Algorithms: Correctness and Complexity

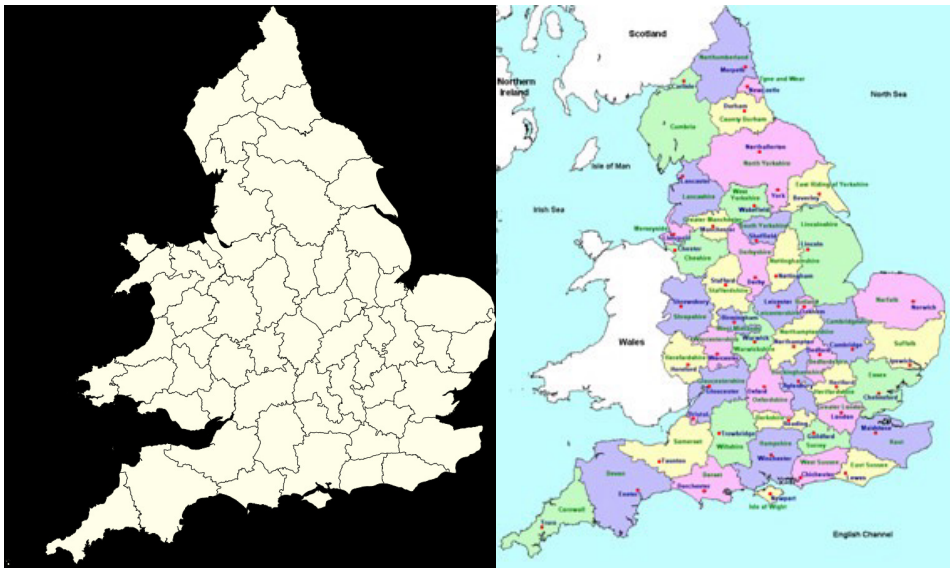
By the end of this course you will

- Have a better understanding of how to design algorithms.
- Have a better understanding of how to design algorithms that work
- and that you can demonstrate work.
- Have a better understanding of how to compare the efficiency of algorithms.
- Have a better understanding of how to design efficient algorithms.
- Be better able to recognize problem patterns.
- Be better able to apply solution patterns.
- Understand the limits of algorithms:
  - Some problems have no algorithmic solution.
  - Some problems have no efficient algorithmic solution.

3 Problems

The map colouring problem

In 1852 Francis Guthrie noticed that he could colour a map of the counties of England using only 4 colours without two adjacent counties being coloured the same.



We ignore bodies of water, as they are usually assigned an extra colour (not counted) not used for territories.

Exclaves: An exclave is a part of a territory separated from the main part.

(For example Kalingrad Oblast is an exclave of Russia, creating borders with Lithania and Poland that otherwise it would not have.)

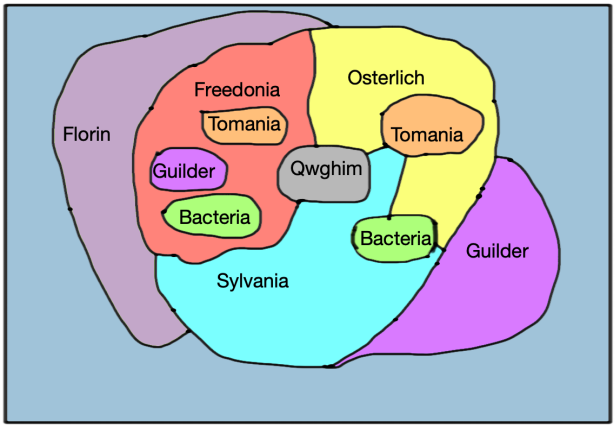
Some questions?

- Can all maps (without exclaves) be coloured with 4 colours?
  - (This is the 4-color conjecture.)
- How can we find a colouring or decide that there isn't one, given a map (possibly with exclaves) and a set of colours?

Note:

- The first question is a yes/no question.
- An answer to the second question is an algorithm.

Given this map



and 4 colours, the algorithm should find a 4-colouring or declare that none exists.

The exam scheduling problem

A medium sized university has 20,000 students taking 1,800 courses with 1,500 final exams.

There are 48 (nonoverlapping) exam slots.

- Can the exams be scheduled so that no student has two exams at the same time?
- What is the minimum number of time slots required?

In general:

- How can we find an assignment of exams to slots, given a set of exams, a set of slots, and the set of pairs of exams that share at least one student, or determine that there isn't one?

The register allocation problem

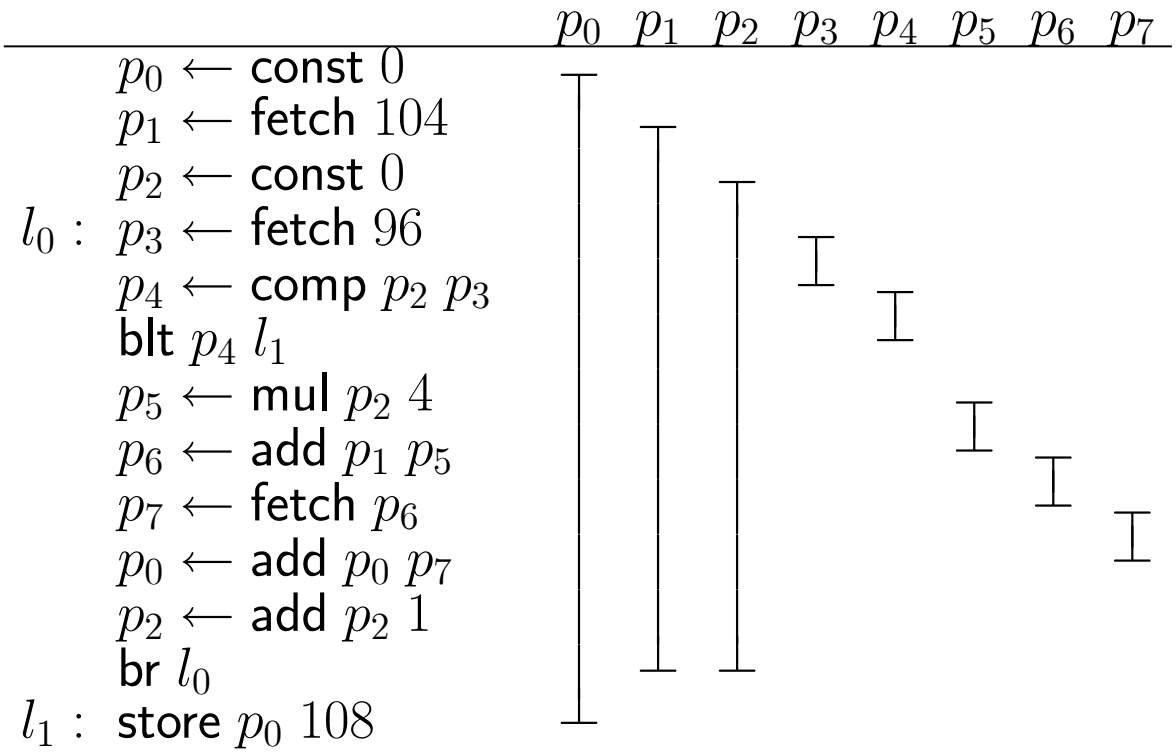
Many compilers generate assmbly (or machine) code in multiple passes. Here is a typical pass structure

- Parse: Parse the code to create a tree representation of the source code.
- Analyse: Analyse the tree, looking for errors and inferring types for nodes.
- Generate: Generate machine code, pretending that the machine has an unbounded set of registers $\{p_0, p_1, p_2, ..p_\infty\}$
- Register allocation: For each pretend register, assign a real register from $\{r_0, r_1, r_2, .., r_k\}$ to represent it. And if that is not possible introduce “register spilling code” and try again until successful. Then rewrite the code.

We'll look at the Register Allocation pass.

Here is an example of output from the Generate pass.

The "live range" of each register is shown at right.



Can we renumber the registers so that only 4 register are used without chaning the meaning of the code?

To avoid changing meaning, no two contemporaneous pretend register can be assigned to the same real register.

Yes! This mapping will work

$$\left\{\begin{array}{l} p_0 \mapsto r_0, \\ p_1 \mapsto r_1, \\ p_2 \mapsto r_2, \\ p_3 \mapsto r_3, \\ p_4 \mapsto r_3, \\ p_5 \mapsto r_3, \\ p_6 \mapsto r_3, \\ p_7 \mapsto r_3 \end{array}\right\}$$

$$\begin{array}{ll} & p_0 \leftarrow \mathsf{const}\ 0 \\ & p_1 \leftarrow \mathsf{fetch}\ 104 \\ & p_2 \leftarrow \mathsf{const}\ 0 \\ l_0: & p_3 \leftarrow \mathsf{fetch}\ 96 \\ & p_4 \leftarrow \mathsf{comp}\ p_2\ p_3 \\ & \mathsf{blt}\ p_4\ l_1 \\ & p_5 \leftarrow \mathsf{mul}\ p_2\ 4 \\ & p_6 \leftarrow \mathsf{add}\ p_1\ p_5 \\ & p_7 \leftarrow \mathsf{fetch}\ p_6 \\ & p_0 \leftarrow \mathsf{add}\ p_0\ p_7 \\ & p_2 \leftarrow \mathsf{add}\ p_2\ 1 \\ & \mathsf{br}\ l_0 \\ l_1: & \mathsf{store}\ p_0\ 108 \end{array} \qquad \begin{array}{ll} & r_0 \leftarrow \mathsf{const}\ 0 \\ & r_1 \leftarrow \mathsf{fetch}\ 104 \\ & r_2 \leftarrow \mathsf{const}\ 0 \\ l_0: & r_3 \leftarrow \mathsf{fetch}\ 96 \\ & r_3 \leftarrow \mathsf{comp}\ r_2\ r_3 \\ & \mathsf{blt}\ r_3\ l_1 \\ & r_3 \leftarrow \mathsf{mul}\ r_2\ 4 \\ & r_3 \leftarrow \mathsf{add}\ r_1\ r_3 \\ & r_3 \leftarrow \mathsf{fetch}\ r_3 \\ & r_0 \leftarrow \mathsf{add}\ r_0\ r_3 \\ & r_2 \leftarrow \mathsf{add}\ r_2\ 1 \\ & \mathsf{br}\ l_0 \\ l_1: & \mathsf{store}\ p_0\ 108 \end{array}$$

The problem:

- How can we find an assignment of pretend registers to real registers, given a set of pretend registers, a set of real registers, and the set of pairs of pretend registers that are contemporaneous, or determine that there isn't one?

Problem abstraction

We can see that the following problems are equivalent

- Colour a map with k colours.
- Assign exams to k slots.
- Renumber the registers of machine code to k registers.

We'll represent all these problems with a new problem:

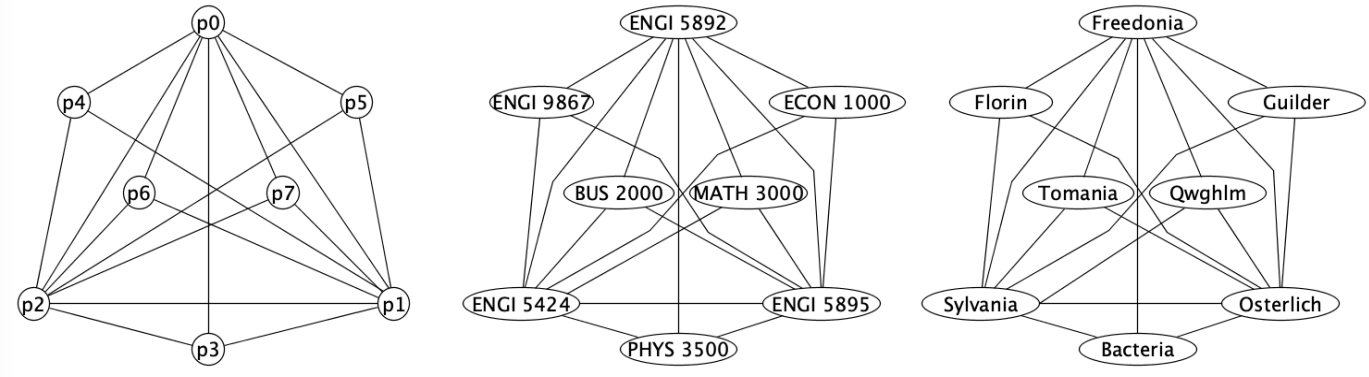
Node colouring a graph

Given an undirected graph (V, E) and a number k, is there a mapping from V to a set of k colours so that no two nodes connected by an edge are mapped to the same color.

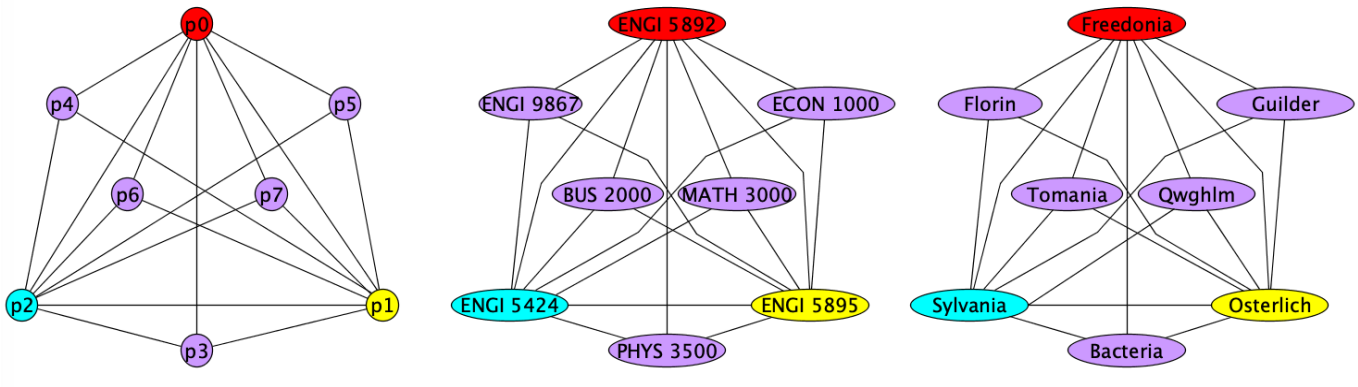
$$\forall\,(p, q) \in E \cdot m(p) \neq m(q)$$

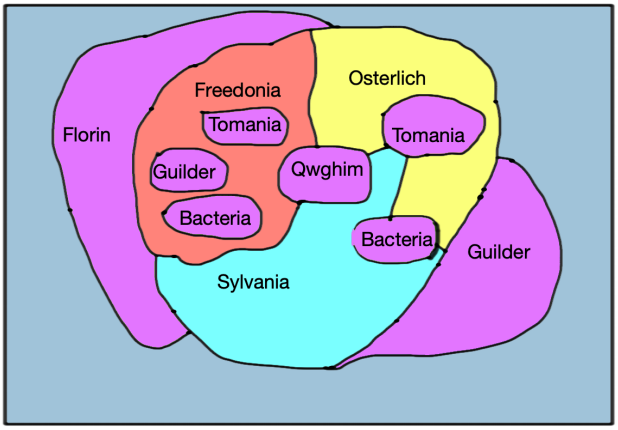
Nodes	Edge between	Colours
Territories	Border between	Colours
Courses	Share student	Slots
Pretend registers	Contemporaneous	Real Registers

These problem instance are equivalent



A solution to one instance is a solution to all.





An algorithm for one problem will work for all.

Can we find an algorithm?

A backtracking algorithm

Represent a map from nodes to colours by a set of node/int pairs. $\{(v_0, 0), (v_1, 0), (v_2, 1)$

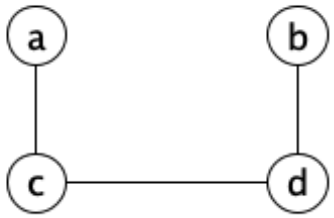
Represent failure with a special value $Failure$ and success by a set M representing a map.

This algorithm exhaustively searches for a solution.

```
fun color(V : Set[Node], E : Set[Edge], k : Int)
    if V = ∅ then return ∅
    else
        let v be any member of V
        return colorRec(V − {v} , E, {(v, 0)})
```

```
fun colorRec(V′, E, k,M : Set[Node × Int])
    if V′ = ∅ then
        return M
    else
        let v be any member of V′
        for c ∈ {0, ..k} // from 0 up to (but not including k)
            if no edge in E joins v to a node u such that
            (u, c) ∈ M
                let M′ := M ∪ {(v, c)}
                let r := colorRec(V′ − {v} , E, k, M′)
                if r ≠ Failure then // It's a success
                    return r
        return Failure
```

Simulate the algorithm on the previous page for this graph and 2 colours.



Some questions about this algorithm and the node colouring problem?

Does the algorithm work?

- How can we define what "work" means?
- How could we prove that it works?

Is the algorithm efficient?

- What counts as efficient?
- How many steps does it take (worst case) attempt to colour a graph with n nodes with k colours?
- What flavour is this function: linear? cubic? worse?

Can we prove that there is no efficient algorithm?

Can we prove that, if there is an efficient algorithm for this problem, then there are efficient algorithm for some other problem that is generally regarded as difficult?

Is there an efficient algorithm that is approximately correct?

- E.g., one that might fail when it shouldn't, but always succeeds when k is twice the 'chromatic number".
- (The chromatic number $\chi(G)$ of a graph G is the minumum number of colours needed to colour G.)

Is there an efficient algorithm that works for some graphs?

- E.g., one that works for all planar graphs?
- (Planar graphs can be drawn on a plane with no two edges crossing.)