HW 11 textbook problems and hints

Problem 1 7.2.3 (6) (Horspool for binary strings)

3. How many character comparisons will be made by Horspool's algorithm in searching for each of the following patterns in the binary text of 1000 zeros?

   a. 00001

   b. 10000

   c. 01010

Author's hint:

3. For each pattern, fill in its shift table and then determine the number of character comparisons (both successful and unsuccessful) on each trial and the total number of trials.

Problem 2 7.2.7 (9) (Boyer-Moore for binary strings)

7. How many character comparisons will the Boyer-Moore algorithm make in searching for each of the following patterns in the binary text of 1000 zeros?

   a. 00001

   b. 10000

   c. 01010

Author's hint:

7. For each pattern, fill in the two shift tables and then determine the number of character comparisons (both successful and unsuccessful) on each trial and the total number of trials.

Problem 3: 7.2.8 (4) (does Boyer-Moore still work with just one table?)

8. a. Would the Boyer-Moore algorithm work correctly with just the bad-symbol table to guide pattern shifts?

   b. Would the Boyer-Moore algorithm work correctly with just the good-suffix table to guide pattern shifts?

Author's hint:

8. Check the description of the Boyer-Moore algorithm.

Problem 4: 7.2.11 (3, 5) (right cyclic shift)

11. You are given two strings S and T, each n characters long. You have to establish whether one of them is a right cyclic shift of the other. For example, PLEA is a right cyclic rotation of LEAP, and vice versa. (Formally, T is a right cyclic shift of S if T can be obtained by concatenating the $(n-i)$-character suffix of S and the i-character prefix of S for some $1 \leq i \leq n$.)

    a. Design a space-efficient algorithm for the task. Indicate the space and time efficiencies of your algorithm.

    b. Design a time-efficient algorithm for the task. Indicate the time and space efficiencies of your algorithm.

Author's hint:

11. a. A brute-force algorithm fits the bill here.

    b. Enhance the input before a search.

Problem 5: 7.3.4 (5) (probability that n keys all hash to the same table location)

4. Find the probability of all n keys being hashed to the same cell of a hash table of size m if the hash function distributes keys evenly among all the cells of the table.

Author's hint:

4. The question is quite similar to computing the probability of having the same result in n throws of a fair die.

Problem 6: 7.4.3 (6)

(minimum order of a B tree that guarantees no more than 3 disk accesses in a tree with 10^8 elements)

3. Find the minimum order of the B-tree that guarantees that the number of disk accesses in searching in a file of 100 million records does not exceed 3. Assume that the root's page is stored in main memory.

Author's hint:

3. Find this value from the inequality in the text that provides the upper-bound of the B-tree's height.

Problems 7-8 : 8.1.10, 8.1.11 (12 each) (Longest path in a dag, Maximum square submatrix)

10. *Longest path in a dag* a. Design an efficient algorithm for finding the length of a longest path in a dag. (This problem is important both as a prototype of many other dynamic programming applications and in its own right because it determines the minimal time needed for completing a project comprising precedence-constrained tasks.)

    ▷ b. Show how to reduce the coin-row problem discussed in this section to the problem of finding a longest path in a dag.

11. ▶ *Maximum square submatrix* Given an $m \times n$ boolean matrix B, find its largest square submatrix whose elements are all zeros. Design a dynamic programming algorithm and indicate its time efficiency. (The algorithm may be useful for, say, finding the largest free square area on a computer screen or for selecting a construction site.)

Author's hint:

10. a. Topologically sort the dag's vertices first.

    b. Create a dag with $n + 1$ vertices: one vertex to start and the others to represent the coins given.

11. Let $F(i, j)$ be the order of the largest all-zero submatrix of a given matrix with its low right corner at (i, j) Set up a recurrence relating $F(i, j)$ to $F(i - 1, j)$, $F(i, j - 1)$, and $F(i - 1, j - 1)$.

Problem 9: 8.1.12 (10) (World Series odds)

10. ▷ *World Series odds* Consider two teams, A and B, playing a series of games until one of the teams wins n games. Assume that the probability of A winning a game is the same for each game and equal to p and the probability of A losing a game is $q = 1 - p$. (Hence, there are no ties.) Let $P(i, j)$ be the probability of A winning the series if A needs i more games to win the series and B needs j more games to win the series.

    a. Set up a recurrence relation for $P(i, j)$ that can be used by a dynamic programming algorithm.

    b. Find the probability of team A winning a seven-game series if the probability of it winning a game is 0.4.

    c. Write a pseudocode of the dynamic programming algorithm for solving this problem and determine its time and space efficiencies.

Author's hint:

10. a. In the situation where teams A and B need i and j games, respectively, to win the series, consider the result of team A winning the game and the result of team A losing the game.

    b. Set up a table with five rows $(0 \le i \le 4)$ and five columns $(0 \le j \le 4)$ and fill it by using the recurrence derived in part (a).

    c. A pseudocode should be guided by the recurrence set up in part (a). The efficiency answers follow immediately from the table's size and the time spent on computing each of its entries.

Instructor note:

In a 7-game series (such as the real American baseball World Series), the first team to win 4 games wins the series. 7 is the maximum number of games that can be played before one of the teams must win four games. But if one team wins 4 games sooner, the series ends immediately.