
Data Structures and Algorithms in Python

Michael T. Goodrich

Department of Computer Science
University of California, Irvine

Roberto Tamassia

Department of Computer Science
Brown University

Michael H. Goldwasser

Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter

1

Python Primer

Hints

Reinforcement

- R-1.1)** The modulo operator could be useful here.
- R-1.2)** Use bit operations.
- R-1.3)** Keep track of the smallest and largest value while looping.
- R-1.4)** Although there is a formula for this, the easy thing to do is to write a loop.
- R-1.5)** How can you describe the range of integers for the sum?
- R-1.6)** Consider modifying the range over which you loop.
- R-1.7)** How can you describe the range of integers for the sum?.
- R-1.8)** Give your answer in terms of n and k .
- R-1.9)** Where does the sequence start and end? What is the step size?
- R-1.10)** Use a negative step size.
- R-1.11)** Those look like powers of two!
- R-1.12)** Use `randrange` to pick the index of the chosen element.

Creativity

- C-1.13)** The Python function does not need to be passed the value of n as an argument.
- C-1.14)** Note that both numbers in the pair must be odd.
- C-1.15)** The simple solution just checks each number against every other one, but we will discuss better solutions later in the book. But make sure you don't compare a number to itself.
- C-1.16)** Think about the semantics of `data[j] = data[j] * factor`.
- C-1.17)** Try it out and see if it works!
- C-1.18)** What are the factors of each number?
- C-1.19)** Use the `chr` function with appropriate range

- C-1.20)** Consider randomly swapping an element to the first position, then randomly swapping a remaining element to the second position, and so on.
- C-1.21)** Use a list to store all the lines.
- C-1.22)** Go back to the definition of dot product and write a for loop that matches it.
- C-1.23)** Use a try-except structure.
- C-1.24)** You can use the condition `ch in 'aeiou'` to test if a character is a vowel.
- C-1.25)** Consider each character one at a time.
- C-1.26)** Try a case analysis for each pair of integers and an operator.
- C-1.27)** Either buffer the bigger value from each pair of factors, or repeat the loop in reverse to avoid the buffer.
- C-1.28)** Use the `**` operator to compute powers.

Projects

- P-1.29)** There are many solutions. If you know about recursion, the easiest solution uses this technique. Otherwise, consider using a list to hold solutions. If this still seems too hard, then consider using six nested loops (but avoid repeating characters and make sure you allow all string lengths).
- P-1.30)** This is the same as the logarithm, but you can use recursion here rather than calling the log function.
- P-1.31)** While not always optimal, you can design your algorithm so that it always returns the largest coin possible until the value of the change is met.
- P-1.32)** Do a case analysis to categorize each line of input.
- P-1.33)** Write your program to loop continually until a quit operation is entered. In each iteration, collect a sequence of button pushes, and then output the result from processing that sequence of pushes.
- P-1.34)** Define a way of indexing all the sentences and the location in each one and then work out a way of picking eight of these locations for a typo.
- P-1.35)** Use a two-dimensional list to keep track of the statistics and a one-dimensional list for each experiment.
- P-1.36)** You need some way of telling when you have seen the same word you have before. Feel free to just search through your list of words to do this here.

Chapter

2

Object-Oriented Programming

Hints

Reinforcement

- R-2.1)** Think of applications that could cause a death if a computer failed.
- R-2.2)** Consider an application that is expected to change over time, because of changing economics, politics, or technology.
- R-2.3)** Consider the File or Window menus.
- R-2.4)** Consider using get and set methods for accessing and modifying the values.
- R-2.5)** Read about exception handling in Chapter 1.
- R-2.6)** Read about exception handling in Chapter 1.
- R-2.7)** Read about default parameter values in Chapter 1.
- R-2.8)** Try to make the last card over its limit.
- R-2.9)** The code should look very similar to `--add--`.
- R-2.10)** Create a vector of the appropriate length and then set its coordinates.
- R-2.11)** You will need to define the `--radd--` method.
- R-2.12)** Create a vector of the appropriate length and then set its coordinates.
- R-2.13)** You should be able to reuse your implementation of `--mul--`.
- R-2.14)** Remember that you are returning a single number (not a vector).
- R-2.15)** Use the `isinstance` function to determine the operand type.
- R-2.16)** If we were to increase the stop value, one at a time, at what point would a new value appear in the range?
- R-2.17)** Review the definition of inheritance diagram, and begin your drawing with object as the highest box.
- R-2.18)** Your program should output 42, which Douglas Adams considers to be the answer to the ultimate question of life, universe, and everything.
- R-2.19)** Try it out.

- R-2.20)** Think about what happens when a new instance of class Z is created and when methods of class Z are called.
- R-2.21)** Think about code reuse.
- R-2.22)** Be especially careful when the two sequences do not have the same length.
- R-2.23)** Be especially careful when one sequence is a prefix of another.

Creativity

- C-2.24)** Create a separate class for each major behavior.
- C-2.25)** Use the `isinstance` function to determine the operand type.
- C-2.26)** Think about how the internal counter should be initialized.
- C-2.27)** Consider the difference between the target value and the start of the range, and the step size for that range.
- C-2.28)** The key is being able to accurately track how many times `charge` has been called thus far during a month.
- C-2.29)** You will need to keep track of how much payment has been received in the current month.
- C-2.30)** Make sure to test your modified code.
- C-2.31)** Model your solution after our other subclasses of `Progression`.
- C-2.32)** Use the `sqrt` function in the `math` module.

Projects

- P-2.33)** If you have not had calculus, you can look up the formula for the first derivative of a polynomial on the Internet.
- P-2.34)** You don't have to use GUI constructs; simple text output is sufficient, say, using X's to indicate the values to print for each bar (and printing them sideways).
- P-2.35)** Use three different classes, for each of the actors, and provide methods that perform their various tasks, as well as a simulator engine that performs the periodic operations.
- P-2.36)** When a fish dies, set its associated cell back to **None**.
- P-2.37)** Use random number generation for the strength field.
- P-2.38)** Create a separate class for each major behavior. Find the available books on the Internet, but be sure they have expired copyrights.
- P-2.39)** Look up the formulas for area and perimeter on the Internet.

Data Structures and Algorithms in Python

Michael T. Goodrich

Department of Computer Science
University of California, Irvine

Roberto Tamassia

Department of Computer Science
Brown University

Michael H. Goldwasser

Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter

3

Algorithm Analysis

Hints

Reinforcement

- R-3.1)** Use powers of two as your values for n .
- R-3.2)** Set the running times equal, use algebra to simplify the equation, and then various powers of two to home in on the right answer.
- R-3.3)** Set both sides equal to each other to determine this.
- R-3.4)** Any growing function will have a “flatter” curve on a log-log scale than it has on a standard scale.
- R-3.5)** Think of another way to write $\log n^c$.
- R-3.6)** Characterize this in terms of the sum of all integers from 1 to n .
- R-3.7)** Use the fact that if $a < b$ and $b < c$, then $a < c$.
- R-3.8)** Simplify the expressions, and then use the ordering of the seven important algorithm-analysis functions to order this set.
- R-3.9)** Review the definition of big-Oh and use the constant from this definition.
- R-3.10)** Start with the product and then apply the definition of the big-oh for $d(n)$ and then $e(n)$.
- R-3.11)** Use the definition of the big-oh and add the constants (but be sure to use the right n_0).
- R-3.12)** You need to give a counterexample. Try the case when $d(n)$ and $e(n)$ are both $O(n)$ and be specific.
- R-3.13)** Use the definition of the big-oh first to $d(n)$ and then to $f(n)$ (but be sure to use the right n_0).
- R-3.14)** First show that the max is always less than the sum.
- R-3.15)** Simply review the definitions of big-oh and big-omega. This one is easy.
- R-3.16)** Recall that $\log n^k = k \log n$.
- R-3.17)** Notice that $(n + 1) \leq 2n$ for $n \geq 1$.

R-3.18) $2^{n+1} = 2 \cdot 2^n$.

R-3.19) Make sure you don't get caught by the fact that $\log 1 = 0$.

R-3.20) Use the definition of big-omega, but don't get caught by the fact that $\log 1 = 0$.

R-3.21) Use the definition of big-omega, but don't get caught by the fact that $\log 1 = 0$.

R-3.22) If $f(n)$ is a positive nondecreasing function that is always greater than 1, then $\lceil f(n) \rceil \leq f(n) + 1$.

R-3.23) Consider the number of times the loop is executed and how many primitive operations occur in each iteration.

R-3.24) Consider the number of times the loop is executed and how many primitive operations occur in each iteration.

R-3.25) Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the outer loop.

R-3.26) Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the outer loop.

R-3.27) Consider the number of times the inner loop is executed and how many primitive operations occur in each iteration, and then do the same for the two outer loops.

R-3.28) You can do all rows except for $n \log n$ just by setting the function equal to the value and solving for n . For the $n \log n$ function, the easiest technique is unfortunately to simply use trial-and-error on a calculator.

R-3.29) The $O(\log n)$ calculation is performed n times.

R-3.30) The $O(n)$ calculation is performed $\log n$ times.

R-3.31) Consider the cases when all entries of S are even or odd.

R-3.32) First characterize the running time of Algorithm D using a summation.

R-3.33) Discuss how the definition of the big-oh fits into Al's claim.

R-3.34) Recall the definition of the Harmonic number, H_n .

Creativity

C-3.35) Use sorting as a subroutine.

C-3.36) Note that 10 is a constant!

C-3.37) Think of a function that grows and shrinks at the same time without bound.

C-3.38) Use induction, a visual proof, or bound the sum by an integral.

C-3.39) 1

C-3.40) Use the log identity that translates $\log bx$ to a logarithm in base 2.

C-3.41) 1

C-3.42) Consider the sum of the maximum number of visits each friend can make without visiting his/her maximum number of times.

C-3.43) You need to line up the columns a little differently.

C-3.44) Characterize the number of bits needed first.

C-3.45) Consider computing a function of the integers in S that will immediately identify which one is missing.

C-3.46) Consider the first induction step.

C-3.47) Consider the contribution made by one line.

C-3.48) Look carefully at the definition of big-Oh and rewrite the induction hypothesis in terms of this definition.

C-3.49) Use the definition of big-omega, and make $n = 1$ and $n = 2$ your base cases.

C-3.49) Use the definition of big-Omega, and make $n = 1$ and $n = 2$ your base cases.

C-3.50) Consider writing a pseudo-code description of this algorithm and note its loop structure.

C-3.51) Try to bound from above each term in this summation.

C-3.52) Try to bound a significant number of the terms from below.

C-3.53) Number each bottle and think about the binary expansion of each bottle's number.

C-3.54) Use an auxiliary array that keeps counts for each value.

Projects

P-3.55) Choose representative values of the input size n , and run at least 5 tests for each size value n .

P-3.56) Try to reuse your code as much as possible.

P-3.57) You should try several runs over many different problem sizes.

P-3.58) Do a type of “binary search” to determine the maximum effective value of n for each algorithm.

Chapter

4

Recursion

Hints

Reinforcement

- R-4.1)** Don't forget about the space used by the function stack.
 - R-4.2)** This is probably the first power algorithm you were taught.
 - R-4.3)** Be sure to get the integer division right.
 - R-4.4)** You can model your figure after Figure 4.11.
 - R-4.5)** You should draw small boxes or use a big paper, as there are a lot of recursive calls.
 - R-4.6)** Start with the last term.
 - R-4.7)** Process the string left to right.
 - R-4.8)** Look for a geometric series.
-

Creativity

- C-4.9)** Consider returning a tuple, which contains both the minimum and maximum value.
- C-4.10)** The integer part of the base-two logarithm of n is the number of times you can divide by two before you get a number less than 2.
- C-4.11)** Consider reducing the task of telling if the elements of a sequence are unique to the problem of determining if the last $n - 1$ elements are all unique and different than the first element.
- C-4.12)** You need subtraction to count down from m or n and addition to do the arithmetic needed to get the right answer.
- C-4.13)** Define a recurrence equation.
- C-4.14)** 1
- C-4.15)** Start by removing the first element x and computing all the subsets that don't contain x .

- C-4.16)** You can use syntax `print(ch, end= ' ')` to print one character `ch` at a time, without extraneous spaces.
- C-4.17)** Check the equality of the first and last characters and recur (but be careful to return the correct value for both odd- and even-length strings).
- C-4.18)** Write your recursive function to first count vowels and consonants.
- C-4.19)** Consider whether the last element is odd or even and then put it at the appropriate location based on this and recur.
- C-4.20)** Begin by comparing the first and last elements in a range of indices in A .
- C-4.21)** The beginning and the end of a range of indices in S can be used as arguments to your recursive function.
- C-4.22)** You can rely on bitwise operations to interpret n in binary.

Projects

- P-4.23)** Review use of the `os` module.
- P-4.24)** Use recursion in your main solution engine.
- P-4.25)** Consider a small example to see why the binary representation of the counter is relevant.
- P-4.26)** Note the recursive nature of the problem.
- P-4.27)** Review use of the other methods of the `os` module.

Chapter

5

Array-Based Sequences

Hints

Reinforcement

- R-5.1)** Although the speed may differ, the asymptotics should be similar.
- R-5.2)** Keep track of the maximum data size thus far.
- R-5.3)** You might want to make the list relatively large before you begin to remove entries.
- R-5.4)** Note that an index such as -3 is equivalent to the traditional index $n - 3$ for a list of length n .
- R-5.5)** Now we are charging more for the growing, so we need to store more cyber-dollars with each element. Calculate the number needed for growing and this will help you determine the number you need to save, which in turn tells you one less than you need to charge each push.
- R-5.6)** You are going to need two (non-nested) loops.
- R-5.7)** You don't need to sort A .
- R-5.8)** Do the observed results match what you expected?
- R-5.9)** The alphabets for most alphabet-based languages are included in the Unicode character encoding standard.
- R-5.10)** Review the list comprehension syntax from Section 1.9.2.
- R-5.11)** You should use nested loops.
- R-5.12)** Consider how to build a list of subtotals, one for each nested list.

Creativity

- C-5.13)** Use list comprehension to make an initial list of a specific size.
- C-5.14)** Consider randomly shuffling the deck one card at a time.
- C-5.15)** Consider how much cyber “money” is saved up from one expansion to the next.
- C-5.16)** The existing `_resize` method can be used to shrink the array.

- C-5.17)** You can predict precisely when a resize occurs during this process, and take the sum of those costs.
- C-5.18)** Apply the amortized analysis accounting technique using a monetary accounting scheme with extra funds for both insertions and removals.
- C-5.19)** Consider, after any resize takes place, how many subsequent operations are needed to force another resize.
- C-5.20)** Try to oscillate between growing and shrinking.
- C-5.21)** See Code Fragment 5.4 for an example use of Python's time module.
- C-5.22)** You might design an experiment similar to that in Code Fragment 5.4.
- C-5.23)** See Code Fragment 5.4 for an example use of Python's time module.
- C-5.24)** Use the time module together with loops such as those given on page 205.
- C-5.25)** You must be very careful if modifying a list at the same time that you loop through its elements!
- C-5.26)** It might help to sort B .
- C-5.27)** You might wish to use an auxiliary array of size at most $4n$.
- C-5.28)** Argue why you have to look at all the integers in L .
- C-5.29)** Be sure to allow for the case where every pair (x,y) in A and every pair (y,z) in B have the same y value.
- C-5.30)** You should be able to achieve $O(n)$ time.
- C-5.31)** Recall that `binary_sum` from Section 4.4.2 computes the sum of numbers in a one-dimensional list.

Projects

- P-5.32)** The entries $A[i][j][k]$ and $B[i][j][k]$ are the ones that need to be added.
- P-5.33)** Matrix addition is defined so that if $C = A + B$, then $C[i, j] = A[i, j] + B[i, j]$. Matrix multiplication is defined so that if $C = AB$, where A is a $c \times d$ matrix and B is a $d \times e$ matrix, then $C[i, j] = \sum_{k=0}^d A[i, k]B[k, j]$. That is, C is a $c \times e$ matrix.
- P-5.34)** You will probably need separate encrypt and decrypt mappings for the upper- and lower-case characters.
- P-5.35)** The original CaesarCipher implementation was already effectively a substitution cipher, with a specifically chosen encoder pattern.

P-5.36) If you get the constructor to use the correct encoder string, everything else should work.

P-5.37) A good way to generate a random encryption array is to start with the alphabet array. Then for each letter in this array, randomly swap it with some other letter in the array.

Chapter

6

Stacks, Queues, and Deques

Hints

Reinforcement

- R-6.1)** Use a paper and pencil with eraser to simulate the stack.
- R-6.2)** If a stack is empty when pop is called, its size does not change.
- R-6.3)** Transfer items one at a time.
- R-6.4)** First check if the stack is already empty.
- R-6.5)** Use two loops.
- R-6.6)** Give a recursive definition.
- R-6.7)** Use a paper and pencil with eraser to simulate the queue.
- R-6.8)** If a queue is empty when dequeue is called, its size does not change.
- R-6.9)** Each successful dequeue operation causes that index to shift circularly to the right.
- R-6.10)** Consider how the queue might be configured within the underlying array.
- R-6.11)** Read the documentation of `collections.deque`, if needed.
- R-6.12)** Use a paper and pencil to simulate the deque.
- R-6.13)** You may use the return value of a removal method as a parameter to an insertion method.
- R-6.14)** You may use the return value of a removal method as a parameter to an insertion method. Think about how to effectively use the stack for temporary storage.

Creativity

- C-6.15)** Pop the top integer, but remember it.
- C-6.16)** Use a new instance variable to store the capacity limit.
- C-6.17)** Use an expression such as `[None] * k` to build a list of `k` **None** values.
- C-6.18)** You will need to do three transfers.
- C-6.19)** After finding what's between the `<` and `>` characters, the tag is only the part before the first space (if any).
- C-6.20)** Use a stack to reduce the problem to that of enumerating all permutations of the numbers $\{1, 2, \dots, n - 1\}$.
- C-6.21)** Use the stack to store the elements yet to be used to generate subsets and use the queue to store the subsets generated so far.
- C-6.22)** Use a stack.
- C-6.23)** You can still use `R` as temporary storage, as long as you never pop its original contents.
- C-6.24)** Rotate elements within the queue.
- C-6.25)** Consider using one stack to collect incoming elements, and another as a buffer for elements to be delivered.
- C-6.26)** Think of using one stack for each end of the deque.
- C-6.27)** Think of how you might use `Q` to process the elements of `S` twice.
- C-6.28)** Use a new instance variable to store the capacity limit.
- C-6.29)** You might start by combining the code of a dequeue followed by an enqueue, and then simplify.
- C-6.30)** Think of the queues like boxes and the integers like red and blue marbles.
- C-6.31)** Lazy and Crazy should only go across once.

Projects

- P-6.32)** Suggested instance variables are described in the book.
- P-6.33)** What is the index of a new element? What is the index of the old element that is lost?
- P-6.34)** You will need to use a stack.
- P-6.35)** How does this functionality compare to a deque?
- P-6.36)** Keep information about the purchase shares and prices in a queue, and then match those against sales. Care must be taken if only part of a purchase block is sold.
- P-6.37)** Start one stack at each end of the array, growing toward the center.

Hints

Reinforcement

- R-7.1)** It is okay to have an algorithm running in linear time.
- R-7.2)** This concatenation operation need not search all of L and M .
- R-7.3)** Consider passing a node as a parameter.
- R-7.4)** Performing the swap for a singly linked list will take longer than for a doubly linked list.
- R-7.5)** You need to keep track of where you start or your method will have an infinite loop.
- R-7.6)** Your only need to go around one of the lists once.
- R-7.7)** You must adjust links so that the first node is moved to the end of the list.
- R-7.8)** Consider a combined search from both ends. Also, recall that a link hop is an assignment of the form " $p = p.\text{next}$ " or " $p = p.\text{prev}$ ".
- R-7.9)** Splice the end of L into the beginning of M .
- R-7.10)** Is there a scenario in which these substitutions fail?
- R-7.11)** Keep track of the maximum thus far while walking the list
- R-7.12)** Within a method of the class, you may access nonpublic members
- R-7.13)** Start looking at the beginning of the list.
- R-7.14)** Consider parameterizing the method with a node of the list.
- R-7.15)** Model your solution on the original implementation with appropriate symmetry.
- R-7.16)** Be careful when working with an empty list.
- R-7.17)** Be careful to repair the list in the neighborhood abandoned by the moved node.

R-7.18) Implement the move-to-front using a pencil and eraser. Better yet, write the six letters on separate pieces of paper and simulate the actions physically.

R-7.19) Consider the two extreme cases of how we could distribute m accesses across n elements.

R-7.20) The first should be last, both physically and in terms of how long ago it has been accessed.

R-7.21) For this lower bound, assume that when an element is accessed we search for it by traversing the list starting at the front.

R-7.22) You can either clear the underlying list or start over with a new list.

R-7.23) You will need to adjust instances of the nested `_Item` class.

Creativity

C-7.24) Admittedly, it is not clear that there is any advantage to the sentinel for this purpose.

C-7.25) You should be able to avoid the conditional within enqueue.

C-7.26) Make sure to leave the head and tail members of both lists with appropriate values.

C-7.27) View the chain of nodes following the head node as themselves forming another list.

C-7.28) Recur on the first $n - 1$ positions.

C-7.29) Consider changing the orientation of links while making a single pass through the list.

C-7.30) Think carefully about the orientation of the linked list.

C-7.31) Consider using an abstraction that is a subset of the positional list ADT.

C-7.32) You should replace the `first()` and `last()` methods with a method abstracting the cursor.

C-7.33) You will need to carefully switch next and prev pointers and properly manage the sentinels.

C-7.34) Watch out for the special case when p and q are neighbors.

C-7.35) See Section 2.3.4 for discussion of iterators.

C-7.36) Watch out for special cases when the length is one or less.

C-7.37) To get you started, consider if the smallest and largest values add to V . If not, you should be able to eliminate one of the two as unnecessary.

C-7.38) It would be helpful to implement a swap subroutine.

- C-7.39)** Carefully map the public methods of the queue interface to the concrete behaviors of the PositionalList class.
- C-7.40)** Note well that there may be fewer than n elements included in the most recent n accesses, due to duplication.
- C-7.41)** Be sure to handle the case where every pair (x,y) in A and every pair (y,z) in B have the same y value.
- C-7.42)** You should keep track of the number of game entries explicitly.
- C-7.43)** Convert the two parts to two separate lists as sublists.

Projects

- P-7.44)** Use a position instance variable to keep track of the cursor location.
- P-7.45)** There is a trade-off between insertion and searching depending on whether the entries in L are sorted.
- P-7.46)** It is okay to be inefficient in this case.
- P-7.47)** Keep all cards in a single list, and use four positions to demark the beginning of the respective suits.

Hints

Reinforcement

R-8.1) Be sure not to get confused between depth (which is for a single node) and height (which is for the entire tree).

R-8.2) Recall that the worst-case running time for the depth method is $O(n)$.

R-8.3) Use the definitions of height and depth, and note that the height of the tree has to be realized by some path from the root to an external node. In addition, using induction is the easiest justification technique to use with this proposition.

R-8.4) Use a new parameter, n_p , which refers to the number of positions in the subtree rooted at p .

R-8.5) Review the binary tree ADT.

R-8.6) Consider using dummy nodes.

R-8.7) You need to give four values—the minimum and maximum number of internal nodes and the same for external nodes. In addition, one of these four values is 1.

R-8.8) Use the formulas presented in the book that relate external nodes, internal nodes, and height.

R-8.9) Consider how any proper binary tree with n nodes can be related to a proper binary tree with $n - 2$ nodes.

R-8.10) Although this is an abstract class, you may rely on its other (abstract) methods.

R-8.11) Each subtree is rooted at a node of the tree; give the value associated with each such node.

R-8.12) This one is a real puzzler, and it doesn't even use the operators $+$ and $\times$.

R-8.13) Review how arithmetic expressions can be represented with binary trees.

R-8.14) Assume that the container of children is implemented as a positional list.

R-8.15) We'll get you started...

```
class MutableLinkedBinaryTree(LinkedBinaryTree):
    def add_root(self, e):
        return self._add_root(e)
```

R-8.16) Think about a tree that is really a path.

R-8.17) Recall the definition of the level numbering and use this definition when passing information from a node to its children.

R-8.18) All of these methods can be easily performed using formulas involving level numbers.

R-8.19) Use a substitution of $g(p) = f(p) + 1$.

R-8.20) Draw the tree starting with the beginning and end characters of the two character strings.

R-8.21) Draw the tree on a separate sheet of paper and then mark nodes as they are visited, writing down the output produced for each visit.

R-8.22) Draw the tree on a separate sheet of paper and then mark nodes as they are visited, writing down the output produced for each visit.

R-8.23) Draw a tree with one node, one with two nodes, and then one with three nodes.

R-8.24) Draw a tree with one node and then one with three nodes (and note it is impossible to draw a proper binary tree with exactly two nodes).

R-8.25) We already got you started. . .

R-8.26) You can add all of c 's children to the fringe at once.

R-8.27) Draw the tree again and use a pencil and paper to mark how the recursive calls move around the tree.

R-8.28) Argue that this method basically performs the same computations as a familiar tree traversal method.

R-8.29) Consider the information and computations that need to be made in each of the three visits during an Euler tour traversal.

R-8.30) Regular expressions can be helpful for parsing such strings.

Creativity

C-8.31) Use the fact that we can build T from a single root tree via a series of pairs of `insertLeft` and `insertRight` operations.

C-8.32) The minimum value will occur when T has one external node.

- C-8.33)** Consider how you could possibly combine a tree with maximum depth with a tree with minimum depth.
- C-8.34)** First show that there is a node with three external node children, and then consider what changes when the children of that node are all removed.
- C-8.35)** Use the definition to derive a recursive algorithm.
- C-8.36)** Explore the different ways you can have empty nodes on the bottom level.
- C-8.37)** Explore with pen and paper.
- C-8.38)** Can you tell how many elements are removed?
- C-8.39)** There are many special cases to consider when p and q neighbor each other.
- C-8.40)** Try to avoid conditionals when possible.
- C-8.41)** Recursion can be very helpful.
- C-8.42)** Build the new tree in preorder fashion.
- C-8.43)** The answer to the first part of (c) is "yes".
- C-8.44)** Use a tree traversal.
- C-8.45)** Derive a formula that relates the depth of a position p to the depths of positions adjacent to p .
- C-8.46)** Modify an algorithm for computing the depth of each position so that it computes path lengths at the same time.
- C-8.47)** Try to compute node heights and balance factors at the same time.
- C-8.48)** Draw a picture of a proper binary tree and its preorder traversal and then hold this picture up to mirror.
- C-8.49)** First derive a formula for $post(p) - pre(p)$.
- C-8.50)** Think about what could be the worst-case number of nodes that would have to be traversed to answer each of these queries.
- C-8.51)** See Section 2.3.4 for discussion of iterators.
- C-8.52)** Use induction to show that no crossing edges are produced.
- C-8.53)** Use induction to show that no crossing edges are produced.
- C-8.54)** The y coordinates can be the same as in the binary tree case. The trick, then, is to find a good substitute for the inorder number used in the binary tree drawing algorithm.
- C-8.55)** `import os; help(os.walk).`
- C-8.56)** Consider a recursive algorithm.
- C-8.57)** Consider a recursive algorithm.
- C-8.58)** It helps to know the relative depths of p and q .

- C-8.59)** Be careful—the path establishing the diameter might not include the root of the tree.
- C-8.60)** Consider what numbers $f(p) + 1$, $f(q) + 1$, and $f(a) + 1$ look like in binary.
- C-8.61)** Parameterize your recursion by an index into the list of tokens which begins a subtree.
- C-8.62)** For the base case, you must determine whether the token is a numeric literal or a string variable.
- C-8.63)** Use the inherited postorder method.

Projects

- P-8.64)** What are natural update methods for this representation?
- P-8.65)** Use a Python list for the children.
- P-8.66)** You will need to reimplement the `_make_position` utility.
- P-8.67)** The essential part of this structure is just a binary tree, so take each part one step at a time.
- P-8.68)** This is a huge project, so plan accordingly.
- P-8.69)** Review the definition of this representation to get the details right.
- P-8.70)** Use recursion where appropriate.

Chapter

9

Priority Queues

Hints

Reinforcement

- R-9.1)** Each `remove_min()` operation takes $O(\log n)$ time.
- R-9.2)** Observe that the keys are guaranteed to satisfy the heap-order property. What other property does T need to satisfy in order to be a heap?
- R-9.3)** Use a simple list-based priority queue and a pencil with a good eraser.
- R-9.4)** There is a very good reason this exercise appears in this chapter.
- R-9.5)** Be prepared!
- R-9.6)** Sounds too good to be true.
- R-9.7)** Mimic the illustration style used in the book.
- R-9.8)** Mimic the illustration style used in the book.
- R-9.9)** Think about where insertion-sort has to put each added element and design your sequence so that insertion-sort has to put each next element as far as possible.
- R-9.10)** Where might the second smallest key be?
- R-9.11)** If the smallest is at the top of the heap...
- R-9.12)** Consider transforming the keys to invert their order.
- R-9.13)** Mimic the illustration style used in the book for insertion-sort and selection-sort.
- R-9.14)** Consider the heap-order property and the definition of the level number of a node in a tree.
- R-9.15)** Recall the definition of a complete binary tree.
- R-9.16)** The answers are “yes,no,yes.” Now all you have to do is to give examples for the yeses and a reason for the no.
- R-9.17)** The preorder sequence starts out 0, 1, 3, 7, ...
- R-9.18)** Consider the last $n/2$ terms in this sum.

- R-9.19)** Try to construct a heap that has larger elements in left subtrees.
- R-9.20)** Try to construct a heap that has larger elements in right subtrees.
- R-9.21)** Mimic the illustration style used in the book.
- R-9.22)** Mimic the illustration style used in the book.
- R-9.23)** You need to be very careful about how you partition the keys between the subtrees rooted at the children of the root.
- R-9.24)** Structure the insertions so that each requires lots of down-heap bubbling.
- R-9.25)** Mimic the illustration style used in the book.

Creativity

- C-9.26)** Figure out a way to time stamp the entries in the priority queue.
- C-9.27)** Figure out a way to time stamp the entries in the priority queue.
- C-9.28)** Is it ever possible that a new element gets a key that is strictly smaller than a previously inserted element?
- C-9.29)** Beware: `pop(0)` is expensive for a Python list.
- C-9.30)** Use a loop.
- C-9.31)** Use a loop.
- C-9.32)** Do simple up-and-down searches in the tree to locate the last node each time.
- C-9.33)** Think about what changes need to be made when leaves are created or destroyed.
- C-9.34)** Consider the binary expansion of $n - 1$, n , and $n + 1$.
- C-9.35)** Note that the entries do not need to be reported in sorted order. Use binary recursion on the subtrees of the heap and think about where the keys smaller than k are stored in the heap T .
- C-9.36)** Think carefully about how location-aware entries can be implemented efficiently.
- C-9.37)** Use Calculus.
- C-9.38)** Study the combine step in the bottom-up heap construction algorithm.
- C-9.39)** Make sure you understand the proper semantics when the parameter is smaller than all heap elements.
- C-9.40)** Make sure you understand the proper semantics when the parameter is smaller than all heap elements.
- C-9.41)** Use a suitably constructed heap.
- C-9.42)** Start by using the bottom-up construction.

- C-9.43)** Process elements one at a time, always storing the largest k that you have seen.
- C-9.44)** Create a new nested key type that wraps the provided keys to invert comparisons.
- C-9.45)** Write a short function that computes the number of 1's in the binary expansion of an integer by using the bitwise “and” operation.
- C-9.46)** Rather than using elements as keys in the priority queue, use the result of the key function for each element.
- C-9.47)** Partition the array into a sorted part and an unsorted part and use swaps to move elements around.
- C-9.48)** Partition the array into a sorted part and an unsorted part and use swaps to move elements around.
- C-9.49)** Use the right portion of the array to store the heap.
- C-9.50)** You will need two priority queues.
- C-9.51)** Use adaptable priority queues.
- C-9.52)** You will need two data structures that somehow keep “links” between each other.

Projects

- P-9.53)** Use as large of inputs as you can for experimentation.
- P-9.54)** Experiment with for which values of k your new implementation outperforms the original.
- P-9.55)** Use a heap for the priority queue.
- P-9.56)** Study again the bottom-up heap construction algorithm.
- P-9.57)** Use a heap for the CPU job priority queue.
- P-9.58)** Decide early how you are going to implement the “pointers” for your location-aware entries.

Chapter 10

Maps, Hash Tables, and Skip Lists

Hints

Reinforcement

- R-10.1)** This is more challenging because `__delitem__` does not return the deleted value.
- R-10.2)** You must rely on the `iter` method to access the contents of the map.
- R-10.3)** You should not directly call `__iter__`, but you can mimic its style.
- R-10.4)** The first insertion is $O(1)$, the second is $O(2)$, ...
- R-10.5)** Review the API for `PositionalList` from Section 7.4.
- R-10.6)** Think about which of the schemes use the array supporting the hash table exclusively and which of the schemes use additional storage external to the hash table.
- R-10.7)** Use of Python's `__hash__` method is discussed on page 415.
- R-10.8)** There is a lot of symmetry and repetition in this string, so avoid a hash code that would not deal with this.
- R-10.9)** Try to mimic the figure in the book.
- R-10.10)** Try to mimic the figure in the book.
- R-10.11)** The failure occurs because no empty slot is found. For the drawing, try to mimic the figure in the book.
- R-10.12)** Try to mimic the figure in the book.
- R-10.13)** Think of the worst-case time for inserting every entry in the same cell in the hash table.
- R-10.14)** Mimic the way the figure is drawn.
- R-10.15)** Allow the user to express the maximum load factor as an optional parameter to the constructor.
- R-10.16)** It is okay to insert a new entry on “top” of the deactivated entry object.

- R-10.17)** You will need to keep track of the number of probes in order to apply quadratic probing.
- R-10.18)** Think of where the entry with minimum key is stored.
- R-10.19)** The crucial methods are `__getitem__` and `__setitem__`.
- R-10.20)** Since the map will still contain n entries at the end, you can assume that each `__delitem__` operation takes the same asymptotic time.
- R-10.21)** What happens in the case where the middle table value equals k ?
- R-10.22)** Assume that a skip list is used to implement the sorted map.
- R-10.23)** Mimic the style of the figures in the book.
- R-10.24)** You must link out the removed entry's tower from all the lists it belongs to.
- R-10.25)** You will need to rely on the `iter` behavior to find an arbitrary element of the set.
- R-10.26)** For each element of the smaller set, check to see if it is contained in the larger set.
- R-10.27)** Something from this chapter should be helpful!

Creativity

- C-10.28)** The existing `__setitem__` implementation can serve as a reasonable model for this new method.
- C-10.29)** You might provide an implementation directly within the `ProbeHashMap`, but you'll need to borrow some techniques from the `HashMapBase` class.
- C-10.30)** You might provide an implementation directly within the `ChainHashMap`, but you'll need to borrow some techniques from the `HashMapBase` class.
- C-10.31)** 1
- C-10.32)** Prepare a concise table of your experimental results.
- C-10.33)** Fortunately, a Python list can be heterogeneous!
- C-10.34)** Oops. We meant to say to reimplement the `HashMapBase` class. Feel free to subclass the nested `Item` class.
- C-10.35)** You need to do some shifting of entries to close up the "gap" just made, but you should only do this for entries that need to move.
- C-10.36)** For part a, note that the symmetry will halve the range of possible values. For part b, note that such automatic collisions will not occur.
- C-10.37)** Perhaps you might define a nonpublic method that generates probe indices.
- C-10.38)** Consider the way `find_range` was implemented for `SortedTableMap`.

- C-10.39)** Try out some examples.
- C-10.40)** Do a “double” binary search.
- C-10.41)** Dovetail two binary searches.
- C-10.42)** Think first about how you can determine the number of 1’s in any row in $O(\log n)$ time.
- C-10.43)** Think about first sorting the pairs by cost.
- C-10.44)** 1
- C-10.45)** Consider augmenting each node v in a higher level with the number of missing entries in the gap from v to the next node over.

- C-10.47)** Make sure to start with a new empty set (or make a copy of one of the two initial sets).
- C-10.48)** Think of how you could transform D into L .
- C-10.49)** Maintain a secondary `PositionalList` instance that represents the FIFO order

Projects

- P-10.50)** In a Unix/Linux system, a good place to start is `/usr/dict`.
- P-10.51)** It is okay to generate these phone numbers more-or-less at random.
- P-10.52)** You might consider embedding a next pointer within each item composite.
- P-10.53)** Sentinels can be used in place of the theoretical $-\infty$ and $+\infty$.
- P-10.54)** Try to make your screen images mimic the skip list figures in the book.
- P-10.55)** For each word t that results from a minor change to s , you can test if t is in W in $O(1)$ time.

Chapter

11

Search Trees

Hints

Reinforcement

- R-11.1)** Recall the definition of where we perform an insertion in a binary search tree.
- R-11.2)** You will need to draw 8 trees, but they are all small.
- R-11.3)** You can enumerate them with pictures.
- R-11.4)** Try a few examples of five-entry binary search trees.
- R-11.5)** Try a few examples of five-entry AVL trees.
- R-11.6)** Use a loop to express the repetition
- R-11.7)** There is one of each type. Which one is which?
- R-11.8)** Mimic the figure in the book.
- R-11.9)** Mimic the figure in the book.
- R-11.10)** Think about the data movements needed in an array list representation of a binary tree.
- R-11.11)** Carefully note the heights of all subtrees before the deletion, and after the deletion but before the restructuring.
- R-11.12)** Carefully note the heights of all subtrees before the deletion, and after the deletion but before the restructuring.
- R-11.13)** Carefully trace the potential heights of various subtrees.
- R-11.14)** Use a pencil with a good eraser.
- R-11.15)** Each entry is splayed to the root in increasing order.
- R-11.16)** No. Why not?
- R-11.17)** It is not k_1 . Why?
- R-11.18)** You will need at list five entries to find a counterexample.
- R-11.19)** Use the correspondence rules described in the chapter.
- R-11.20)** Use a pencil with a good eraser.
- R-11.21)** Use a pencil with a good eraser.

R-11.22) Consider looking at the (2,4) tree and red-black tree definitions again.

R-11.23) Recall the definition of a binary search tree, in general.

R-11.24) Some have $O(\log n)$ worst-case height and some have $O(n)$ worst-case height. Make sure you know which. Also, try to get the constant factors right in this case.

R-11.25) You need to create a node that does not satisfy the AVL balance condition, but would be acceptable in a red-black tree. A good example would be a tree with at least 6 nodes, but no more than 16.

R-11.26) Note that the black-path length must have been identical for each path downward from p .

R-11.27) Note that the black-path length must have been identical for each path downward from p .

R-11.28) Note that the black-path length must have been identical for each path downward from p .

Creativity

C-11.29) The method is similar to priority-queue sorting.

C-11.30) Review what it means for splay trees to have $O(\log n)$ amortized time performance.

C-11.31) You should make a single call to utility `_subtree_search`.

C-11.32) Show that $O(n)$ rotations suffice to convert any binary tree into a *left chain*, where each internal node has an external right child.

C-11.33) Where might the search path for k diverge from a path to one of the other keys?

C-11.34) Consider the maximum number of times the recursive method is called on a position that is not within the subrange.

C-11.35) Consider a top-down recursive approach.

C-11.36) Make sure that the result is a valid AVL tree.

C-11.37) Note that this method returns a single integer, so it is not necessary to visit all s items that lie in the range. You will need to extend the tree data structure, adding a new field to each node.

C-11.38) How do the rebalancing actions affect the information stored at each node?

C-11.39) Use triangles to represent subtrees that are not affected by this operation, and think of how to cascade the imbalances up the tree.

C-11.40) Think carefully about how the balance of a node and its ancestors changes immediately after an insertion or deletion.

- C-11.41)** Just consider the operations that could change the leftmost position or who points to it.
- C-11.42)** How do the rebalancing actions affect the minimum?
- C-11.43)** Have each node of the tree maintain references to its inorder neighbors.
- C-11.44)** How do the rebalancing actions affect the inorder relationships?
- C-11.45)** Carefully review the steps of deletion when given a direct reference to the position to be deleted.
- C-11.46)** These operations will be easier if you know the size of each subtree.
- C-11.47)** Is it possible for an splay tree to also be a red-black tree?
- C-11.48)** Study closer the balance property of an AVL tree and the rebalance operation. Also, make a node high up in tree have its AVL balance depend on the node that just got inserted.
- C-11.49)** Study closer the balance property of an AVL tree and the rebalance operation.
- C-11.50)** Find the right place to “splice” one tree into the other to maintain the (2,4) tree property. Also, it is okay to destroy the old versions of T and U .
- C-11.51)** Find the right place to “splice” one tree into the other to maintain the red-black tree property.
- C-11.52)** You don’t need to use induction here.
- C-11.53)** Think about a way of using the structure of the binary search tree itself to indicate color.
- C-11.54)** Search down for k and cut along this path. Now consider how to “glue” the pieces back together in the right order.
- C-11.55)** Consider the red and black meaning of the three possible balance factors in an AVL tree.
- C-11.56)** Since you know the node x will eventually become the root, maintain a tree of nodes to the left of x and a tree of nodes to the right of x , which will eventually become the two children of x .
- C-11.57)** The analysis in the book works also for half-splay trees, with minor modifications.
- C-11.58)** If you are having trouble with this problem, you may wish to gain some intuition about splay trees by “playing” with an interactive splay tree program on the Internet.
- C-11.59)** Force such a replacement to take place and then attempt to use an existing position instance to the item that served as the replacement.
- C-11.60)** Make sure that each item remains at its original node instance.

Projects

P-11.61) We've provided the implementations; you need to develop the experiment.

P-11.62) In this case, you will need a skip list implementation to test.

P-11.63) Remember that a single node of the tree might store multiple (key,value) pairs.

P-11.64) The order is implicit in the tree, so adding these methods should not be hard.

P-11.65) Think carefully about how to uniquely represent the position of a single (key,value) pair.

P-11.66) Use a recursive method to do the conversion.

P-11.67) The most significant challenge is how to handle the insertion of duplicate, given that the original tree search will stop when it finds the existing key.

P-11.68) Review the cases for zig-zag, zig-zig, and zig. Make sure you do splaying right before doing anything else.

P-11.69) First figure out a way that works assuming that all keys in existing mergeable heaps are distinct, and then work out how this is not strictly necessary.

Chapter 12

Sorting and Selection

Hints

Reinforcement

- R-12.1)** Argue in more detail about why the merge-sort tree has height $O(\log n)$.
- R-12.2)** Recall the definition of a recursion trace from Chapter 4.
- R-12.3)** Consider “padding” out the input with infinities to make n a power of 2. How does this affect the running time?
- R-12.4)** Consider an input with duplicates.
- R-12.5)** Consider an input with duplicates.
- R-12.6)** You need a different way to handle the equal case in the merge procedure.
- R-12.7)** Consider using something like the merge for merge-sort.
- R-12.8)** Derive a recurrence equation for this algorithm assuming n is a power of 2. Does it look familiar? It should.
- R-12.9)** You want each choice of pivot to form a very bad split.
- R-12.10)** Recall what is the best possible split we can get for a given pivot and then derive a recurrence equation assuming we get this kind of a split. This equation should look familiar.
- R-12.11)** To gain intuition, work out the first few splits on the sequence $(1, 1, 1, 1, 1, 1, 1, 1, 2)$.
- R-12.12)** Clearly the flaw must involve a case where a pass of the outermost loop completes with the value of left precisely equal to right.
- R-12.13)** Develop a test case in which left equals right immediately prior to the evaluation of line 14.
- R-12.14)** $1/n^2$ is the same as $1/2^{2\log n}$.
- R-12.15)** What is the maximum number of external nodes that a binary tree of height n can have?

- R-12.16)** Recall that to sort n elements with a comparison-based algorithm requires $\Omega(n \log n)$ time.
- R-12.17)** No. Why not?
- R-12.18)** Work out some examples with triples first. Then move on to d -tuples.
- R-12.19)** The two running times are not the same.
- R-12.20)** There are only two possible key values.
- R-12.21)** Try to mimic the partition method used in the in-place quick-sort algorithm.
- R-12.22)** Check out the discussion comparing the various sorting algorithms.
- R-12.22)** Check out the discussion comparing the various sorting algorithms.
- R-12.23)** Consider the complexity of comparisons versus using elements as indices into an array.
- R-12.24)** Think of the worst possible way to choose pivots in this algorithm.

Creativity

- C-12.25)** How do you know S and T have the same elements in them?
- C-12.26)** Sort first.
- C-12.27)** Can you adapt the merge algorithm of Code Fragment 12.3 to directly manipulate nodes of the list.
- C-12.28)** Merge-sort is a particularly good choice for a linked list.
- C-12.29)** A queue of queues can be very helpful.
- C-12.30)** It would be easier if the last element in the array were still the pivot...
- C-12.31)** For the overall worst case, recall the worst case for choosing the last element as the pivot.
- C-12.32)** You need to use an induction hypothesis that $T(n) \leq cn \log n$, for some constant c .
- C-12.33)** Carefully consider how to maintain the stated invariant when classifying each additional element.
- C-12.34)** Sort the votes, and then determine who received the maximum number of votes.
- C-12.35)** Think of a data structure that can be used for sorting in a way that only stores k elements when there are only k keys.
- C-12.36)** Shoot for an $O(n)$ expected running time.

- C-12.37)** Develop a meaningful way to break ties during comparisons .
- C-12.38)** Sort A and B first.
- C-12.39)** Think of alternate ways of viewing the elements.
- C-12.40)** Find a way of sorting them as a group that keeps each sequence contiguous in the final listing.
- C-12.41)** Sort first.
- C-12.42)** Try to modify the merge-sort algorithm to solve this problem.
- C-12.43)** Try to modify the insertion-sort algorithm to solve this problem.
- C-12.44)** 1
- C-12.45)** Consider the graph of the equation $m = a + b$ for a fixed value of m .
- C-12.46)** Perform a selection first on some appropriate order statistics.
- C-12.47)** Try to design an efficient divide-and-conquer algorithm.
- C-12.48)** You will need two-passes through the data at each level of recursion.
- C-12.49)** Use in-place quick-sort as a starting point.
- C-12.50)** Think about what would be the perfect pivot in an algorithm like quick-sort.
- C-12.51)** Use linear-time selection in an appropriate way.
- C-12.52)** Think of an alien version of quick-sort.
- C-12.53)** You could dynamically define a new key type with its own definition for comparisons.
- C-12.54)** For (a), revisit the definition of the randomized quick-sort algorithm. For (b), argue why the probability that $C_{i,j}(x) = 1$ is at most $1/2^j$ and why the dependence between $C_{i,j}(x)$'s only helps. For (c), review the book's discussion of geometric sums. For (d), just plug in the equation for μ and do the math. For (e), argue about all n elements from the bound on a single one.
- C-12.55)** The recurrence equation denotes two recursive calls, but one is smaller than the other.

Projects

- P-12.56)** Think about how to define subproblems concisely and store them on the stack. You then can use a while loop to process problems from and to this stack. Also, please see the chapter discussion about in-place quick-sort for more hints.
- P-12.57)** Implement the version that is not in-place first.

P-12.58) An almost sorted sequence could be one with at most a linear number of inversions.

P-12.59) An almost sorted sequence could be one with at most a linear number of inversions.

P-12.60) Be sure to perform enough tests so that your results are trustworthy.

P-12.61) Use good testing inputs to verify that your method is stable. Also, be sure to copy the elements of the list in and out of the bucket array.

P-12.62) One good animation style uses vertical lines various lengths to represent the different elements.

Chapter 13

Text Processing

Hints

Reinforcement

- R-13.1)** The empty string is one of them.
- R-13.2)** Recall the definitions of prefix and suffix.
- R-13.3)** Mimic the style of the text-matching figures in the book.
- R-13.4)** Mimic the style of the text-matching figures in the book.
- R-13.5)** Mimic the style of the text-matching figures in the book.
- R-13.6)** Use the algorithm presented in the book.
- R-13.7)** Use the version of the algorithm presented in the book.
- R-13.8)** Draw the entire table for the dynamic programming algorithm.
- R-13.9)** All answers are encoded in the table.
- R-13.10)** Simulate a running of the algorithm presented in the book.
- R-13.11)** Don't forget to include the space character.
- R-13.12)** Mimic the drawing style used in the book.
- R-13.13)** Mimic the drawing style used in the book.
- R-13.14)** Mimic the drawing style used in the book.

Creativity

- C-13.15)** Make the text and the pattern very periodic.
- C-13.16)** Use symmetry to redesign the search from right to left, yet still returning the index at which the pattern *starts*.
- C-13.17)** Use symmetry to redesign the search from right to left, including the definition of the “last” map.
- C-13.18)** Use symmetry to redesign the search from right to left, including the definition of the failure function.
- C-13.19)** After finding a complete match, make sure to skip ahead past the end of that match before continuing.

- C-13.20)** After finding a complete match, make sure to skip ahead past the end of that match before continuing.
- C-13.21)** After finding a complete match, make sure to skip ahead past the end of that match before continuing.
- C-13.22)** The justification is similar to the argument that the number of iterations in `find_kmp` is $O(n)$.
- C-13.23)** Consider modifying the KMP matching algorithm.
- C-13.24)** Convert this problem to a noncircular pattern-matching problem.
- C-13.25)** The failure function can now take advantage of the fact that it knows what does match in the mismatched location.
- C-13.26)** You need to incorporate a failure function with the Boyer-Moore heuristics.
- C-13.27)** Keep around extra information in the table for the dynamic programming algorithm.
- C-13.28)** Anatjari should use a greedy algorithm.
- C-13.29)** First give as many quarters as possible.
- C-13.30)** Don't use normal denominations like you would find in a country on earth.
- C-13.31)** We can use a greedy algorithm.
- C-13.32)** There is a surprising similarity between this problem and the matrix chain-product problem.
- C-13.33)** Consider using a prefix trie.
- C-13.34)** Start by building a suffix trie.
- C-13.35)** Review the LCS algorithm.
- C-13.36)** Use a greedy algorithm.
- C-13.37)** Review the LCS algorithm.
- C-13.38)** Use dynamic programming.
- C-13.39)** Consider using a greedy algorithm.
- C-13.40)** Use brute force, first to enumerate all pairs (a, b) such that a is in A and b is in B .
- C-13.41)** Use dynamic programming.
- C-13.42)** Build a prefix tree for X and a suffix tree for Y ...
- C-13.43)** Start by locating the leaf that corresponds to the end of the string.
- C-13.44)** Start by locating the leaf that corresponds to the end of the string.
- C-13.45)** Recall how you identify the branches of the suffix trie that can be compressed.

Projects

- P-13.46)** Stick to the smaller strings, since LCS is a quadratic algorithm.
- P-13.47)** The edit distance algorithm is a dynamic program based on the LCS problem.
- P-13.48)** You can find large documents on the Internet.
- P-13.49)** You can find large documents on the Internet.
- P-13.50)** You can find large documents on the Internet.
- P-13.51)** Try using inputs that are likely to cause both best-case and worst-case running times for various algorithms.
- P-13.52)** You can rely on our implementation of trees and priority queues.
- P-13.53)** Create some way of visualizing your standard trie so that you can verify that it is being constructed correctly.
- P-13.54)** Create some way of visualizing your compressed trie so that you can verify that it is being constructed correctly.
- P-13.55)** Create some way of visualizing your prefix trie so that you can verify that it is being constructed correctly.
- P-13.56)** Use an inverted file data structure.
- P-13.57)** Use an inverted file data structure and store page ranks.

Hints

Reinforcement

- R-14.1)** Use the drawing style of the book for the figure.
- R-14.2)** Recall that an n -vertex component of a simple undirected graph can have no more than $n(n-1)/2$ edges.
- R-14.3)** Fix a canonical ordering of the nodes.
- R-14.4)** Fix a canonical ordering of the nodes.
- R-14.5)** When drawing the Euler tour notice that every in has an out.
- R-14.6)** Recall the properties of inserting and removing elements in a doubly linked list.
- R-14.7)** Don't forget to update the edge list E .
- R-14.8)** Don't forget to update the edge list E .
- R-14.9)** How can the edges method be implemented?
- R-14.10)** How can the edges method be implemented?
- R-14.11)** The point of this exercise is to explore the trade-offs of the various representations. If you are having trouble, please review these trade-offs.
- R-14.12)** Review the pseudo-code while observing the different running times needed to implement each step when the graph is represented with an adjacency matrix.
- R-14.13)** Use a pencil and paper.
- R-14.14)** Work out an example with a complete graph of six vertices.
- R-14.15)** Work out an example with a complete graph of six vertices.
- R-14.16)** Follow the algorithm descriptions and drawing styles given in the book.
- R-14.17)** Mimic the drawing style used in the book.
- R-14.18)** If you wish, you may run our implementation to determine the order.

- R-14.19)** Are there any pairs of vertices that are not related?
- R-14.20)** Design a counting scheme that charges each level in T with the transitive closure edges that go from that level to lower levels.
- R-14.21)** Use the transitive closure algorithm given in the book.
- R-14.22)** Model the courses and their prerequisites as a directed graph.
- R-14.23)** Using the drawing style given in the book.
- R-14.24)** The essential property of course is to observe the edge directions. So work out the pseudo-code with this in mind.
- R-14.25)** Use the drawing style given in the book, and show the order in which the edges are added to the MST.
- R-14.26)** Use the drawing style given in the book, and show the order in which the edges are added to the MST.
- R-14.27)** Use an MST algorithm.
- R-14.28)** The black lines represent unexplored edges. What about the others?
- R-14.29)** Edges in the DFS tree are represented by solid lines. What about the others?
- R-14.30)** The black lines represent unexplored edges. What about the others?
- R-14.31)** Solid black lines are edges in the original input graph. What about the others?
- R-14.32)** Traversed edges are shown with dashed blue lines. What about the others?
- R-14.33)** Solid black edges have not been relaxed yet. What about the others?
- R-14.34)** Edges being considered in the current iteration for the MST are shown in thick blue lines. What about the others?
- R-14.35)** Edges being considered in the current iteration for the MST are shown in thick blue lines. What about the others?
- R-14.36)** Note how much time is spent in each step of George's algorithm.

Creativity

- C-14.37)** Make sure to remove any incident edges.
- C-14.38)** Make sure to remove the edge from the adjacency map for each incident vertex.
- C-14.39)** Use a data structure described in this book.
- C-14.40)** Suppose that such a nontree edge is a cross edge, and argue based upon the order the DFS visits the end vertices of this edge.

- C-14.41)** The DFS method must be changed so that the repetition stops when discovering v .
- C-14.42)** There has to be a good traversal that does this.
- C-14.43)** Maintain a set of all vertices upon which a recursive call to DFS is currently active.
- C-14.44)** Modify the `dfs_complete` function in order to tag each vertex with a component number when it is first discovered.
- C-14.45)** The solution involves a traversal discussed in this chapter.
- C-14.46)** Number the vertices 0 to $n - 1$.
- C-14.47)** Suppose there is a forward edge or back edge and show why it would not be a nontree edge.
- C-14.48)** Suppose there is such an edge and show why it would not be a nontree edge.
- C-14.49)** Justify this by induction on the length of a shortest path from the start vertex.
- C-14.50)** Take each part in turn and use induction or contradiction as justification techniques.
- C-14.51)** Starting with the source vertex in the queue, repeatedly remove the vertex from the front of the queue and insert any of its unvisited neighbors to the back of the queue.
- C-14.52)** Do a BFS search in G .
- C-14.53)** Start out greedy and patch in the places this approach misses.
- C-14.54)** Perform “baby” searches from each station.
- C-14.55)** Try to compute the *diameter* of the tree T by a pruning procedure, starting at the leaves (external nodes).
- C-14.56)** Perform a traversal of T to compute distances to leaves.
- C-14.57)** What is the in-degree of the vertex labeled i in a compact graph?
- C-14.58)** Change the function to be optimized, but keep it indexed in terms of the a fixed listing of vertices.
- C-14.59)** Think about a shortest path with negative edges.
- C-14.60)** Be greedy.
- C-14.61)** Give G lots of edges and make every edge count for lots updates in the heap.
- C-14.62)** Examine closely the justification for why Dijkstra’s algorithm works correctly and note the place where it breaks if negative-weight edges.
- C-14.63)** The greedy algorithm presented in this problem is not guaranteed to find the shortest path. Why not?

- C-14.64)** Maintain an auxiliary set of vertices that have already been discovered.
- C-14.65)** Think about the important fact about minimum spanning trees.
- C-14.66)** How does the number of connected components change from one iteration of Barůvka's algorithm to the next?
- C-14.67)** Use the tree-based union/find data structure from Chapter 10 along with one of the MST algorithms given in Chapter 12 and a good sorting algorithm.
- C-14.68)** This is not a shortest-path problem.
- C-14.69)** Use a greedy algorithm.
- C-14.70)** Model this problem as an optimal path problem that goes between two vertices in a directed graph without cycles.
- C-14.71)** Define an augmented graph from the input graph and perform the appropriate computation on the augmented graph.
- C-14.72)** If $M^2(i, j) = 1$, then there is a path of length ≤ 2 (a path traversing at most 2 edges) from vertex i to vertex j in the graph G . Alternatively, if $M^2(i, j) = 0$, then there is no such path. Now generalize from here...
- C-14.73)** The running time is more than linear.

Projects

- P-14.74)** Test the basic structure by having a method that visualizes the adjacency matrix.
- P-14.75)** Test the basic structure by having a method that visualizes the edge list.
- P-14.76)** Test the basic structure by having a method that visualizes the adjacency list.
- P-14.77)** Implement the static undirected version first, then add the extra methods.
- P-14.78)** Have some way of printing out each phase of the algorithm for debugging purposes.
- P-14.79)** Use uniform and Gaussian distributions for the edge weights.
- P-14.80)** You might use the `cs1graphics` Python module to implement the visualization.
- P-14.81)** Do a Breadth-First Search from each node in the network.

Chapter 15

Memory Management and B-Trees

Hints

Reinforcement

- R-15.1)** Perform an Internet search to determine a good estimate on the number of atoms on earth.
- R-15.2)** Start with the description provided in the book.
- R-15.3)** Revisit the definition of an (a, b) tree.
- R-15.4)** The definition of an order- d deals with the minimum and maximum number of children an internal node can have. Please see the book for details.
- R-15.5)** Draw the memory cache and manually process the requests using a pencil with a good eraser.
- R-15.6)** Draw the memory cache and manually process the requests using a pencil with a good eraser.
- R-15.7)** Draw the memory cache and manually process the requests using a pencil with a good eraser.
- R-15.8)** Use a pencil with a good eraser.

Creativity

- C-15.9)** Review the external-memory sorting algorithm.
- C-15.10)** Keep the top one or two blocks of the stack in main memory.
- C-15.11)** Keep queue runs in blocks.
- C-15.12)** Consider an alternate linked list implementation that uses “fat” nodes.
- C-15.13)** Note that each valid node v and its children in a $(2,4)$ tree correspond to a red-black subtree of height 2. In a $(4,8)$ tree, you will need bigger subtrees.
- C-15.14)** Consider the extreme cases.

- C-15.15)** Try to block order- B sized sub “trees” in the skip list.
- C-15.16)** Start from sequence solution for the union-find problem.
- C-15.17)** A single scan suffices.
- C-15.18)** Each request can “see into the future” to see when is the next time existing blocks will be accessed next.
- C-15.19)** In an initial scan, keep track of the best candidate majority value, x , and a counter that keeps track of the number of times you have seen a copy of x versus some other integer.
- C-15.20)** Consider what happens to a page that is accessed a lot and then never accessed again.
- C-15.21)** The answer just uses some simple logarithm identities.

Projects

- P-15.22)** Make sure to use typical memory sizes and do a long simulation.
- P-15.23)** Let a and b be definable parameters or constants. And let insertion be the first update method you program.
- P-15.24)** Start with insertion as the first update operation you code up, and use a simple uniform distribution of keys to perform the experiments.