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 (6) 10.2.1 (sources and sinks with negative weights)

1. Since maximum-flow algorithms require processing edges in both directions, it is convenient to modify the adjacency matrix representation of a network as follows: If there is a directed edge from vertex i to vertex j of capacity u_{ij}, then the element in the ith row and the jth column is set to u_{ij}, while the element in the jth row and the ith column is set to $-u_{ij}$; if there is no edge between vertices i and j, both these elements are set to zero. Outline a simple algorithm for identifying a source and a sink in a network presented by such a matrix and indicate its time efficiency.

Author's hint

:

1. What properties of the adjacency matrix elements stem from the source and sink definitions, respectively?

Problem #5 (6) 10.2.3a (Maximum flow solution unique?) Explain your answers.

3. a. Does the maximum-flow problem always have a unique solution? Would your answer be different for networks with different capacities on all their edges?

.

Author's hint:

3. Of course, the value (capacity) of an optimal flow (cut) is the same for any optimal solution. The question is whether distinct flows (cuts) can yield the same optimal value.

Problem #6 (5) 10.2.4a (Maximum flow single source and sink)

4. a. Explain how the maximum-flow problem for a network with several sources and sinks can be transformed to the same problem for a network with a single source and a single sink.

Author's hint:

4. a. Add extra vertices and edges to the network given.

Problem #7 (8) 10.4.1 (stable marriage example)

1. Consider an instance of the stable marriage problem given by the ranking matrix

	A	B	C
α	1,3	2,2	3,1
β	3,1	1,3	2,2
γ	2,2	3,1	1,3

For each of its marriage matchings, indicate whether it is stable or not. For the unstable matchings, specify a blocking pair. For the stable matchings, indicate whether they are man-optimal, woman-optimal, or neither. (Assume that the Greek and Roman letters denote the men and women, respectively.)

Author's hint:

1. A marriage matching is obtained by selecting three matrix cells, one cell from each row and column. To determine the stability of a given marriage matching, check each of the remaining matrix cells for containing a blocking pair.

Problem #8 (4) 10.4.2 (stable marriage check algorithm)

2. Design a simple algorithm for checking whether a given marriage matching is stable and determine its time efficiency class.

Author's hint:

2. It suffices to consider each member of one sex (say, the men) as a potential member of a blocking pair.

Problem #9 (6) 10.4.5 (Huffman TF)

5. Determine the time-efficiency class of the stable-marriage algorithm

   (a) in the worst case.

   (b) in the best case.

Author's hint:

5. The time efficiency is clearly defined by the number of proposals made. You may but are not required to provide the exact number of proposals in the worst and best cases, respectively; an appropriate Θ class will suffice.

Problem #10 (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 $d_0 \times d_1$, $d_1 \times d_2$, $\ldots$, $d_{n-1} \times d_n$, 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 #11 (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.