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

Instructor's Solutions Manual

WILEY

Chapter 5 Array-Based Sequences

Hints and Solutions

Reinforcement

R-5.1) Hint Although the speed may differ, the asymptotics should be similar.

R-5.2) Hint Keep track of the maximum data size thus far.

R-5.2) Solution

```
import sys                                  # provides getsizeof function
data = [ ]
for k in range(n):
  oldsize = sys.getsizeof(data)
  data.append(None)                         # increase length by one
  if sys.getsizeof(data) != oldsize:
    print("Capacity reached at", len(data) - 1)
  oldsize = sys.getsizeof(data)
```

R-5.3) Hint You might want to make the list relatively large before you begin to remove entries.

R-5.4) Hint Note that an index such as -3 is equivalent to the traditional index $n-3$ for a list of length n.

R-5.4) Solution

```
def __getitem__(self, k):
  if k < 0:
    k += self._n                            # adjustment for negative index
  if not 0 <= k < self._n:
    raise IndexError('invalid index')
  return self._A[k]                         # retrieve from array
```

R-5.5) Hint 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-5.5) Solution Let us assume that one cyber-dollar is enough to pay for the execution of each push operation in S, excluding the time spent for growing the array. Now, however, growing the array from size k to size $2k$ requires $3k$ cyber-dollars. Once again, we will need to account for this cost with our "bank account" in the elements of the last half of the array list. To grow from 2^i to 2^{i+1}, we need $3*2^i$ cyber dollars. Thus, from the second half of the array list—the last 2^{i-1} elements we need to have 6 cyber-dollars apiece stored away. So, overall, we need to charge 7 cyber-dollars for each push operation: 6 for future growth and 1 for insertion.

R-5.6) Hint You are going to need two (non-nested) loops.

R-5.6) Solution

```
def insert(self, k, value):
  if self._n == self._capacity:                  # not enough room
    B = self._make_array(2*self._capacity)       # new (bigger) array
    for j in range(k):
      B[j] = self._A[j]
    B[k] = value
    for j in range(k, self._capacity):
      B[j+1] = self._A[j]
    self._capacity = 2 * self._capacity
    self._A = B
  else:
    for j in range(self._n, k, -1):              # shift rightmost first
      self._A[j] = self._A[j-1]
    self._A[k] = value                           # store newest element
  self._n += 1
```

R-5.7) Hint You don't need to sort A.

R-5.7) Solution

```
def missing(A):
  found = [False] * len(A)
  for val in A:
    if found[val]:
      return val            # this is a duplicate
    else:
      found[val] = True
```

R-5.8) Hint Do the observed results match what you expected?

R-5.9) Hint The alphabets for most alphabet-based languages are included in the Unicode character encoding standard.

R-5.9) Solution The encoder and decoder lists and the various uses of value 26 would have be changed to the new alphabet and its size. In addition, all the places that use the literal A to refer to the first letter would now have to be changed to the first letter in the new alphabet. In this case, it would be better to define a final static int FIRSTLETTER, set it to the first letter, and use it instead of A. This assumes the messages are still in upper-case, of course.

R-5.10) Hint Review the list comprehension syntax from Section 1.9.2.

R-5.10) Solution

```
def __init__(self, shift):
  self._forward = ''.join(chr((k + shift)%26 + ord('A')) for k in range(26))
  self._backward = ''.join(chr((k − shift)%26 + ord('A')) for k in range(26))
```

R-5.11) Hint You should use nested loops.

R-5.12) Hint Consider how to build a list of subtotals, one for each nested list.

R-5.12) Solution `sum(sum(subset) for subset in data)`

Creativity

C-5.13) Hint Use list comprehension to make an initial list of a specific size.

C-5.14) Hint Consider randomly shuffling the deck one card at a time.

C-5.14) Solution For each i from 0 to $n-1$, swap the element at index i with a randomly chosen element from the list (possibly itself).

C-5.15) Hint Consider how much cyber "money" is saved up from one expansion to the next.

C-5.16) Hint The existing _resize method can be used to shrink the array.

C-5.17) Hint You can predict precisely when a resize occurs during this process, and take the sum of those costs.

C-5.18) Hint Apply the amortized analysis accounting technique using a monetary accounting scheme with extra funds for both insertions and removals.

C-5.19) Hint Consider, after any resize takes place, how many subsequent operations are needed to force another resize.

C-5.20) Hint Try to oscillate between growing and shrinking.

C-5.20) Solution This problem is different from the previous, because the threshold of N/2 is unfortunately aligned with the doubling rule used for an increase. Assume that an underlying array has capacity k for large k. When we push a $(k+1)$st element, the array will be doubled to capacity

$2k$, at a cost of $\Omega(k)$. If we then pop two elements, the $k-1$ remaining elements is less than half of the array's capacity, so a resize to shrink the array will be performed with cost of $\Omega(k)$. By repeatedly pushing two elements and popping two elements, there will be a linear amount of work for every two operations.

C-5.21) Hint See Code Fragment 5.4 for an example use of Python's time module.

C-5.22) Hint You might design an experiment similar to that in Code Fragment 5.4.

C-5.23) Hint See Code Fragment 5.4 for an example use of Python's time module.

C-5.24) Hint Use the time module together with loops such as those given on page 205.

C-5.25) Hint You must be very careful if modifying a list at the same time that you loop through its elements!

C-5.25) Solution

```
def remove_all(data, value):
  # first we compact all desired elements toward the front
  keep = 0
  for walk in range(len(data)):
    if data[walk] != value:
      data[keep] = data[walk]
      keep += 1
  # next we clear the rest of the list (in reverse).
  # This step can be coded more efficiently with Python syntax
  # data[keep: ] = [ ]
  while len(data) > keep:
    data.pop( )
```

C-5.26) Hint It might help to sort B.

C-5.26) Solution Sort the array B then scan it from front to end looking for the repeated entries. Each pair of repeated integers will be consecutive in the sorted listing.

C-5.27) Hint You might wish to use an auxiliary array of size at most $4n$.

C-5.27) Solution Use a Boolean array of size at most $4n$, where index i is true if and only if i is in the sequence. Initially, all cells are false, then process the sequence setting cell i to true for each integer i in L. At the end, scan the array for a false cell. The corresponding index is not in the list. Note that such a cell must exist, since there are at least $2n$ k-bit positive integers. It takes $O(n)$ time to compute this value.

C-5.28) Hint Argue why you have to look at all the integers in L.

C-5.28) Solution Any algorithm that misses even one integer in L and reports that some integer i is not in L is overlooking the possibility that the missed value in L is i.

C-5.29) Hint 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) Hint You should be able to achieve $O(n)$ time.

C-5.31) Hint Recall that binary_sum from Section 4.4.2 computes the sum of numbers in a one-dimensional list.

Projects

P-5.32) Hint The entries $A[i][j][k]$ and $B[i][j][k]$ are the ones that need to be added.

P-5.33) Hint 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) Hint You will probably need separate encrypt and decrypt mappings for the upper- and lower-case characters.

P-5.35) Hint The original CaesarCipher implementation was already effectively a substitution cipher, with a specifically chosen encoder pattern.

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

P-5.37) Hint 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.