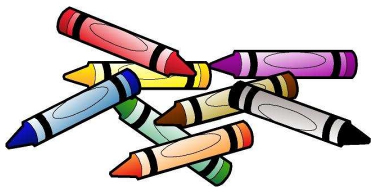
CSC112 Fundamentals of Computer Science Spring 2016
Lab 7 – Recursion: Graph Coloring

In this lab we will use recursion to solve a famous problem in graph theory known as the **graph coloring problem**.

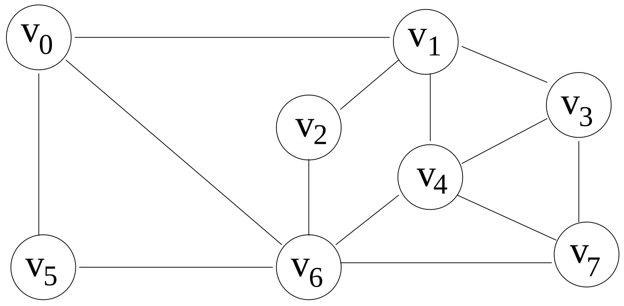
Let $G = (V, E)$ be an undirected graph. In this notation, $V = \{ v_1, v_2, v_3, ..., v_n \}$ is the set of vertices of the graph and $E = \{ e_1, e_2, e_3, ..., e_m \}$ is the set of edges.

The *graph coloring problem* is stated as follows: You are given k colors and your task is to assign colors to the vertices so that no two adjacent[1] vertices have the same color. It sounds easy, but a key aspect of the problem is that (assuming three or more colors): when you choose a color for a vertex (after the first vertex), you have no way of knowing whether your choice will lead to a solution.



A related problem is finding the **chromatic number** of a graph. The *chromatic number* of a graph is the minimum number of colors needed to solve the graph coloring problem.

You can try your hand at graph coloring on the graph illustrated below. Can this graph be colored using 4 colors ? Can it be colored with 3 colors ? How about 2 colors ?



Graph coloring falls into a class of problems called $\mathcal{NP}$-complete problems. The theory of $\mathcal{NP}$-complete problems is well beyond the scope of this course. Suffice it to say, that after 70+ years of trying, the only known algorithms for all of the $\mathcal{NP}$-complete problems amount to "***guess at an answer and then check if you guessed correctly.***" If you are unlucky at guessing you may end up trying all possibilities. The number of possible guesses using k colors on a graph with n vertices is k^n. I.e., the number of things to check grows exponentially with the number of vertices.

So how do I systematically try all possibilities ?

The graph coloring problem can be solved by a **recursive backtracking** algorithm. At each level in the recursion, consider a new vertex. A color choice is made for that vertex, and the consequences of that choice are explored recursively. If the recursive exploration leads to a solution, GREAT – you have found a solution. You must have guessed correctly. However if the recursive exploration does not lead to a solution, then you must un-do your choice and make another choice. If you have exhausted all choices you must return an indication of failure. That will lead a higher level of recursion to attempt a different choice.

[1]Two vertices are adjacent if and only if there is an edge between them.

If the top level of recursion has exhausted all its choices and none of them led to a solution, then no solution exists.

See the attached handout for a description of a general backtracking algorithm. For the graph coloring problem, the base case occurs when all of the vertices have been successfully assigned a color. As you examine the possibilities for each vertex, it is pointless to consider colors which are already assigned to an adjacent vertex. Skip those colors. Each color which is distinct from the adjacent vertices' colors is a potential correct choice for the current vertex, Loop over all such potentially correct colors and recursively explore the consequences.

Input Input is the number of colors (given as a command line parameter), and a graph. The graph will be given to you as a file with the following format: The first number in the file is the number of vertices n. The subsequent n lines of the file will represent the adjacency matrix of the graph G. Each line is a sequence of space-separated 0's and 1's giving the adjacency matrix in row-major order. An value of 1 in row i and column j in the adjacency matrix indicates an edge between vertices v_i and v_j. An value of 0 indicates no edge.

Output Your output should be a table with two columns. The left column contains the vertex numbers, $0, 1, 2, 3, 4..., n-1$. The right column contains the color assigned to that vertex. Colors must be numbered $0, 1, 2, 3, 4..., k-1$.

Other Program Requirements

Your program must:

1. Be written in an object oriented style.
2. INDENT the statements to reflect the effects of control statements. I.e., `if`, `else`, `while`, `for`, etc.
3. Accept input parameters on the command line.
   - Your program must run correctly with:

     ```
     % a.out k filename
     ```

     where `k` is the number of colors, and `filename` is the name of a file representing the graph adjacency matrix.
4. Check for errors whenever allocating memory and whenever opening a file.

Who cares about coloring anyway ?

Graph coloring is actually a very useful problem to solve. A **compiler** is a program that translates your high-level language into machine instructions. Consider the following example of "machine" instructions (on the next page):

```
(1)     input a
(2)     input b
(3)     c = a + b
(4)     d = a * b
(5)     x = c + 1
(6)     y = 2 * d
(7)     z = x + d
(8)     w = z + y
(9)     return w
```

As the machine instructions proceed, values are created. For example, line (3) creates the value `a+b` and stores it in `c`. The last use of that value occurs on line (5); after that, the value `a+b` is no longer referenced. So, we can think of lines (3) through (5) as the **lifetime** of the value created on line (3). In general, the lifetime of a value is the sequence of instructions from the point where the value is first created to the last use of the value. The lifetime of the value `a*b` created on line (4) extends to line (7).

For efficiency, temporary values (e.g., `a+b`) should be stored in a general purpose **register** within the CPU.[2] Obviously, if the lifetime of two values overlap, then those two values must be stored in two different registers.[3] Unfortunately, registers are in short supply.[4]

One of the tasks that a compiler must perform is to assign registers to store the temporary values created by a sequence of operations. This is called the **register allocation problem**. As it turns out, the register allocation problem is identical to graph coloring. We construct a graph as follows: Create a vertex in a graph for each value created by the instructions under consideration. Whenever the respective lifetimes of two values overlap, we add an edge between the two vertices representing those values.[5]

The available pool of registers in the register allocation problem corresponds directly to the colors in a graph coloring problem. I.e., values which can not be assigned to the same register are in a one-to-one correspondence with graph vertices which can not be assigned the same color.

Scheduling Problems;

Many scheduling problems are also equivalent to graph coloring. Suppose you have N courses that need to be scheduled for the Fall 2016 semester. Each course requires a time slot during the week (e.g. MWF at 12:30). There is a fairly long list of reasons why two courses must not be scheduled at the same time. For examples: two courses might taught by the same faculty member; two courses are taken concurrently by a significant number students; graduate student TA's must be available during undergraduate class times, so required graduate courses must not be scheduled at the same time.

The courses in the class scheduling problem correspond to the vertices of a graph. Courses that can not be offered at the same time are connected by an (interference) edge in that graph. The time available time slots correspond to the colors. Adjacent vertices (courses which must not be offered at the same time) must be assigned a different color (different time slot in the week).

[2]Storing data to main memory is much slower than storing directly in a register.
[3]Conversely, if the lifetimes of two values do not overlap, then a register can be reused for both values.
[4]In most CPU designs fewer than 16 general purpose registers are available.
[5]Edges in the register allocation problem are called *interference edges*.

Turn In:

Save all your work in a directory named “Lab7”. Change to your home directory (the parent directory of “Lab7”), and create a file named “lab7.tar” using the command:

```
tar cf lab7.tar Lab7
```

Use `sftp` to upload the file “lab7.tar” to your account on telesto.

This Week’s Cartoon:

Color Bugs Bunny for Easter

Bugs is relaxing on Big Beach, Maui