Design and Analysis of Algorithms April 25, 2012
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Dana Moshkovitz and Bruce Tidor Handout 19

Problem Set 7 Solutions

This problem set is due **at 9:00pm** on **Wednesday, April 25, 2012**.

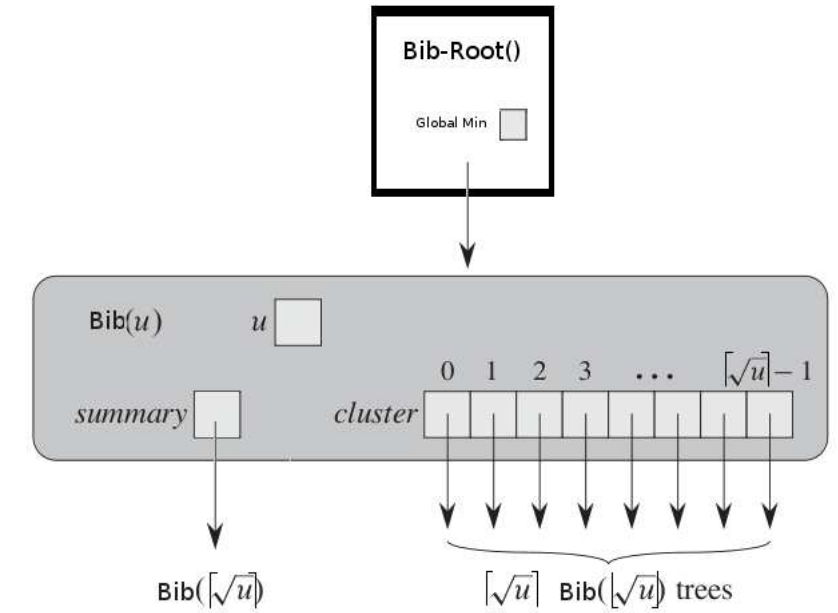
Problem 7-1. Seeing the Forest for the van Emde Boas Trees

After listening to a lecture on van Emde Boas Trees, Ben Bitdiddle decides that he can simplify the implementation by not storing the max and min fields in each of the vEB substructures separately. Instead, he decides to insert them into the tree, while still keeping track of the global min in a separate variable. He dubs this revised version of the data structure the "Bib Tree".

(a) Write pseudo-code implementations of the BIB-INSERT, BIB-DELETE, and BIB-MIN functions for the Bib Tree.

Solution:

Here is a graphic of the structure of a Bib Tree as described:



BIB-MIN is the simplest operation, if we are trying to get the global min: we simply return the value that is already stored. (Here, we use $V.root$ to represent the structure storing metadata, which is not recursive and only exists at the top level.)

BIB-MIN(V) :

1 **return** $V.root.min$

In the code for BIB-INSERT, we define the base case to be when the size of the universe is 2; any small constant would suffice here.

The base case Bib Tree structure (the leaves of the tree) is distinct from that of the upper level ones; rather than having an array of clusters and a summary Bib Tree, it stores only a bit array which indicates the presence (or absence) of the elements governed by it.

```
BIB-INSERT(V, x) :
1  if V.u == 2
2       V.A[x] = 1
3  else
4       BIB-INSERT(V.cluster[high(x)], low(x))
5       BIB-INSERT(V.summary, high(x))
```

We'll also define a helper function called BIB-RECURSIVE-MIN, to get the smallest element not stored in the global min. (This is necessary for BIB-DELETE).

```
BIB-RECURSIVE-MIN(V) :
 1  if V.u == 2
 2       if V.A[0] == 1
 3            return 0
 4       elseif V.A[1] == 1
 5            return 1
 6       else return NIL
 7  else c = BIB-RECURSIVE-MIN(V.summary)
 8       if c == NIL
 9            return NIL
10       else i = BIB-RECURSIVE-MIN(V.cluster[c])
11            return c√u + i
```

There are a two key cases for BIB-DELETE: if we delete the global min, we have to find the next min and delete it from the recursive substructures, as the global min is not stored recursively. Otherwise, it suffices to simply delete the element from the appropriate cluster, and delete the cluster from the summary (which is itself a Bib Tree of size $\sqrt{u}$) if necessary.

BIB-DELETE(V, x) :

```
1  if x == V.root.min
2       newmin = BIB-RECURSIVE-MIN(V)
3       BIB-DELETE(V, newmin)
4       V.root.min = newmin
5  elseif V.u == 2
6  V.A[x] = 0
7  else
8  BIB-DELETE(V.cluster[high(x)], low(x))
9  if BIB-RECURSIVE-MIN(V.cluster[high(x)]) == NIL // Is the cluster now empty?
10      BIB-DELETE(V.summary, high(x))
```

It is possible to avoid the extra BIB-RECURSIVE-MIN call in line 7 of BIB-DELETE above by defining an additional variable to count the number of items stored in each Bib Tree structure.

(b) Write the recurrences for the run-times of the operations in the revised data structure and solve the recurrences.

Solution:

BIB-MIN is not recursive, and so just takes $O(1)$ time.

BIB-INSERT has 2 recursive calls to BIB-INSERT, so has recurrence $T(u) = 2T(\sqrt{u}) + O(1)$. We can solve this recurrence by substitution: Define $m = \lg u$, so that $u = 2^m$ and $\sqrt{u} = 2^{m/2}$, and $S(m) = T(2^m) = T(u)$.

Then $S(m) = 2S(m/2) + O(1)$, so $S(m) = \Theta(m)$ by the Master method and $T(u) = \Theta(\lg u)$.

BIB-RECURSIVE-MIN likewise has recurrence $T(u) = 2T(\sqrt{u}) + O(1)$ and therefore also takes time $\Theta(\lg u)$.

Finally, as written above BIB-DELETE has a worst-case running time of $T(u) = 2T(\sqrt{u}) + T_{\text{BIB-RECURSIVE-MIN}} = 2T(\sqrt{u}) + \Theta(\lg u)$.

Using substitution, as above, we get $S(m) = 2S(m/2) + \Theta(m)$, which solves to $\Theta(m \lg m) = \Theta(\lg u \lg \lg u)$ by the Master method.

(c) Ben Bitdiddle decides to use the Bib-Tree data structure in his implementation of Prim's algorithm (§23.2). You can assume that the input graphs of interest all have integer weights in the range from 1 to n and that edge weights are allowed to repeat.

Show how Ben would make use of his Bib-Tree and provide an analysis of time complexity for this implementation of Prim.

Solution:

We cannot use the Bib Tree directly for Prim's algorithm, because the Bib Tree as described in the previous parts does not have the capability of storing multiple elements with the same key. In order to make it possible to use the Bib Tree for Prim's algorithm, we make the following modifications:

1. At the leaves of the Bib Tree (the base case, where the universe size is $u = 2$), make $V.A$ into an array of pointers rather than a bit array as it was previously. Each pointer is NIL if no vertex has that key, and is a pointer to the head of a linked list containing all vertices with that key otherwise.
2. In the base case of BIB-INSERT, if we are inserting a value v with key k: rather than setting $V.A[k]$ to be 1, we instead add v to the end of the linked list at $V.A[k]$, or create a new linked list with one element if $V.A[k]$ was previously NIL. Additionally, when we insert (k, v), we store a pointer to its position in that linked list with the vertex v, which will become useful later for the modified BIB-DELETE.
3. When BIB-DELETE(k, v) is called, at the base case, we can delete v from the linked list at k by following the pointer stored with v upon its insertion.

We implement EXTRACT-MIN by getting a vertex whose current key is the global min; then we must call BIB-DELETE on that key (with the vertex) to remove the vertex from the linked list, and potentially replace the global min with the next smallest element.

In the worst case, this takes as much time as BIB-DELETE, which takes $\Theta(\lg n \lg\lg n)$ time by the previous part of the pset.

INSERT(k, v) is simply a call to the modified BIB-INSERT procedure.

DECREASE-KEY(k, v) involves a call to the modified BIB-DELETE followed by one to BIB-INSERT, which takes a total of $\Theta(\lg n \lg\lg n)$ time.

Prim's algorithm takes $|V|T_{\text{EXTRACT-MIN}} + |E|T_{\text{DECREASE-KEY}}$, so with our modified Bib Trees this takes $\Theta((E + V) \lg n \lg\lg n)$ time.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2012