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 10 Maps, Hash Tables, and Skip Lists

Hints and Solutions

Reinforcement

R-10.1) Hint This is more challenging because __delitem__ does not return the deleted value.

R-10.1) Solution

```
def pop(self, k, d=None):
  try:
   value = self[k]
   del self[k]
   return value
  except KeyError:
    if d is not None:
      return d
    else:
      raise KeyError('key not found')
```

R-10.2) Hint You must rely on the iter method to access the contents of the map.

R-10.2) Solution

```
def items(self):
  for k in iter(self):
    yield (k, self[k])
```

This implementation would only run in $O(n^2)$ time if using the UnsortedTableMap because of the average linear time for each call to **self**[k].

R-10.3) Hint You should not directly call __iter__, but you can mimic its style.

R-10.3) Solution

```
def items(self):
  for item in self._table:
    yield (item._key, item._value)
```

R-10.4) Hint The first insertion is $O(1)$, the second is $O(2)$, ...

R-10.5) Hint Review the API for PositionalList from Section 7.4.

R-10.6) Hint 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) Hint Use of Python's `__hash__` method is discussed on page 415.

R-10.8) Hint There is a lot of symmetry and repetition in this string, so avoid a hash code that would not deal with this.

R-10.8) Solution Either polynomial hash codes or cyclic shift hash codes would be good.

R-10.9) Hint Try to mimic the figure in the book.

R-10.9) Solution

0	1	2	3	4	5	6	7	8	9	10
13	94 39				44 88 11			12 23	16 5	20

R-10.10) Hint Try to mimic the figure in the book.

R-10.10) Solution

0	1	2	3	4	5	6	7	8	9	10
13	94	39	16	5	44	88	11	12	23	20

R-10.11) Hint The failure occurs because no empty slot is found. For the drawing, try to mimic the figure in the book.

R-10.11) Solution

0	1	2	3	4	5	6	7	8	9	10
13	94	39	11		44	88	16	12	23	30

The probe sequence when inserting value 5 fails to ever find the sole remaining empty slot (at index 4).

R-10.12) Hint Try to mimic the figure in the book.

R-10.12) Solution

0	1	2	3	4	5	6	7	8	9	10
13	94	23	88	39	44	11	5	12	16	20

R-10.13) Hint Think of the worst-case time for inserting every entry in the same cell in the hash table.

R-10.13) Solution The worst-case time is $O(n^2)$. The best case is $O(n)$.

R-10.14) Hint Mimic the way the figure is drawn.

R-10.14) Solution

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		12	18	41	30	36	25		54			38	10			28		

R-10.15) Hint Allow the user to express the maximum load factor as an optional parameter to the constructor.

R-10.16) Hint It is okay to insert a new entry on "top" of the deactivated entry object.

R-10.17) Hint You will need to keep track of the number of probes in order to apply quadratic probing.

R-10.18) Hint Think of where the entry with minimum key is stored.

R-10.19) Hint The crucial methods are `__getitem__` and `__setitem__`.

R-10.20) Hint 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) Hint What happens in the case where the middle table value equals `k`?

R-10.21) Solution Yes, this variant always produces the same result as the original. To see this note that if the target key is not contained in the collection, the recursion proceeds in identical fashion. The only issue is when reaching a case where the target key equals the current "middle" key in the table range. This means a match was found and in the original version, the index of that match is immediately returned. In the new version, the recursion continues to the lefthand side, within index range [`low`, `mid`−1]. Since map keys are distinct, the target value will not be found in that range, and it must be that everything in that range is strictly smaller. By convention, that recursion will be an unsuccessful search and will return the index one greater than the upper extend of the range. Therefore, it returns index `mid`, which is the index at which the exact match was found.

R-10.22) Hint Assume that a skip list is used to implement the sorted map.

R-10.22) Solution For this problem, we assume we are using a skip list as the implementation for the sorted map. With a skip list, the expected running time for maintaining a maxima set if we insert n pairs such that each pair has lower price and performance than one before it is $O(n \log n)$, since each pair is inserted and does not remove any others. Thus, the i^{th} insertion runs in $O(\log i)$ time, and the total expected running time is $\sum_{i=1}^{n} \log i$, which is $O(n \log n)$.

If, on the other hand, each pair had a lower price and higher performance than the one before it, then the overall running time is $O(n)$, because each new pair causes the removal of the pair that came before it. Therefore the size of the map is constant, and thus all its operations run in worst-case $O(1)$ time.

(If a SortedTableMap were used, with lower price/performance pairs, each pair is inserted at the beginning of the array list, and thus the insertion time dominates the search time, for overall $O(n^2)$ time. With lower price/higher performance pairs, the SortedTableMap would use $O(n)$ time overall.)

R-10.23) Hint Mimic the style of the figures in the book.

R-10.24) Hint You must link out the removed entry's tower from all the lists it belongs to.

R-10.25) Hint You will need to rely on the iter behavior to find an arbitrary element of the set.

R-10.25) Solution

```
def pop(self):
  if len(self) == 0:
    raise KeyError('set is empty')
  else:
    element = next(iter(self))      # get first element from iterator
    self.discard(element)
    return element
```

R-10.26) Hint For each element of the smaller set, check to see if it is contained in the larger set.

R-10.27) Hint Something from this chapter should be helpful!

R-10.27) Solution We should use a sorted multiset, as we want to maintain multiple entries with the same birthday and we want to be able to do neighboring queries.

Creativity

C-10.28) Hint The existing __setitem__ implementation can serve as a reasonable model for this new method.

C-10.28) Solution

```
def setdefault(self, k, v):
  for item in self._table:
    if k == item._key:
      return item._value
  # did not find match for key
  self._table.append(self._Item(k,v))
  return v
```

C-10.29) Hint You might provide an implementation directly within the ProbeHashMap, but you'll need to borrow some techniques from the HashMapBase class.

C-10.30) Hint You might provide an implementation directly within the ChainHashMap, but you'll need to borrow some techniques from the HashMapBase class.

C-10.31) Hint 1

C-10.32) Hint Prepare a concise table of your experimental results.

C-10.33) Hint Fortunately, a Python list can be heterogeneous!

C-10.34) Hint Oops. We meant to say to reimplement the HashMapBase class. Feel free to subclass the nested item class.

C-10.35) Hint 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-10.35) Solution When we remove an entry from a hash table that uses linear probing without using deleted element markers, we must ensure that any cluster of consecutively filled cells remains intact, unless no entries in that cluster after a newly introduced gap have a hash value that falls before such a gap. Assume that we wish to delete an entry stored at index j and that the cluster it belongs to goes until an empty cell at index k. For ease of exposition only, we assume $k > j$, thus the cluster does not wrap around the end of the array. To fill the hole in index j, we wish to find the next entry of the cluster that has a hash value $h \leq j$. If no such entry exists, we may leave a hole at j. Otherwise, assume index i holds the first such entry. We move that entry to j and then we repeat the process to fill the hole that results at index i, finding the next subsequent entry of the cluster with hash value $h \leq i$. Note that the entire process takes time linear in the length of the original cluster of filled cells.

C-10.36) Hint 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) Hint Perhaps you might define a nonpublic method that generates probe indices.

C-10.38) Hint Consider the way find_range was implemented for SortedTableMap.

C-10.38) Solution Simply do a binary search to find an element equal to k. Then step back through the array until you reach the first element equal to k. Finally, step forward through the area reporting items until you reach the first key that is not equal to k. This takes $O(\log n)$ time for the search and then at most s time to search back to the beginning of the run of k's and s time return all of the elements with key k. Therefore, we have a solution running in at most $O(s + \log n)$ time.

C-10.39) Hint Try out some examples.

C-10.39) Solution The original version of Code Fragment 10.8 does not guarantee that it finds the leftmost occurrence when duplicates exist, because it stops searching as soon as it finds a match. For example, if the entire table was filled with duplicates, it will report the middle index.

In contrast, the version in Exercise R-10.21 does guarantee that it identifies the leftmost of all duplicates, as it always searches in the subrange to the left of one match, to potentially find another match.

C-10.40) Hint Do a "double" binary search.

C-10.40) Solution In order to the find the k^{th} smallest key in the union of the keys from S and T, we can do a "double" binary search. In other words, we will begin by examining the $k/2^{th}$ element in the array list S. Next, we will find the largest element in T that is less than $S[k/2]$ by binary search. Then, we will add the indices of the elements we were examining in S and T. If the sum equals k, then the max of the two elements is our result. If the sum is greater than k, then we will do a binary search to the right (upwards) in S. If the sum is less than k, then we will do a binary search to the left (downwards) in S. This is followed once again by searching in T for largest element less than the current element in S, etc. This method does a binary search on S which requires $O(\log n)$ "probes." However, for each probe of the search, it does a binary search on T which takes $O(\log n)$ time. Thus, the entire method takes $O(\log^2 n)$ time.

C-10.41) Hint Dovetail two binary searches.

C-10.42) Hint Think first about how you can determine the number of 1's in any row in $O(\log n)$ time.

C-10.42) Solution To count the number of 1's in A, we can do a binary search on each row of A to determine the position of the last 1 in that row. Then we can simply sum up these values to obtain the total number of 1's in A. This takes $O(\log n)$ time to find the last 1 in each row. Done for each of the n rows, then this takes $O(n \log n)$ time.

C-10.43) Hint Think about first sorting the pairs by cost.

C-10.43) Solution Sort the pairs by cost. Then scan this list looking at the performance values. Remove any that have performance values worse than the (unremoved) pair that came before.

C-10.44) Hint 1

C-10.45) Hint 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) Hint Make sure to start with a new empty set (or make a copy of one of the two initial sets).

C-10.48) Hint Think of how you could transform D into L.

C-10.49) Hint Maintain a secondary PositionalList instance that represents the FIFO order

Projects

P-10.50) Hint In a Unix/Linux system, a good place to start is /usr/dict.

P-10.51) Hint It is okay to generate these phone numbers more-or-less at random.

P-10.52) Hint You might consider embedding a next pointer within each item composite.

P-10.53) Hint Sentinels can be used in place of the theoretical $-\infty$ and $+\infty$.

P-10.54) Hint Try to make your screen images mimic the skip list figures in the book.

P-10.55) Hint For each word t that results from a minor change to s, you can test if t is in W in $O(1)$ time.