© 2014 Sudarshan S. Chawathe

Name: ____

Please refer to the previous assignment for important instructions on the allowable use of resources and the requirements for electronic and hardcopy submission. Use the newsgroup for questions, clarifications, and general discussion related to this homework and class. Ensure that everything you submit is neat and easy to comprehend; otherwise, it will be ignored, with zero credit. You are encouraged to answer the three ⋆ questions for extra credit but do note that those questions are graded much more strictly than the others.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) The pen-and-paper questions in this assignment need to be answered on separate sheets of paper that must be neatly attached to your hardcopy submission with a single staple. Note the number of additional pages attached here: ____

3. (1 pt.) Package and submit your solutions to the programming questions via `http://cs.umaine.edu/~chaw/u/`. *After* submitting your work, fill in the following:

   File name: ______________________
   Size, in bytes: ______________________
   MD5 checksum: ______________________
   Timestamp: ______________________

4. (17 pts.) A *k-tree*, for $k > 1$, either is empty or consists of a node r, called its root, and a sequence of $c \leq k$ subtrees, each of which is a nonempty k-tree. The roots of these c subtrees are the *children* of r, and r is their *parent*.

   Depict all distinct (i.e., nonisomorphic) 4-trees with n nodes, for $n = 0, 1, 2, 3, 4, 5$.

5. (10 pts.) A *labeled k-tree* is a k-tree in which each node with $c > 1$ children is associated with a sequence of labels $l_1, l_2, \ldots, l_{c-1}$.

   Indicate the number of distinct labeled 4-trees with n nodes, for $n = 0, 1, 2, 3, 4, 5$, with labels drawn from the domain $\{1, 2, 3\}$. A tree need not contain all the labels from the domain, and labels may be repeated. You do *not* need to depict all the trees, but you must explain clearly how you arrive at the answers. (Depict at least a few representative trees.) There is no credit for answers without proper explanations.

6. (20 pts.) A *k-search-tree* is a labeled k-tree with the following additional constraints on each node n with $c > 1$ children, where we define $l_0 = -\infty$ and $l_c = \infty$ for convenience.

   - The labels are strictly increasing; that is, $l_i < l_{i+1}$ for $i = 0, 1, 2, \ldots c - 1$.
   - If $x_1, x_2, \ldots, x_c$ are the children of n, then all labels in the subtree rooted at x_i are greater than the label l_{i-1} and less than the label l_i, for $i = 1, 2, \ldots, c$.

Depict all distinct labeled 4-search-trees with n nodes, for $n = 0, 1, 2, 3, 4, 5$, with labels drawn from the domain $\{1, 2, 3, 4, 5\}$.

7. (10 pts.) Depict a k-search-tree in which some label l occurs in two distinct locations, or explain clearly why no such tree exists.

8. (30 pts.) Describe efficient algorithms for the following, using the k-search-trees defined earlier. For each algorithm, provide (1) a clear English description, (2) detailed pseudocode, and (3) a proof that the algorithm's running time is $O(h)$ where h is the height of the k-search-tree.

   (a) Locating a given label in the tree or determining that it is absent from the tree.

   (b) Inserting a given label into the tree or determining that it is already in the tree.

   (c) Removing a given label from the tree or determining that the tree does not contain that label.

9. (10 pts.) Conduct an experimental evaluation of the three algorithms for the maximum contiguous subsequence sum problem described in the textbook.[1] Ensure that your results are meaningful, given the limited clock accuracy and other factors. In your electronic submission, include a *plain text ASCII* file that tabulates the running times for input sequences of varying lengths. Describe in that file how the input sequences are generated, and why that generation process is suitable. (Example: Do not exclusively use sequences that contain only positive or only negative numbers.) Explain any discrepancies between your results and those predicted by the textbook. Also include in your submission a *portable PDF* [sic] file that depicts the running time data in a suitable graphical form.

   Note the file names here:
   text results and explanation: ______________________________
   graphical results: ______________________________

10. (90 pts.) Implement your algorithms of Question 8 as part of the simple record-manager application described below. Package and submit your source code as in the previous assignment, being sure to include adequate documentation. Poorly written or poorly documented code is likely to receive zero credit.

    The record manager stores *key-data pairs* of the form (k, d) where k is a non-negative integer and d is floating point number. (You may assume that k and d fit in Java's `int` and `float` types.) All user interaction with the record manager is through a text-mode command language based on the standard-input/standard-output interface. The input consists of one command per line. The record manager reads and responds to each command in turn. Except for the `xp` command, the response to each command is also a single newline-terminated line. The syntax and semantics of the commands, and the desired outputs, are as follows. We use the symbol ␣ to denote a single space character.

[1] Mark Allen Weiss, *Data Structures and Problem Solving Using Java*, 4th edition (Addison-Wesley, 2010), §5.3.

command	actions
`c␣k`	Create a record manager that contains no records and that uses a k-search-tree with the given value of `k`. The output is an empty line (single newline character).
`s␣k␣d`	The pair (k, d) is stored in the record manager. If a pair of the form (k, d') already exists, then the new pair replaces it. The output is an empty line.
`e␣k`	If the record manager contains the key k then the output is `1` else the output is `0`.
`r␣k`	The data d associated with key k is retrieved from the record manager. The output is d. If the key k is absent from the record store, the output consists of an empty line.
`d␣k`	If there is a record with key k then it is deleted. The output is an empty line.
`xs`	The output is the size of the tree (a single integer).
`xh`	The output is the height of the tree.
`xa`	The output is a list of the key-data pairs in the tree, in preorder. All pairs are printed on a single line, with a single space separating adjacent pairs. Each pair (k, d) is printed using the parenthesized format `(`k`␣`d`)`.
`xb`	The output is similar to that of `xa` but the nodes are to be listed in postorder.
`xp`	The output consists of the following *two-dimensional text representation* of the current state of the tree used to store the records. This representation of a tree t contains one line for each node of t. The line representing a node $n \in t$ consists of n's (k, d) pair, printed as in the `xa` command, prefixed with $3d$ spaces, where d is the depth of n in t. Further, the line representing a node is printed before those of its descendants, and all the lines representing the descendants of a node precede the line representing that node's right sibling, if any.

Ensure that (1) your program produces exactly the output described above, with no spurious text such as extra spaces or newlines, prompts, or other messages and (2) your program reads from standard input and writes to standard output, with no additional assumptions on either.

11. (10 pts.) Conduct an experimental evaluation of each command of your record-manager implementation, following the guidelines of Question 9.

    Note the file names here:
    text results and explanation: ____________________
    graphical results: ____________________

12. (20 pts.) $\star$ Describe an efficient algorithm for generating all k-search-trees with n nodes, with labels drawn from L, where k, n, and L are provided as input. Provide an English description and pseudocode. Explain why the algorithm is correct. Determine and prove its running time. Implement the algorithm and conduct an experimental study of the running time. Package and submit your implementation as in earlier questions.

13. (20 pts.) $\star$ Describe modifications to the algorithms of Question 8 that ensure that the height of a k-search-tree with n keys is $O(\log n)$. Follow the instructions of Question 12.

14. (10 pts.) $\star$ Provide an efficient algorithm for pretty-printing a k-search-tree based on conventional graphical notation using ASCII graphics. Use the newsgroup for elaboration of this question and document and package your results as in Question 12.

Guidelines and reminders

- Refer to the class newsgroup for questions, hints, and additional guidelines.
- Do not define packages for your homework code.
- Format your code to fit well in 80-character lines.
- Use descriptive names for variables, classes, etc.
- Follow Java conventions for cases of names (e.g., MyClass, aMember).
- Test your code well outside of any IDE you may use.
- Ask for clarifications if you have any doubts.