MA/CSSE 473 – Design and Analysis of Algorithms

Homework 14 (49 points total) Solution

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

Problems to write up and turn in:

1. (5) 9.4.4 (maximal Huffman codeword length) Once you have figured out the answer, describe a set of probabilities (or frequencies) that make the maximum happen.

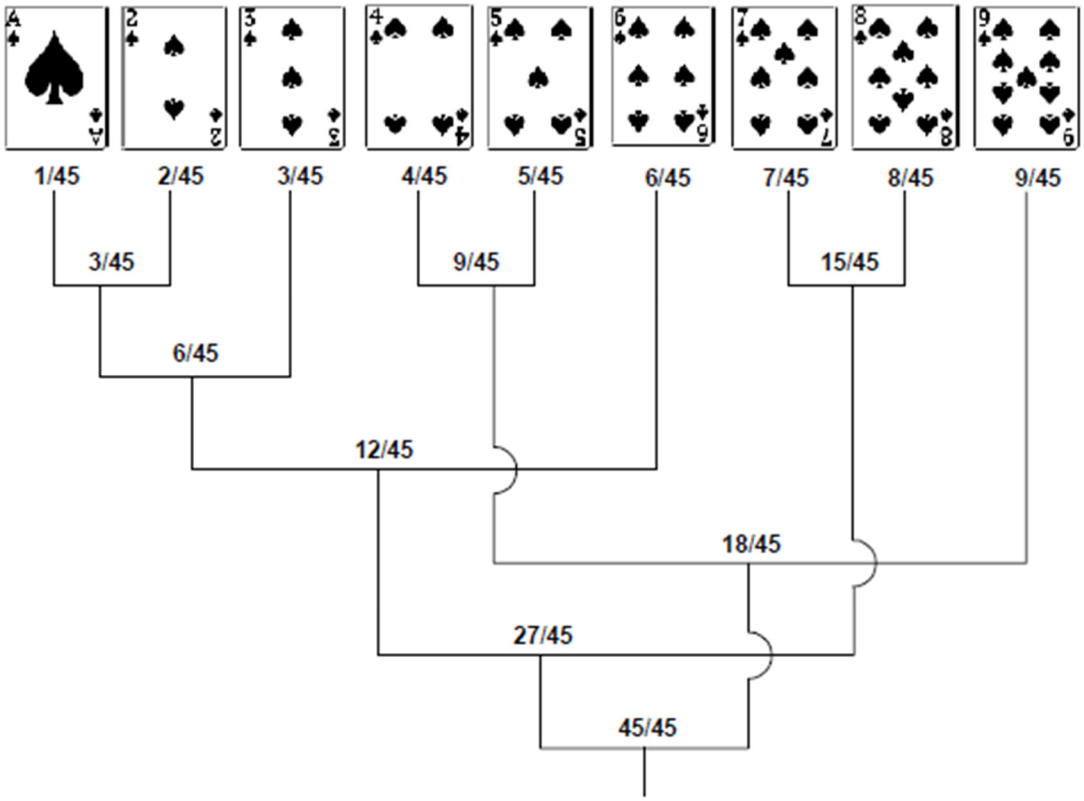
   The binary tree with n leaves can be no larger than n – 1. Can a Huffman tree be that tall? The answer is yes. Let the probabilities of the n characters be $p_1 = ½$, $p_2 = ¼$, $p_3 = 1/8$, … $p_{n-1} = 1/(2^{n-1})$, $p_n = 1/(2^{n-1})$. The first tree merger will combine the last two characters into a tree with total probability $1/(2^{n-2})$, which is also the probability of c_{n-2}. This continues from right to left, so that in i^{th} non-leaf node of the final tree, the left subtree has a single node (c_i), and the right subtree has c_{i+1}, … , c_n arranged into a similar tree with height n – i – 1. So the total height of the tree is n-1. The codeword for c_i (i=1..n-1) is i-1 1's followed by a 0, and the codeword for c_n is n-1 1's.

2. (10) 9.4.10 (card guessing)

The probabilities of a selected card be of a particular type is given in the following table:

card	ace	deuce	three	four	five	six	seven	eight	nine
probability	1/45	2/45	3/45	4/45	5/45	6/45	7/45	8/45	9/45

Huffman's tree for this data looks as follows:



The first question this tree implies can be phrased as follows: "Is the selected card a four, a five, or a nine?" . (The other questions can be phrased in a similar fashion.)

The expected number of questions needed to identify a card is equal to the weighted path length from the root to the leaves in the tree:

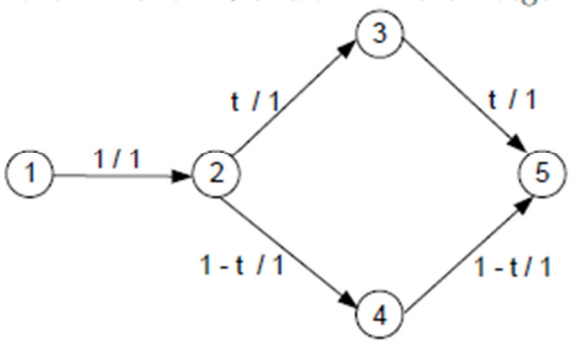
$$\bar{l} = \sum_{i=1}^{9} l_i p_i = \frac{5\cdot 1}{45} + \frac{5\cdot 2}{45} + \frac{4\cdot 3}{45} + \frac{3\cdot 5}{45} + \frac{3\cdot 6}{45} + \frac{3\cdot 7}{45} + \frac{3\cdot 8}{45} + \frac{2\cdot 9}{45} = \frac{135}{45} = 3.$$

4. (6) 10.2.1 (sources and sinks with negative weights)

A vertex is a source if and only if there are no negative entries in its row, and it is a sink if there are no positive entries in its row. Finding a source and a sink simply requires looking through all of the elements of the matrix, so it is $\Theta(n^2)$.

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

a. The maximum-flow problem may have more than one optimal solution. In fact, there may be infinitely many of them if we allow (as the definition does) non-integer edge flows. For example, for any $0 \leq t \leq 1$, the flow depicted in the diagram below is a maximum flow of value 1. Exactly two of them—for $t = 0$ and $t = 1$—are integer flows.



The answer does not change for networks with distinct capacities: e.g., consider the previous example with the capacities of edges $(2, 3)$, $(3, 5)$, $(2, 4)$, and $(4, 5)$ changed to, say, $2, 3, 4$, and 5, respectively.

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

Add two vertices, which will be the source and sink of the new network. Connect the new source to each of the original sources and connect each of the original sinks to the new sink, by edges with some large capacity M. It is sufficient tolet M be the sum of the capacities of all edges that leave sources of the original network.

7. (8) 10.4.1 (stable marriage example)

There are the total of $3! = 6$ one-one matchings of two disjoint 3-element sets:

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

$\{(\alpha, A), (\beta, B), (\gamma, C)\}$ is stable: no other cell can be blocking since each man has his best choice. This is obviously the man-optimal matching.

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

$\{(\alpha, A), (\beta, C), (\gamma, B)\}$ is unstable: (γ, A) is a blocking pair.

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

$\{(\alpha, B), (\beta, A), (\gamma, C)\}$ is unstable: (β, C) is a blocking pair.

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

$\{(\alpha, B), (\beta, C), (\gamma, A)\}$ is stable: all the other cells contain a 3 (the lowest rank) and hence cannot be a blocking pair. This is neither a man-optimal nor a woman-optimal matching since it's inferior to $\{(\alpha, A), (\beta, B), (\gamma, C)\}$ for the men and inferior to $\{(\alpha, C), (\beta, A), (\gamma, B)\}$ for the women.

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

$\{(\alpha, C), (\beta, A), (\gamma, B)\}$ is stable: no other cell can be blocking since each woman has her best choice. This is obviously the woman-optimal matching.

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

$\{(\alpha, C), (\beta, B), (\gamma, A)\}$ is unstable: (α, B) is a blocking pair.

8. (4) 10.4.2 (stable marriage check algorithm)

2. **Stability-checking algorithm**

Input: A marriage matching M of n (m, w) pairs along with rankings of the women by each man and rankings of the men by each woman
Output: "yes" if the input is stable and a blocking pair otherwise
for $m \leftarrow 1$ **to** n **do**
  for each w such that m prefers w to his mate in M **do**
    if w prefers m to her mate in M
      return (m, w)
return "yes"

With appropriate data structures, it is not difficult to implement this algorithm to run in $O(n^2)$ time. For example, the mates of the men and the mates of the women in a current matching can be stored in two arrays of size n and all the preferences can be stored in the n-by-n ranking matrix containing two rankings in each cell.

9. (6) 10.4.5 (time efficiency of stable marriage Algorithm)

5. a. The worst-case time efficiency of the algorithm is in $\Theta(n^2)$. On the one hand, the total number of the proposals, $P(n)$, cannot exceed n^2, the total number of possible partners for n men, because a man does not propose to the same woman more than once. On the other hand, for the instance of size n where all the men and women have the identical preference list 1, 2, ..., n, $P(n) = \sum_{i=1}^{n} i = n(n+1)/2$. Thus, if $P_w(n)$ is the number of proposals made by the algorithm in the worst case,

$$n(n+1)/2 \leq P_w(n) \leq n^2,$$

i.e., $P_w(n) \in \Theta(n^2)$.

b. The best-case time efficiency of the algorithm is in $\Theta(n)$: the algorithm makes the minimum of n proposals, one by each man, on the input that ranks first a different woman for each of the n men.

10. 8.3.11bc [8.3.10bc] matrix chain multiplication.

b. Let $m(n)$ be the number of different ways to compute a chain product of n matrices $A_1 \cdot \ldots \cdot A_n$. Any parenthesization of the chain will lead to multiplying, as the last operation, some product of the first k matrices $(A_1 \cdot \ldots \cdot A_k)$ and the last $n-k$ matrices $(A_{k+1} \cdot \ldots \cdot A_n)$. There are $m(k)$ ways to do the former, and there are $m(n-k)$ ways to do the latter. Hence, we have the following recurrence for the total number of ways to parenthesize the matrix chain of n matrices:

$$m(n) = \sum_{k=1}^{n-1} m(k)m(n-k) \quad \text{for } n > 1, \quad m(1) = 1.$$

Since parenthesizing a chain of n matrices for multiplication is very similar to constructing a binary tree of n nodes, it should come as no surprise that the above recurrence is very similar to the recurrence

$$b(n) = \sum_{k=1}^{n-1} b(k)b(n-1-k) \quad \text{for } n > 1, \quad b(0) = 1,$$

for the number of binary trees mentioned in Section 8.3. Nor is it surprising that their solutions are very similar, too: namely,

$$m(n) = b(n-1) \quad \text{for } n \geq 1,$$

where $b(n)$ is the number of binary trees with n nodes. Let us prove this assertion by mathematical induction. The basis checks immediately: $m(1) = b(0) = 1$. For the general case, let us assume that $m(k) = b(k-1)$ for all positive integers not exceeding some positive integer n (we're using the strong version of mathematical induction); we'll show that the equality holds for $n+1$ as well. Indeed,

$$\begin{aligned}
m(n+1) &= \sum_{k=1}^{n} m(k)m(n+1-k) \\
&= [\text{using the induction's assumption}] \sum_{k=1}^{n} b(k-1)b(n-k) \\
&= [\text{substituting } l = k-1] \quad \sum_{l=0}^{n-1} b(l)b(n-1-l) \\
&= [\text{see the recurrence for } b(n)] \quad b(n).
\end{aligned}$$

c. Let $M[i,j]$ be the optimal (smallest) number of multiplications needed for computing $A_i \cdot ... \cdot A_j$. If k is an index of the last matrix in the first factor of the last matrix product, then

$$M[i,j] = \max_{1 \le k \le j-1} \{M[i,k] + M[k+1,j] + d_{i-1} d_k d_j\} \text{ for } 1 \le i < j \le n,$$
$$M[i,i] = 0.$$

This recurrence, which is quite similar to the one for the optimal binary search tree problem, suggests filling the $n+1$-by-$n+1$ table diagonal by diagonal as in the following algorithm:

Algorithm *MatrixChainMultiplication*$(D[0..n])$
//Solves matrix chain multiplication problem by dynamic programming
//Input: An array $D[0..n]$ of dimensions of n matrices
//Output: The minimum number of multiplications needed to multiply
//a chain of n matrices of the given dimensions and table $T[1..n, 1..n]$
//for obtaining an optimal order of the multiplications
for $i \leftarrow 1$ **to** n **do** $M[i,i] \leftarrow 0$
for $d \leftarrow 1$ **to** $n-1$ **do** //diagonal count
    for $i \leftarrow 1$ **to** $n-d$ **do**
        $j \leftarrow i+d$
        $minval \leftarrow \infty$
        for $k \leftarrow i$ **to** $j-1$ **do**

            $temp \leftarrow M[i,k] + M[k+1,j] + D[i-1] * D[k] * D[j]$
            if $temp < minval$
                $minval \leftarrow temp$
                $kmin \leftarrow k$
        $T[i,j] \leftarrow kmin$
return $M[1,n]$, T

To find an optimal order to multiply the matrix chain, call *OptimalMultiplicationOrder*$(1,n)$ below:
Algorithm *OptimalOrder*(i,j)
//Outputs an optimal order to multiply n matrices
//Input: Indices i and j of the first and last matrices in $A_i...A_j$ and
//table $T[1..n, 1..n]$ generated by *MatrixChainMultiplication*
//Output: $A_i...A_j$ parenthesized for optimal multiplication
if $i = j$
    print("A_i")
else
    $k \leftarrow T[i,j]$
    print("(")
    OptimalOrder(i,k)
    OptimalOrder$(k+1,j)$
    print(")")