Introduction to Algorithms Day 28
Massachusetts Institute of Technology 6.046J/18.410J
Singapore-MIT Alliance SMA5503
Professors Erik Demaine, Lee Wee Sun, and Charles E. Leiserson Handout 27

Problem Set 7 Solutions

MIT students: This problem set is due in lecture on *Day 26*.

Reading: Chapters 17

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation instructor and time, the date, and the names of any students with whom you collaborated.

MIT students: Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 7-1. Do exercise 17.1-1 on page 409 of CLRS.

Solution:

No, a sequence of MULTIPUSH operations could make the amortized bound $O(k)$.

Exercise 7-2. Do exercise 17.3-4 on page 416 of CLRS.

Solution:

The total cost of executing n stack operations, assuming the stack begins with s_0 objects and finishes with s_n objects is bounded by $2n + s_0 - s_n$.

Exercise 7-3. Do exercise 17.3-7 on page 416 of CLRS.

Solution:

You use an unsorted array, so insert takes $O(1)$ worst-case time. For DELETE-LARGER-HALF, you use the linear-time median algorithm to find the median, then you use PARTITION to partition the array around the median, then you delete the larger side of the partition in $O(1)$ time.

For the amortized analysis, insert each item with 2 tokens on it. When you perform a DELETE-LARGER-HALF operation, each item in the list pays 1 token for the operation. When you delete the larger half, the tokens on these items are redistributed on the remaining items. If each item on the list starts with 2 tokens, they each have one after the median finding, and then each item in the deleted half gives its token to one of the remaining items. Thus, there are always two tokens per item and we get constant amortized time.

Exercise 7-4. Do exercise 17.4-1 on page 424 of CLRS.

Solution:

To keep insertion time reasonable. Insertion into a dynamic open-address hash table can be made to run in $O(1)$ time by expanding when $\alpha \geq .8$ and contracting when $\alpha \leq .2$.

Problem 7-1. Reducing the space in the van Emde Boas structure

In this problem, we will use hashing to modify the van Emde Boas data structure presented in lecture in order to reduce its space usage.

Recall the problem statement: In the ***fixed-universe successor*** problem, a data structure must maintain a dynamic subset S of the universe $U = \{0, \ldots, u-1\}$. The data structure must support the operations of inserting elements into S, deleting elements from S, finding the successor (next element in S) from any element in U, and finding the predecessor (previous element in S) from any element in U.

Recall the outline of the van Emde Boas data structure: The universe $U = \{0, \ldots, u-1\}$ is represented by a ***widget*** of size u. Each widget W of size $|W|$ stores an array *sub*$[W]$ of $\sqrt{|W|}$ ***recursive subwidgets*** *sub*$[W][0]$, *sub*$[W][1], \ldots,$ *sub*$[W][\sqrt{|W|}-1]$ each of size $\sqrt{|W|}$. In addition, each widget W stores a ***summary widget*** *summary*$[W]$ of size $\sqrt{|W|}$, representing which subwidgets are nonempty. Each widget W also stores its minimum element *min*$[W]$ separately from all the subwidgets. Finally, each widget W maintains the value *max*$[W]$ of its maximum element.

For reference, the van Emde Boas algorithms for insertion and finding successors in $O(\lg\lg u)$ time are given as follows. For any widget W, and for any x in the universe of possible elements in W, define $\mathit{high}(x)$ and $\mathit{low}(x)$ to be nonnegative integers so that $x = \mathit{high}(x)\sqrt{|W|} + \mathit{low}(x)$. Thus, $\mathit{high}(x)$ and $\mathit{low}(x)$ are both less than $\sqrt{|W|}$, and represent the high-order and low-order halves of the bits in the binary representation of x.

```
vEB-INSERT$(x, W)$
1  if $x < \mathit{min}[W]$
2     then exchange $x \leftrightarrow \mathit{min}[W]$
3  if subwidget $\mathit{sub}[W][\mathit{high}(x)]$ is nonempty, that is, $\mathit{min}[\mathit{sub}[W][\mathit{high}(x)]] \neq$ NIL
4     then vEB-INSERT$(\mathit{low}(x), \mathit{sub}[w][\mathit{high}(x)])$
5     else $\mathit{min}[\mathit{sub}[W][\mathit{high}(x)]] \leftarrow \mathit{low}(x)$
6          vEB-INSERT$(\mathit{high}(x), \mathit{summary}[W])$
7  if $x > \mathit{max}[W]$
8     then $\mathit{max}[W] \leftarrow x$
```

```
vEB-SUCCESSOR$(x, W)$
1  if $x < \mathit{min}[W]$
2     then return $\mathit{min}[W]$
3  if $\mathit{low}(x) < \mathit{max}[\mathit{sub}[W][\mathit{high}(x)]]$
4     then $j \leftarrow$ vEB-SUCCESSOR$(\mathit{low}(x), \mathit{sub}[W][\mathit{high}(x)])$
5          return $\mathit{high}(x)\sqrt{|W|} + j$
6     else $i \leftarrow$ vEB-SUCCESSOR$(\mathit{high}(x), \mathit{summary}[W])$
7          return $i\sqrt{|W|} + \mathit{min}[\mathit{sub}[W][i]]$
```

(a) Argue that the van Emde Boas data structure uses $\Theta(u)$ space. (*Hint:* Derive a recurrence for the space $S(u)$ occupied by a widget of size u.)

Solution:

The space $S(u)$ occupied by the data structure is given by the recurrence

$$S(u) = (1 + \sqrt{u})S(\sqrt{u}) + O(\sqrt{u}) ,$$

because in each widget there are $\sqrt{u}$ recursive subwidgets, 1 recursive summary widget, and an array of size $O(\sqrt{u})$.

First we prove that $S(u) \leq c_1 u - c_2$ by the substitution method. Assume by induction that $S(k) \leq c_1 k - c_2$ for all $k < u$. Then

$$\begin{aligned}
S(u) &\leq (1 + \sqrt{u})(c_1\sqrt{u} - c_2) + O(\sqrt{u}) \\
&= c_1\sqrt{u} + c_1 u - c_2 - c_2\sqrt{u} + O(\sqrt{u}) \\
&= c_1 u - \Big((c_2 - c_1 - O(1))\sqrt{u} + c_2\Big) \\
&\leq c_1 u ,
\end{aligned}$$

provided that c_2 is chosen large enough. The constant c_1 must be chosen large enough to satisfy the base case.

Second we prove that $S(u) \geq cu$ by the substitution method. Assume by induction that $S(k) \geq ck$ for all $k < u$. Then

$$\begin{aligned} S(u) &\geq (1+\sqrt{u})c\sqrt{u} + O(\sqrt{u}) \\ &= c\sqrt{u} + cu + O(\sqrt{u}) \\ &\geq cu \,. \end{aligned}$$

The constant c must be chosen small enough to satisfy the base case.

Consider the following modifications to the van Emde Boas data structure.

1. Empty widgets are represented by the value NIL instead of being explicitly represented by a recursive construction.
2. The structure *sub*$[W]$ containing the subwidgets

   $$\textit{sub}[W][0], \textit{sub}[W][1], \ldots, \textit{sub}[W][\sqrt{|W|} - 1]$$

   is stored as a dynamic hash table (as in Section 17.4 of CLRS) instead of an array. The key of a subwidget *sub*$[W][i]$ is i, so we can quickly find the ith subwidget *sub*$[W][i]$ by a single search in the hash table *sub*$[W]$.
3. As a consequence of the first two modifications, the hash table *sub*$[W]$ only stores the *nonempty* subwidgets. The NIL values of the empty subwidgets are not even stored in the hash table. Thus, the space occupied by the hash table *sub*$[W]$ is proportional to the number of nonempty subwidgets of W.

Whenever we insert an element into an empty (NIL) widget, we ***create*** a widget using the following procedure, which runs in $O(1)$ time:

```
CREATE-WIDGET($x$)          ▷ Returns a new widget containing just the element $x$.
1  allocate a widget structure $W$
2  $min[W] \leftarrow x$
3  $max[W] \leftarrow x$
4  $summary[W] \leftarrow$ NIL
5  $sub[W] \leftarrow$ a new empty dynamic hash table
6  return $W$
```

In the next two problem parts, you will develop the insertion and successor operations for this modified van Emde Boas structure. It suffices to simply describe the necessary changes from the vEB-INSERT and vEB-SUCCESSOR operations detailed above. In any case, you should give special attention to the interaction with the hash table *sub*$[W]$.

(b) Give an efficient algorithm for inserting an element into the modified van Emde Boas structure, using CREATE-WIDGET as a subroutine.

Solution:

The algorithm is similar to vEB-INSERT. One main change is that the two cases are distinguished based on testing whether a particular key is stored in the hash table *sub*$[W]$. A second main change is that when the key is not in the hash table, a new widget is created using CREATE-WIDGET. We summarize with the pseudocode:

```
MODIFIED-INSERT($x, W$)
 1  if $W$ = NIL
 2     then $W \leftarrow$ CREATE-WIDGET($x$)
 3     else  if $x < min[W]$
 4              then exchange $x \leftrightarrow min[W]$
 5           if the hash table $sub[W]$ has an entry for key $high(x)$
 6              then MODIFIED-INSERT($low(x), sub[W][high(x)]$)
 7              else  $W' \leftarrow$ CREATE-WIDGET($x$)
 8                    insert into hash table $sub[W]$ the subwidget $W'$ with key $high(x)$
                                            ▷ Sets $sub[W][high(x)] \leftarrow W'$
 9                    MODIFIED-INSERT($high(x), summary[W]$)
10           if $x > max[W]$
11              then $max[W] \leftarrow x$
```

(c) Give an efficient algorithm for finding the successor of an element in the modified van Emde Boas structure.

Solution:

The algorithm is identical to vEB-SUCCESSOR, except that references to *sub*$[W][i]$ translate into searches in the hash table *sub*$[W]$ for key i.

(d) Using known results, argue that the running time of your modified insertion and successor algorithms run in $O(\lg\lg u)$ expected time, under the assumption of simple uniform hashing.

Solution:

Each recursive call used to perform $O(1)$ instructions, and now additionally performs $O(1)$ additional hash-table operations. Thus, under the assumption of simple uniform hashing, the total cost goes up by an expected constant factor from the normal van Emde Boas structure.

(e) Prove that the space occupied by the modified data structure is $O(n)$. You may ignore the possibility of deletions, and assume that only insertions and successor operations are performed.

Solution:

Each widget by itself (ignoring its subwidgets and summary widgets) takes $O(1)$ space. We store a widget only if its *min* field is occupied by an element. The hash table increases the space by a constant factor (amortizing over the constant cost of each subwidgets). Thus the space is $O(n)$.

Problem 7-2. The cost of restructuring red-black trees

There are four basic operations on red-black trees that perform ***structural modifications***: node insertions, node deletions, rotations, and color modifications. We have seen that RB-INSERT and RB-DELETE use only $O(1)$ rotations, node insertions, and node deletions to maintain the red-black properties, but they may make many more color modifications.

(a) Describe a legal red-black tree with n nodes such that calling RB-INSERT to add the $(n+1)$st node causes $\Omega(\lg n)$ color modifications. Then describe a legal red-black tree with n nodes for which calling RB-DELETE on a particular node causes $\Omega(\lg n)$ color modifications.

Solution:

For RB-INSERT, consider a complete red-black tree with an even number of levels in which nodes at odd levels are black and nodes at even levels are red. When a node is inserted as a child of one of the leaves, then $\Omega(\lg n)$ color changes will be needed to fix the colors of nodes on the path from the inserted node to the root. For RB-DELETE, consider a complete red-black tree in which all nodes are black. If a leaf is deleted, then the ”double blackness” will be pushed all the way up to the root, with a color change at each level (case 2 of RB-DELETE-FIXUP), for a total of $\Omega(\lg n)$ color changes.

Although the worst-case number of color modifications per operation can be logarithmic, we shall prove that any sequence of m RB-INSERT and RB-DELETE operations on an initially empty red-black tree causes $O(m)$ structural modifications in the worst case.

(b) Some of the cases handled by the main loop of the code of both RB-INSERT-FIXUP and RB-DELETE-FIXUP are ***terminating***: once encountered, they cause the loop to terminate after a constant number of additional operations. For each of the cases of RB-INSERT-FIXUP and RB-DELETE-FIXUP, specify which are terminating and which are not. (hint: Look at Figures 13.5, 13.6, and 13.7).

Solution:

All cases except for case 1 of RB-INSERT-FIXUP and case 2 of RB-DELETE-FIXUP are terminating.

We shall first analyze the structural modifications when only insertions are performed. Let T be a red-black tree, and define $\Phi(T)$ to be the number of red nodes in T. Assume that 1 unit of potential can pay for the structural modifications performed by any of the three cases of RB-INSERT-FIXUP.

(c) Let T' be the result of applying Case 1 of RB-INSERT-FIXUP to T. Argue that $\Phi(T') = \Phi(T) - 1$.

Solution:

Case 1 of RB-INSERT-FIXUP reduces the number of red nodes by one, a fact that can be seen in Figure 13.4 in CLRS. Hence, $\Phi(T') = \Phi(T) - 1$.

(d) Node insertion into a red-black tree using RB-INSERT can be broken down into three parts. List the structural modifications and potential changes resulting from lines 1-16 of RB-INSERT, from nonterminating cases of RB-INSERT-FIXUP, and from terminating cases of RB-INSERT-FIXUP.

Solution:

Lines 1-16 of RB-INSERT cause one node insertion and a unit increase in potential. The nonterminating case of RB-INSERT-FIXUP (Case 1) makes three color changes and decreases the potential by one. The terminating cases of RB-INSERT-FIXUP (Cases 2 and 3) cause one rotation each and do not affect the potential.

(e) Using part (d), argue that the amortized number of structural modifications performed by any call of RB-INSERT is $O(1)$.

Solution:

The number of structural modifications and amount of potential change resulting from lines 1-16 of RB-INSERT and from the terminating cases of RB-INSERT-FIXUP are constant, so the amortized cost of these parts are constant. The nonterminating case of RB-INSERT-FIXUP may repeat up to $O(\lg n)$ times, but its amortized cost is 0, since by our assumption the unit decrease in the potential pays for the structural modifications needed. Therefore, the worst-case amortized cost of RB-INSERT is constant.

We now wish to prove that there are $O(m)$ structural modifications when there are both insertions and deletions. Let us define, for each node x,

$$w(x) = \begin{cases} 0 & \text{if } x \text{ is red ,} \\ 1 & \text{if } x \text{ is black and has no red children ,} \\ 0 & \text{if } x \text{ is black and has one red child ,} \\ 2 & \text{if } x \text{ is black and has two red children .} \end{cases}$$

Now we redefine the potential of a red-black tree T as

$$\Phi(T) = \sum_{x \in T} w(x) \ ,$$

and let T' be the tree that results from applying any nonterminating case of RB-INSERT-FIXUP or RB-DELETE-FIXUP to T.

(f) Show that $\Phi(T') \leq \Phi(T) - 1$ for all nonterminating cases of RB-INSERT-FIXUP. Argue that the amortized number of structural modifications performed by any call of RB-INSERT-FIXUP is $O(1)$.

Solution:

From Figure 13.4, we see that Case 1 of RB-INSERT-FIXUP makes the following changes to the tree:

- Changes a black node with two red children to a red node (node C), resulting in a potential change of -2.
- Changes a red node to a black node with one red child (node A in the top diagram; node B in the bottom diagram), resulting in no potential change.
- Changes a red node to a black node with no red children (node D), resulting in a potential change of 1.

The total change in potential is -1, which pays for the structural modifications performed, and thus the amortized cost of Case 1 (nonterminating case) is 0. Because the terminating cases of RB-INSERT-FIXUP cause constant structural changes and constant change in potential, since $w(v)$ is based solely on node color and the number of color changes caused by terminating cases is constant. The amortized cost of the terminating cases is at most constant. Hence, the overall amortized cost of RB-INSERT is constant.

(g) Show that $\Phi(T') \leq \Phi(T) - 1$ for all nonterminating cases of RB-DELETE-FIXUP. Argue that the amortized number of structural modifications performed by any call of RB-DELETE-FIXUP is $O(1)$.

Solution:

Figure 13.7 shows that Case 2 of RB-DELETE-FIXUP makes the following changes to the tree:

- Changes a black node with no red children to a red node (node D), resulting in a potential change of -1.
- If B is red, then it loses a black child, with no effect on potential.
- If B is black, then it goes from having no red children to having one red child, resulting in a potential change of -1.

The total change in potential is either -1 or -2, depending on the color of B. In either case, one unit of potential pays for the structural modifications performed, and thus the amortized cost of Case 2 (nonterminating case) is at most 0. Because the terminating cases of RB-DELETE cause constant structural changes and constant change in potential, since $w(v)$ is based solely on node color and the number of color changes caused by terminating cases is constant. The amortized cost of the terminating cases is at most constant. Hence, the overall amortized cost of RB-DELETE-FIXUP is constant.

(h) Complete the proof that in the worst case, any sequence of m RB-INSERT and RB-DELETE operations performs $O(m)$ structural modifications.

Solution:

Since the amortized cost of each operation is bounded above by a constant, the actual number of structural modifications for any sequence of m RB-INSERT and RB-DELETE operations on an initially empty red-black tree cause $O(m)$ structural modifications in the worst case.