

# CS51 Assignment 3: Modern Tines

Design document due: Monday, 25 February 2008 at 5:00 PM

Implementation due: Friday, 29 February 2008 at 5:00 PM

If taking late days, must be submitted by Tuesday, 4 March 2008 at  
5:00 PM

*Total Points: 70 (including 10 style points)*

The International Conference of Door-to-Door Spork Salespeople will hold their biennial meeting this year at American Legion Post 067 in Rich Hill, Kansas. Their hosts, the local chapter of the Spork Purveyors Organization (SPORK), need your help. On the third afternoon of the convention, they would like to have several hour-long presentations, some running simultaneously. The conferees have already signed up for which talks they wish to attend, and now SPORK needs to schedule the afternoon based on their members' preferences. They would like to know if there is a schedule that will allow every member to attend his or her preferred talks, and if so, what that schedule should be. They'd also like you to use data abstraction.

## 1 Partners

You will complete this assignment with your partner from assignment 2. As before, you should submit exactly the same answers and code. The only difference between your two submissions should be the answer to the question about how long each of you worked on the problem set.

### 1.1 Late Days

Again, each of you expend your own personal late days but will use the *same* number of late days on the assignment. If you plan to use late days on this assignment, you should make sure that this is acceptable to your partner. You may not turn in two separate assignments if you wish to take late days and your partner does not.

## 1.2 Submission

Both you and your partner must submit a copy of the final assignment—although only one partner need email in a copy of the design document.

## 2 Graph coloring

A *graph* is a collection of *nodes* and *edges*, in which an edge connects two nodes. In an *undirected graph*, we do not distinguish the direction of an edge between its two nodes, and we do not allow an edge from a node to itself; thus, an edge from  $A$  to  $B$  implies an edge from  $B$  to  $A$ , and there cannot be an edge from  $A$  to  $A$ .

This assignment presents a way to use a graph algorithm to solve SPORK's scheduling problem. We begin by introducing a particular graph-theoretic idea:

**Definition.** A *k-coloring* of an undirected graph is an assignment of colors to nodes, using a maximum of  $k$  different colors, such that no two adjacent nodes have the same color. Two nodes are adjacent if and only if there is an edge between them.

For example, we may use a  $k$ -coloring to color areas on a map such that no two adjacent areas share a color. There is a well-known theorem that all maps are 4-colorable, but of course this is not true of graphs in general<sup>1</sup>.

### 2.1 An algorithm

There are several ways to  $k$ -color a graph; unfortunately, they all require *exponential* time in the worst case<sup>2</sup>. One solution is to use an algorithm called *recursive backtracking*:

1. Pick a node and color it.
2. If the coloring is invalid, change your last color choice and try again, backtracking to the previously colored node when you run out of colors.
3. If all the nodes are colored and the coloring is valid, return the solution; if you backtrack all the way and have tried every color on the first node, there is no solution.

If we consider a *coloring* to be a state, the process becomes a depth-first search for a *goal state* in the space of colorings.

---

<sup>1</sup>For any  $k$ , one can construct a graph that is **not**  $k$ -colorable. Can you think of how?

<sup>2</sup>While the general, optimal algorithm is exponential, there exist sub-optimal and special-case algorithms that run in polynomial time. We, however, will implement a general, optimal algorithm.

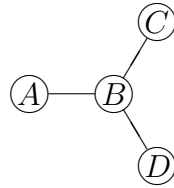


Figure 1: A 2-colorable graph



Figure 2: Two 3-colorable (but not 2-colorable) graphs

## 2.2 An interference graph

You might be wondering how graphs and colors can be of use to spork salespeople or conference organizers. Consider a trivial conference with four talks ( $A$ ,  $B$ ,  $C$ , and  $D$ ), two timeslots (red and green), and three conferees (Bubbles, Buttercup, and Blossom). Buttercup would like to attend talks  $A$  and  $B$ , Blossom would like to attend  $B$  and  $C$ , and Bubbles would like to attend  $B$  and  $D$ . We can construct an interference graph (Figure 1). An edge between two talks/nodes means that someone wants to attend both of them. They “interfere,” and optimally should not be scheduled for the same timeslot. If we can “color” the four talks with our two timeslots/colors, then we have a schedule; if not, then there is no schedule to be found.

In this case, we can color  $A$ ,  $C$ , and  $D$  red, and  $B$  green, and we have a schedule. However, we can easily construct an interference graph that is not 2-colorable (Figure 2).

## 3 Design document

(10 points total)

As assignments become more complicated, careful design will become all the more essential. A design document is a fairly formal presentation of your ideas about how to attack a problem set. In your `cs51/asst3` directory<sup>3</sup>, write your design in a file called `design.txt`.

Here are some issues to focus on in your architectural design:

- A specification of all the abstractions you will need (e.g., dictionary, graph, schedule, etc.). This specification should include an explanation of the abstraction and the interface for it.

<sup>3</sup>Which, surely, you’ve created by now and populated with files copied from `~lib51/assigns/asst3/`.

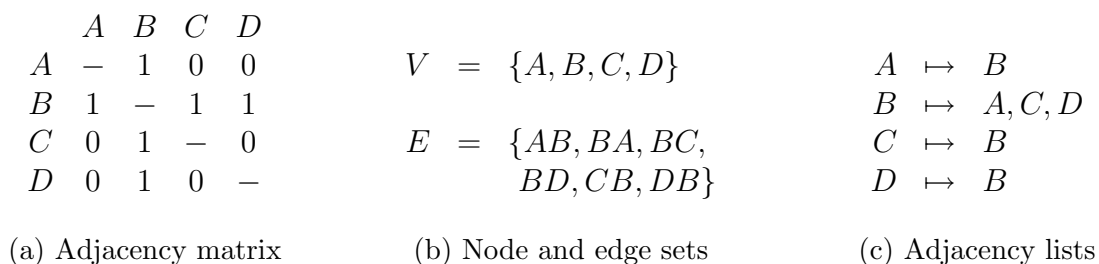


Figure 3: Three graph representations

- A description of the overall program with a breakdown of the key functions you'll need to write.
- Rationales for your design decisions (i.e., why you chose a particular data structure or algorithm).

You should also comment on your design from last week; what worked, what didn't, and how are you fixing any problems you found in your design strategy?

**Exercise 1.***10 points*

Email your design document to your TF by Monday, 25 February 2008 at 5:00 PM.

## 4 Implementation

*(50 points total)*

### 4.1 Data representation

There are several ways to represent graphs in a program, such as a matrix, adjacency sets, or a node set and an edge set (Figure 3). Under what circumstances do you think one representation would be preferable to another?

How you represent graphs is up to you. You may use the `dict51`-dictionary implementation if you like, but if you want to try something else, go for it. However you represent graphs, you'll need an interface to manipulate them. Here's an interface for functional, undirected graphs:

<code>(graph-empty)</code>	a new, empty graph
<code>(graph-connect g n1 n2)</code>	a graph like <code>g</code> , but with nodes <code>n1</code> and <code>n2</code> adjacent
<code>(graph-adjacent? g n1 n2)</code>	are <code>n1</code> and <code>n2</code> adjacent in <code>g</code> ?
<code>(graph-neighbors g n)</code>	list of <code>n</code> 's neighbor nodes in <code>g</code>
<code>(graph-nodes g)</code>	list of all nodes in <code>g</code>

**Exercise 2.***11 points*

- (a) [2 points] For the interface to implement *undirected* graphs, what contract needs to exist between `graph-connect` and `graph-adjacent`??
- (b) [2 points] Oops! The interface above is missing a method that is necessary if we want to be able to represent every undirected graph. Can you figure out what kind of graph cannot be created with the interface above? What is missing? Include it in your implementation.
- (c) [7 points] Implement the interface for graphs.

All code in the next section must be fully abstract with respect to graphs. That is, it should deal with them using only the five (or six) interface function above, and should work with **any** graph representation that implements the interface.

## 4.2 Color it already

The spork sellers have already compiled a list of interfering pairs of talks, so given that and a list of color-coded time slots, they want you to figure if there's a schedule, and if so, what it is.

### Exercise 3.

*30 points*

Write a function (`spork-schedule talk-pairs timeslots`) that returns an association list<sup>4</sup> mapping talks to timeslots if a valid schedule exists, or `#f` otherwise. Think carefully about modularity—you may find code that implements non-domain-specific functionality to be useful later. Your function must use recursive backtracking, and it should treat graphs as an opaque, abstract data type.

`Spork-schedule` should work like this:

```
> (spork-schedule '((a . b) (b . c) (b . d)) '(red))
#f
> (spork-schedule '((a . b) (b . c) (b . d)) '(red green))
((a . red) (c . red) (b . green) (d . red))
> (spork-schedule '((a . b) (b . c) (b . d) (c . d)) '(red green))
#f
> (spork-schedule '((a . b) (b . c) (b . d) (c . d)) '(red green blue))
((a . red) (b . blue) (d . green) (c . red))
> (spork-schedule '() '(red green))
()
```

The order of the pairs in a coloring doesn't matter, and colorings are equivalent up to permutation of colors. That is, these are all equivalent colorings:

```
((c . red) (a . red) (b . green) (d . red))
((d . red) (a . red) (c . red) (b . green))
((d . green) (a . green) (c . green) (b . red))
((a . green) (b . red) (d . green) (c . green))
```

---

<sup>4</sup>You may find the Scheme function `assoc` useful here.

**Exercise 4.**

5 points

We have provided a number of tests for `spork-schedule`, but you should add more tests to prove to us that your program works. You must include a test case for which your particular implementation of graph coloring would have to backtrack at some point. You should also write thorough tests for any helper functions you write, including the graph interface. Thorough tests would test each path in your helper functions.

**Exercise 5.**

4 points

(a) [2 points] One of your classmates came up with a novel coloring algorithm that, for commonly occurring scheduling conflicts, runs in polynomial time.<sup>5</sup> Suppose that you've been given a function (`color-in-polynomial graph colorlist`). How much of your code would you have to change for `spork-schedule` to use the new coloring routine in place of your own? Type your brief answer as a comment in `asst3.scm`; writing the new `spork-schedule` in terms of `color-in-polynomial` (in a comment) is likely to receive full credit.

(b) [2 points] SPORK is having trouble making a seating chart for their dinner on the first night. A number of spork sellers are not on speaking terms, and putting a feuding pair at the same table could be explosive. If SPORK prepares a list of who is mad at whom and decides how many tables they would like, how long would it take for you to write a function to perform table assignments? Type your answer in a comment; then write (`spork-table-assignment feuding-pairs table-list`)<sup>6</sup>.

## 5 When you are finished

About how long did this take your group to complete? How confident are you with your code? How long did you spend on the assignment, and how long did your partner spend on this assignment? Your answers will not affect your grade. Type your answer in a comment at the end of `asst3.scm`.

Submit your work with `make submit`.

---

<sup>5</sup>They always say that at Harvard you learn the most from your classmates!

<sup>6</sup>Don't worry about the size of the tables or balancing people between them—all we care about is keeping the feuding pairs separated. There will be other people at the dinner that aren't listed in `feuding-pairs`, but SPORK will work that part out themselves.