Version: 29-Apr-2021 11:50 AM

CS 2223 D21 Term. Homework 4

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples as I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d21/#policies.
- Due Date for this assignment is **6PM Tuesday May 4th** which is the day before the end of the term.
- Submit your assignments electronically using the canvas site for CS2223. Login to canvas.wpi.edu and locate HW4. You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID where USERID is your CCC user id.
- Submission information is found at the end of this document.

Primary Instructions

This homework assignment is about problem solving. Using the data structures and algorithms presented over the past few weeks (including this week), this assignment gives you the opportunity to take the ideas presented in class and solve problems.

For this assignment, copy all of the files in the **`algs.hw4.submission`** package into your **`USERID.hw4`** package.

Q1 Expressions (30)
Q1.4 Bonus (1)

Q2 Graphs and Breadth First Search (60)
Q2.4 Bonus (1)

Q3. Random Graphs (10)

Q4. Bonus Points (2)

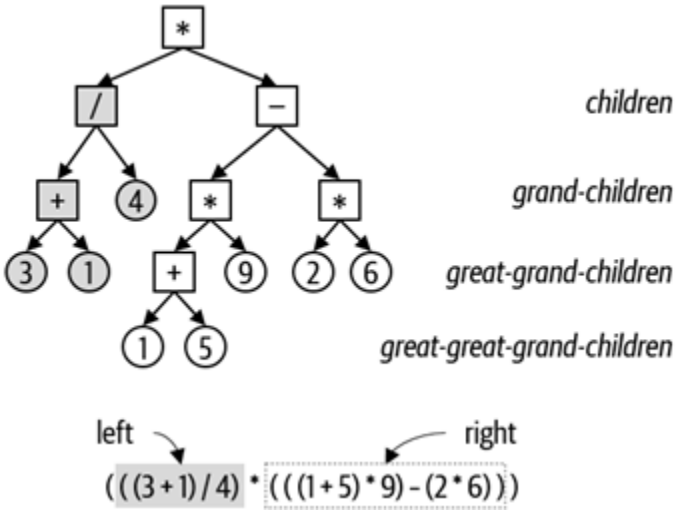
Version: 29-Apr-2021 11:50 AM

Q1. Expression Trees (30 pts)

The only binary structure we have seen is a Binary Search Tree. There are others. One of the most common is an *Expression Tree*, where each node is either a Value or represents a binary operation, such as "+" or "-"). A valid expression is either a **Value** or it combines two **Expression**s using a binary function:

- `3` — Any numeric **`Value`** is valid.
- `(3+2)` — Add a left **`Value`** 3 with a right **`Value`** 2.
- `(((1+5)*9)-(2*6))` — Subtract a right **`Expression`** (2*6) from a left **`Expression`** ((1+5)*9).

Expressions can combine and grow to be as large as desired — the expression below has seven mathematical operations and eight numeric values. Linked lists cannot model this non-linear expression. While we use the term Expression Tree, there is no class to represent the tree with a root field. You are only dealing with raw `Expression` nodes. Check out **`algs.hw4.Example`**.



The top `Multiply` node has two children nodes, ultimately leading to four grandchildren nodes (one of which is the `Value` 4), six great-grandchildren nodes, and two great-great grandchildren nodes. To compute the numeric value of this `Multiply` node, first recursively evaluate its left subtree expression to produce the numeric value of `1.0`. In similar recursive fashion, its right expression subtree evaluates to `42.0`, so the computed numeric value of the original expression is `1.0*42.0 = 42.0`.

So, you guessed it, this is your third time implementing an `Evaluate` class. This time you will read input formatted using postfix notation (as you did for HW1) and construct the top-level `Expression` node representing the entire expression tree.

Q1.1 Complete new type of nodes for the expression tree (10 pts)

Complete the implementation of subclasses `Subtract`, `Divide`, `Multiple` – pattern these implementations after the sample classes provided.

Version: 29-Apr-2021 11:50 AM

Q1.2 Complete Q1 class (10 pts)

In HW1 you converted a postfix expression into an infix expression and evaluated it using **`FixedCapacityStack`** objects. In similar fashion, modify Q1 to work with a **`FixedCapacityStack<Expression>`** to construct **`Expression`** nodes, resulting in an Expression tree. Once done, you can call both `format()` and `eval()` to demonstrate it works on the following inputs:

- `"9 7 /"` report a computation of 1.285714…
- `"3 1 + 4 / 1 5 + 9 * 2 6 * - *"` report a computed value of 42.0
- `"1 2 * 2 3 * + 9 8 7 6 / * + -"` report a computed value of -10.33333

In all cases, the Infix formatted expression should also be printed using `format()`.

Q1.3 Define a new operation to perform (10pts)

In the same way that you can define the height of a Binary Search Tree, you can also define the height of an *Expression Tree* in terms of its nodes.

The height of the sample expression tree shown earlier is 4. The height of a single `Value` leaf node is 0.

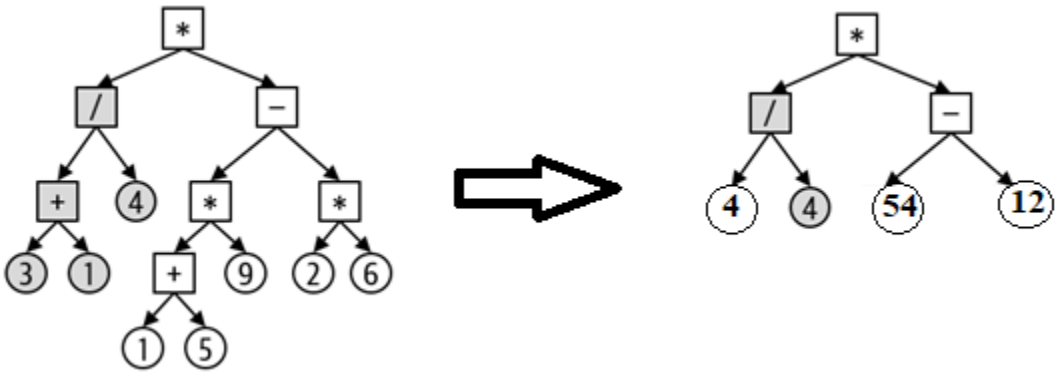
Add the following new operation to the `Expression` class.

```
/** Compute the height of the Expression tree. */
public abstract int height();
```

Update the other classes accordingly and modify the Q1 class to also print the height of the expression trees entered.

Q1.4 BONUS Question (1 pt)

Add a method `truncate(int depth)` to `Expression` that reduces an expression to a specific depth by evaluating all expressions at a specific depth and replacing their subtrees with Values. For example, `truncate(2)` would have the following impact, since after this invocation, no `Expression` node will exist at depth 3.



Hint: To make this work, you will need to ensure that `left` and `right` attributes are not **`final`**.

Version: 29-Apr-2021 11:50 AM

Q2 Learn to work with graphs and Breadth First Search (60 total pts)

On Day 21, I showed how to work with highway maps as graphs. In the example I showed, I arbitrarily picked two vertices to conduct the search. For this assignment, you are to work with Breadth First Search to compute specific paths through the highway graph. Feel free to use the ideas from **`algs.days.day21.BreadthFirstPaths`**.

Q2.1 Standard Paths (25 pts)

There are two scenarios to tackle:

- Using Breadth First Search, compute the shortest path (in terms of total number of highway segments) from the western-most highway location to the eastern most highway location. Complete the implementation of **`westernMostVertex`** and **`easternMostVertex`**. Be sure to identify the label associated with each vertex and print the total number of edges in the path.
- Using Breadth First Search, compute the shortest path (in terms of total number of highway segments) from the southern-most highway location to the northern-most highway location. Complete the implementation of **`southernMostVertex`** and **`northernMostVertex`**. Be sure to identify the label associated with each vertex and print the total number of edges in the path.

Given the Massachusetts Highway system, you will find that the following labels are associated with these different most distance edge points. The following are the actual labels in some random order. Only you will be able to know which ones are West, East, South or North once you complete your changes.

- MA23/NY23@NY/MA
- MA28@ShoRd
- MA88@CheWebbLn
- MA150/NH150@MA/NH

Q2.2 Demonstrate why Depth First Search is inappropriate here (10 pts)

For the same two considered scenarios above, now complete a Depth First Search and report the total number of edges for each one, as you did with Breadth First Search.

Q2.3 Eliminate Mass Pike from consideration (25 pts)

In some GPS applications, the user can request to avoid Toll Roads. For this question, you are to complete the implementation of **`Information remove_I90_segments(Information info)`** which is in the Q2 class. This method will return a new **`Information`** object containing *a new graph* for all Massachusetts highway segments except those from the Mass Pike (I-90). Note that you cannot remove an edge from a **`Graph`** object, so instead you are going to create a new graph using its original vertices but you will only add *edges that are not wholly contained by the I-90 Mass Turnpike*.

Now go back and compute the above standard paths for Breadth First Search only. Describe the impact of not being able to use the Mass Pike. Describe the change (if there is one) on both scenarios, specifically regarding the total number of edges.

Q2.4 Bonus (1 pts)

In the **`algs.hw4.map.GPS`** class there is a method that computes the distance in miles between any two GPS coordinates. Use this method to compute the length in mileage for the shortest computed paths for the two pairs of edge points. Now use https://google.com/maps using the same GPS endpoints and determine if google can find a SHORTER path between the two pairs of edge points. Show the total distance traveled by google (just scrape the information or provide a screenshot?) and compare your mileage results.

Consider using/modifying the **`BFSMapAnimation`** example from day 21 to generate images.

Version: 29-Apr-2021 11:50 AM

Q3 Exploring Random Directed Graphs (10)

This question asks you to generate 10,000 random directed graphs and inspect certain properties of these directed graphs.

Conduct this experiment with two different kinds of random directed graphs.

- For a graph of N vertices, there are N*(N-1) possible directed edges. For each of these (u, v) possible edges, add the edge if `Math.random() < 0.5`
- For a graph of N vertices, there are N*(N-1) possible directed edges. For each of these (u, v) possible edges, add the edge if `Math.random() < 1.0/N`

These random directed graphs will have different properties. **For N in the range from 2 to 15**, print a table that shows (a) whether a cycle exists within the graph; (b) whether all vertices in the graph are reachable from **vertex 0**. Make sure that your graph is still a simple graph with no self-loops.

Your output will look like the following (with some variation since these are random trials):

```
Graphs with probability 0.5
N       #Cycles         #Connected
2       2493            4957
3       6122            5004
4       8680            5959
5       9740            7074
6       9958            8156
7       9998            8927
8       10000           9353
9       10000           9653
10      10000           9835
11      10000           9871
12      10000           9942
13      10000           9965
14      10000           9980

Graphs with probability 1/N
N       #Cycles         #Connected
2       2448            5027
3       3249            2618
4       3718            1421
5       3989            810
6       4222            448
7       4431            295
8       4631            176
9       4776            96
10      4877            51
11      5056            45
12      5310            19
13      5244            9
14      5411            9
```

Version: 29-Apr-2021 11:50 AM

Q3.1 Bonus Question (1pt) Explanations?

So it appears that when the probability of an edge is 1/N then a cycle exists about 66% of the time (if you extend to say N=50). But if you tweak the probability to be 0.5/N, then it drops to about 16%. Is there a pattern you can discern? What if probability is K/N? can you state the pattern in terms of K?

Version: 29-Apr-2021 11:50 AM

Q4. BONUS question: Canada (2 points)

Find the Western-most and the Eastern-most points on the Canadian Highway System. Conduct a Breadth first search from one to the other. What's this? You can't get there from here? What about doing a path from the Northern-most point to the Southern-most? What? That doesn't connect either? What is going on?

https://en.wikipedia.org/wiki/Trans-Canada_Highway#/media/File:TransCanadaHWY.png

If you show this same graph in BFSMapAnimation, there are similarities, though not drawn to scale.

Q4.1 (1 pt) First Fix the northern one
In searching from the Northernmost point to the Easternmost point, there are some segments that do not appear to be reachable.

What are the two vertex identifiers (and GPS coordinates) of the northern-most section that, apparently, has a gap? **Just add this edge, and you will be able to get past it**.

Q4.2 (1pt) Now try to fix the eastern one.
In searching from the Easternmost point to the Westernmost point, the search stops. Why does it stop? What is missing? And what is your practical solution to fix this problem?

Change Notes

1. Fixed the bonus question and properly adjusted the points for the assignment to be out of 100.
2. For Q3, there are N*(N-1) possible edges in directed graph
3. For Q3, my numbers were way off because I wasn't preventing self-loops.