MA/CSSE 473 HW 14 textbook problems and hints

Problem #1 (5) 9.4.4 (maximal Huffman codeword length)

4. What is the maximal length of a codeword possible in a Huffman encoding of an alphabet of n characters?

Once you have figured out the answer, describe a set of probabilities (or frequencies) that make that maximum happen.

Author's hint:

4. The maximal length of a codeword relates to the height of Huffman's coding tree in an obvious fashion. Try to find a set of n specific frequencies for an alphabet of size n for which the tree has the shape yielding the longest codeword possible.

Problem #2 (10) 9.4.10 (card guessing)

10. *Card guessing* Design a strategy that minimizes the expected number of questions asked in the following game [Gar94], #52. You have a deck of cards that consists of one ace of spades, two deuces of spades, three threes, and on up to nine nines, making 45 cards in all. Someone draws a card from the shuffled deck, which you have to identify by asking questions answerable with yes or no.

Author's hint:

10. A similar example was discussed at the end of Section 9.4. Construct Huffman's tree and then come up with specific questions that would yield that tree. (You are allowed to ask questions such as: Is this card the ace, or a seven, or an eight?)

Problem #3 (10) Kruskal proof

Problem on Kruskal's algorithm:

The questions

(a) How do we know that v was already part of some connected component of G'?

Does the addition of e to C satisfy the hypothesis of the lemma? For each statement below, explain why it is true.

(b) G' is a subgraph of some MST for G:

(c) C is a connected component of G':

(d) e connects a vertex in C to an vertex in G – C:

(e) e satisfies the minimum-weight condition of the lemma:

The algorithm:

- To find a MST for a connected undirectedG:
  - Start with a graph G' containing all of the n vertices of G and no edges.
  - for i = 1 to n – 1:
    - Among all of G's edges that can be added without creating a cycle, add one (call it e) that has minimal weight.

The property we are trying to prove: Before every loop execution, G' is a subgraph of some MST of G.

Proof is (of course) by induction on i.
BASE CASE: When I = 1, G' consists of only vertices of G. Since all vertices must be part of any MST for G, G is a subgraph of every MST.

INDUCTION STEP. Assume that G' is a subgraph of an MST for G. Choose e according to the above algorithm. Show that G' $\cup${e} is a sungraph of an MST of G.

The Lemma we want to use: Let G be a weighted connected graph with a MST T; let G′ be any subgraph of T, and let C be any connected component of G′. If we add to C an edge $e=(v,w)$ that has minimum-weight among all of the edges that have one vertex in C and the other vertex not in C, then G has an MST that contains the union of G′ and e.

In order to be able to use the lemma, we have to pick a connected component of C, and show that it satisfies the conditions of the lemma. We let v be one of the vertices of e, and let C be the connected component of G' that contains v.

Within this context, answer the 5 questions above.

Problem #4 (20) 8.3.11bc [8.3.10bc] matrix chain mltiplication

11. ***Matrix chain multiplication*** Consider the problem of minimizing the total number of multiplications made in computing the product of n matrices

$$A_1 \cdot A_2 \cdot \ldots \cdot A_n$$

whose dimensions are $d_0 \times d_1$, $d_1 \times d_2$, . . . , $d_{n-1} \times d_n$, respectively. Assume that all intermediate products of two matrices are computed by the brute-force (definition-based) algorithm.

a. Give an example of three matrices for which the number of multiplications in $(A_1 \cdot A_2) \cdot A_3$ and $A_1 \cdot (A_2 \cdot A_3)$ differ at least by a factor of 1000.

b. How many different ways are there to compute the product of n matrices?

c. Design a dynamic programming algorithm for finding an optimal order of multiplying n matrices.

Author's hint:

b. You can get the answer by following the approach used for counting binary trees.

c. The recurrence relation for the optimal number of multiplications in computing $A_i \cdot \ldots \cdot A_j$ is very similar to the recurrence relation for the optimal number of comparisons in searching in a binary tree composed of keys $a_i, \ldots, a_j$.

Problem #5 (5) 9.2.8 (efficiency of find in union-by-size)

8. Prove that the time efficiency of $find(x)$ is in $O(\log n)$ for the union-by-size version of quick union.

Author's hint:

8. The argument is very similar to the one made in the section for the union-by-size version of quick find.

Problem #6 (5) 11.1.1 (lower bound for alternating disk algorithm)

1. Prove that any algorithm solving the alternating-disk puzzle (Problem 11 in Exercises 3.1) must make at least $n(n+1)/2$ moves to solve it. Is this lower bound tight?

Author's hint:

1. Is it possible to solve the puzzle by making fewer moves than the brute-force algorithm? Why?

Problem #7 (5) 11.1.4 (fake coin minimum number of guesses)

4. Consider the problem of identifying a lighter fake coin among n identical-looking coins with the help of a balance scale. Can we use the same information-theoretic argument as the one in the text for the number of questions in the guessing game to conclude that any algorithm for identifying the fake will need at least $\lceil \log_2 n \rceil$ weighings in the worst case?

Author's hint:

4. Reviewing Section 5.5, where the fake-coin problem was introduced, should help in answering the question.

Problem #8 (12) 11.1.10 (matrix multiplication and squaring) (6, 6)

10. a. Can we use this section's formulas that indicate the complexity equivalence of multiplication and squaring of integers to show the complexity equivalence of multiplication and squaring of square matrices?

    b. Show that multiplication of two matrices of order n can be reduced to squaring a matrix of order $2n$.

Author's hint:

10. a. Check whether the formulas hold for two arbitrary square matrices.

    b. Use a formula similar to the one showing that multiplication of arbitrary square matrices can be reduced to multiplication of symmetric matrices.

Problem #9 (9) 11.2.10ab [11.2.8ab] (advanced fake-coin problem) (4, 5)

8. *Advanced fake-coin problem* There are $n \geq 3$ coins identical in appearance; either all are genuine or exactly one of them is fake. It is unknown whether the fake coin is lighter or heavier than the genuine one. You have

a balance scale with which you can compare any two sets of coins. That is, by tipping to the left, to the right, or staying even, the balance scale will tell whether the sets weigh the same or which of the sets is heavier than the other, but not by how much. The problem is to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier then the genuine ones.

a. Prove that any algorithm for this problem must make at least $\lceil \log_3(2n+1) \rceil$ weighings in the worst case.

b. Draw a decision tree for an algorithm that solves the problem for $n = 3$ coins in two weighings.

Author's hint

8. a. How many outcomes does this problem have?

b. Draw a ternary decision tree that solves the problem.

:

Problem #10 (5) 11.3.1 (Chess decidable?) Explain your answer.

1. A game of chess can be posed as the following decision problem: given a legal positioning of chess pieces and information about which side is to move, determine whether that side can win. Is this decision problem decidable?

.

Author's hint:

1. Check the definition of a decidable decision problem.

Problem #11 (8) 11.3.2 (tractable?) Explain your answer.

2. A certain problem can be solved by an algorithm whose running time is in $O(n^{\log_2 n})$. Which of the following assertions is true?

a. The problem is tractable.

b. The problem is intractable.

c. None of the above.

Author's hint:

2. First, determine whether $n^{\log_2 n}$ is a polynomial function. Then read carefully the definitions of tractable and intractable problems.