Assignment 7

CSE 326: Data Structures and Algorithms

Autumn Quarter, 1999, University of Washington.

Steven Tanimoto, Instructor

Due dates: Phase 1: Wednesday, November 24, Phase 2: Wednesday, December 8.

Purpose: This assignment offers an opportunity to explore data structures in a more realistic context than you did with the AVL exercises. In the search-engine option, you'll process text, build a large index. and provide ways to access the information. In the graphics option, you'll explore how to use spatial indexes to rapidly retrieve a set of geometric objects based on location.

Choose either the Web Search Engine option, the Rectangles option, or the Image Analysis option. For Phase 1, do the basic program. For Phase 2, select some options and complete those.

Web Search Engine Option

Work either individually or in a team of 2-3 people.

THE BASIC PROGRAM: Implement a program that takes a list of file names, processes each one to build an index, and accepts and processes queries on the index.

Each entry in the index should contain a key (which should normally be a word occuring in at least one of the files) and a list of "occurrences". An occurrence consists of (A) the name and path of a file containing the word, and (B) that line of text in the file which contains the first occurrence of the word in that file, and (C) a count of the occurrences of that word in that file.

The index should be structured either as a b-tree whose nodes are implemented as files or as an open hash table whose data is stored mostly in files.

The decision of exactly how much data to put into each node (file) is yours. However, you should justify that decision in terms of performance issues. A limit at the fine-grained end of the spectrum is one file per word. However, this seems like too many files, and so you should consider nodes that contain multiple words.

Individuals should choose either index building or query processing and do just that one job. Groups of two should usually split up the job into index building and query processing. Groups of three should divide the work so that one person is in charge of processing the text files to extract the information to go into the index, another is responsible for the index structure itself – tree insertion and maintenance. The third person would be responsible for obtaining queries from a user and processing the queries.

THE EXTENSIONS: Individuals should choose one suboption and do a nice job with it. Partnerships should choose two suboptions. Teams of 3 should choose 3 suboptions.

Suboptions include:

1. Taking a list of URLs rather than a list of files.

2. Crawling part of the web by analyzing each document to extract links and follow them.

3. Making the index into a clickable web page or web site.

4. Accepting queries over the web and matching the query against index entries, returning a list of hits.

Use one or more of the following techniques: B-trees, hashing, graph search (where the graph is some part of the World Wide Web).

Graphics Option

Work either individually or in a team of 2-3 people.

Suppose that a computer-based drawing program allows the user to draw rectangles and enter them, each with a string label, into a drawing. The drawing is actually just a set of labelled rectangles. The rectangles are assumed to have their sides parallel to coordinate axes, to keep things simple.

When the user wants to edit a rectangle, let's say that the user has only to click inside the rectangle and then, if several rectangles include that point, keep clicking until the desired rectangle is highlighted.

THE BASIC PROGRAM: Your job is to take a set of rectangles and build a tree structure that makes the operation of finding those rectangles that contain a particular point very efficient.

Read up on k-d trees starting on p.321. Also consider quad trees, described starting on p.325.

Provide a comparison to show the relative performance when using the tree vs doing a linear search.

THE EXTENSIONS:

Suboptions include:

1. Actually create a drawing program to let the user input rectangles and then select rectangles based on containment. (This is an option. The alternative is to read in the data from a file.)

   This option will require the use of a graphics package. If you are familiar with Java and the AWT, this would be a good way to go.

2. Compare 2-d trees with quad trees. Implement both kinds of trees and the rectangle retrieval operations. For a given set of rectangles, how does the size and shape (e.g., balance) of the 2-d tree compare with that of the quad tree? What are the mean and standard deviations of the depths of the leaves for each kind of tree? How many comparison operations are performed in doing the same sequence of operations on each of the two kinds of trees?

3. Implement a program that allows the user to create an object-oriented drawing in which each object can have parts and the parts can have parts, etc. Each "part" should be a subset of rectangles (that you might assume to be nonoverlapping, but this is not that important). When the user selects an object, the part(s) containing the rectangle(s) selected would be highlighted, rather than the rectangles themselves.

4. Extend your system to work with arbitrary closed polygons rather than only rectangles.

Image Analysis Option

If you choose this option, work individually. For phase I, implement all the basic functionality. For phase II, choose one of the three suboptions and do that.

THE BASIC PROGRAM: The idea in this program is to read in a file that contains pixel values for an image, and then determine the regions in the image that may correspond to objects, the background, etc. The data structures you will use are (a) a two-dimensional array, and (b) a forest of trees where each tree is an instance of a special type of tree called an up-tree whose links all point upwards from the nodes to their parents. The up-tree and its methods are described on pages 307-316.

After your algorithm has run to completion, the pixels of the image will each be a part of a tree that represents the set of pixels for one region.

This forest of trees should then be sorted in order of decreasing size, and the first K of them should be printed out as images. The user should get to specify the value of K.

More information about this option will be provided in class.

THE EXTENSIONS: Choose one of the following three suboptions.

1. Implement path compression on the trees and show how performance improves as a result – give the number of link traversals required with and without path compression.

2. Implement a flexible criterion for growing the regions in which each region (tree) in the image has a range of pixel values rather than a single value, and two trees can be merged if the union of their ranges is a new range that does not exceed a certain width given by the user.

3. Modify the algorithm so that it works with a color image – each pixel would be represented by a three-element record, with a value for red, a value for green, and a value for blue.

Additional details regarding demonstrations and what to turn in will be given later.