Graph coloring

Kevin Ly

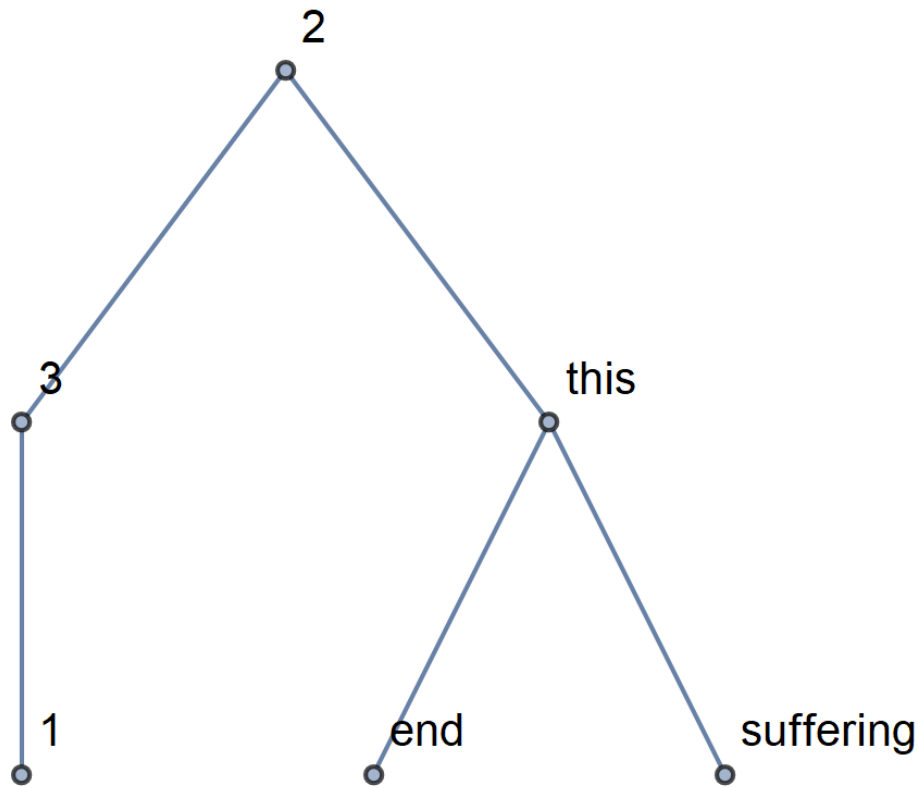
September 20, 2019

Itinerary

- Introduction to simple graphs
- A brief game of matching
- The coloring problem and DSATUR
- Networkx demo

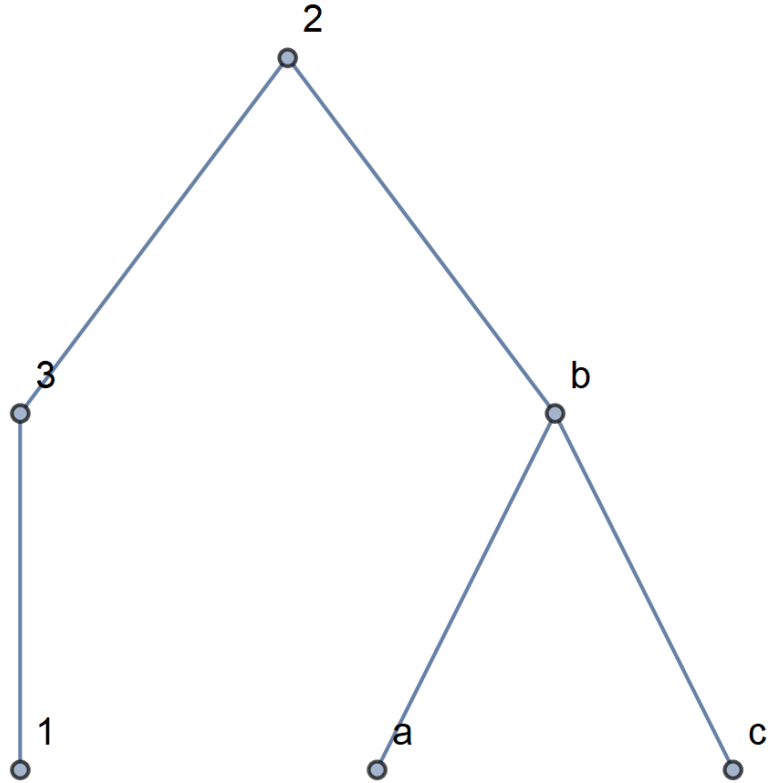
Graphs primer

A *simple* graph G consists of a set V and a set of pairs E. The pairs must be chosen from V, i.e. any $e \in E$ must be a subset $e \subseteq V$ and $|e| = 2$. V is the set of *vertices* of the graph and E is the set of its *edges*



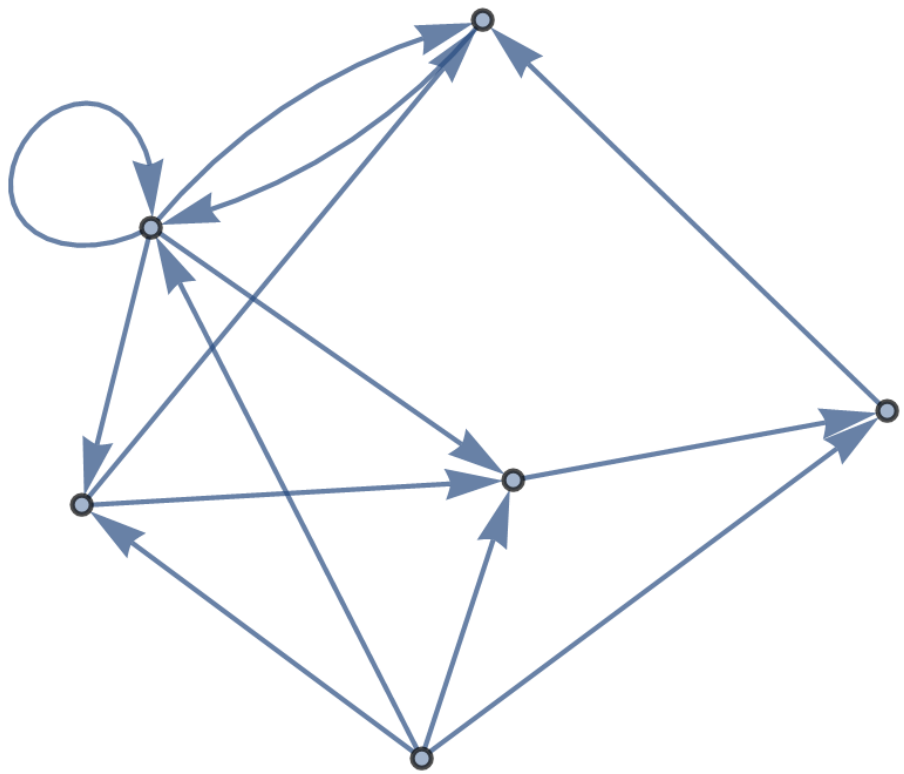
Graphs primer

This graph is described by $V = \{1, 2, 3, a, b, c\}$ and $E = \{\{1, 3\}, \{a, b\}, \{c, b\}, \{b, 2\}, \{2, 3\}\}$



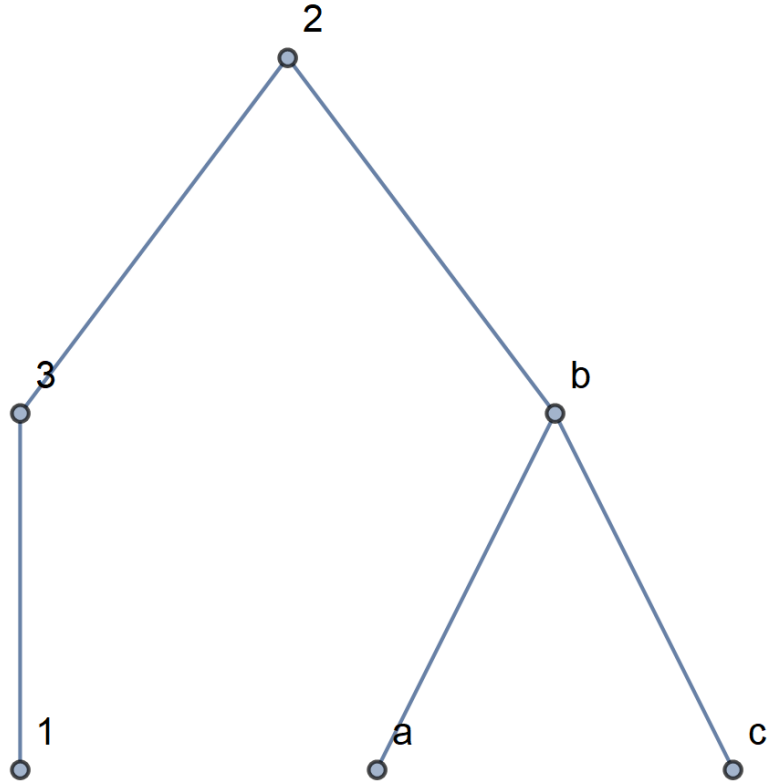
Graphs primer

There are many types of graphs; one can include loops and directions; a random example is shown below. In this case the graph is no longer considered simple. You can also have multiple edges, weighted edges, etc. We will focus on simple graphs.



Graphs primer

The degree of a vertex is the number of edges incident (connected) to it. Here $\deg(1) = 1, \deg(b) = 3$, etc.



Graphs primer

A simple graph G may be represented by a $N \times N$ matrix A, where $N = |V|$. For example, enumerating the vertices $V = \{v_1, v_2, \ldots, v_N\}$, the matrix elements are

$$A_{ij} = \begin{cases} 1, & \{v_i, v_j\} \in E \\ 0, & \text{otherwise} \end{cases} . \tag{1}$$

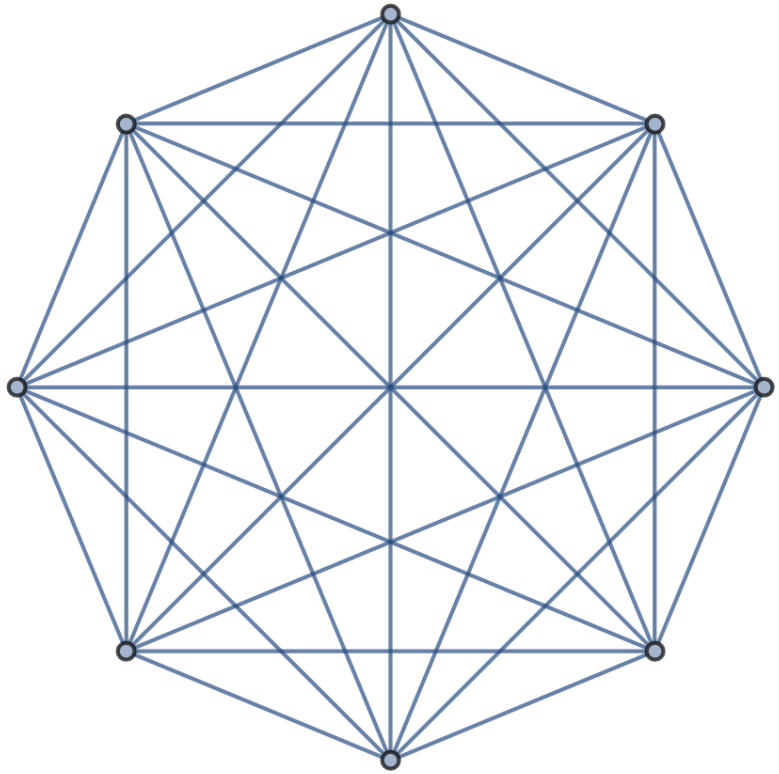
The matrix A is called the *adjacency matrix*. For a graph with weighted and directed edges, the elements of its adjacency matrix need not be restricted to 1s and 0s, and it does not have to be symmetric. **This is not the only matrix that one can construct from a given graph** G

Graphs primer

The adjacency matrix has some useful features. For example, suppose A as the adjacency matrix of a graph G and A' for G'. The two graphs are isomorphic if permuting the rows (or columns) of A gives A', i.e. the graphs G and G' can be drawn in identical manner. The study of graphs with their adjacency matrices is called *spectral graph theory*. Many useful results do not require spectral graph theory, and instead rely on the more intuitive visual representation; hence, we will not pursue this theory here.

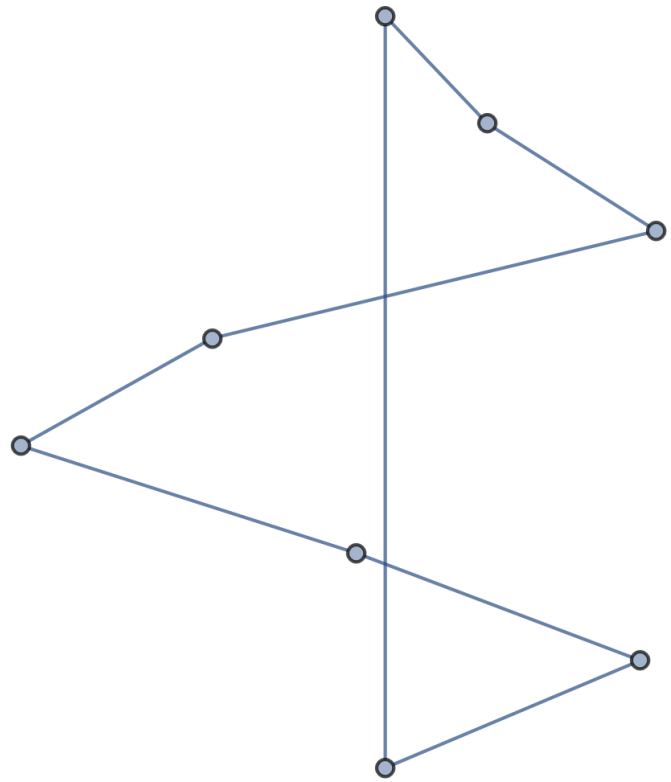
Special graphs

The *complete graph* with n vertices is the simple graph consisting of all possible edges, i.e. every vertex is adjacent (connected) to every other vertex. The complete graph with $n = 8$ is shown below



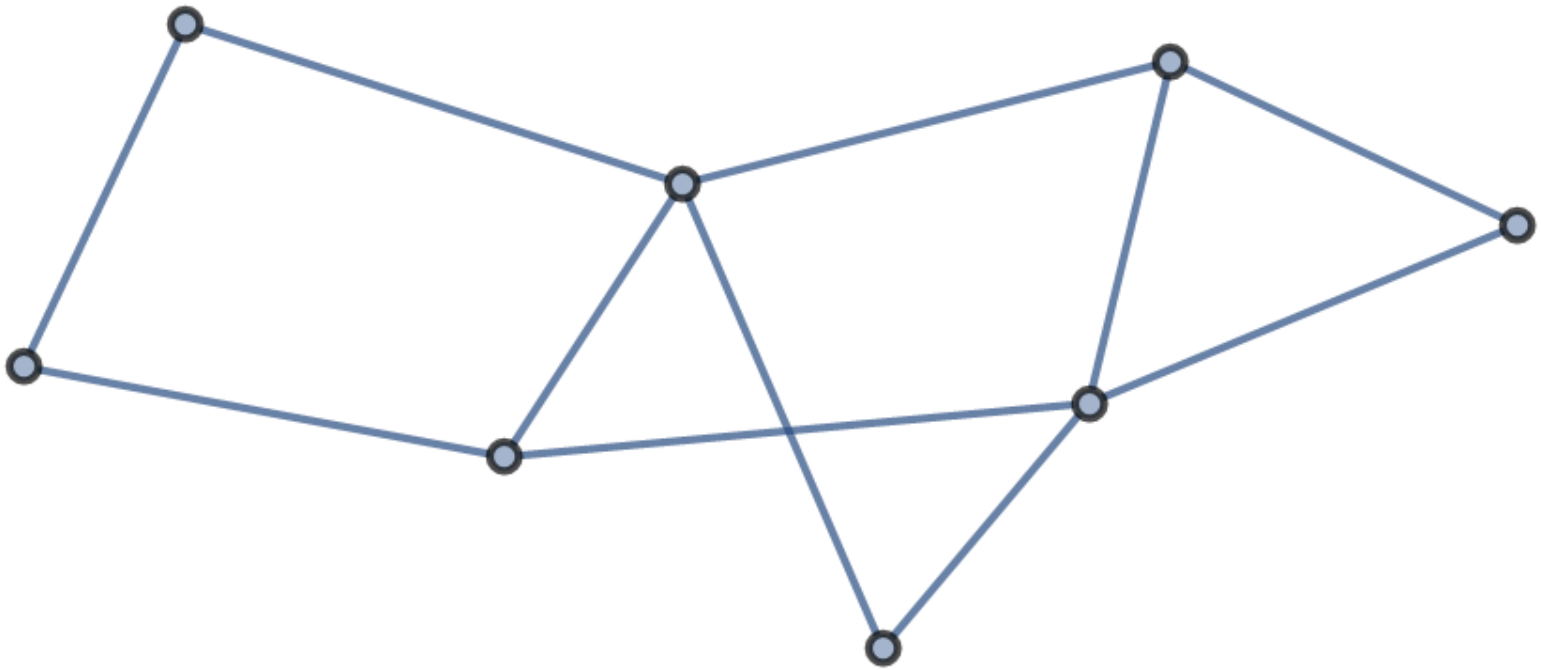
Special graphs

The *cycle graph* of n vertices consists of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and n edges "in sequence," $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_n, v_1\}$. Removing any one of these edges gives the *line graph* of n vertices. The cycle with $n = 8$ is shown below.



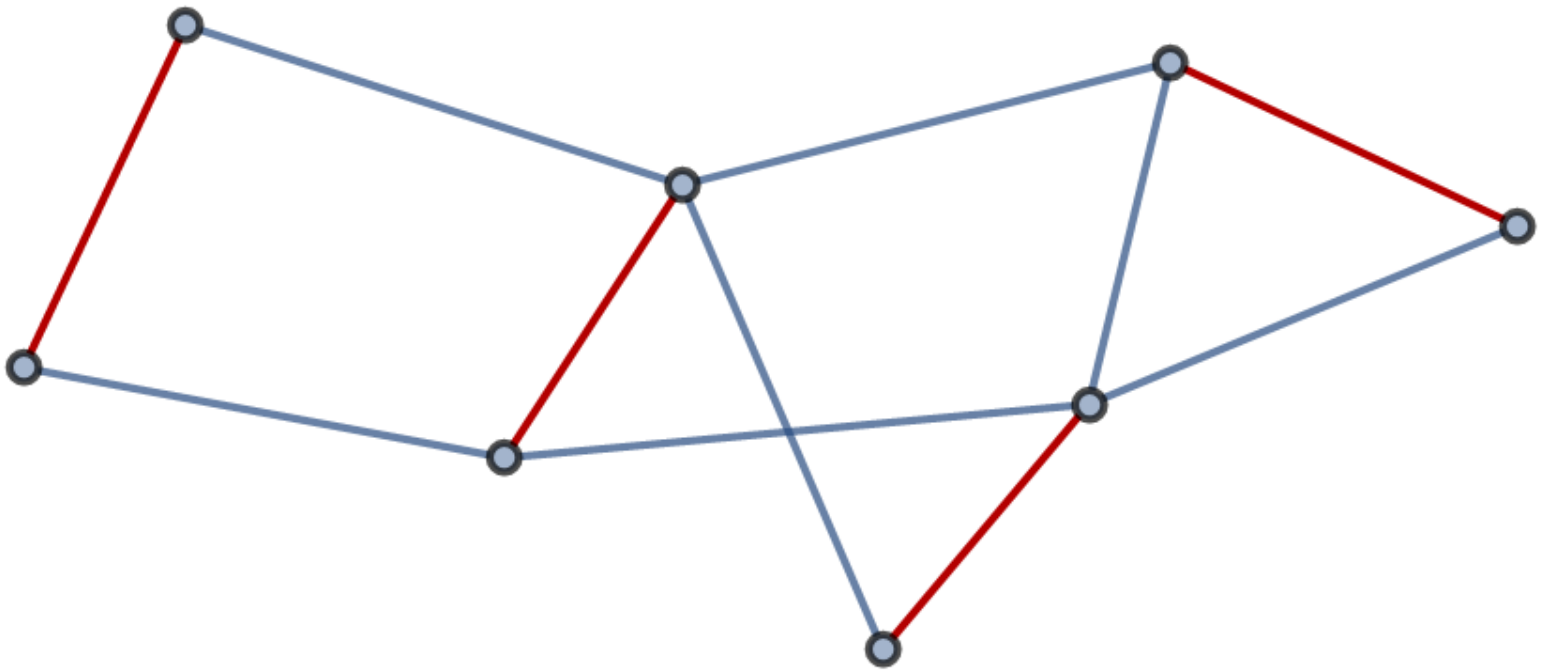
Application: matching

There are 8 vertices and 11 edges here (randomly generated). Can you find 4 edges such that every vertex is incident to one of these edges?



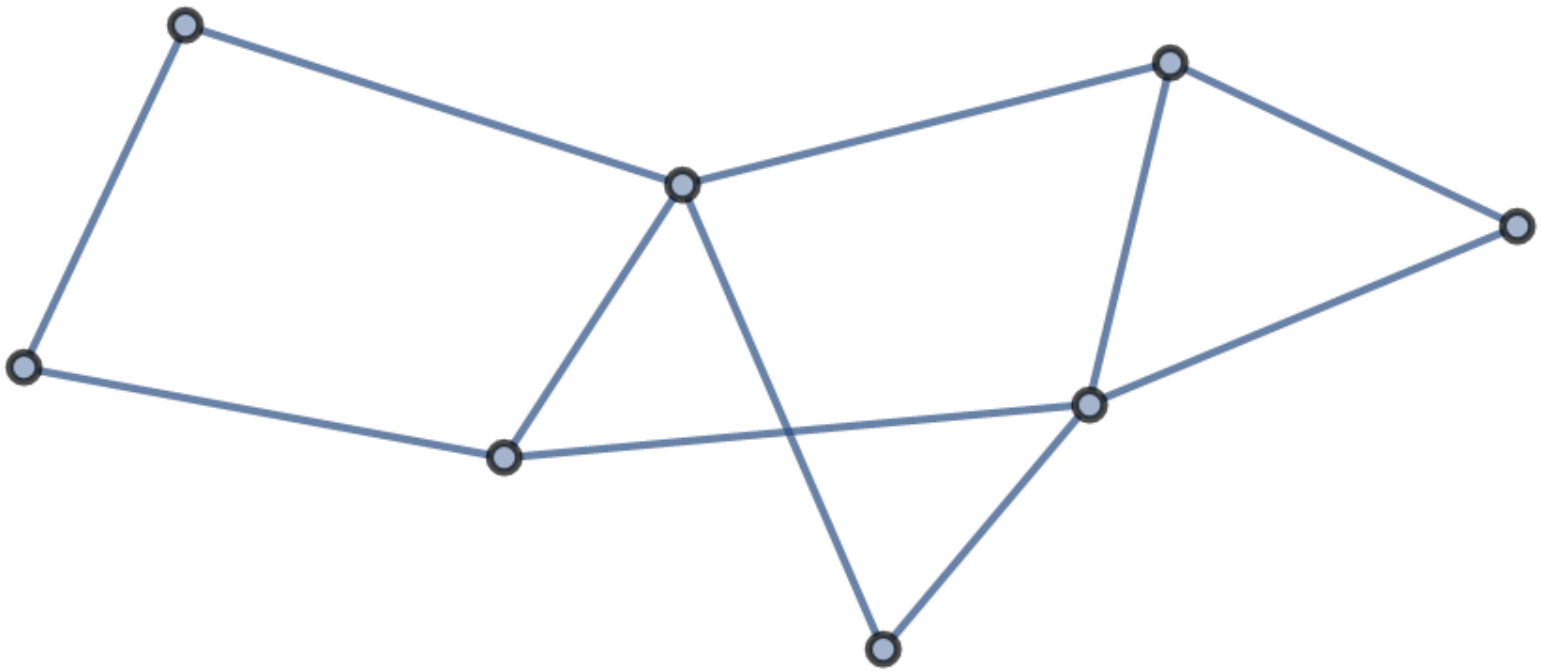
Application: matching

A subset of edges from a graph which share no vertices is called a *matching*. If the matching gets every vertex in the graph, it is said to be a *perfect* matching. This graph happens to have a perfect matching. The first example graph shown does not.



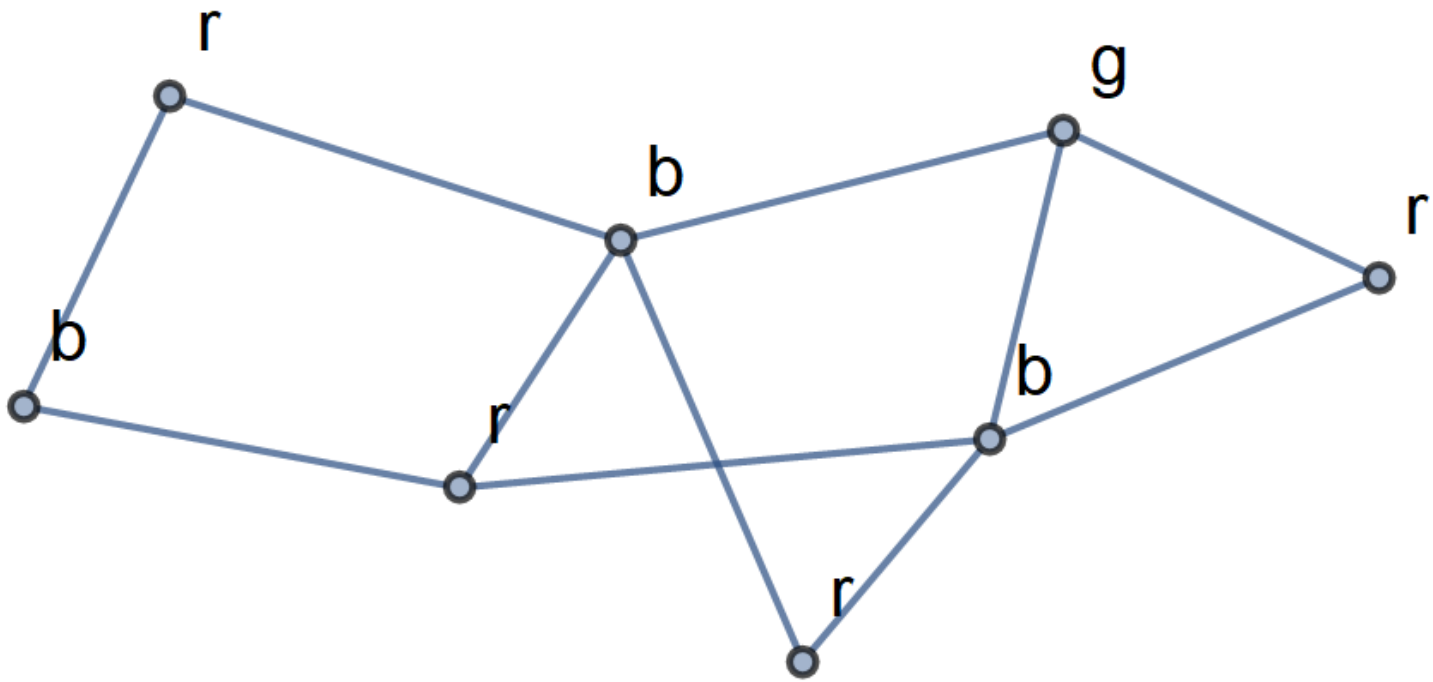
Coloring

Suppose you want to color the vertices such that any two adjacent vertices do not have the same color. What is the smallest number of colors you need?



Coloring

It's not very hard to see that you can do it with just 3 colors. Can it be done with 2?



Coloring

It cannot be done with 2 colors (the complete graph with 3 vertices is a subgraph!). The minimum number of colors is then 3, and this is referred to as the *chromatic number* of the graph. Currently there is no algorithm that is able to determine the chromatic number of an arbitrary simple graph in polynomial time.

Coloring

Applications:

- Scheduling classes: given that two classes cannot occur at the same time if they have at least one mutual student, what is the minimum number of time slots required to accommodate all classes?
- Allocating registers: upon compilation, variables will be assigned to registers on the processor for efficiency; any two variables that are needed at the same time should not be on the same register. What is the minimum number of registers required?

Coloring

A straightforward, greedy algorithm:

1. Put the vertices in some (chosen) sequence $\{v_1, \ldots, v_n\}$ and assign to v_1 the first color.
2. For every subsequent vertex v_i, check if assigning it to the first color works; if not, move on to the next, etc. If none of the current colors work, assign it a new color.

At the ith step, this routine can require checking for $i - 1$ conflicts, and thus at worst this algorithm has complexity $\mathcal{O}(n^2)$. This is definitely not guaranteed to give the minimum number of colors and the performance depends on the chosen sequence.

Coloring

The DSATUR (degree of saturation) algorithm is similar to the one just described (at each step, check all current colors to see if they can be used) but provides a sequence. Let the *saturation degree* of a vertex be the number of adjacent vertices which are colored.

1. Find the vertex with the highest saturation degree. If there are multiple such vertices, chosen among them the one with the highest degree. If there are multiple such vertices, choose one randomly from them.
2. For the chosen vertex, assign it a color by manually checking if any of the already used colors is viable, as previously described.

This has the same complexity $\mathcal{O}(n^2)$, but requires keeping track of more things like the saturation degree.

Coloring

DSATUR example, from Lewis (2016):

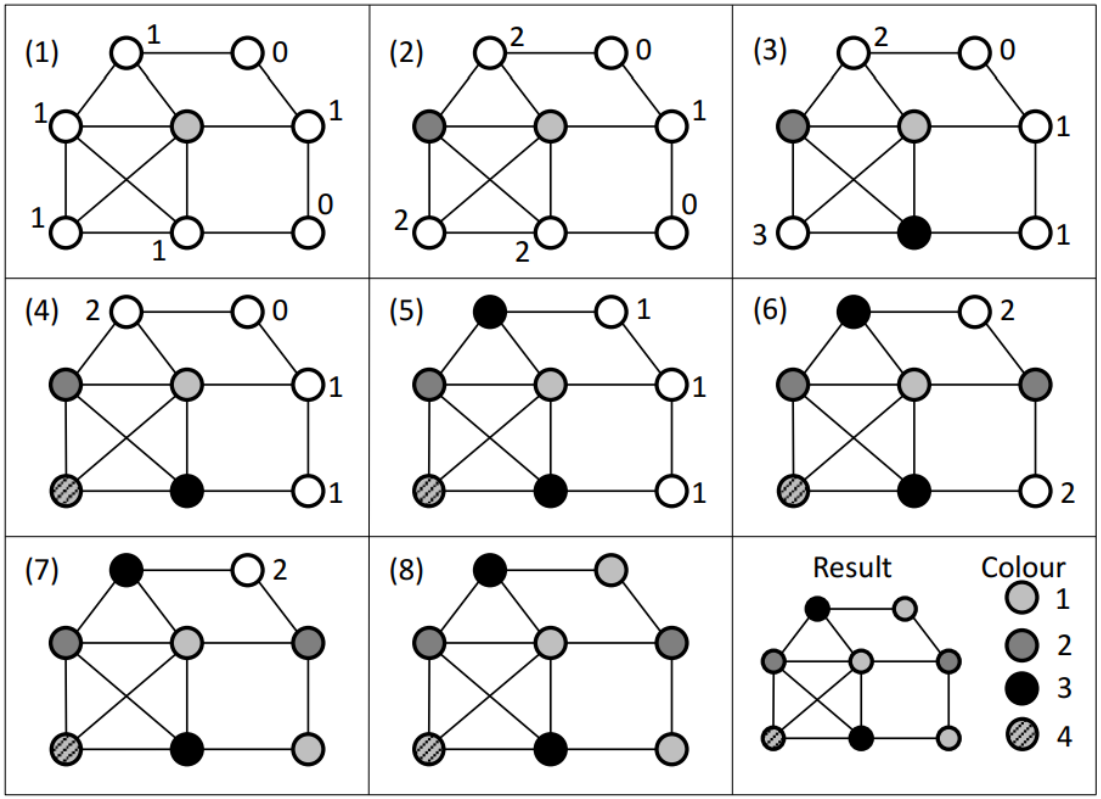


Fig. 2.9 Example application of DSATUR. Here, uncoloured vertices (members of X) are shown in white, and have their saturation degrees written alongside

Coloring

Demo in Networkx

Coloring

What can you expect?

- Suppose that, among all the vertices in a graph G the one with the largest degree is v. The procedure of assigning colors just by manually checking every previously colored vertex is guaranteed to give at most $\deg(v) + 1$ colors
- If the chromatic number of a graph happens to be 2, DSATUR will give an optimal coloring.
- If the graph happens to be a cycle, DSATUR will give an optimal coloring (2 or 3 colors). If the graph happens to be a wheel graph (take a cycle, arrange the vertices on a circle, put a vertex in the middle and connect it to all the others), DSATUR will give an optimal coloring (3 or 4 colors).
- There are certain graphs of a special topology that, in spite of being 3-colorable, DSATUR will color with n colors on $\mathcal{O}(n)$ vertices (Spinrad and Vijayan, 1985).

Summary

- Simple graphs (no weights, no directions) can be useful!
- Simple graphs pose difficult problems...
- Many commonly used algorithms are based on manually checking whether or not assigning a color conflicts with all previously colored vertices. The difference among them is the order in which you color the vertices.
- There is another algorithm, RLF, which when tested on random graphs, performs better on average than DSATUR for random simple graphs. I didn't realize this until I finished writing this talk...

References

- Tom Leighton and Martin van Dijk, *Mathematics for Computer Science*. MIT OpenCouseWare, https://ocw.mit.edu/.
- R. M. R. Lewis, *A Guide to Graph Coloring*. Springer.
- Networkx, https://networkx.github.io/.