Computer Science 431
Algorithms
The College of Saint Rose
Spring 2015

Problem Set 1: Data Structures

Due: 11:59 PM, Sunday, January 25, 2015

For the first problem set, you are to complete a set of programming tasks to familiarize or refresh your knowledge of some of the fundamental data structures we will be working with all semester.

You may work alone or in groups of size 2 or 3 on this assignment. Only one submission per group is needed.

Getting Set Up

Create a directory or folder for your work for this problem set. You may use any IDE you wish, but be sure your programs run at the command line.

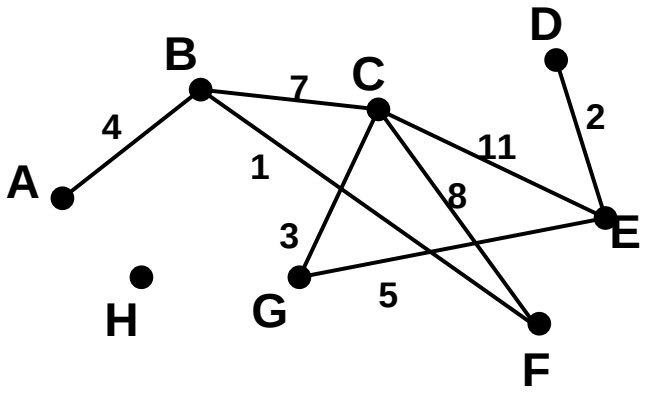
Writing your own Simple ADT

Write the class described in Bailey Problem 3.6, p. 65. You may limit your implementation to include a default constructor that creates a `BitVector` with slots initially for 10 `boolean` values, a second constructor that takes a parameter specifying the number of slots, and the following public methods: `add` at the end, `add` at a given position, `contains`, `get`, `indexOf`, `clear`, `remove` (by position), `set`, `size`, and `toString`. Also include an implementation of `BitVectorIterator` and a method `iterator` of your `BitVector` class that constructs and returns a `BitVectorIterator`. Include a `main` method that thoroughly tests your class and all of its constructors and methods.

Use Bailey's `Vector` class implementation and its corresponding `Iterator` as a reference. Also refer to Bailey Ch. 8 for more information about iterators if you are not familiar with the idea. (17 points)

Creating a Simple Graph

We will be working with graph structures in some of our programming tasks. The standard Java libraries contain a wide variety of useful data structures, but not graphs. We will make use of the graph structures in Bailey's "Java Structures" package. Familiarize yourself with that implementation. Then write a program `LittleGraph.java` that constructs a graph that represents the one we saw in class, then prints it out using the `Graph`'s `toString` method. (5 points)



A Graph Representing Highway Data

For your final task in this assignment, you will begin working with some real world data derived from highway systems. This same data will be used in other assignments later this semester.

A big advantage of working with this kind of data is that it has a connection to reality, and that we can visualize the data and the results of our manipulations of that data with the Google Maps API. This data is collected by the Clinched Highway Mapping (CHM) Project (`http://cmap.m-plex.com/`). I have taken some of the data from the CHM collaborators and converted into a format that is more convenient for us to load into a graph structure and use. Much more about the project is available at `http://courses.teresco.org/chm/`, but everything you need to know should be on this sheet.

The Data

The data is in "`.gra`" files which have the following format:

- The first line consists of two numbers: the number of vertices, $|V|$, (we'll call them "waypoints") and the number of edges, $|E|$, (road segments that connect adjacent waypoints).
- The next $|V|$ lines descibe the waypoints. Each line consists of a string describing a waypoint (its "label"), followed by its latitude and longitude as floating-point numbers.
- The last $|E|$ lines describe the road segments. Each line consists of two numbers specifying the waypoint numbers (0-based and in the order read in from this file) connected by this road segment, followed by a string with the name of the road or roads that form this segment.

You can find a few dozen example graph files linked from `http://courses.teresco.org/chm/graphs.html`. For example, `usai.gra` describes the entire U.S. Interstate Highway system. `canyt.gra` describes a much smaller system: the territorial highway system in the Yukon. The links of most interest to you are the "download" and "view" links.

Over the course of the semester, you will develop a Java program or programs that can read in graph data, store it appropriately in memory, and perform a variety of operations on that data.

Your tasks:

- Develop a class `Waypoint` that represents that data for a single waypoint. It should include fields for the waypoint name and its latitude and longitude values, a constructor, accessors for the three components, and appropriate `equals` and `toString` methods. There is a class `LatLng.java` in the shared area (`/home/shared/chm/src` on mogul) that you would likely find useful here. (8 points)

- Come up with an appropriate graph structure (something that holds the appropriate data in the vertex and edge labels) to hold this data and write a Java application to construct one from a data file whose name is given at the command line (*i.e.*, in `args[0]`). You are strongly encouraged, but not required, to use the graph implementations from Java Structures. Big hint: labels need not be simple objects like `String`s or `Integer`s. You can use any object type (including those you define yourself, like the `Waypoint` described above) for those labels. (15 points)

- Your program should then print out, in a nice format, a list of all waypoints that have been stored in the graph (*i.e.*, it should iterate over and print all vertex labels). (3 points)

- Your program should then print out the northernmost, southernmost, easternmost, and westernmost waypoints in the graph. Print all information (label and coordinates) of each extreme point. (4 points)

- Your program should then print out the shortest and longest road segments. Print all information (the labels of the endpoints, the edge label itself, and the length of the edge). Here, you will need to iterate over graph edges. (6 points)

Notes:

- While you might use an array or a `Vector` or an `ArrayList` to track some information during the construction of your graph, the only persistent structure in your implementation should be an instance of a `Graph` class.

- You should never need to use any casts if you declare and construct your `Graph` with the appropriate type parameters. For example, my graph uses `Waypoint` objects as vertex labels and `String`s as edge labels. So my declaration is:

  ```
  Graph<Waypoint,String> data;
  ```

  and it is constructed as

  ```
  data = new GraphListUndirected<Waypoint,String>();
  ```

- All of the information required to be printed after the graph is constructed should be extracted from the `Graph` structure, not remembered during the reading of the data file in other variables.

Submitting

Before 11:59 PM, Sunday, January 25, 2015, submit your problem set for grading. To complete the submission, upload an archive (a `.7z` or `.zip`) file containing all required files using Submission Box at `http://sb.teresco.org` under assignment "PS1".

Grading

This assignment is worth 65 points, which are distributed as follows:

Feature	Value	Score
`BitVector` fields	2	
`BitVector` resizes as needed	2	
`BitVector` constructor(s)	1	
`BitVectorIterator` implementation	4	
`BitVector` other methods	5	
`BitVector` main method with tests	3	
`LittleGraph` Program	5	
`Waypoint` object implementation	8	
Declaration and construction of `Graph` object	2	
Get file name from command line	1	
Read file into graph structure	12	
Print waypoints from graph structure	3	
Print extreme points from graph structure	4	
Print longest/shortest from graph structure	6	
Style, documentation, and formatting	7	
Total	65	