CS5800: Algorithms — Virgil Pavlu

Homework 8

Name:
Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. (50 points)

Implement a hash for text. Given a string as input, construct a hash with words as keys, and word counts as values. Your implementation should include:

- a hash function that has good properties for text
- storage and collision management using linked lists
- operations: insert(key,value), delete(key), increase(key), find(key), list-all-keys

Output the list of words together with their counts on an output file. For this problem,you cannot use built-in-language datastuctures that can index by strings (like hashtables). Use a language that easily implements linked lists, like C/C++.
You can test your code on "Alice in Wonderland" by Lewis Carroll, at link.
The test file used by TA will probably be shorter.
Try these three values for $m = MAXHASH : 30, 300, 1000$. For each of these m values, produce a histogram over the lengths of collision lists. You can also calculate variance of these lengths.
If your hash is close to uniform in collisions, you should get variance close to zero, and almost all list-lengths around $\alpha = n/m$.
If your hash has long lists, we want to know how many and how long, for example print the lengths of the longest 10% of the lists.
(Extra Credit) Find a way to record not only word counts, but also the positions in text. For each word, besides the count value, build a linked list with positions in the given text. Output this list together with the count.

Solution:

2. (50 points)

Implement a red-black tree, including binary-search-tree operations *sort, search, min, max, successor, predecessor* and specific red-black procedures *rotation, insert, delete.* The *delete* implementation is **Extra Credit** (but highly recommended).

Your code should take the input array of numbers from a file and build a red-black tree with this input by sequence of "inserts". Then interactively ask the user for an operational command like "insert x" or "sort" or "search x" etc, on each of which your code rearranges the tree and if needed produces an output. After each operation also print out the height of the tree.

You can use any mechanism to implement the tree, for example with pointers and struct objects in C++, or with arrays of indices that represent links between parent and children. You cannot use any tree built-in structure in any language.

Solution:

3. (50 points)

Study the skiplist data structure and operations. They are used for sorting values, but in a datastructure more effficient than lists or arrays, and more guaranteed than binary search trees. Review Slides skiplists.pdf and Visualizer. The demo will be a sequence of operations (asked by

TA) such as for example insert 20, insert 40, insert 10, insert 20, insert 5, insert 80, delete 20, insert 100, insert 20, insert 30, delete 5, insert 50, lookup 80, etc

Solution:

4. (50 points)

Implement binomial heaps as described in class and in the book. You should use links (pointers) to implement the structure as shown in the figure 1. Your implementation should include the operations: Make-heap, Insert, Minimum, Extract-Min, Union, Decrease-Key, Delete

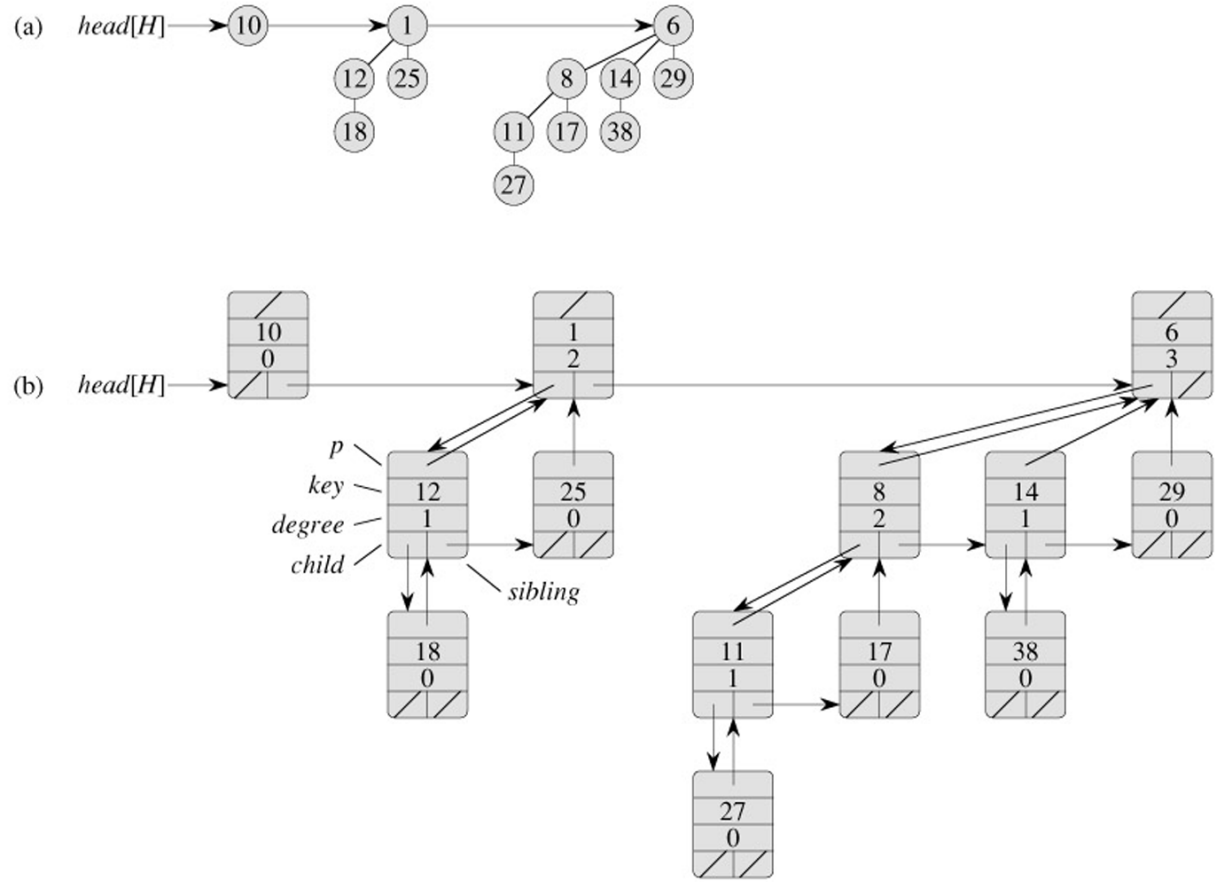

Figure 1: binomial heaps

Make sure to preserve the characteristics of binomial heaps at all times: (1) each component should be a binomial tree with children-keys bigger than the parent-key; (2) the binomial trees should be in order of size from left to right. Test your code several arrays set of random generated integers (keys).

Solution:

5. (25 points)

(Exercise 16.3-3) Consider an ordinary binary min-heap data structure supporting the instructions $Insert$ and $Extract-Min$ that, when there are n items in the heap, implements each operation in

$O(\lg n)$ worst-case time. Give a potential function Φ such that the amortized cost of $Insert$ is $O(\lg n)$ and the amortized cost of $Extract-Min$ is $O(1)$, and show that your potential function yields these amortized time bounds. Note that in the analysis, n is the number of items currently in the heap, and you do not know a bound on the maximum number of items that can ever be stored in the heap.

Solution:

$$\Phi = k * n$$

Min Heap	c_i	$\Phi_i - \Phi_{i-1}$	$\hat{c}_i$		
Insert	$\lg n$	k	$\lg n + k$	$\rightarrow$	$O(\lg n)$
Extract-Min	$\lg n$	$-k$	$\lg n - k$	$\rightarrow$	$O(1)$

When you insert, we increase n by 1, so the resulting delta with be k, and when we extract the min, we decrease n by 1, so the resulting delta is $-k$. Analyzing the amortized cost for insert is simple; because k is a constant and they are being summed together, $\lg n$ is asymptotically larger and dominates. Analyzing the amortized cost for Extract-Min is more difficult because the constant is being subtracted. If we choose a large enough constant though, it will pick up most of the cost and we can view the amortized cost as asymptotically constant.