String length	Binary tree	Hash table	Suffix tree	Compressed tree
8	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0
32	0.1	0.0	0.0	0.0
64	0.3	0.4	0.3	0.0
128	2.4	1.1	0.5	0.0
256	17.1	9.4	3.8	0.2
512	31.6	67.0	6.9	1.3
1,024	1,828.9	96.6	31.5	2.7
2,048	11,441.7	941.7	553.6	39.0
4,096	$>$ 2 days	5,246.7	out of	45.4
8,192		$>$ 2 days	memory	642.0
16,384				1,614.0
32,768				13,657.8
65,536				39,776.9

Figure 3.13: Run times (in seconds) for the SBH simulation using various data structures

Chapter Notes

Optimizing hash table performance is surprisingly complicated for such a conceptually simple data structure. The importance of short runs in open addressing has to more sophisticated schemes than sequential probing for optimal hash table performance. For more details, see Knuth [Knu98].

Our triangle strip optimizing program, *stripe*, is described in [ESV96]. Hashing techniques for plagiarism detection are discussed in [SWA03].

Surveys of algorithmic issues in DNA sequencing by hybridization include [CK94, PL94]. Our work on interactive SBH reported in the war story is reported in [MS95a].

3.10 Exercises

Stacks, Queues, and Lists

3-1. *[3]* A common problem for compilers and text editors is determining whether the parentheses in a string are balanced and properly nested. For example, the string (((())())() contains properly nested pairs of parentheses, which the strings)()(and ()) do not. Give an algorithm that returns true if a string contains properly nested and balanced parentheses, and false if otherwise. For full credit, identify the position of the first offending parenthesis if the string is not properly nested and balanced.

3-2. *[3]* Write a program to reverse the direction of a given singly-linked list. In other words, after the reversal all pointers should now point backwards. Your algorithm should take linear time.

3-3. *[5]* We have seen how dynamic arrays enable arrays to grow while still achieving constant-time amortized performance. This problem concerns extending dynamic arrays to let them both grow and shrink on demand.

(a) Consider an underflow strategy that cuts the array size in half whenever the array falls below half full. Give an example sequence of insertions and deletions where this strategy gives a bad amortized cost.

(b) Then, give a better underflow strategy than that suggested above, one that achieves constant amortized cost per deletion.

Trees and Other Dictionary Structures

3-4. *[3]* Design a dictionary data structure in which search, insertion, and deletion can all be processed in $O(1)$ time in the worst case. You may assume the set elements are integers drawn from a finite set $1, 2, .., n$, and initialization can take $O(n)$ time.

3-5. *[3]* Find the overhead fraction (the ratio of data space over total space) for each of the following binary tree implementations on n nodes:

(a) All nodes store data, two child pointers, and a parent pointer. The data field requires four bytes and each pointer requires four bytes.

(b) Only leaf nodes store data; internal nodes store two child pointers. The data field requires four bytes and each pointer requires two bytes.

3-6. *[5]* Describe how to modify any balanced tree data structure such that search, insert, delete, minimum, and maximum still take $O(\log n)$ time each, but successor and predecessor now take $O(1)$ time each. Which operations have to be modified to support this?

3-7. *[5]* Suppose you have access to a balanced dictionary data structure, which supports each of the operations search, insert, delete, minimum, maximum, successor, and predecessor in $O(\log n)$ time. Explain how to modify the insert and delete operations so they still take $O(\log n)$ but now minimum and maximum take $O(1)$ time. (Hint: think in terms of using the abstract dictionary operations, instead of mucking about with pointers and the like.)

3-8. *[6]* Design a data structure to support the following operations:

- *insert(x,T)* – Insert item x into the set T.
- *delete(k,T)* – Delete the kth smallest element from T.
- *member(x,T)* – Return true iff $x \in T$.

All operations must take $O(\log n)$ time on an n-element set.

3-9. *[8]* A *concatenate* operation takes two sets S_1 and S_2, where every key in S_1 is smaller than any key in S_2, and merges them together. Give an algorithm to concatenate two binary search trees into one binary search tree. The worst-case running time should be $O(h)$, where h is the maximal height of the two trees.

Applications of Tree Structures

3-10. *[5]* In the *bin-packing problem*, we are given n metal objects, each weighing between zero and one kilogram. Our goal is to find the smallest number of bins that will hold the n objects, with each bin holding one kilogram at most.

- The *best-fit heuristic* for bin packing is as follows. Consider the objects in the order in which they are given. For each object, place it into the partially filled bin with the smallest amount of extra room *after* the object is inserted.. If no such bin exists, start a new bin. Design an algorithm that implements the best-fit heuristic (taking as input the n weights $w_1, w_2, ..., w_n$ and outputting the number of bins used) in $O(n \log n)$ time.
- Repeat the above using the *worst-fit heuristic*, where we put the next object in the partially filled bin with the largest amount of extra room *after* the object is inserted.

3-11. *[5]* Suppose that we are given a sequence of n values $x_1, x_2, ..., x_n$ and seek to quickly answer repeated queries of the form: given i and j, find the smallest value in $x_i, \ldots, x_j$.

(a) Design a data structure that uses $O(n^2)$ space and answers queries in $O(1)$ time.

(b) Design a data structure that uses $O(n)$ space and answers queries in $O(\log n)$ time. For partial credit, your data structure can use $O(n \log n)$ space and have $O(\log n)$ query time.

3-12. *[5]* Suppose you are given an input set S of n numbers, and a black box that if given any sequence of real numbers and an integer k instantly and correctly answers whether there is a subset of input sequence whose sum is exactly k. Show how to use the black box $O(n)$ times to find a subset of S that adds up to k.

3-13. *[5]* Let $A[1..n]$ be an array of real numbers. Design an algorithm to perform any sequence of the following operations:

- *Add(i,y)* – Add the value y to the ith number.
- *Partial-sum(i)* – Return the sum of the first i numbers, i.e. $\sum_{j=1}^{i} A[j]$.

There are no insertions or deletions; the only change is to the values of the numbers. Each operation should take $O(\log n)$ steps. You may use one additional array of size n as a work space.

3-14. *[8]* Extend the data structure of the previous problem to support insertions and deletions. Each element now has both a *key* and a *value*. An element is accessed by its key. The addition operation is applied to the values, but the elements are accessed by its key. The *Partial_sum* operation is different.

- *Add(k,y)* – Add the value y to the item with key k.
- *Insert(k,y)* – Insert a new item with key k and value y.
- *Delete(k)* – Delete the item with key k.

- *Partial-sum(k)* – Return the sum of all the elements currently in the set whose key is less than y, i.e. $\sum_{x_j<y} x_i$.

The worst case running time should still be $O(n \log n)$ for any sequence of $O(n)$ operations.

3-15. *[8]* Design a data structure that allows one to search, insert, and delete an integer X in $O(1)$ time (i.e. , constant time, independent of the total number of integers stored). Assume that $1 \leq X \leq n$ and that there are $m+n$ units of space available, where m is the maximum number of integers that can be in the table at any one time. (Hint: use two arrays $A[1..n]$ and $B[1..m]$.) You are not allowed to initialize either A or B, as that would take $O(m)$ or $O(n)$ operations. This means the arrays are full of random garbage to begin with, so you must be very careful.

Implementation Projects

3-16. *[5]* Implement versions of several different dictionary data structures, such as linked lists, binary trees, balanced binary search trees, and hash tables. Conduct experiments to assess the relative performance of these data structures in a simple application that reads a large text file and reports exactly one instance of each word that appears within it. This application can be efficiently implemented by maintaining a dictionary of all distinct words that have appeared thus far in the text and inserting/reporting each word that is not found. Write a brief report with your conclusions.

3-17. *[5]* A Caesar shift (see Section 18.6 (page 641)) is a very simple class of ciphers for secret messages. Unfortunately, they can be broken using statistical properties of English. Develop a program capable of decrypting Caesar shifts of sufficiently long texts.

Interview Problems

3-18. *[3]* What method would you use to look up a word in a dictionary?

3-19. *[3]* Imagine you have a closet full of shirts. What can you do to organize your shirts for easy retrieval?

3-20. *[4]* Write a function to find the middle node of a singly-linked list.

3-21. *[4]* Write a function to compare whether two binary trees are identical. Identical trees have the same key value at each position and the same structure.

3-22. *[4]* Write a program to convert a binary search tree into a linked list.

3-23. *[4]* Implement an algorithm to reverse a linked list. Now do it without recursion.

3-24. *[5]* What is the best data structure for maintaining URLs that have been visited by a Web crawler? Give an algorithm to test whether a given URL has already been visited, optimizing both space and time.

3-25. *[4]* You are given a search string and a magazine. You seek to generate all the characters in search string by cutting them out from the magazine. Give an algorithm to efficiently determine whether the magazine contains all the letters in the search string.

3-26. *[4]* Reverse the words in a sentence—i.e., "My name is Chris" becomes "Chris is name My." Optimize for time and space.

3-27. *[5]* Determine whether a linked list contains a loop as quickly as possible without using any extra storage. Also, identify the location of the loop.

3-28. *[5]* You have an unordered array X of n integers. Find the array M containing n elements where M_i is the product of all integers in X except for X_i. You may not use division. You can use extra memory. (Hint: There are solutions faster than $O(n^2)$.)

3-29. *[6]* Give an algorithm for finding an ordered word pair (e.g., "New York") occurring with the greatest frequency in a given webpage. Which data structures would you use? Optimize both time and space.

Programming Challenges

These programming challenge problems with robot judging are available at *http://www.programming-challenges.com* or *http://online-judge.uva.es*.

3-1. "Jolly Jumpers" – Programming Challenges 110201, UVA Judge 10038.

3-2. "Crypt Kicker" – Programming Challenges 110204, UVA Judge 843.

3-3. "Where's Waldorf?" – Programming Challenges 110302, UVA Judge 10010.

3-4. "Crypt Kicker II" – Programming Challenges 110304, UVA Judge 850.