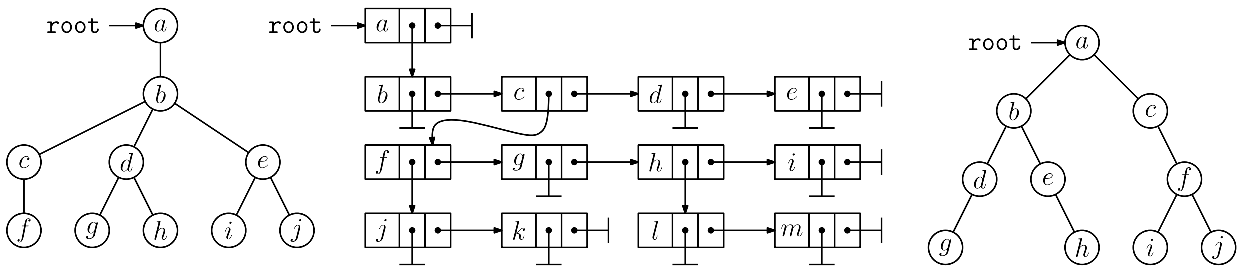
CMSC 420: Spring 2022

Homework 1: Basic Data Structures and Trees

Handed out Tue, Feb 8. Due at **11:59pm, Tue, Feb 15**. Point values given with each problem may vary. **Please see the notes at the end about submission instructions**.

Problem 1. (25 points) Answer the following questions involving the rooted trees shown in Fig. 1.

(a) (4 points) Consider the rooted tree of Fig. 1(a). Draw a figure showing its representation in the "first-child/next-sibling" form.



(a) (b) (c)

Figure 1: Rooted trees.

(b) (4 points) Consider the rooted tree of Fig. 1(b) represented in the "first-child/next-sibling" form. Draw a figure showing the equivalent rooted tree.

(c) (6 points) List the nodes of the tree of Fig. 1(a) in *preorder*. List them in *postorder*.

(d) (9 points) List the nodes of the tree of Fig. 1(c) in *preorder*. List them in *inorder*. List them in *postorder*.

(e) (2 points) Consider the binary tree of Fig. 1(c). Draw a figure showing the tree with inorder threads (as in Fig. 7 from the latex lecture notes for Lecture 3).

Problem 2. (5 points) Present pseudocode for a procedure `int getHeight(Node root)`, which is given the root of a tree represented using the first-child/next-sibling representation, and returns the height of the tree. For full credit your procedure should run in time proportional to the number of nodes in the tree. (For example, given the tree shown in Fig. 1(b), your function would return three.)

Give a short explanation in English how your procedure works. **Hint:** Use recursion.

Problem 3. (5 points) You have a binary trees in which each node, in addition to having links `left`, `right`, has a link `parent`, which points to the node's parent (and note that `root.parent == null`).

Present pseudocode for a function `Node inorderSuccessor(Node p)`, which returns `p`'s inorder successor, that is, the node that follows `p` in an inorder traversal. If `p` is the last node in the inorder traversal, your function should return `null`. (For example, given the tree shown in Fig. 1(c), `inorderSuccessor(c)` would return the node labeled "`i`" and `inorderSuccessor(h)` would return the node labeled "`a`".)

Give a short explanation in English how your procedure works.

Hint: You should **not** assume that this is a binary search tree. If you want to know whether a node `p` is the left or right child of its parent, you can do "`if (p == p.parent.left)`". Of course, beware of dereferencing `null` pointers.

Problem 4. (5 points) Suppose that you have a rooted tree, where all the leaves are at the same depth. We partition the nodes of the tree into levels as follows. The leaves are at level 0, their parents are at level 1, their grandparents are at level 2, and so on up to the root, which is at some level L (see Fig. 2). Let n denote the number of leaf nodes (all at level 0), and generally, for $0 \leq i \leq L$, let n_i denote the number of nodes on level i of this tree.

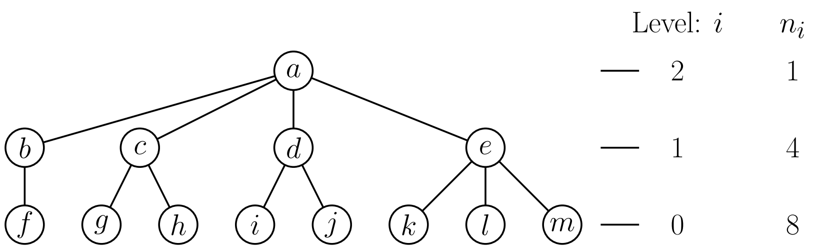


Figure 2: A tree with maximum level $L = 2$.

Suppose that the number of nodes in each level decreases by at least some constant factor, that is, suppose that there is a fixed real number $0 < \alpha < 1$ (which does not depend on n or L) such that $n_{i+1} \leq \alpha n_i$, for $0 \leq i < L$. Prove that there exists a constant c (depending on α) such that $L \leq c \lg n$. (Recall that lg means logarithm base 2.) You should derive the smallest value of c such that this holds.

Hint: If you have difficulty solving this, you can solve the following more concrete version for half credit. Suppose that the number of nodes in each level decreases by at least one third, that is, $n_{i+1} \leq n_i/3$. Prove that $L \leq (\lg n)/(\lg 3)$. (Recall that lg means logarithm base 2.)

Problem 5. (10 points) In this problem, we will consider modification and generalization of the amortized analysis of the dynamic stack algorithm from Lecture 2. We will make two changes: (1) we will slightly change the algorithm and cost model when expanding the stack, and (2) we will allow the stack to contract when the number of elements gets too small.

Throughout, let n denote the number of elements in the stack, and let m denote the size of the current array. Here is a formal description of our new dynamic stack and the actual cost of the two stack operations. We assume that we start with an array of size $m = 1$ containing $n = 0$ elements. Throughout, we maintain the condition that (unless the stack is empty) $\frac{m}{4} < n < m$.

`push(x)`: Add x to the top of the stack and increase n by one. (This is always possible, by our assumption that $n < m$).

If $n < m$ (normal case), we are done, and the actual cost is $+1$. On the other hand, if $n = m$ (overflow case), we double the array size (setting $m \leftarrow 2m$), allocate a new array of this doubled size, copy the contents of the stack into the new array (see Fig. 3(a)). Letting n denote the number of elements *after* the push, the actual cost is $n + 1$ ($+1$ for the push, and n for the time to copy the elements). Observe that the new array is exactly half full.

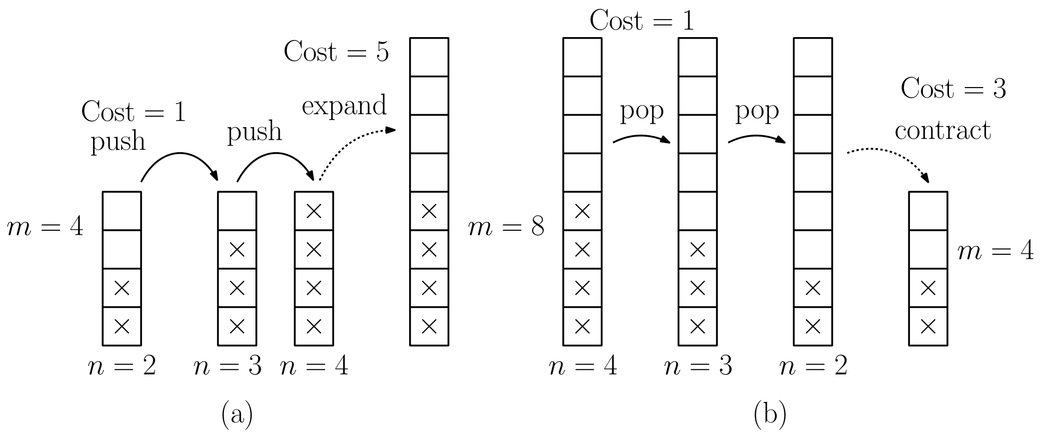


Figure 3: Expanding/Shrinking stack.

`pop()`: If $n = 0$, there is nothing to pop, and we return `null`, at an actual cost of $+1$. Otherwise, we pop the top element from the stack, and decrease n by one.

If after the pop, $n > \frac{m}{4}$ (normal case), we are done, and the actual cost is $+1$. On the other hand, if after the pop we have $n \leq \frac{m}{4}$ and $m \geq 2$ (underflow case), we halve the array size (setting $m \leftarrow \frac{m}{2}$), allocate a new array of this halved size, and copy the contents of the stack into the new array (see Fig. 3(b)). Letting n denote the number of elements *after* the pop, the actual cost is $n + 1$ ($+1$ for the pop, and n for the time to copy the elements). Observe that the new array is exactly half full.

The objective of this problem is to show that, over a long sequence of operations, the amortized cost (that is, the total actual cost divided by the number of operations) is some constant. We assume that we start within an empty stack ($n = 0$ and $m = 1$). Define *run* to be the sequence of operations starting just after the last array reallocation and running through the next array reallocation.

(a) (5 points) Suppose that the array size is m at the start of the run (and hence $n = m/2$), and the run ends with an *expansion* to size $2m$. Prove that there exists is a constant α_1 so that the amortized cost of the run (that is, the total cost of operations divided by the number of operations) is at most α_1.

(b) (5 points) Suppose that the array size is m at the start of the run (and hence $n = m/2$), and the run ends with a *contraction* to size $\frac{m}{2}$. Prove that there exists is a constant α_2 so that the amortized cost of the run (that is, the total cost of operations divided by the number of operations) is at most α_2.

For full credit, in each case compute the smallest value of α that works. You may assume that n is very large, so small additive constant terms do not matter.

Note: Challenge problems are not graded as part of the homework. The grades are recorded separately. After final grades have been computed, I may "bump-up" a grade that is slightly below a cutoff threshold based on these extra points. (But there is no formal rule for this.)

Challenge Problem: Consider the setup in Problem 4 but suppose that the number of nodes decreases even faster. In particular, suppose that $n_{i+1} \leq \sqrt{n_i}$. Prove that there is a constant c such that $L \leq c \lg\lg n$ (that is, the log of the log of n). Since $\sqrt{1} = 1$, we could loop infinitely at the root level. Let's assume we end at level L, where $n_L = 2$.

General note regarding coding in homeworks: A common question at the start of the semester is "how much detail are you expecting?" You will figure this out as the semester goes on, but here are some basic guidelines.

Prove vs. Show: If we ask you to "prove" something, we are looking for a well structured proof. If you are applying induction, please be careful to distinguish your basis case(s) and indicate what your induction hypothesis is. If we ask you to "show," "explain," or "justify", we are usually just expecting a brief English explanation. If you are unsure, please check.

Algorithm vs. Pseudocode: When we ask for an "algorithm" we are expecting a high-level description of some computational process, usually in a combination of English and mathematical notation (e.g., "sort the n keys and locate x using binary search"). For the latter, we are expecting a more detailed step-by-step description that look much more like Java (e.g., "`Node q = p.left`").

Remember that you are writing your code to be read by a human, and not a Java compiler. Please omit extraneous details that are easily converted into Java. For example, it is easier to understand "`i =`$\lceil$`n/m`$\rceil$" than "`int i = (int) Math.ceil((double) n / (double) m))`".

Even if we do not explicitly ask for it, whenever you give an algorithm or pseudocode, **you should always provide a brief English explanation.** This helps the grader understand what your intentions are, and if there is a small error in your code, we can often use your explanation to understand what your actual intentions were. **Even if your solution is technically correct, we reserve the right to deduct points if it is not clear to us why it is correct.**

Submission Instructions: Please submit your assignment as a pdf file through Gradescope. Here are a few instructions/suggestions:

- You can typeset, hand-write, or use a tablet or any combination. We just need a readable pdf file with all the answers. Be generous with figures and examples. If there is a minor error in your pseudo-code, but the figure illustrates that you understood the answer, we can give partial credit.
- When you submit, Gradescope will ask you to indicate which page each solution appears on. **Please be careful in doing this!** It greatly simplifies the grading process for the graders, since Gradescope takes them right to the page where your solution starts. If done incorrectly,

the grader may miss your answer, and you may receive a score of zero. (If so, you can appeal. But hunting around for your answer is troublesome, and it is always best to keep the grader in a good mood!) This takes a few minutes, so give yourself enough time if you are working close to the deadline.

- Try to keep the answer to each subproblem (e.g. 5.2) on a single page. You can have multiple subproblems on the same page, but Gradescope displays one one page at a time. It is easiest to grade when everything needed is visible on the same page. If your answer spans multiple pages, it is a good idea to indicate this to alert the grader. (E.g., write "Continued" or "See next page" at the bottom of the page.)
- Most scanners (including your phone) do not take very good pictures of handwritten text. For this reason, write with dark ink on white paper. Use an image-enhancing app such as CamScanner or Genius Scan to improve the contrast.
- Writing can bleed through to the other side. To be safe, write on just one side of the paper.
- Students often ask me what typesetting system I use. I use LaTeX for text. This is commonly used by academicians, especially in math, CS, and physics, and is worth taking the time to learn if you are thinking about doing research. If you use LaTeX, I would suggest downloading an IDE, such as TeXnicCenter or TeXstudio. I draw my figures using a figure editor called IPE for drawing figures.