ANALYSIS OF ALGORITHMS

LECTURE 6 : GRAPH COLOURING

WHAT?

- Map Colouring Problem
  - Assign colours to countries on a map such that no adjacent countries have the same colour

WHY?

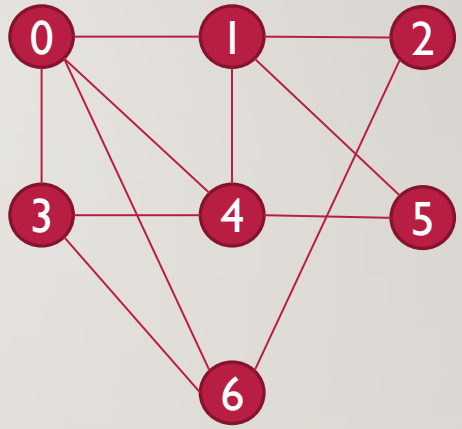
- Map Colouring Problem
  - Assign colours to countries on a map such that no adjacent countries have the same colour
- Why?
  - Visibility
  - Ease of production – Prefer solutions with a smaller number of colours

ABSTRACTION

- Let's examine the sentence again
- Assign **colours** to **countries on a map** such that no **adjacent countries** have the same **colour**
- We want to remove the problem domain from its specification to allow reuse
- Assign **colours** to **vertices** such that no **adjacent vertices** have the same colour

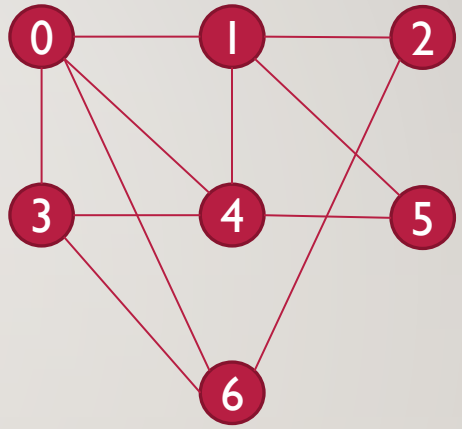
GRAPH COLOURING

- Assign colours to vertices such that no adjacent vertices have the same colour, using the minimum number of colours
- How do we do this?
  - Err… Nobody knows. Well not how to do it efficiently anyway.
  - NP-Complete



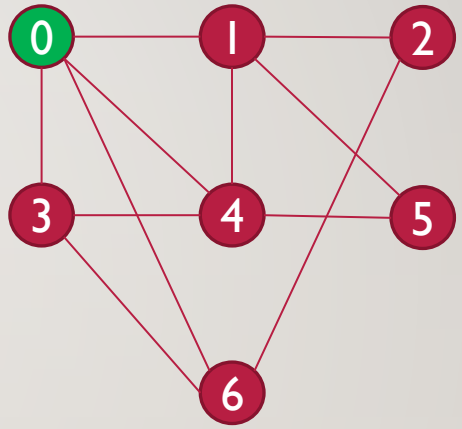
GRAPH COLOURING

- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



GRAPH COLOURING

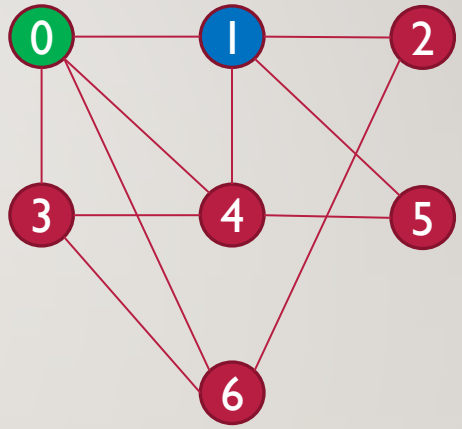
- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, Orange, White

GRAPH COLOURING

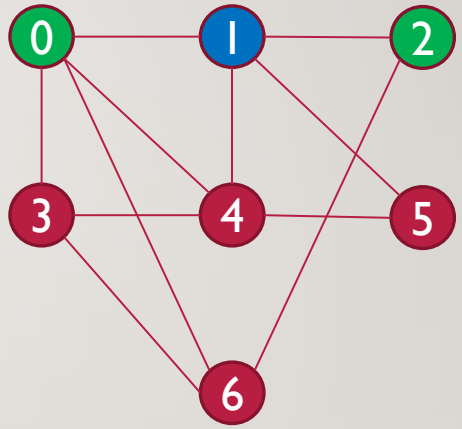
- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, Orange, White

GRAPH COLOURING

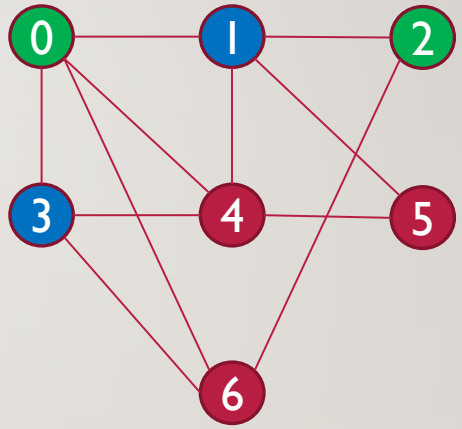
- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, Orange, White

GRAPH COLOURING

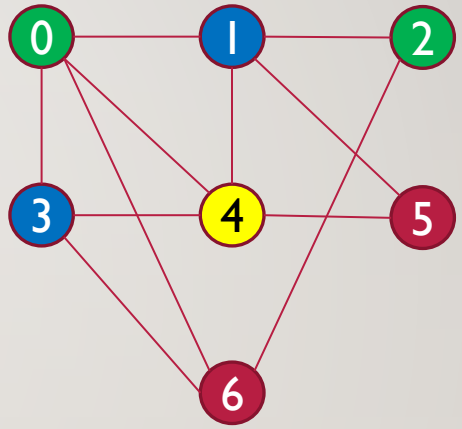
- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, Orange, White

GRAPH COLOURING

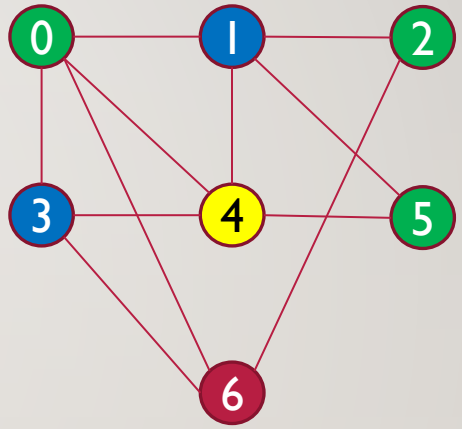
- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, Orange, White

GRAPH COLOURING

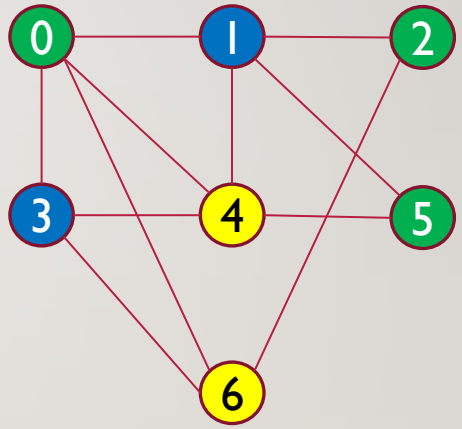
- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, Orange, White

GRAPH COLOURING

- Assign colours to vertices such that no adjacent vertices have the same colour. Prefer solutions that use fewer colours.
- How do we do this?
  - Pretty much the first thing we think of gets a valid colouring, and doesn't do too badly in terms of the number of colours.
  - While there is some uncoloured vertex
    1. Pick any vertex
    2. Set its colour to the first colour that no neighbour is using

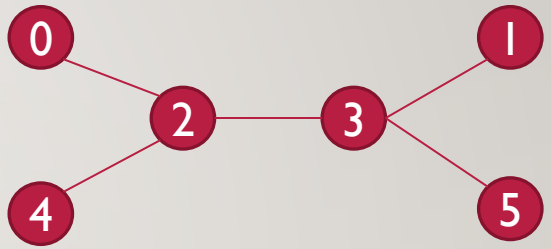


3 colours

Green, Blue, Yellow, Orange, White

GRAPH COLOURING

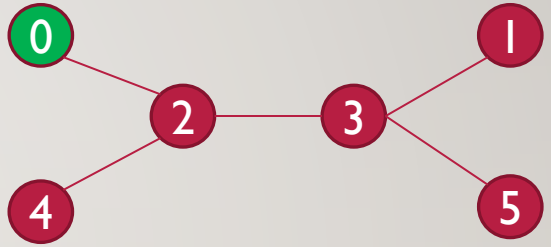
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

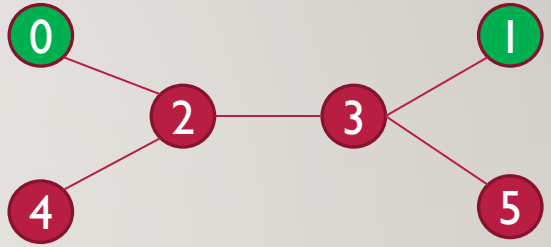
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

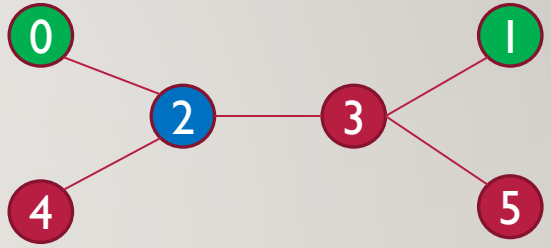
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

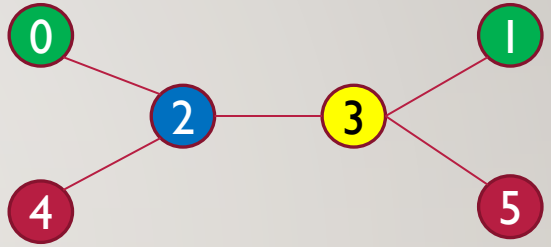
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

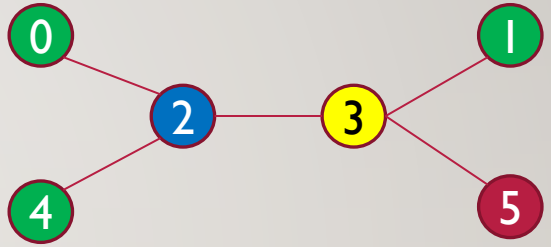
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

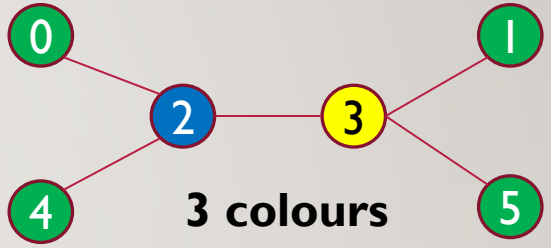
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

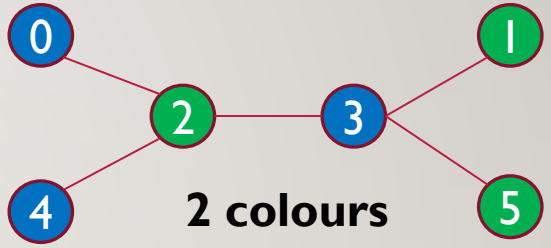
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING

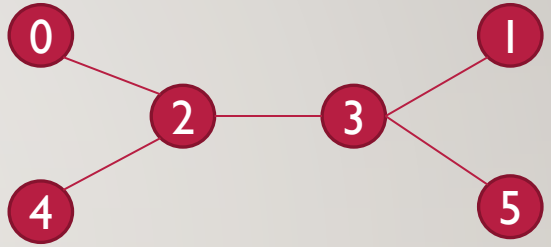
- Let's try another one
- While there is some uncoloured vertex
  1. Pick any vertex
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

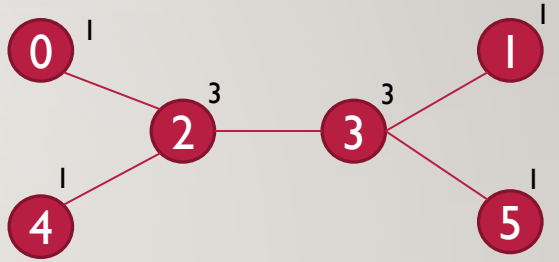
- The vertices are harder to colour based on how many edges they have
- Sort by degree



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

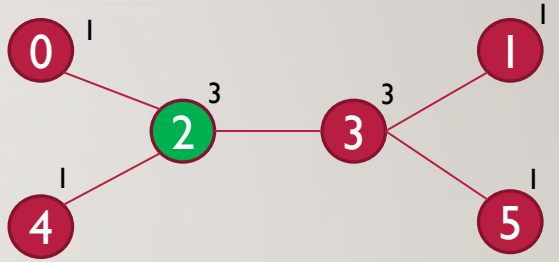
- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

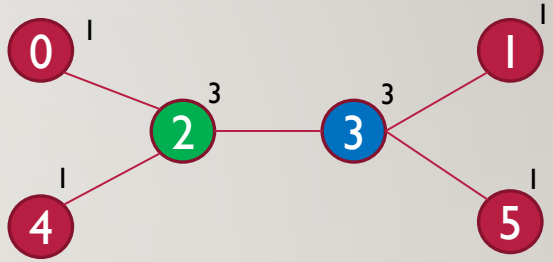
- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

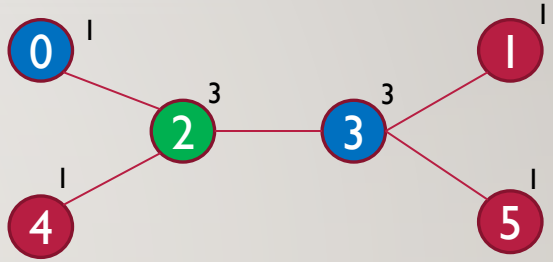
- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

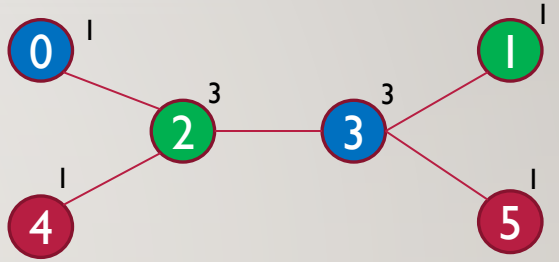
- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

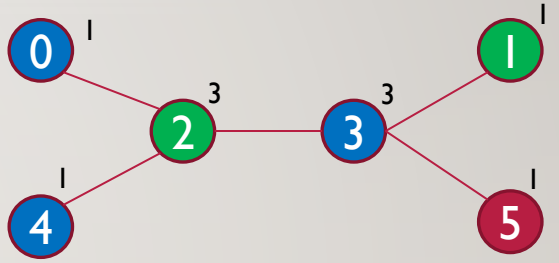
- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

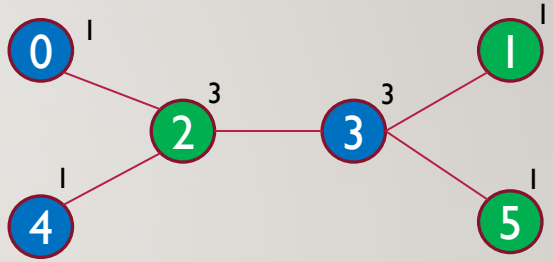
- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



Green, Blue, Yellow, White, Orange

GRAPH COLOURING (IMPROVED)

- The vertices are harder to colour based on how many edges they have
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using



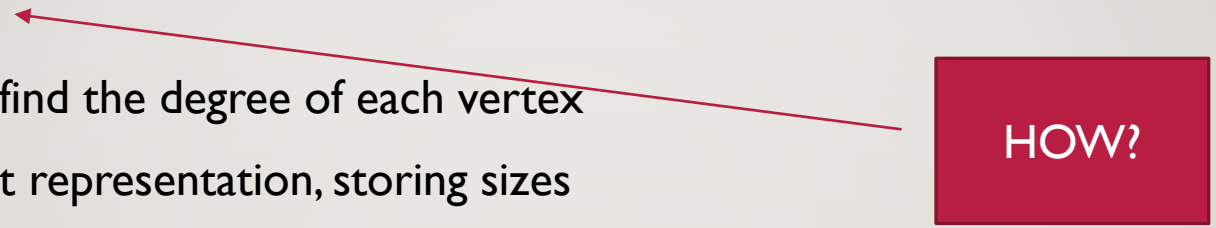
Green, Blue, Yellow, White, Orange

GRAPH COLOURING

- That is a fairly abstract algorithm…
- How do we make it more concrete?
- First, use numbers to represent colours
- Sort by degree
- While there is some uncoloured vertex
  1. Pick the uncoloured vertex with the largest degree
  2. Set its colour to the first colour that no neighbour is using

HOW?

- Sort by degree
  - Will have to find the degree of each vertex
  - Adjacency list representation, storing sizes
  - Then sort using some algorithm… Mergesort? Quicksort?
  - In Java, we can use the built-in **sort** function - $\Theta(nlogn)$

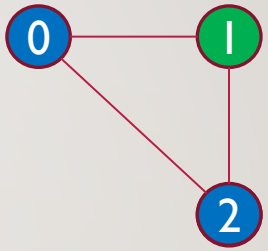


HOW? ATTEMPT 1

- Set its colour to the first colour that no neighbour is using
  - Could keep a variable for the smallest colour s, initially set to 0, indicating that no colours are used
  - Iterate through the neighbours, and increment s if a neighbour is using it
  - $\Theta(n)$

HOW? ATTEMPT 1

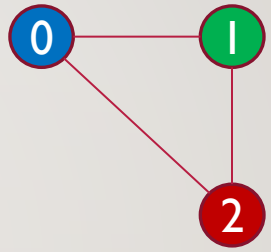
- Set its colour to the first colour that no neighbour is using
  - Could keep a variable for the smallest colour *s*, initially set to 0, indicating that no colours are used
  - Iterate through the neighbours, and increment *s* if a neighbour is using it
  - FAILS! In this case, the *s* will initially be 0. Iterating through the list, we see vertex 0 uses colour 1, so we leave s as *0*. Then we see vertex 1 uses colour 0, so we set *s* to 1.
  - Which means we will set vertex 2's colour to 1 (Blue) which is already used.



0. Green
1. Blue
2. Yellow

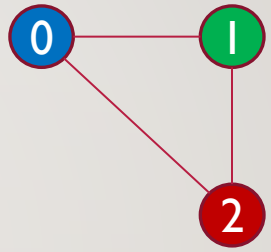
HOW? ATTEMPT 2

- Set its colour to the first colour that no neighbour is using
  - Could go back and check the colour every time s changes. So when s changes to 1, we start iterating from the top of the list again
  - Could be $\Theta(mn)$, potentially $\Theta(n^2)$ in the worst case



HOW? ATTEMPT 3

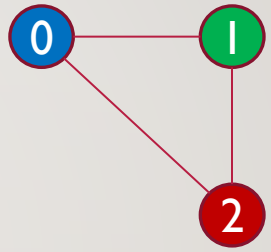
- Set its colour to the first colour that no neighbour is using
  - Could keep track of all colours in a used array, initially set to false.
  - Iterate through all neighbours, setting their colours to true in the list
  - Then search for the first false (unused)



0 (Green)	1 (Blue)	2 (Yellow)
F	F	F

HOW? ATTEMPT 3

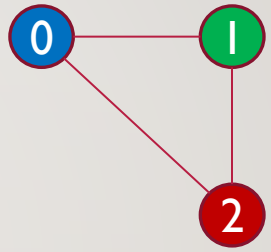
- Set its colour to the first colour that no neighbour is using
  - Could keep track of all colours in a used array, initially set to false.
  - Iterate through all neighbours, setting their colours to true in the list
  - Then search for the first false (unused)



0 (Green)	1 (Blue)	2 (Yellow)
F	T	F

HOW? ATTEMPT 3

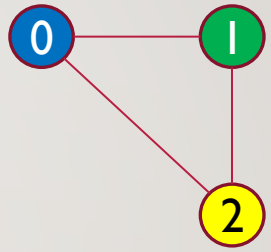
- Set its colour to the first colour that no neighbour is using
  - Could keep track of all colours in a used array, initially set to false.
  - Iterate through all neighbours, setting their colours to true in the list
  - Then search for the first false (unused)



0 (Green)	1 (Blue)	2 (Yellow)
T	T	F

HOW? ATTEMPT 3

- Set its colour to the first colour that no neighbour is using
  - Could keep track of all colours in a used array, initially set to false.
  - Iterate through all neighbours, setting their colours to true in the list
  - Then search for the first false (unused)
  - $O(n)$



0 (Green)	1 (Blue)	2 (Yellow)
T	T	F

GRAPH COLOURING (IMPROVED)

- Sort by degree $\Theta(nlogn)$
- While there is some uncoloured vertex – runs n times
  1. Pick the uncoloured vertex with the largest degree - $\Theta(1)$
  2. Set its colour to the first colour that no neighbour is using - $O(n)$
- Total - $O(nlogn + n^2) = O(n^2)$
- Don't worry if you're not sure if you understood it, because you have to code it this week, so you will understand it after <evil laugh>

HMM… INTERESTING…

- 4 colour theorem
- Next up:
  - Proof of *r*-colourability