

# GRAPH ALGORITHMS

Lecture 19

CS 2110 — Spring 2019

# JavaHyperText Topics

2

“Graphs”, topics:

- 4: DAGs, topological sort
- 5: Planarity
- 6: Graph coloring

# Announcements

3

Monday after Spring Break there will be a CMS quiz about “Shortest Path” tab of JavaHyperText. To prepare:

- Watch the videos (< 15 min) and their associated PDFs (in total 5 pages)
- Especially try to understand the loop invariant and the development of the algorithm

# Announcements

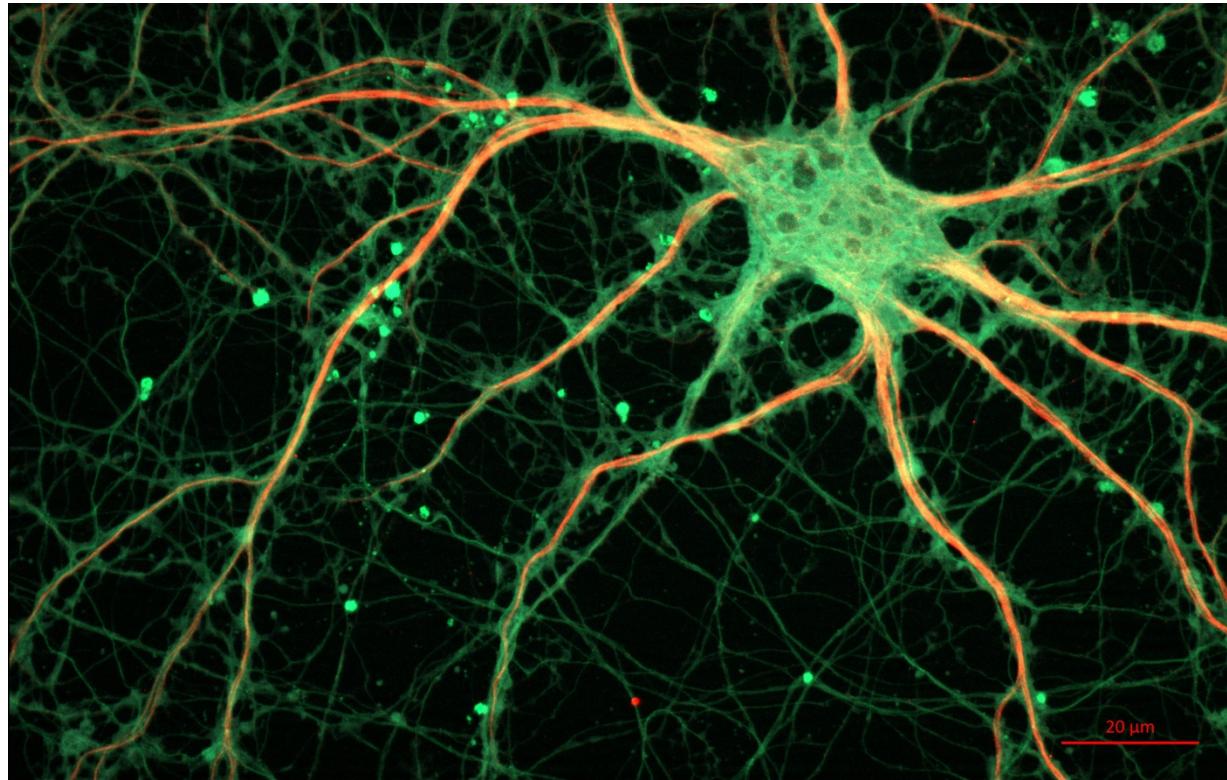
4

- Yesterday, 2018 Turing Award Winners announced
- Won for **deep learning** with **neural networks**
  - Facial recognition
  - Talking digital assistants
  - Warehouse robots
  - Self-driving cars
  - ...see NYTimes article

**Neural networks are graphs!**

# Neural Network

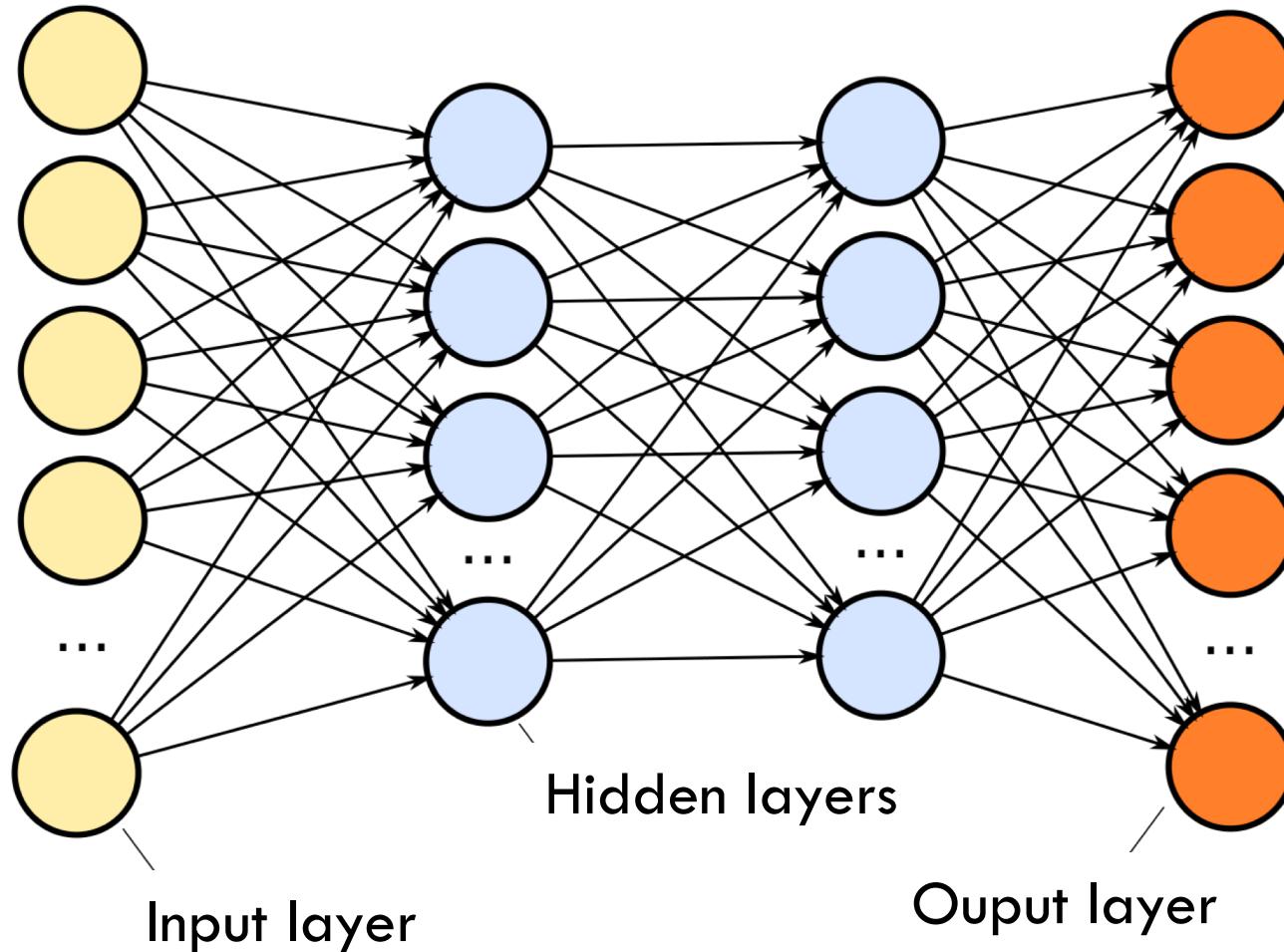
5



Neurons in brain receive input, maybe fire and activate other neurons

# Neural Network

6

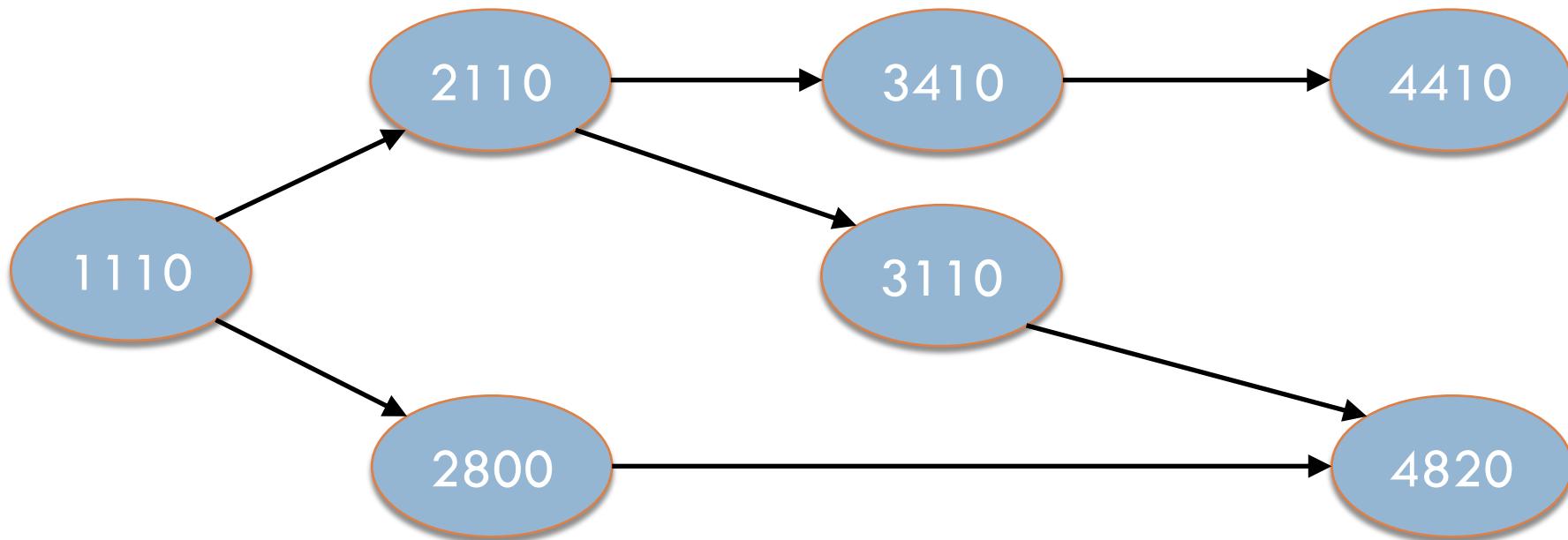


# Sorting

# CS core course prerequisites

(simplified)

8



**Problem:** find an order in which you can take courses without violating prerequisites

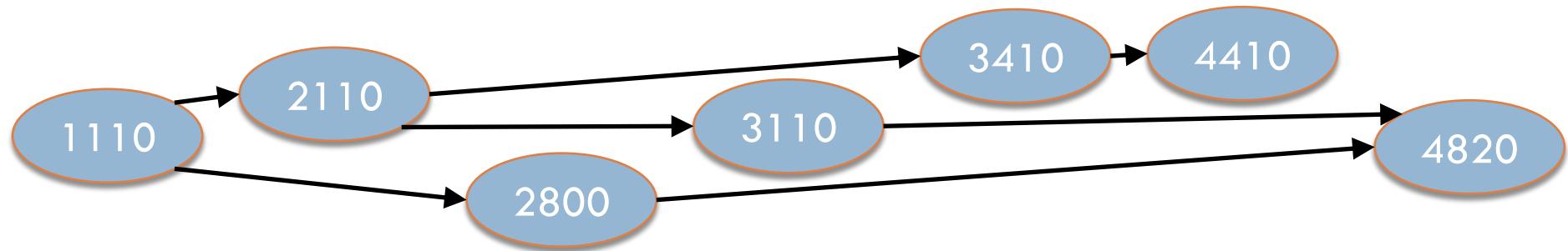
e.g. 1110, 2110, 2800, 3110, 3410, 4410, 4820

# Topological order

9

A **topological order** of directed graph  $G$  is an ordering of its vertices as  $v_1, v_2, \dots, v_n$ , such that for every edge  $(v_i, v_j)$ , it holds that  $i < j$ .

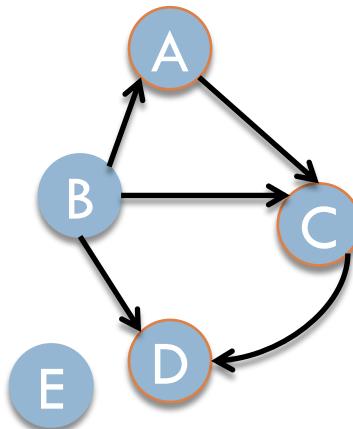
**Intuition:** line up the vertices with all edges pointing left to right.



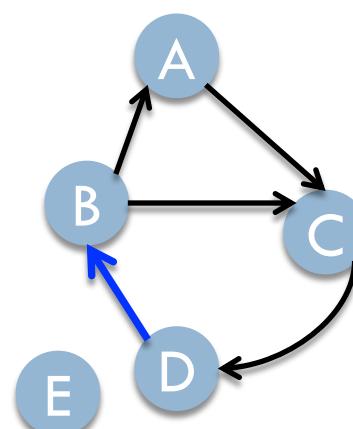
# Cycles

10

- A directed graph can be topologically ordered if and only if it has no cycles
- A **cycle** is a path  $v_0, v_1, \dots, v_p$  such that  $v_0 = v_p$
- A graph is **acyclic** if it has no cycles
- A directed acyclic graph is a **DAG**



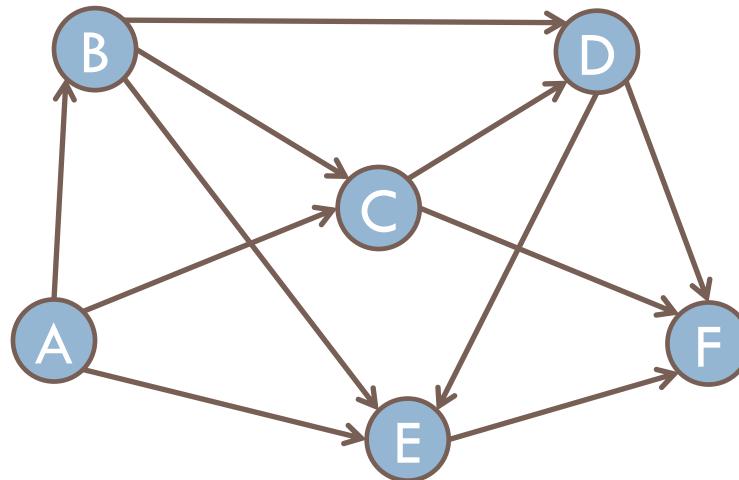
DAG



Not a DAG

# Is this graph a DAG?

11



Yes!  
*It was a DAG.*

- Deleting a vertex with indegree zero would not remove any cycles
- Keep deleting such vertices and see whether graph “disappears”

And the order in which we removed vertices was a topological order!

# Algorithm: topological sort

12

$$k = 0;$$

// inv: k nodes have been given numbers in 1..k in such a way that  
if  $n_1 \leq n_2$ , there is no edge from  $n_2$  to  $n_1$ .

```
while (there is a node of in-degree 0) {
```

Let  $n$  be a node of in-degree 0;

## Give it number k;

Delete  $n$  and all edges leaving it from the graph.

**k = k + 1;**

}

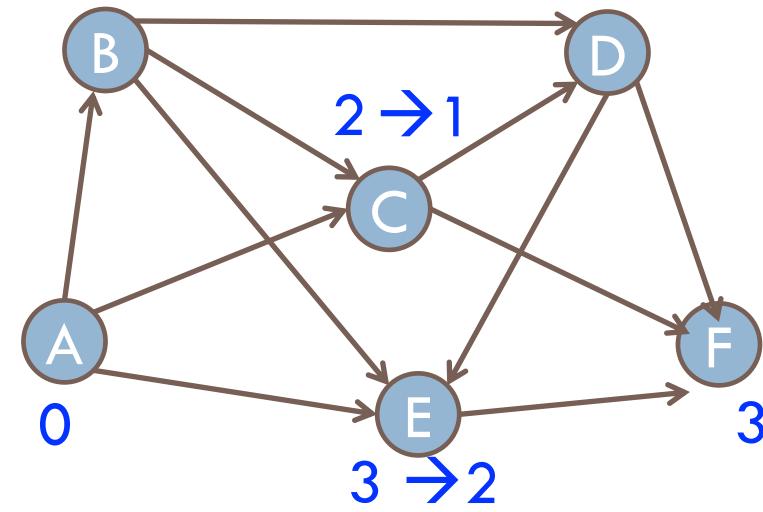
**JavaHyperText shows how to  
implement efficiently:  
 $O(V+E)$  running time.**

...k in such a way that

$$k=0 \rightarrow 1$$

1 → 0

2

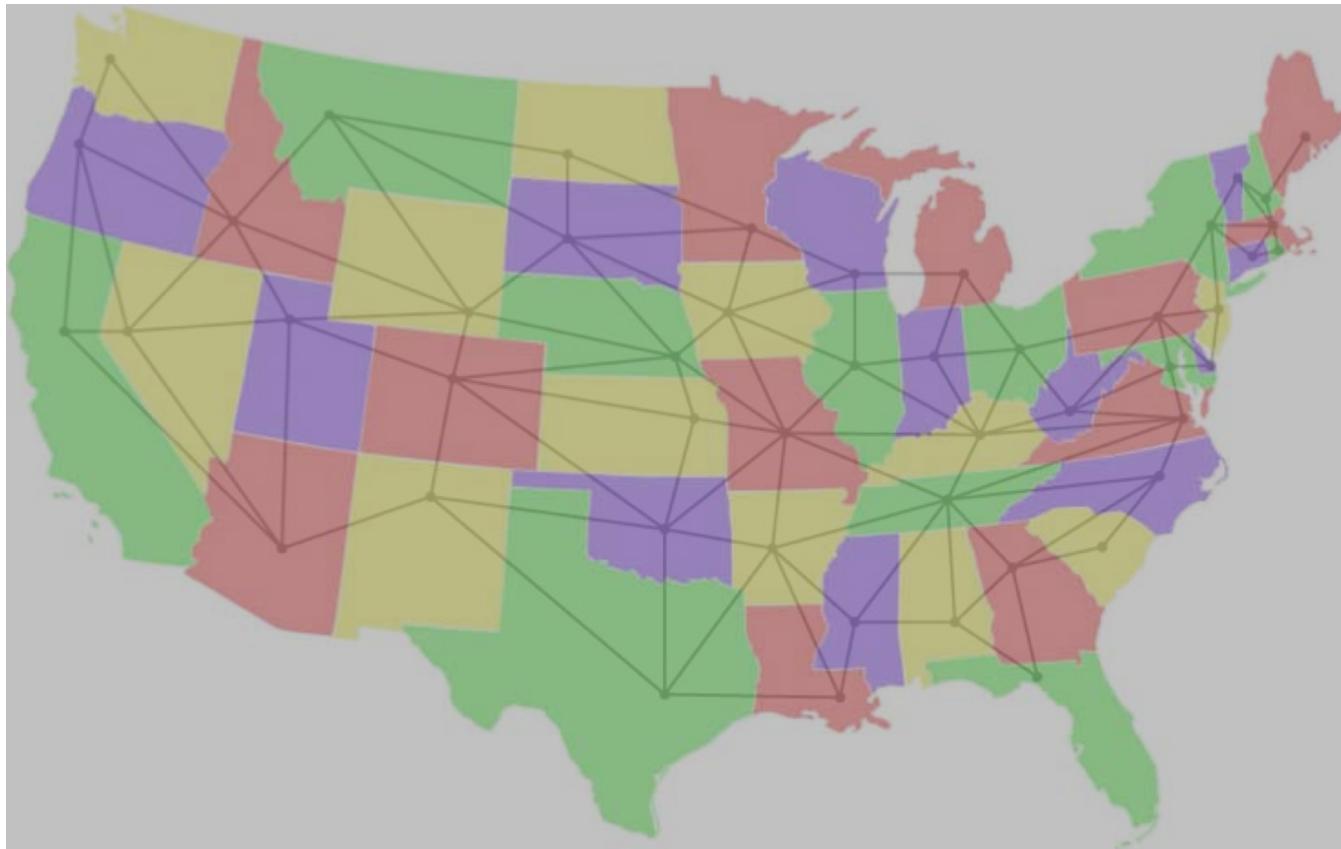


# Graph Coloring

# Map coloring

14

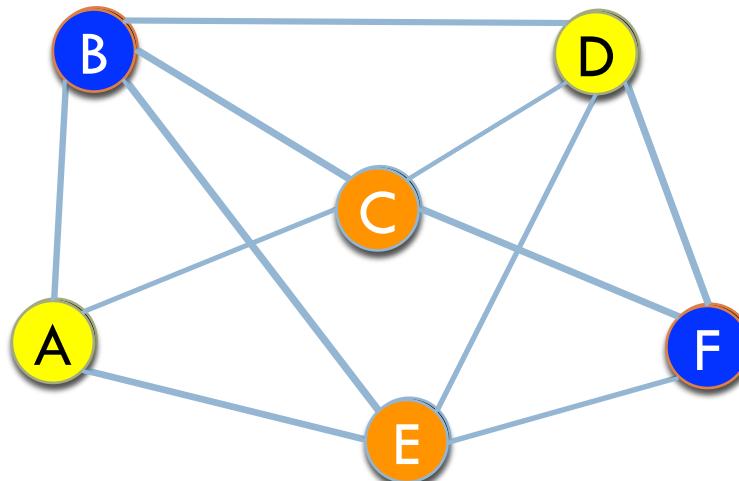
How many colors are needed to ensure adjacent states have different colors?



# Graph coloring

15

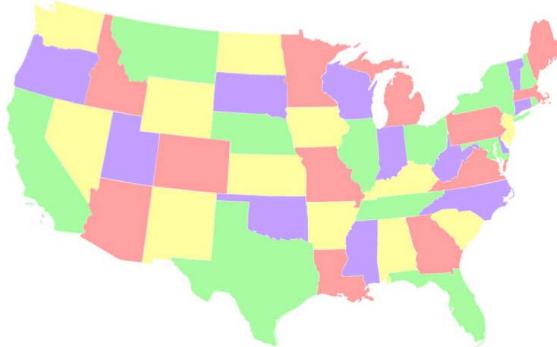
**Coloring:** assignment of color to each vertex.  
Adjacent vertices must have different colors.



How many colors needed?

# Uses of graph coloring

16



**8 Team Round Robin**

Team	Wins	Losses
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		

Team Names

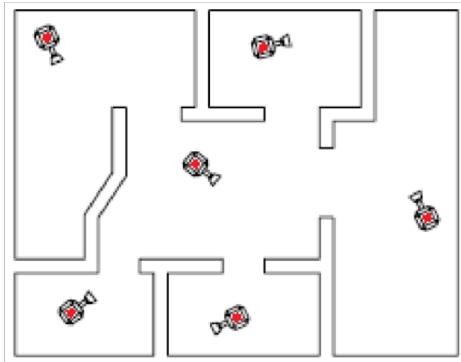


Round 1 Round 2 Round 3 Round 4 Round 5 Round 6 Round 7

2 vs 1	3 vs 4	6 vs 2	7 vs 5	1 vs 3	4 vs 5	7 vs 3
3 vs 8	1 vs 7	7 vs 8	8 vs 4	4 vs 2	8 vs 1	8 vs 2
4 vs 7	8 vs 6	4 vs 1	2 vs 3	5 vs 8	2 vs 7	1 vs 5
5 vs 6	2 vs 5	5 vs 3	6 vs 1	6 vs 7	3 vs 6	6 vs 4

Team vs. Team

[PrintYourBrackets.com](http://PrintYourBrackets.com)



# How to color a graph

17

```
void color() {  
    for each vertex v in graph:  
        c = find_color(neighbors of v);  
        color v with c;  
    }  
  
    int find_color(vs) {  
        int[] used;  
        assign used[c] the number of vertices in vs that are colored c  
  
        return smallest c such that used[c] == 0;  
    }
```

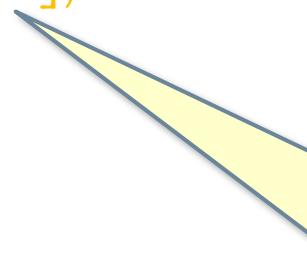
Assume colors are integers 0, 1, ...

# How to color a graph

18

```
void color() {  
    for each vertex v in graph:  
        c= find_color(neighbors of v);  
        color v with c;  
    }  
  
int find_color(vs) {  
    int[] used= new int[vs.length() + 1];  
    for each vertex v in vs:  
        if color(v) <= vs.length():  
            used[color(v)]++;  
    }  
    return smallest c such that used[c] == 0;  
}
```

Assume colors are integers 0, 1, ...



If there are  $d$  vertices, need at most  $d+1$  available colors

# Analysis

19

```
void color() {  
    for each vertex v in graph:  
        c = find_color(neighbors of v);  
        color v with c;  
    }  
  
    int find_color(vs) {  
        int[] used = new int[vs.length() + 1];  
        for each vertex v in vs:  
            if color(v) <= vs.length():  
                used[color(v)]++;  
        }  
        return smallest c such that used[c] == 0;  
    }
```

Total time:  $O(E)$

Time:  $O(\# \text{ neighbors of } v)$

Time:  $O(\text{vs.length}())$

# Analysis

20

```
void color() {  
    for each vertex v in graph:  
        c= find_color(neighbors of v);  
        color v with c;  
    }  
  
int find_color(vs) {  
    int[] used= new int[vs.length() + 1];  
    for each vertex v in vs:  
        if color(v) <= vs.length():  
            used[color(v)]++;  
    }  
    return smallest c such that used[c] == 0;  
}
```

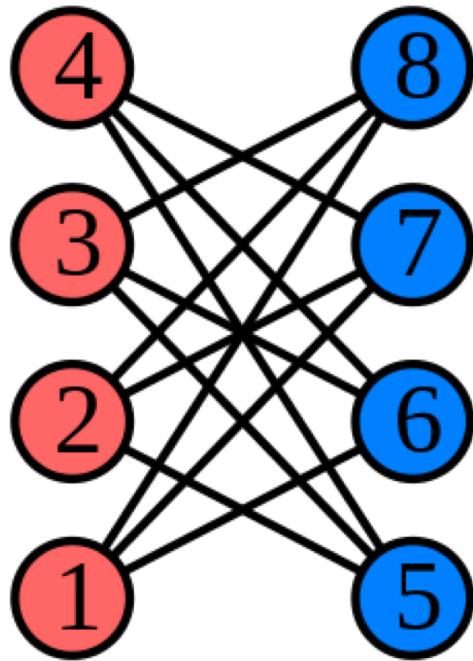
Use the minimum  
number of colors?

Maybe! Depends  
on order vertices  
processed.

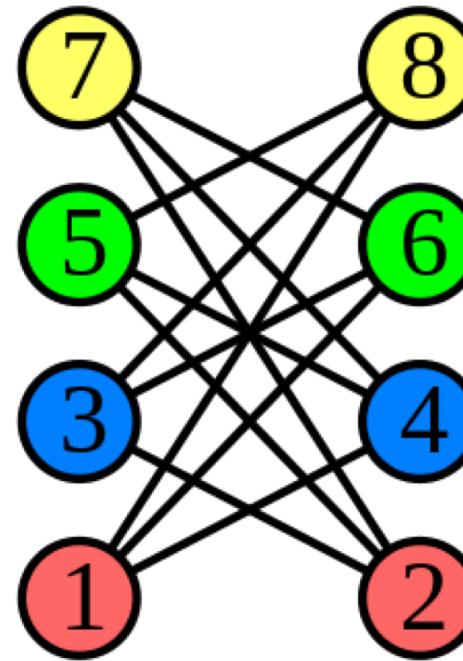
# Analysis

21

Best coloring



Worst coloring



Vertices labeled in order of processing

Only 2 colors needed for this special kind of graph...

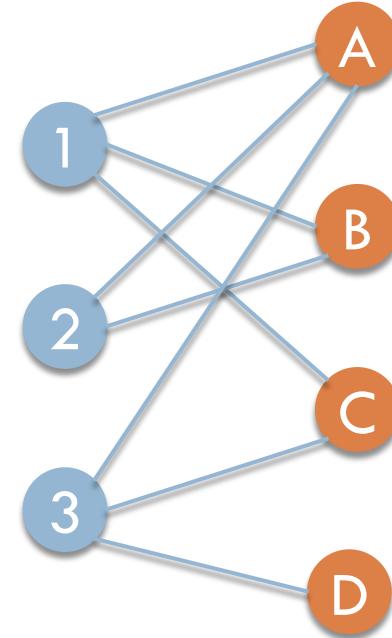
# Bipartite graphs

22

**Bipartite:** vertices can be partitioned into two sets such that no edge connects two vertices in the same set

Matching problems:

- Med students & hospital residencies
- TAs to discussion sections
- Football players to teams



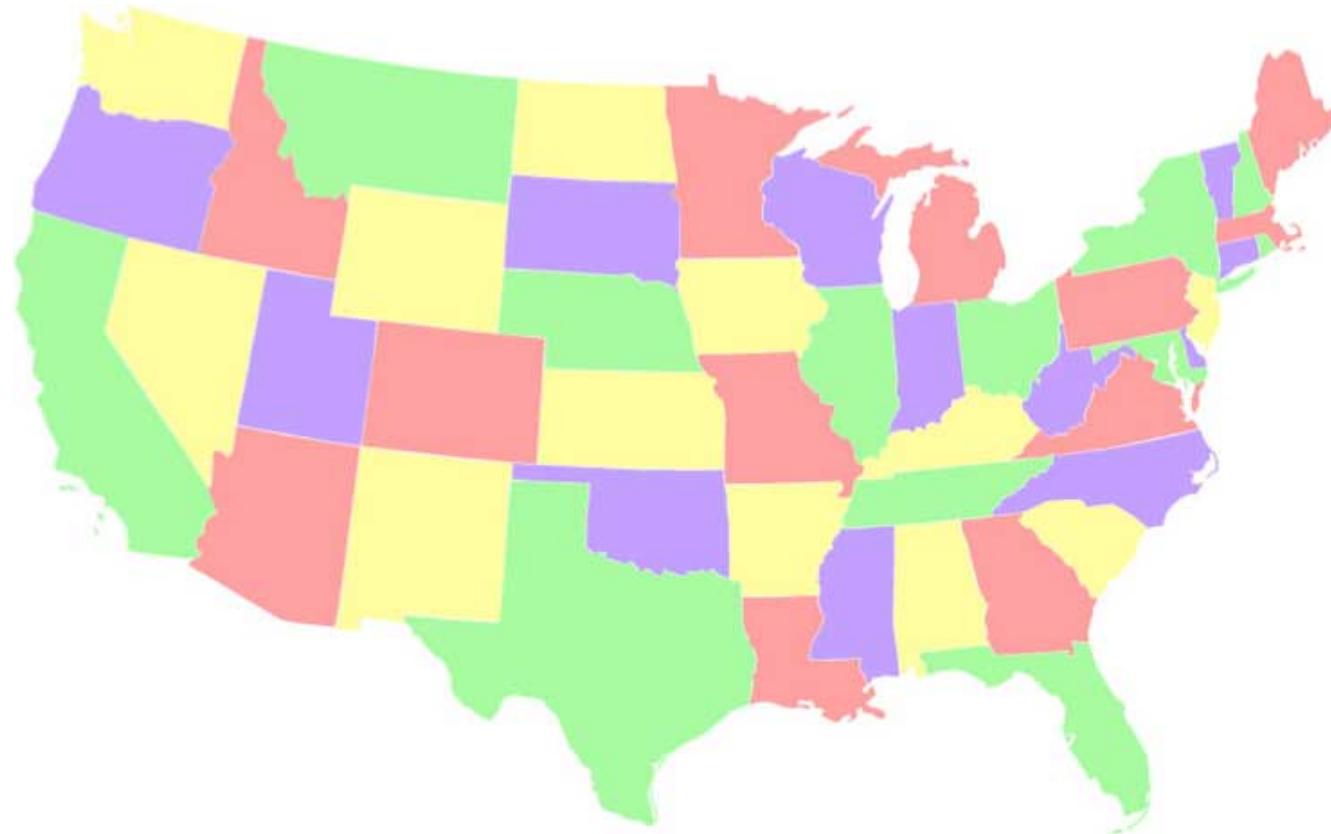
**Fact:**  $G$  is bipartite iff  $G$  is 2-colorable

# Four Color Theorem

23

Every “map-like” graph is 4-colorable

[Appel & Haken, 1976]



# Four Color Theorem

Proof required checking that 1,936 special graphs had a certain property

- Appel & Haken used a computer program to check the 1,936 graphs
- Does that count as a proof?
- Gries looked at their computer program and found an error; it could be fixed

In 2008 entire proof formalized in Coq proof assistant [Gonthier & Werner]: see CS 4160



# Four Color Theorem

25

Every “map-like” graph is 4-colorable

[Appel & Haken, 1976]

...“map-like”?

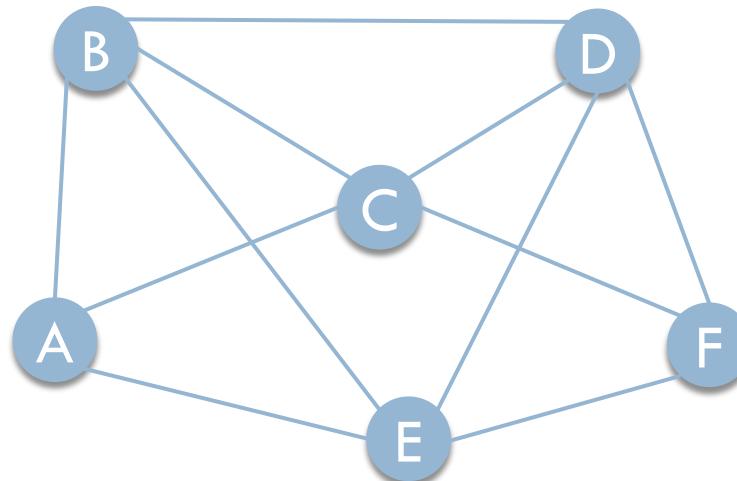
= planar

# Planar Graphs

# Planarity

27

A graph is **planar** if it can be drawn in the plane without any edges crossing

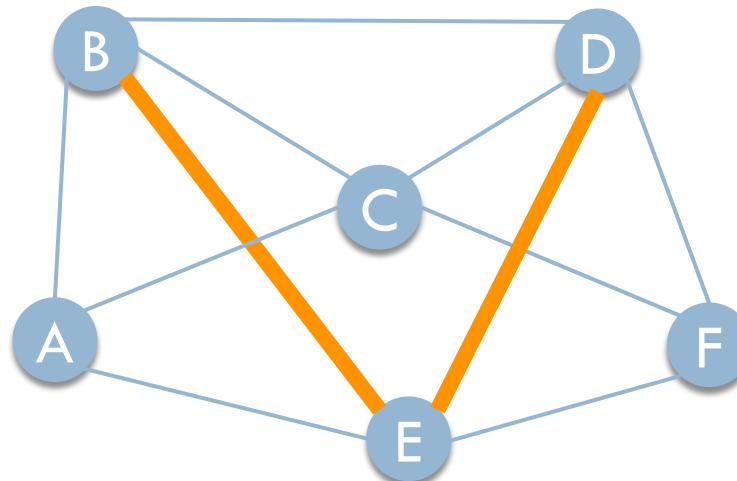


**Discuss:** Is this graph planar?

# Planarity

28

A graph is **planar** if it can be drawn in the plane without any edges crossing

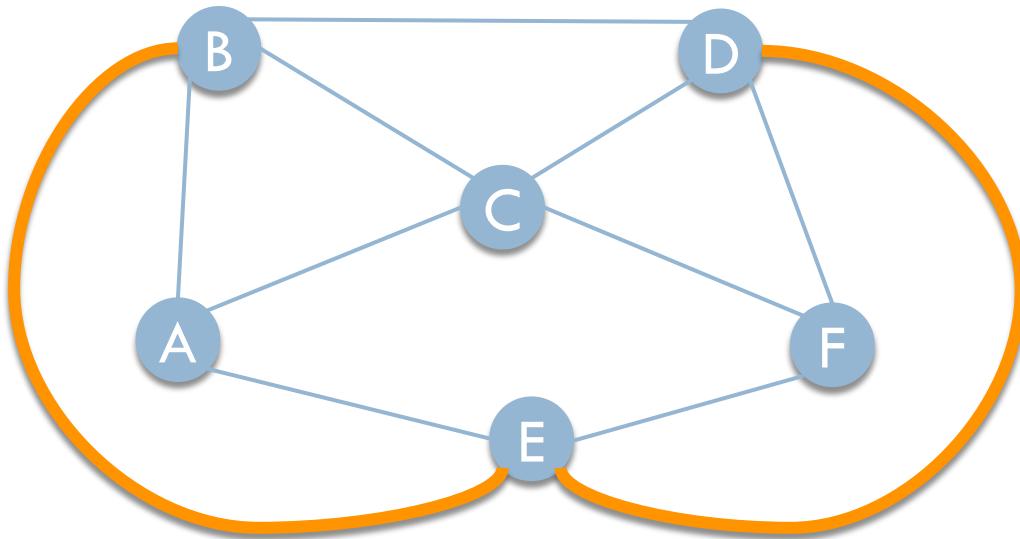


**Discuss:** Is this graph planar?

# Planarity

29

A graph is **planar** if it can be drawn in the plane without any edges crossing

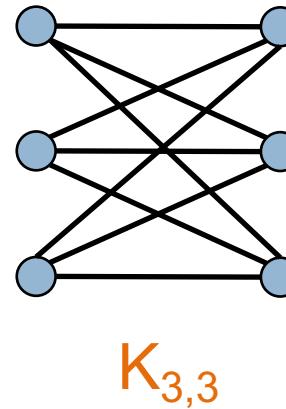
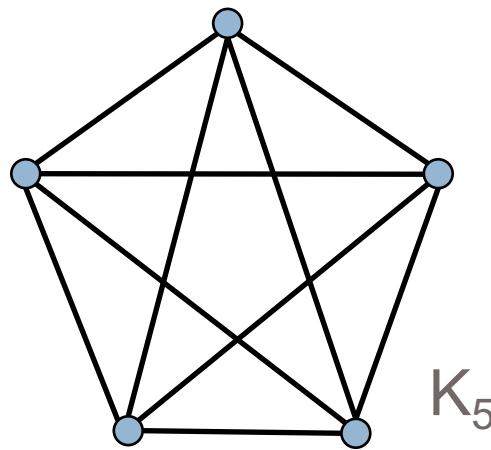


**Discuss:** Is this graph planar?  
**YES!**

# Detecting Planarity

30

## Kuratowski's Theorem:



A graph is planar if and only if it does not contain a copy of  $K_5$  or  $K_{3,3}$  (possibly with other nodes along the edges shown).

# John Hopcroft & Robert Tarjan

31

- **Turing Award in 1986** “for fundamental achievements in the design and analysis of algorithms and data structures”
- One of their fundamental achievements was a  $O(V)$  algorithm for determining whether a graph is planar.



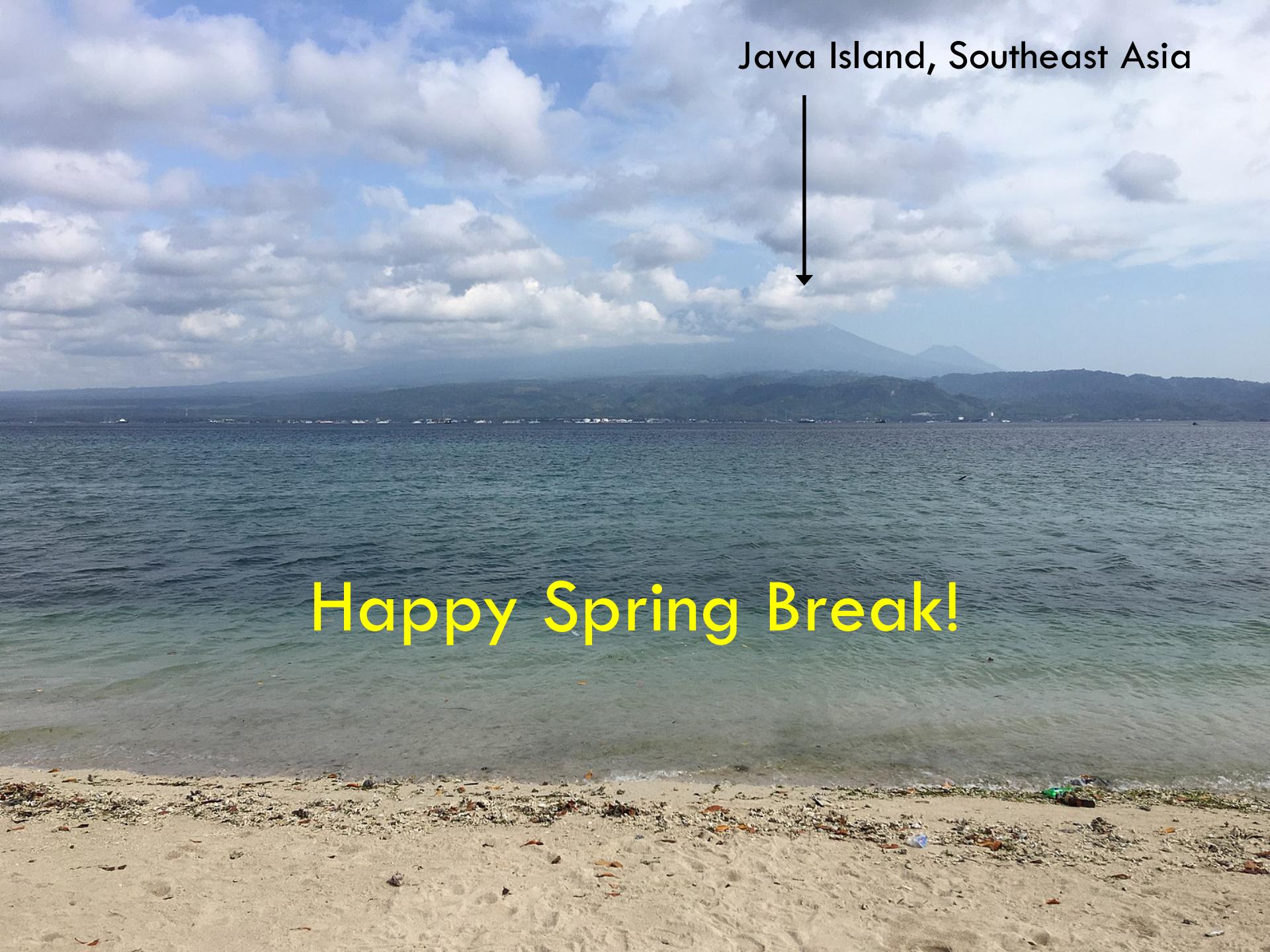


# David Gries & Jinyun Xue

32

## Tech Report, 1988

**Abstract:** We give a rigorous, yet, we hope, readable, presentation of the Hopcroft-Tarjan linear algorithm for testing the planarity of a graph, using more modern principles and techniques for developing and presenting algorithms that have been developed in the past 10-12 years (their algorithm appeared in the early 1970's). Our algorithm not only tests planarity but also constructs a planar embedding, and in a fairly straightforward manner. The paper concludes with a short discussion of the advantages of our approach.

A photograph of a tropical beach scene. In the foreground, there's a sandy beach with some scattered debris like leaves and small pieces of trash. The water is a clear, light blue-green color, gently lapping at the shore. In the middle ground, a range of hills or low mountains is visible across the water. A prominent, dark, conical shape of a volcano rises from behind the hills. The sky is filled with large, white, fluffy clouds against a bright blue sky.

Java Island, Southeast Asia



Happy Spring Break!