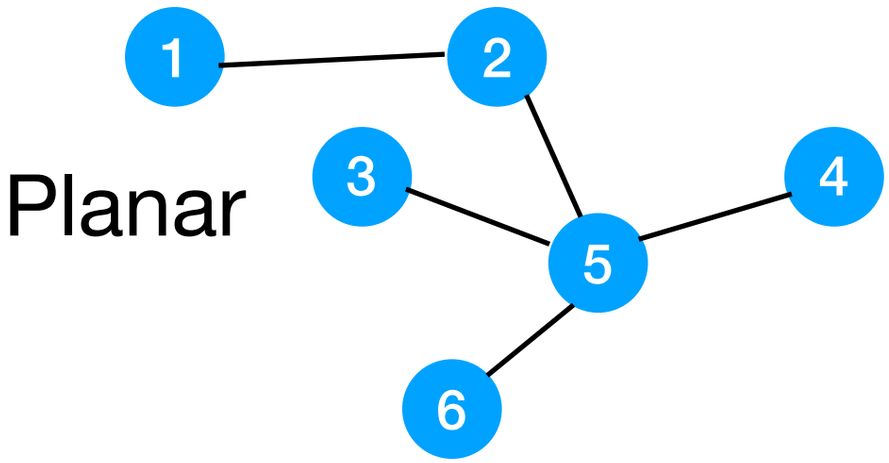


Planarity

- If a graph can be drawn without crossing edges, it is planar.

Planarity

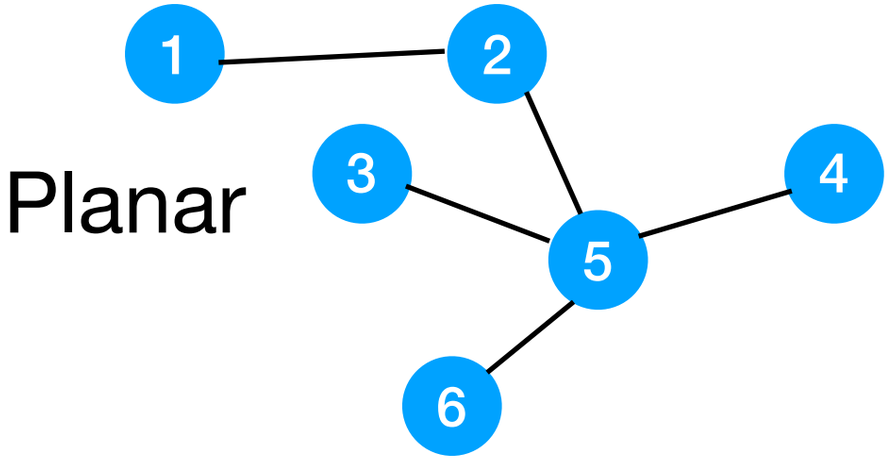
- If a graph can be drawn without crossing edges, it is planar.



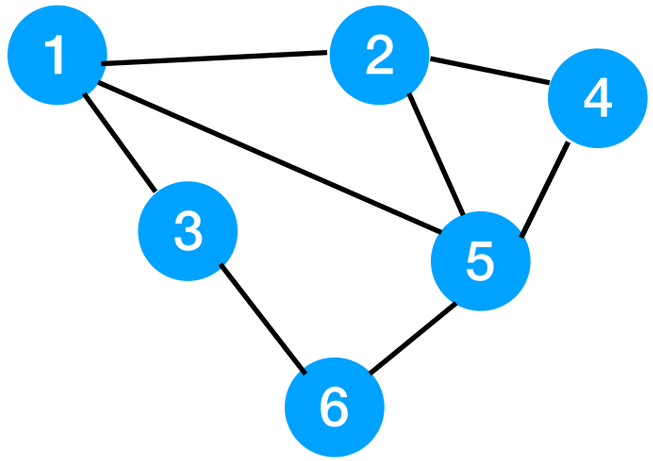
Planar

Planarity

- If a graph can be drawn without crossing edges, it is planar.



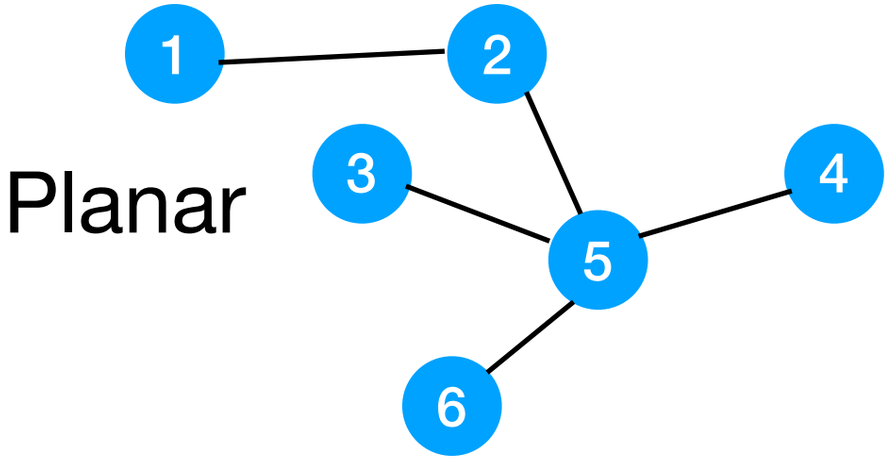
Planar



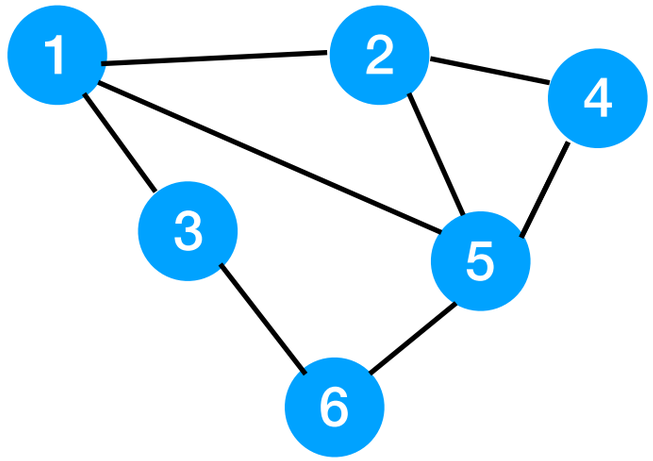
Planar

Planarity

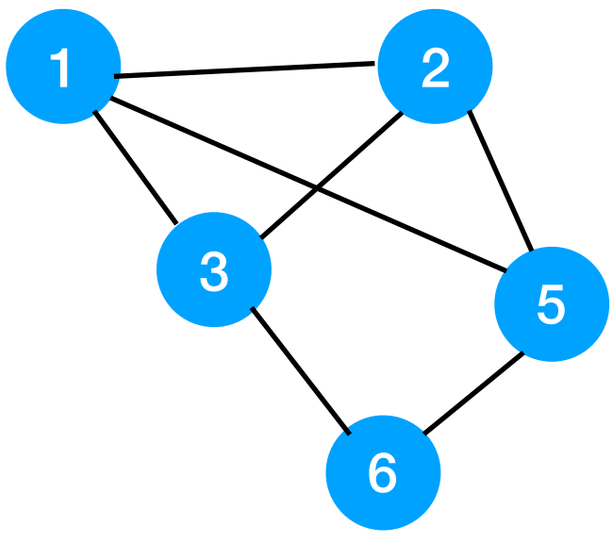
- If a graph can be drawn without crossing edges, it is planar.



Planar

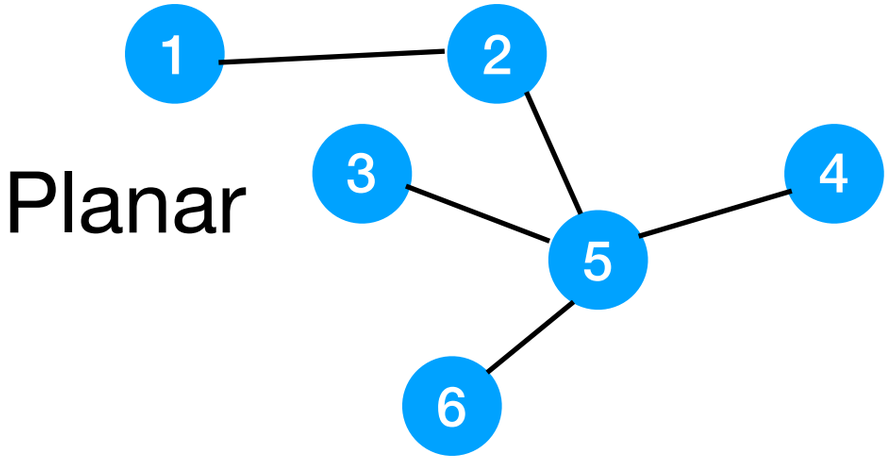


Planar

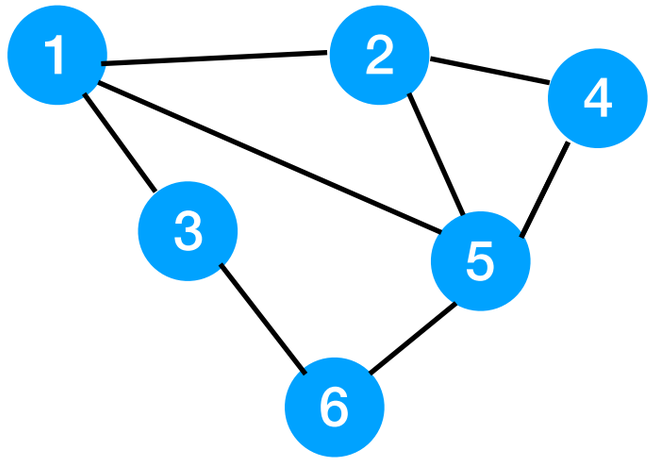


Planarity

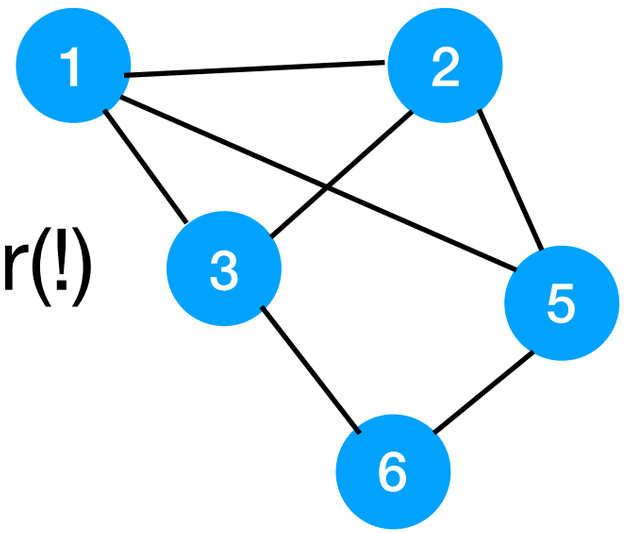
- If a graph can be drawn without crossing edges, it is planar.



Planar



Planar

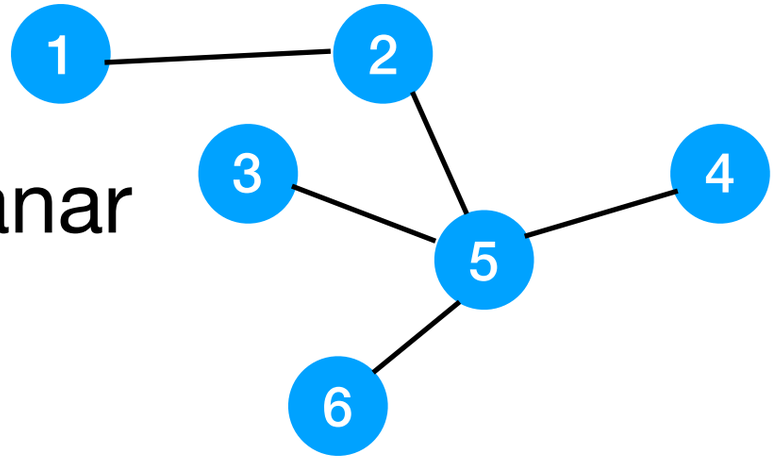


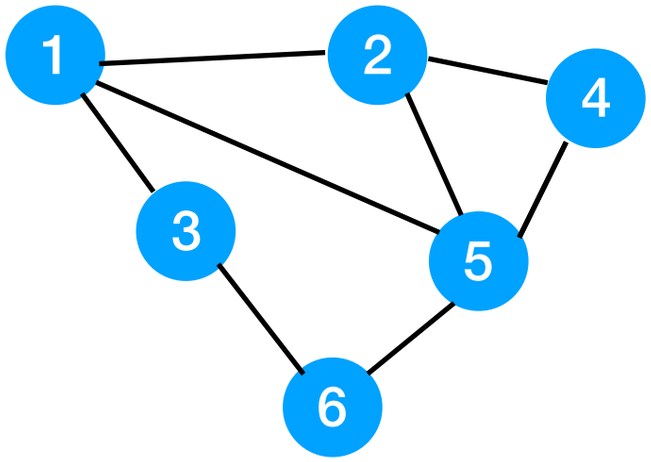
Planar(!)

Planarity

- If a graph can be drawn without crossing edges, it is planar.

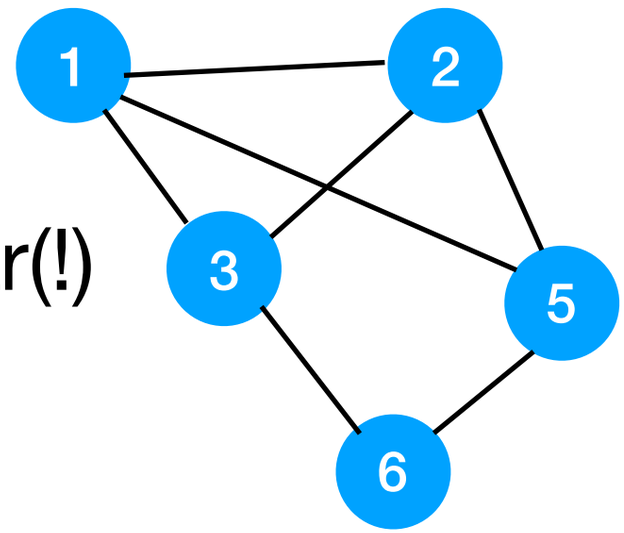
Planar



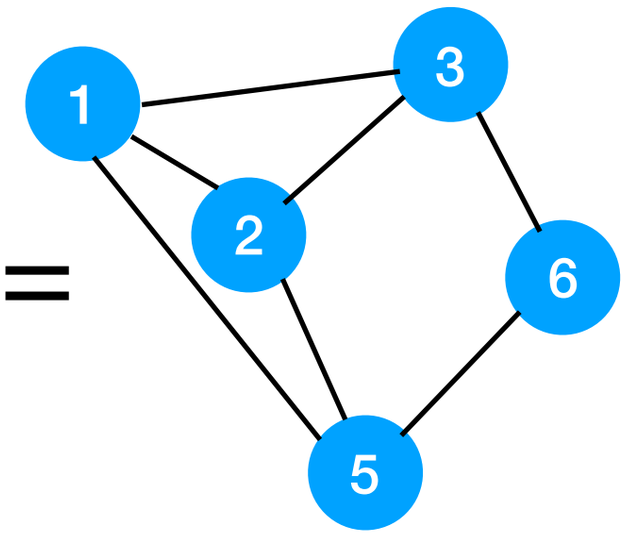


Planar

Planar(!)



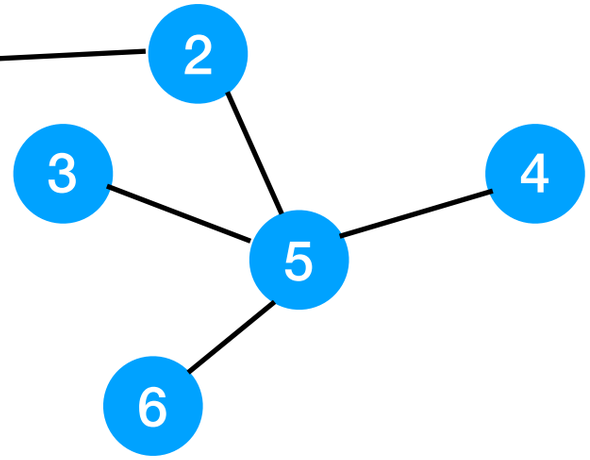
=

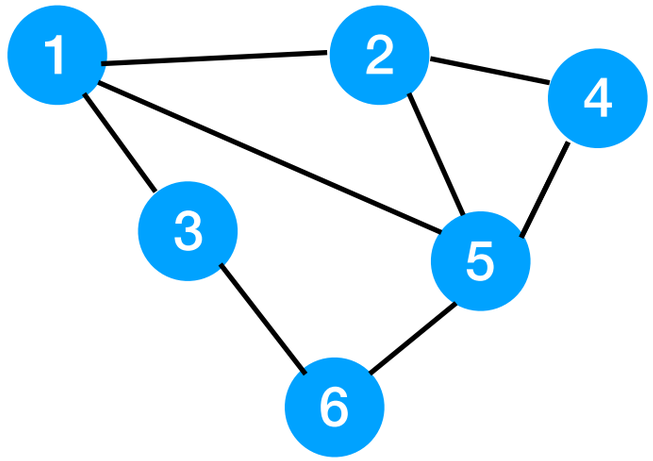


Planarity

- If a graph can be drawn without crossing edges, it is planar.

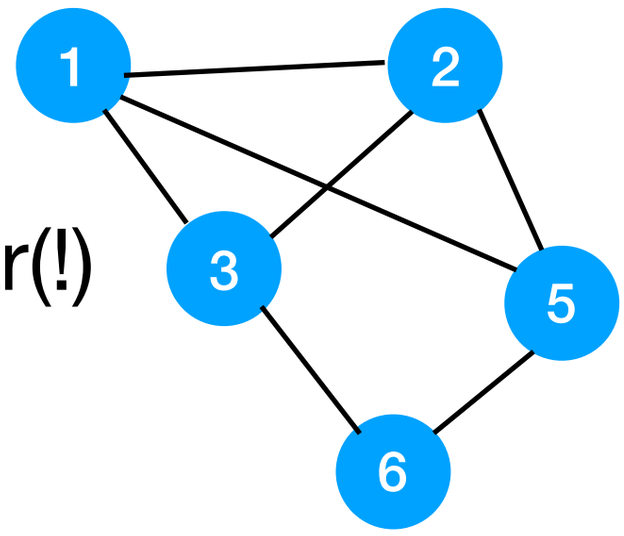
Planar



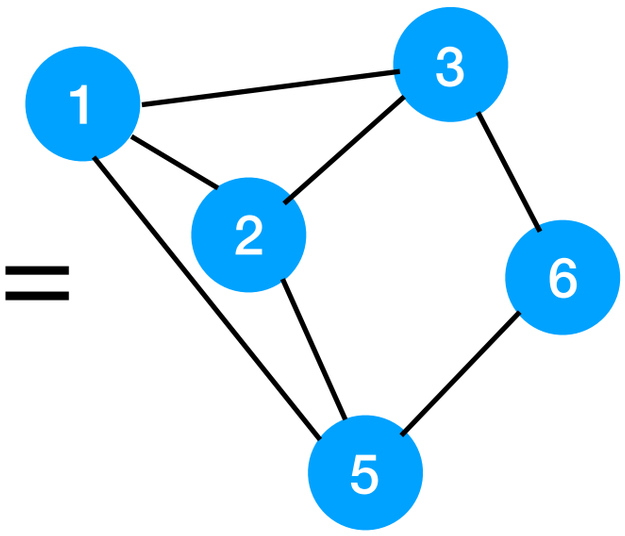


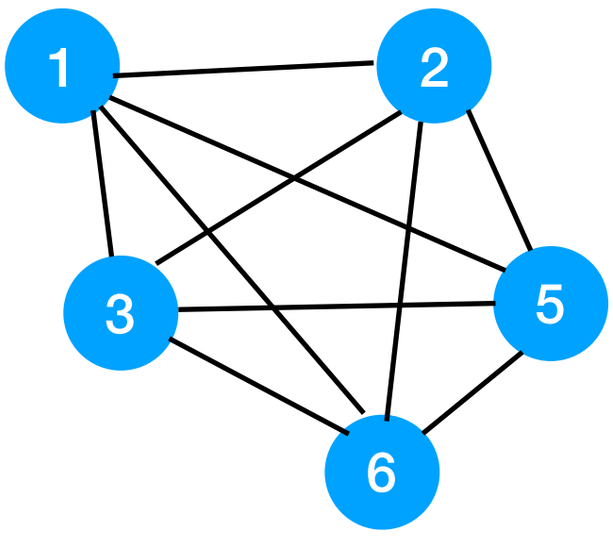
Planar

Planar(!)

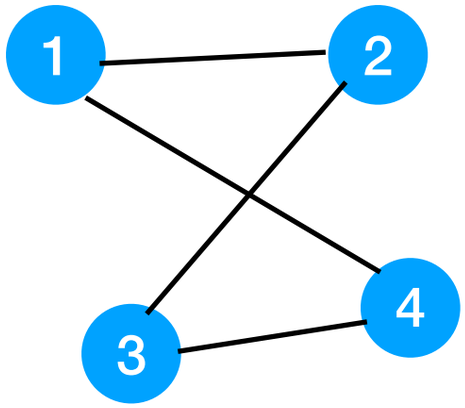


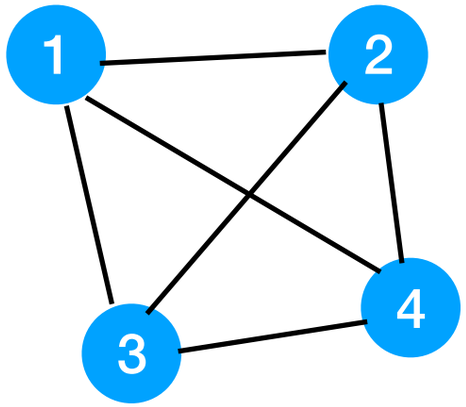
=

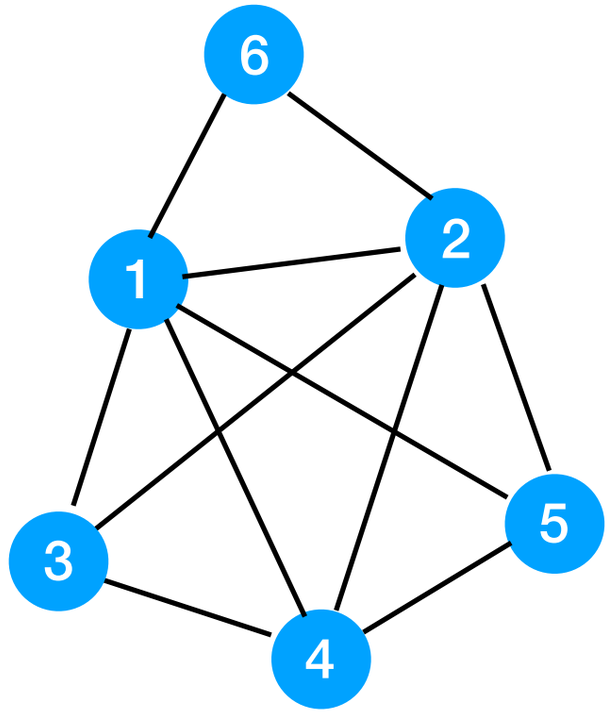




Not planar


1
2
3
4


1
2
3
4


6
1
2
3
4
5

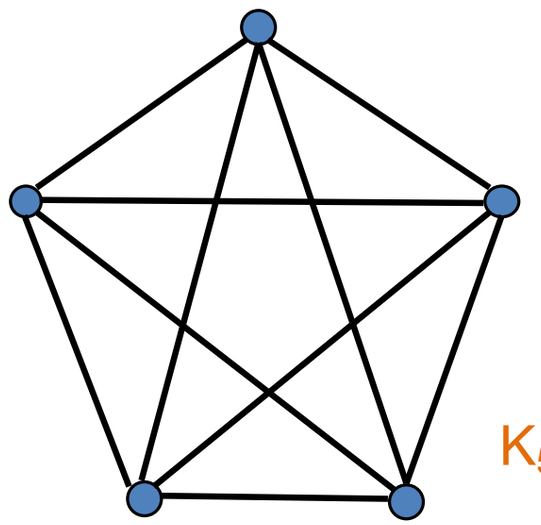
Detecting Planarity

A subgraph of a graph is a graph whose vertex and edge sets are subsets of the larger graph's.

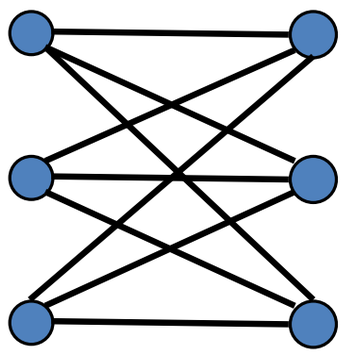
- Elements of the edge subset can only contain nodes in the vertex subset.

- There's a (non-obvious) theorem that says a graph is planar if and only if it does not contain* one of these as a subgraph:

*The definition of "contain" is slightly more general than having one of these directly as a subgraph.



K_5

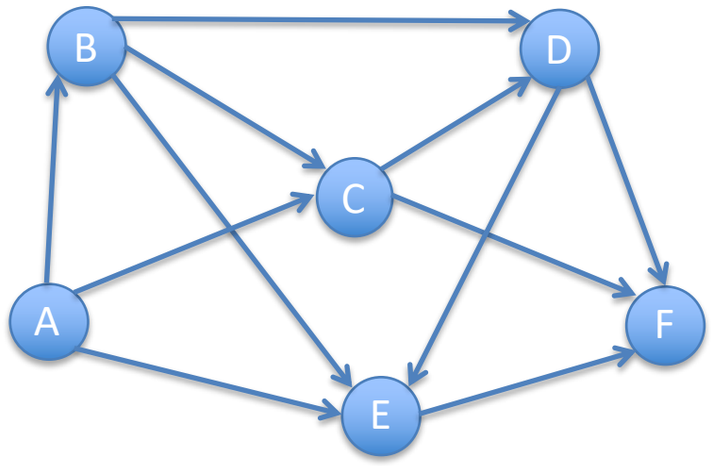


$K_{3,3}$

Detecting DAGs

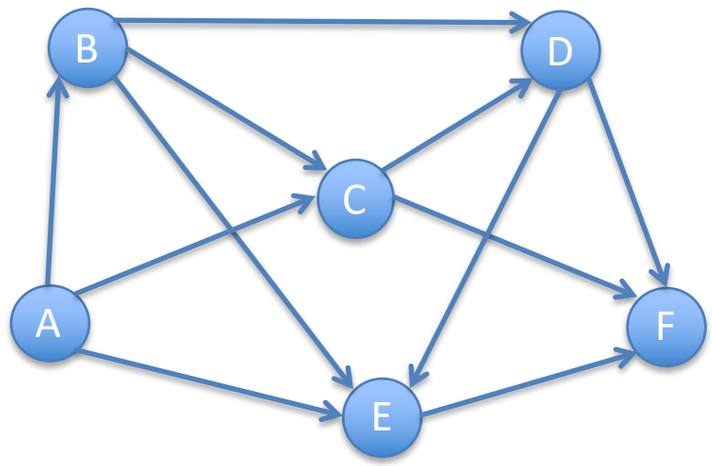
- A DAG, or Directed Acyclic Graph is a…

  graph that is directed and acyclic.



Is this a DAG?

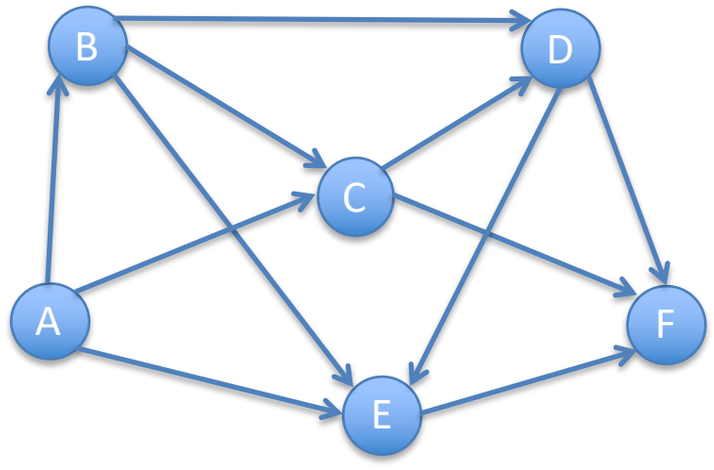
- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.



Is this a DAG?

- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.

Algorithm:



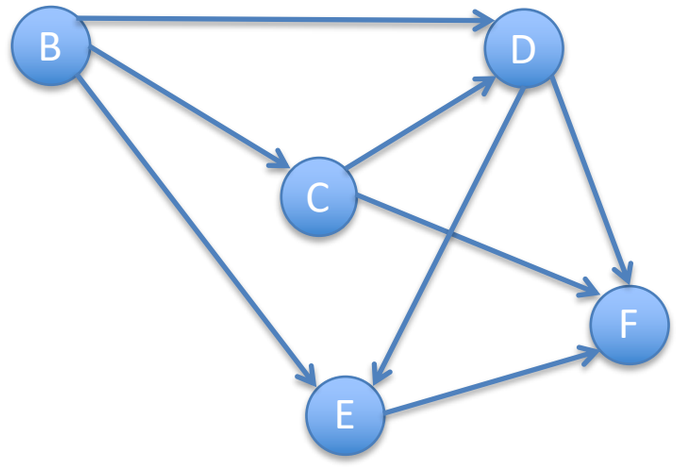
while there is a node with indegree 0:

delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG

Is this a DAG?

- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.



Algorithm:

while there is a node with indegree 0:

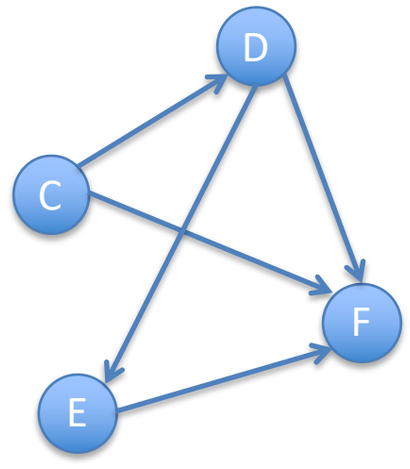
delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG

Is this a DAG?

- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.

Algorithm:



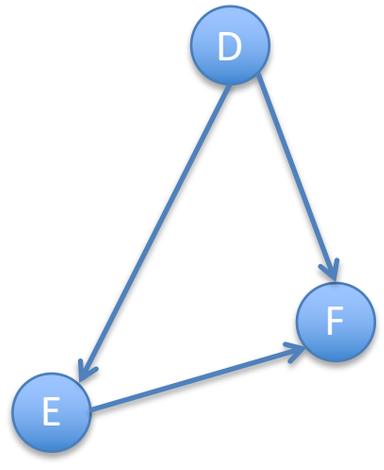
while there is a node with indegree 0:

delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG

Is this a DAG?

- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.



Algorithm:

while there is a node with indegree 0:

delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG

Is this a DAG?

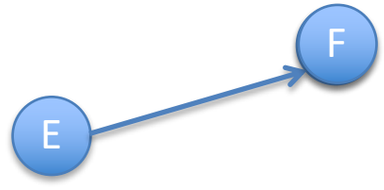
- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.

Algorithm:

while there is a node with indegree 0:

delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG



Is this a DAG?

- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can't be part of a cycle.
  - Edges coming from that node also can't be part of a cycle.

Algorithm:

while there is a node with indegree 0:

delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG

Topological Sort

Topological sort (or toposort):

i = 0

while there is a node with indegree 0:

delete* the node and all edges coming from it

label* the deleted node i

increment i

if the graph is empty, the original graph was a DAG

Topological Sort

Topological sort (or toposort):

i = 0

while there is a node with indegree 0:

delete* the node and all edges coming from it

label* the deleted node i

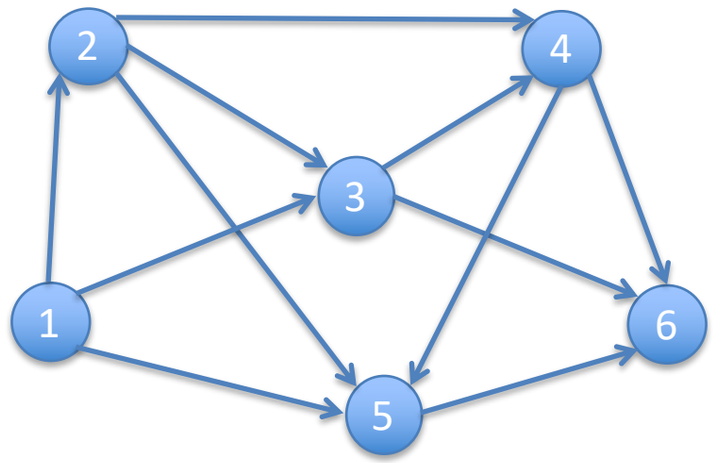
increment i

if the graph is empty, the original graph was a DAG

***This is pseudocode: we probably don't want to actually modify the graph. We'll need to store extra data with nodes and edges, and possibly overlay additional data structures to make it efficient.**

Topological Sort

- Here are the labels we applied to the example graph:

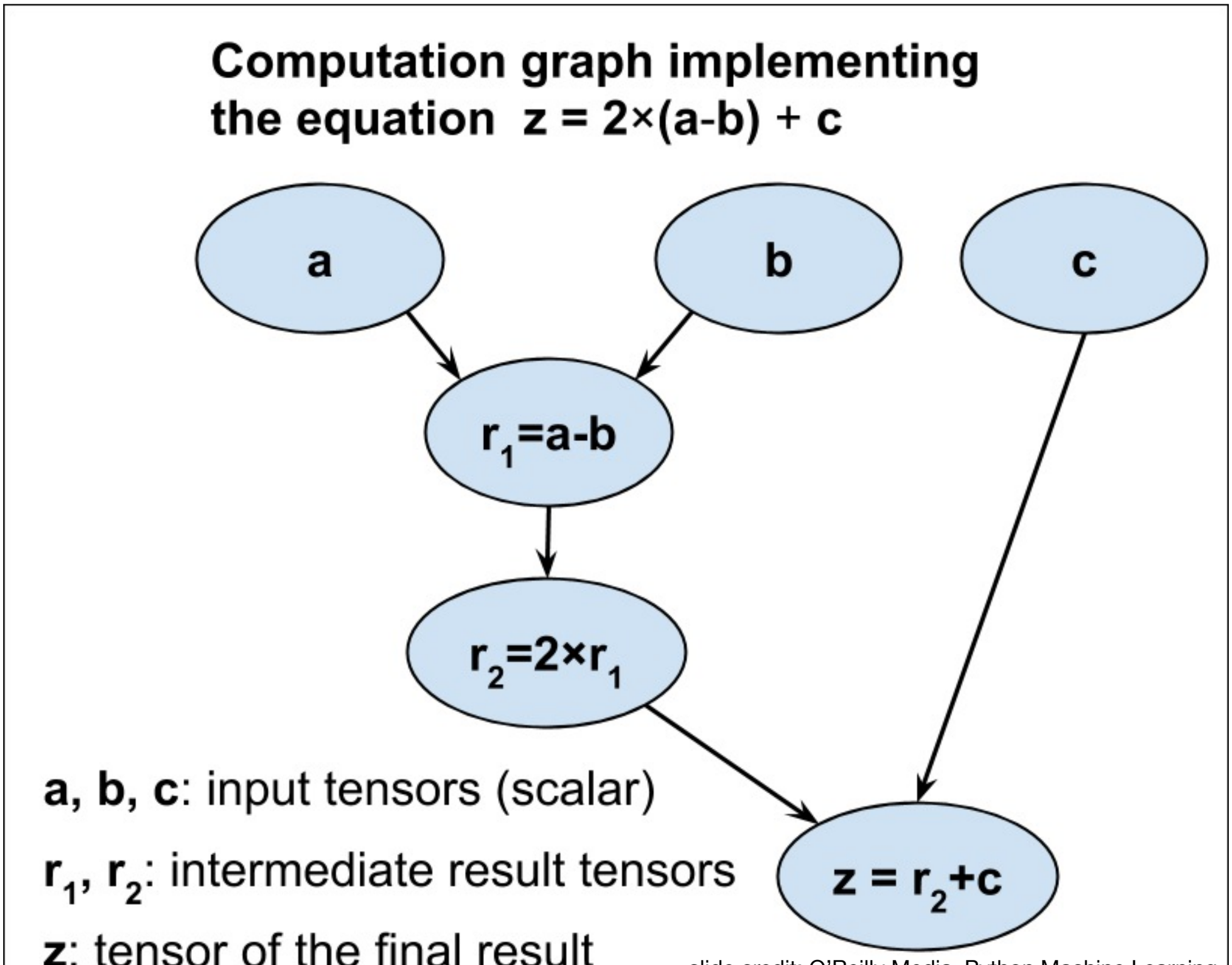


- Property: all edges go from a lower-numbered node to a higher-numbered node.

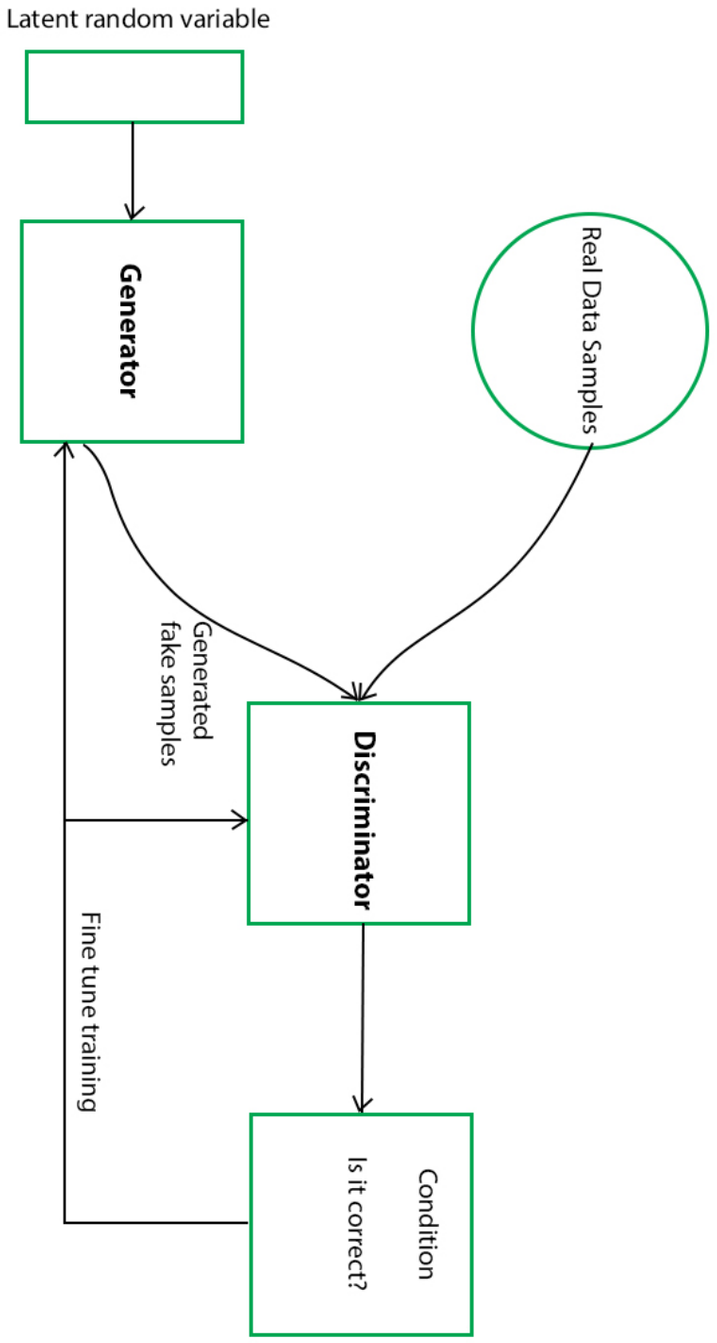
- Useful for dependency resolution, job scheduling,

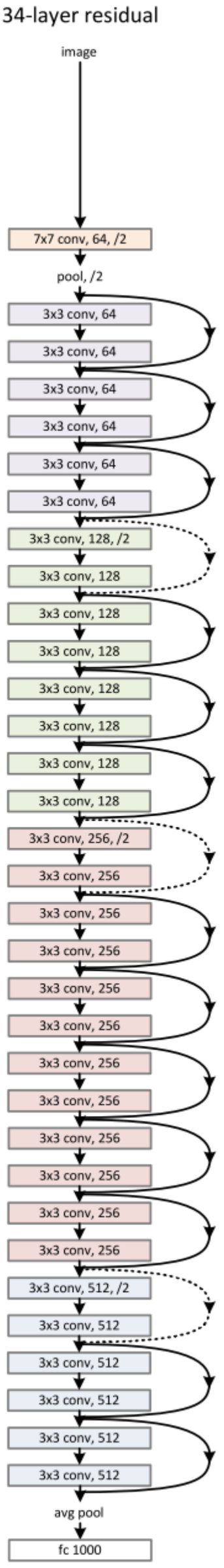
- Ordering is not necessarily unique: could have chosen from among multiple nodes with indegree 0.

Tensorflow Computation Graphs



slide credit: O'Reilly Media, Python Machine Learning


Latent random variable
Generator
Real Data Samples
Generated fake samples
Discriminator
Fine tune training
Condition
Is it correct?


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

Word Problems!