**Lab #9: Abstract Data Types**

Due: Tuesday, April 15, 2014, beginning of class

## Instructions

* You will need your WFU-issued ThinkPad for this lab.
* Submit your answers to the assignment in Sakai using this document and any other material you are asked to submit in the exercises.

**Notes to grader: Only grade the highlighted exercise numbers. The others are only instructions. I count 25 highlighted exercises. So weight them all worth 4 points each.**

## Part 0: Background

The purpose of this lab is to reinforce what you have learned about abstract data types (stacks, queues, trees, and graphs). Background comes from Chapter 8 (Computer Science Illuminated) and from lectures.

## Part 1: Stacks, Queues, and Trees

* Create a folder called ***Lab 9*** on your desktop.
* From the assignment on Sakai, right-click and *save link as* the ***Applets\_for\_Abstract\_Data\_Types.zip*** and ***Abstract\_data\_types\_lab.pdf*** into your Lab 9 folder.
* Once downloaded, double click on the ***Applets\_for\_Abstract\_Data\_Types.zip*** file and extract the directories into your Lab 9 folder.
* Read pages 120 – 124 of the ***Abstract\_data\_types\_lab.pdf*** document and then complete the exercises below.
* Note each exercise calls for you to start a particular applet. To do this, go into the appropriate applets sub-folder in your Lab 9 folder. In that sub-folder, double click on the ***applet\_frame.htm*** file. Once the applet frame starts, depending on your security settings, you may need to explicitly allow the applet to run. Screen captures at the end of this document illustrate how to do this. Note that if the applet does not work with Internet Explorer, right click on applet\_frame.htm and select ‘open with’ and select Firefox.

**Exercise 1: Stacks**

**1.a.** Start the “Stackqueue” applet.

**1.b.** The purpose of this exercise is to see how you could reverse a list of names. Push the three names “Abe,” “Betty,” and “Charles” onto the stack. When you pop the stack 3 times, what do you see in the text field next to “Pop”?

Abe

**Exercise 2: Stacks**

**2.a.** Start the “Stackqueue” applet.

**2.b.** The purpose of this exercise is to figure out whether the mystery data structure is a stack or a queue. Click on *Clear,* then *Random Fill.* Select “mystery” from the pull-down menu.

**2.c.** Click once on *Remove.* Whatever number appears next to the *Remove* button is what you now type into the text field next to *Add.* Then click on *Add.*

39 appeared and was added

**2.d.** Now click on *Remove.* What value appears? Is it the same one or different? What does this mean? Is your mystery object a stack or a queue? Take a screenshot and post-it and write your answer below. You can then *Show* to confirm your conclusion.

39 appears. It is the same. It means the last one in is the first out (LIFO). Or it could be a random chance that the same value was in more than once. Likely, though, this is a stack. Which it is.

**Exercise 3: Binary Search Trees**

**3.a.**  Start the “Trees” applet.

**3.b.**  The purpose of this exercise is to build a binary search tree. Select “Insert alphabetically” from the pull-down menu. Then type the following flowers into the text field, pressing return after each one.

Lily

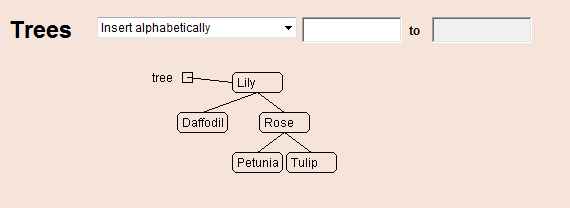
Rose

Daffodil

Tulip

Petunia

**3.c.** Take a screenshot and post it below. Now perform a manual search for “Poppy.” Write each node that you visit during your search. You can confirm your ideas by selecting “Find” from the pull- down and asking the applet to search for “Poppy.”



Could not find Poppy

**3.d.** If you were to manually insert “Poppy” into this tree, what would you put into the boxes and which option would you select?

\_\_\_\_ “Attach as left child” \_\_\_\_\_\_\_\_\_\_\_\_\_ to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

X “Attach as right child” Poppy to Petunia

Check one blank on the left of “Attach...” and fill in the two blanks surrounding “to.”

**Exercise 4: Binary Search Trees**

**4.a.** Start the “Trees” applet.

**4.b.** The purpose of this exercise is to build a binary search tree but choosing different orders for inserting the nodes.

**4.c.** Select ‘Insert alphabetically,’ then type in the following flower names in the given order. (If you don’t want to type a lot, just use the first letter of each name.) Take a screenshot and post it below when done.

Aster

Bluebell

Coreopsis

Daisy

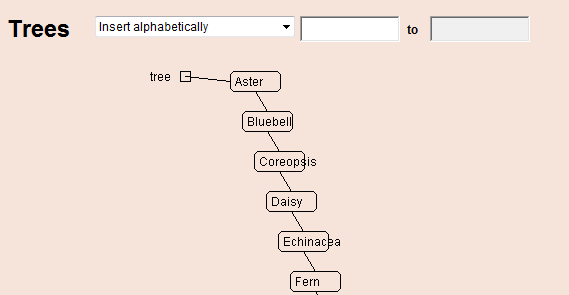
Echinacea

Fern

Gladiolus

**4.d.** What does your tree look like? Describe it in words below.

It is a long line and does not look like a tree at all.



**4.e.** What generalization can you make about inserting elements from a sorted list into a tree?

They will always look like this. If sorted the other way, the tree will go to the left.

**4.f.** Search for “Gladiolus”. How many compares were needed to find this flower? (Note, your tree may extend past the bottom of the applet frame. If so, use your knowledge of how the tree looks to compute the number of compares.)

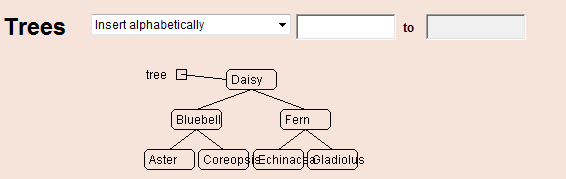
**7**

**4.g.** Could you convince someone that a list is a special kind of tree?

Yes. Just show them this example. A list is a tree in which every node is a right child (or a left child).

**4.h.** Clear the tree and insert the flowers in some other order so that the tree is *balanced* and *complete.* This means that all nodes except the leaf nodes have exactly two children. Use ‘insert alphabetically’ for the root node and ‘Attach as left child’ and ‘Attach as right child’ for the other nodes. After you are successful, take a screenshot and post it below. Also write down the exact order of insertion that you used to get it to look this way.

Daisy, Bluebell, Aster, Coreopsis, Fern, Echinacea, Gladiolus



**4.i.**  Is there more than one order that you could have used that would have created the same balanced, complete tree? Why are multiple different orders possible?

Yes. Because each subtree is independent. So you could do the left then right or right then left and continue this for each subtree.

**4.j.** In your balanced and complete tree, search for “Gladiolus”. How many compares were needed to find this flower?

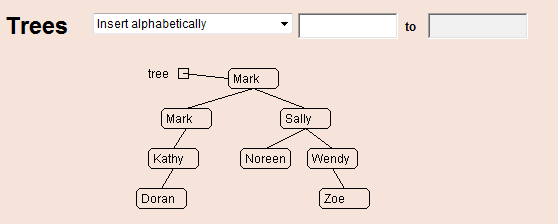
**3**

**Exercise 5: Binary Search Trees**

**5.a.** Start the “Trees” applet.

**5.b.** The purpose of this exercise is to investigate what happens when you have duplicate nodes. Choose “Example 2.”

**5.c.** Select “Insert Alphabetically.” Then type “Mark” (not “mark,” the program is case-sensitive) in the text field and press Return. Take a screenshot and post it below.



**5.d.** Insert another name that is already in the tree. What does this applet do when you ask it to enter a name that is already there?

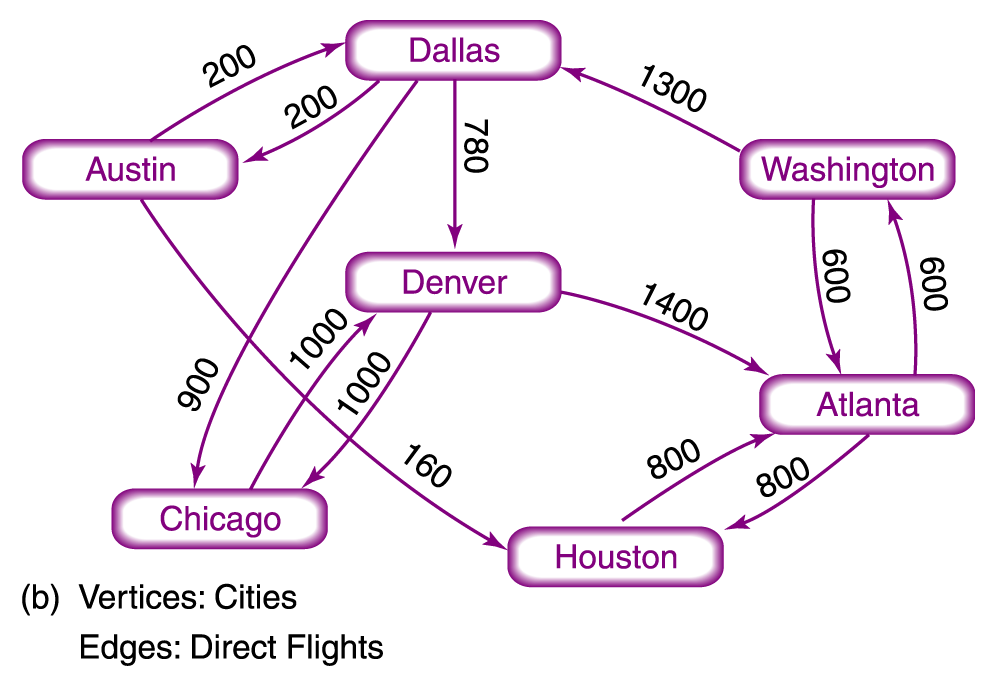
Inserts a left child of the node with the same name and puts the name in that child.

**5.e.** Does “Find” still work? Try to find a name, like “Mark,” that is represented by two nodes. What happens? Describe.

Yes. It searches until if finds the first occurrence of the name.

## Part 2: Graph Representation

Exercises 6-8 in Part 2 utilize the following diagram.



**Exercise 6: Graph Representation**

**6.a.** A graph can be represented in a matrix (table) called an adjacency matrix. Each node is both a row name and a column name. For the graph above, the row name indicates the starting city and the column the ending city. If the edge weight is the same going or coming, then the matrix is symmetric. If not, the matrix is asymmetric. In our case, the adjacency matrix is asymmetric. Complete the matrix below. Where there is not a connection, make the distance 100,000.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Austin** | **Dallas** | **Denver** | **Chicago** | **Houston** | **Atlanta** | **Washington** |
| **Austin** | 0 | 200 |  |  | 160 |  |  |
| **Dallas** | 200 | 0 | 780 | 900 |  |  |  |
| **Denver** |  |  | 0 | 1000 |  | 1400 |  |
| **Chicago** |  |  | 1000 | 0 |  |  |  |
| **Houston** |  |  |  |  | 0 | 800 |  |
| **Atlanta** |  |  |  |  | 800 | 0 | 600 |
| **Washington** |  | 1300 |  |  |  | 600 | 0 |

## Exercise 7: Depth First Search of a Graph

**7.a.** Identify each variable in the pseudocode below, and enter its name in the left column of the table. ***Note: You may find it helpful to open a new window in Word so that you can show the table and the code at the same time.***

**//Depth First Search(startVertex, endVertex)**

**SET found to FALSE**

**PUSH (myStack, startVertex)**

**WHILE (NOT IsEmpty(myStack) AND NOT found)**

**POP (myStack, tempVertex)**

**IF (tempVertex equals endVertex)**

**PRINT endVertex**

**SET found to TRUE**

**ELSE IF (tempVertex not visited)**

**PRINT tempVertex**

**//For the city currently in tempVertex, PUSH its adjacent**

**// unvisited cities onto myStack, picking cities from the**

**// adjacency matrix from left to right in the row for tempVertex.**

**PUSH (myStack, all unvisited vertexes adjacent with tempVertex)**

**Mark tempVertex as visited**

**IF (found)**

**PRINT "Path has been printed"**

**ELSE**

**PRINT "Path does not exist"**

**7.b.** For the above flight graph, use the depth-first search algorithm to find a path from **Dallas to Washington**. (Set *startVertex* to **Dallas** and *endVertex* to **Washington**.) Complete the table by filling in the variable names and record their values as they change. Enter any values written by the program in a row labeled **Output**. Each time a variable changes values enter its value in the next empty column for that variable. In this way, the last value for a variable is its current value. Be sure to make a row for myStack. When you PUSH, enter into the next empty column. When you POP, just put a line through the item on the top of the stack (end of the row). Use the adjacency matrix to check off visited nodes as needed. Add rows or columns if needed.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Variable Name** | **Variable Value** | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| startVertex | Dallas |  |  |  |  |  |  |  |  |
| endVertex | Washington |  |  |  |  |  |  |  |  |
| found | False | True |  |  |  |  |  |  |  |
| tempVertex | Dallas | Chicago | Denver | Atlanta | Washington |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| myStack | ~~Dallas~~ | Austin | Denver | ~~Chicago~~ | ~~Denver~~ | Chicago | ~~Atlanta~~ | Houston | ~~Washington~~ |
|  |  |  |  |  |  |  |  |  |  |
| output | Dallas | Chicago | Denver | Atlanta | Washington |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

**7.c.** Is the path printed correct? If not, comment on what you think went wrong and what you might do to fix it. You do not have write new pseudocode, just give your ideas on a fix.

**Yes.**

## Exercise 8: Shortest Path Algorithm

An important, and widely used algorithm, is finding the shortest path between nodes in a graph. One such algorithm is called Dijkstra’s Shortest Path Algorithm, named for its inventor and famous Dutch computer scientist, Edsger Dijkstra. While you are not responsible for a full understanding of the algorithm, you should see the basics and understand where a stack might come into play. Using the link below, write down what you think is the shortest path from node O to node T (the values on the edges are the length from one node to the next). Then run the example and try to follow along. Did you get the same shortest route?

<http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html>

**8.a.** What is an “unsolved” node?

A node for which the shortest path from the start node has not yet been determined. Once it is determined, the node becomes solved. (Grader: Only the first sentence is required in the answer.)

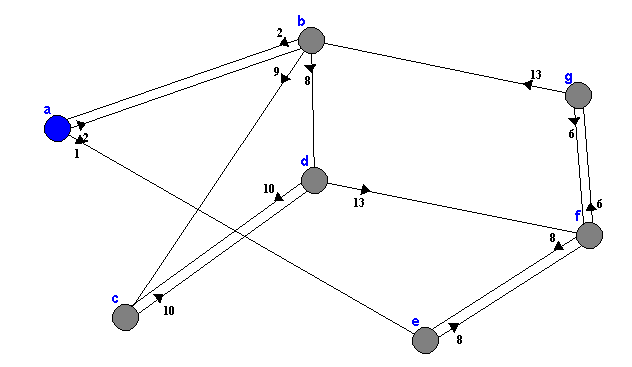
**8.b.** The following link allows you to set up your own network.

<http://www.dgp.toronto.edu/~jstewart/270/9798s/Laffra/DijkstraApplet.html>

Set up the airline route network from the figure above. Divide each distance by 100 and round up since the maximum allowed in the applet is 50. Use the following node name mappings:

(Austin, A) (Dallas, B) (Chicago, C) (Denver, D) (Houston, E) (Atlanta, F) (Washington, G)

Take a screenshot of your graph and paste it below. Grader. Due to rounding student graph may not have the same distances exactly. For example distance from Austin to Houston could be 1 or 2 after dividing by 100 and rounding.



**8.c.** Find the shortest distance from Houston to Dallas. Write the path below and the total distance.

Houston -> Atlanta -> Washington -> Dallas 2,700 miles. Grader. Student may use the distance number from the graph above.

**8.d.** Find the shortest distance from Dallas to Houston. Write the path below and the total distance.

Dallas -> Austin -> Houston 360 miles. Grader. This value may vary depending on how the student rounded the distance.

**8.e.** List 3 other reasonable edge weights other than miles that might be used in an airline route graph.

A few possibilities: Flying time. Fuel used. Ticket price. Grader: Any 3 reasonable answers are fine.

## Exercise 9: The Traveling Salesman Problem (TSP)

The travelling salesman problem (TSP) asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

You can read about the TSP here:

<http://en.wikipedia.org/wiki/Travelling_salesman_problem>

One way to solve this problem is to examine every possible route and see which is the shortest. If there are N cities, then there are N factorial (written as N!) possible routes.

**9.a.** Fill in the following table.

Grader. Students may use scientific notation and do not need to have so many significant digits.

|  |  |
| --- | --- |
| **N** | **N!** |
| 5 | 120 |
| 10 | 3628800 |
| 15 | 1307674368000 |
| 25 | 1.5511210043330983e+25 |

**9.b.** If it takes a computer only one microsecond to determine the distance for a route, how long, in years, would it take to check all the routes for 25 cities? Show your work.

(N! path) \* (1 micro-s/path) / (1,000,000 micro-s/s) / (60 s/min) / (60 min/hr) / (24 hr/day) / (365 day/year)

**Gives 4.919 x 1011 years**

**For further study.** Think about or look up if interested: How do you suppose this problem could be solved practically? Why doesn’t it work just to go from the current city to the next closest that has not yet been visited? You can read more about the TSP at the link given above.