

Data Structures and Algorithms

Exam

Prof. Dr. M. Böhlen

September 10, 2008

9:00 - 11:00

As auxiliary material you may use 1 A4 sheet with your notes. The exam consists of 4 exercises. The four exercises have equal weight. It is important that you argue for your answers and that you present your solutions in a readable form. Write clearly in terms of math, language, and legibility. The clarity of your explanations impacts your grade. The exam lasts 2 hours. Write your name and ID on each solution sheet. All the best.

1 Radix Tree

Given two bit strings $a = a_0a_1\dots a_p$ and $b = b_0b_1\dots b_q$, where each a_i and each b_j is either 0 or 1, we say that a is lexicographically less than b if either

- there exists an integer j , where $0 \leq j \leq \min(p, q)$, such that $a_i = b_i$ for all $0 \leq i \leq j - 1$ and $a_j < b_j$, or
- $p < q$ and $a_i = b_i$ for all $0 \leq i \leq p$

The above lexicographic ordering of bit strings can be used to build a radix tree that stores a set of bit string keys. For example, the radix tree data structure shown in Figure 1 stores the bit strings 0, 10, 100, 011, and 1011.

When searching for a key we go left at a node at depth i if $a_i = 0$ and right if $a_i = 1$.

Task 1.1: Define and initialize a C data structure for radix trees, such as the one illustrated in Figure 1. The nodes shall not store the actual bit string keys.

Task 1.2: Write a function that prints the set of binary strings represented by a radix tree in sorted order.

Task 1.3: Write a function that inserts a bit string B into a radix tree.

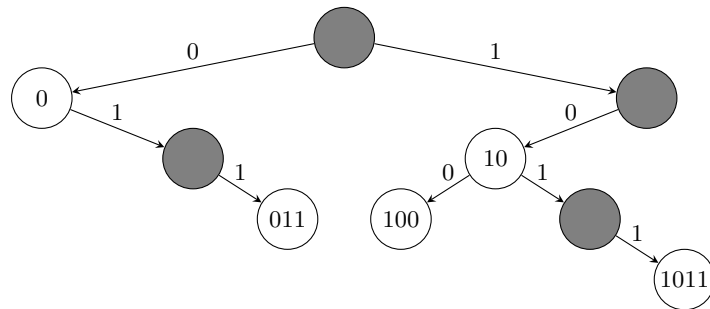


Figure 1: Radix Tree

2 Coin Changing

Assume we have an unlimited number of coins of various denominations. Our goal is to pay out amount A with the smallest number of coins possible.

Depending on the available denominations and amount A we are left with three possibilities:

1. The denomination values are such that a greedy strategy can pay out amount A in an optimal fashion.
2. The denomination values are such that a greedy strategy cannot pay out amount A in an optimal fashion although such an optimal payout does exist.
3. The denomination values are such that no strategy can pay out amount A .

For the tasks below assume an array $d[1..m]$ that keeps track of the available denominations. Without loss of generality assume $d[1] < d[2] < \dots < d[m]$.

Task 2.1: Give an example of a set of denominations and amount A that does not allow to pay out A . Describe the idea of a greedy algorithm. Give an example of a set of denominations and amount A where a greedy algorithm does not yield an optimal solution.

Task 2.2: Give a recursive solution that for all cases determines the minimal number of coins that pays out amount A . Determine the complexity of your solution.

Task 2.3: Give a dynamic programming solution that computes the optimal number of coins needed to pay out amount A . Illustrate your solution with the denominations $d = [1, 3, 4]$ and $A = 9$.

3 Hashing

Assume a hash table with open addressing that is used to index general, possibly empty, ASCII strings. Specifically, the function

$$h(str, i) = \left(\left(\sum_k \text{ASCII}(str[k]) \right) + i^2 \right) \bmod 10$$

is used as hash function ($str[k]$ denotes the k th character of str ; $\text{ASCII}(c)$ is the decimal ASCII code of character c ; $\text{ASCII}('A') = 65$, $\text{ASCII}('L') = 76$, $\text{ASCII}('R') = 82$, etc).

Task 3.1: Show the content of the hash table after the insertion of the strings RED, BAR, BALL, DER, BED (in the given order).

Task 3.2: Discuss possible disadvantages of the proposed hash function.

Task 3.3: Give algorithms to insert elements into and delete elements from the hash table.

4 Recurrences

Consider the following strategies for passing an array of size N as a parameter to a procedure.

1. The array is passed by pointer. The time complexity for passing the array is $\Theta(1)$.
2. The array is passed by copying all elements. The time complexity for passing the array is $\Theta(N)$.
3. The array is passed by copying the subrange that might be accessed by the called procedure. The time complexity for passing the subarray $A[p..q]$ is $\Theta(q - p + 1)$.

Task 4.1: Give a recursive binary search algorithm for finding a number in a sorted array. The array is passed as a parameter to the binary search procedure.

Task 4.2: Give recurrences for the worst case running times of binary search when arrays are passed using each of the three methods above. Solve the recurrences and give good upper bounds on the solutions of the recurrences.

Hint: $\sum_{k=0}^n r^k = \frac{1-r^{n+1}}{1-r}$