CSCI 136
Data Structures & Advanced Programming

Lab : Exam Scheduling

High-level Overview

Input: a set of student schedules

Output: a schedule (set of time slots) for exams where

- Every course is in some time slot
- No student has two exams in the same time slot
- The number of time slots is as small as possible

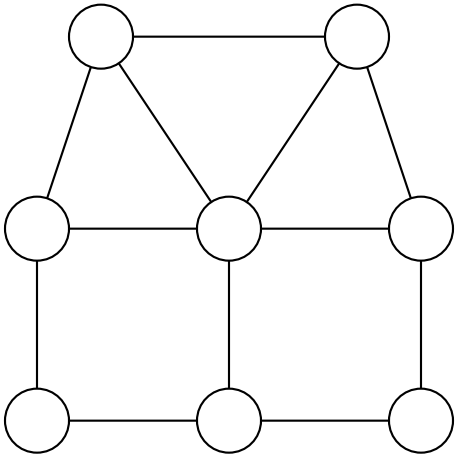
We can complete this task by representing the data using a graph, and then manipulating the graph using a greedy algorithm.

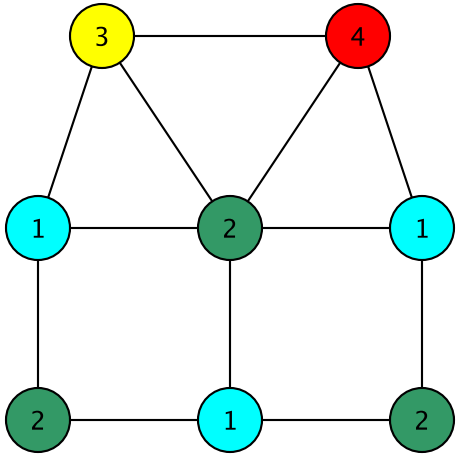
Greedy Algorithms

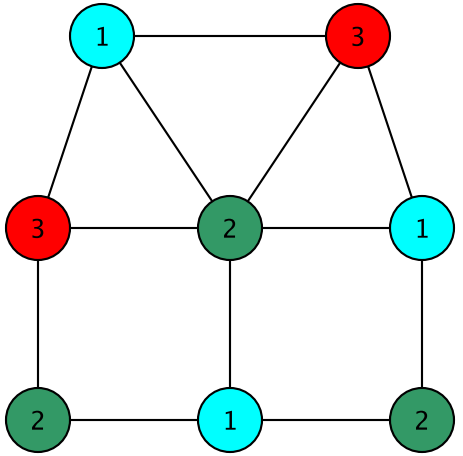
- A greedy algorithm attempts to find a globally optimum solution to a problem by making locally optimum (greedy) choices

- Example: Graph Coloring
  - A (proper) coloring of a graph G = (V,E) is an assignment of a value (color) to each vertex so that adjacent vertices get different values (colors)
  - (Typically, we would try to minimize the number of colors we use)

Greedy Coloring





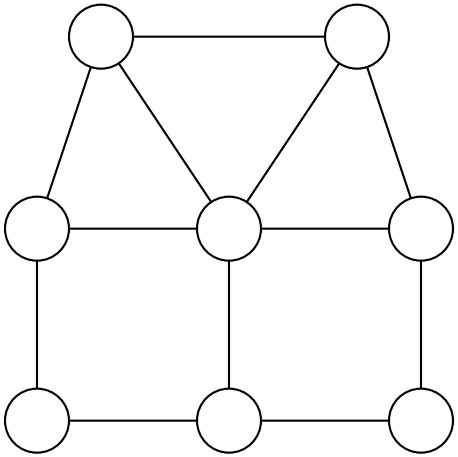


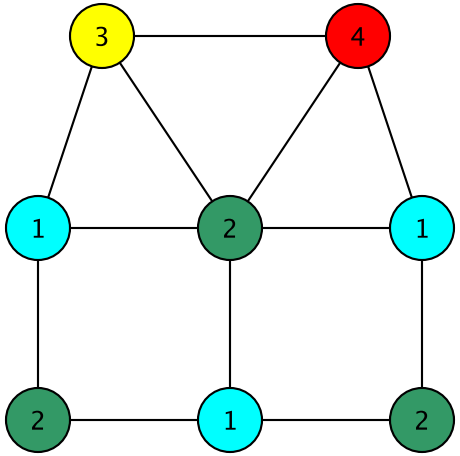
Greedy Coloring : Pseudocode

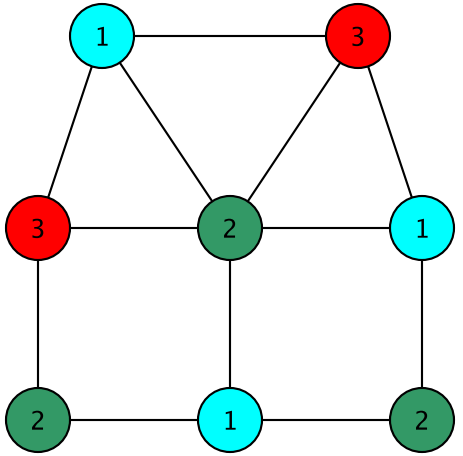
(Builds a structure C of lists of vertices, where each list represents vertices of a single "color")

```
let C = a structure to hold a collection of lists
while G is not empty:
      let L = a new empty list;
      pick some vertex v in G;
      L.add(v);
      for each vertex u ≠ v in G:
            if u is not adjacent to any vertex of L:
                  add u to L
      remove all vertices of L from G
      add L to C
Return C as the coloring
```

Greedy Coloring







Greedy Coloring Observations

- Each list (color class) `L` is a set of vertices, no two of which are adjacent (an *independent set*)

- Each color class is maximal: cannot be made any larger

(if it could have, we would have added that vertex to our list!)

  - The hope is that this results in fewer colors being needed
  - But the solution is not always optimum!

Lab : Exam Scheduling

Goal: Find a schedule (set of time slots) for exams so that

- No student has two exams in the same time slot
- Every course appears in exactly one time slot
- The number of time slots is as small as possible

This is just the graph coloring problem in disguise!

- Each course is a vertex
- Two vertices are adjacent if the courses share student(s)
- A slot must be an independent set of vertices (that is, a color class)

Lab Notes: Using Graphs

- Create a new graph in structure5:
  - GraphListDirected, GraphListUndirected,
  - GraphMatrixDirected, GraphMatrixUndirected

```
Graph<V,E> conflictGraph = new GraphListUndirected<V,E>();
```

Lab : Useful Graph Methods

- `void add(V label)`
  - add vertex to graph
- `void addEdge(V vtx1, V vtx2, E label)`
  - add edge between vtx1 and vtx2
- `Iterator<V> neighbors(V vtx1)`
  - Get iterator for all neighbors to vtx1
- `boolean isEmpty()`
  - Returns true iff graph is empty
- `Iterator<V> iterator()`
  - Get vertex iterator
- `V remove(V label)`
  - Remove a vertex from the graph
- `E removeEdge(V vLabel1, V vLabel2)`
  - Remove an edge from graph