CSCI 2300 — Introduction to Algorithms
Homework 4 (document version 1.1) — DUE March 6, 2020
Greedy Algorithms, Huffman Encoding, and Divide-and-Conquer

- This homework is due by 11:59PM on the due date above and must be submitted in Submitty as a single PDF called `upload.pdf`.
- Just like the labs, please avoid using Google to find suggestions or solutions. You are allowed to use the textbooks and class notes as references. We strongly encourage you to use the office hours, labs/recitations, and Discussion Forum to successfully complete this assignment.
- Also, you are allowed to consult with your classmates, collaborating in line with the policies set forth in the syllabus. You must write up your own separate solution; do not copy-and-paste anything.
- Remember the goal is to use your own brain to work these problems out, which will help you develop the skills to do well on the exams and, more importantly, become a substantially better computer scientist.
- For your solutions and proofs, please write clearly and concisely, and use rigorous formal arguments and pseudocode that mimics the style used in the Dasgupta textbook. Be consistent.
- In general, if you are asked to provide an algorithm, the best thing to do is to provide a clear description in English (e.g., "Use BFS, but with the following change..." or "Run BFS two times as follows...").
- Be as concise as you can in your answers. Long answers are difficult to grade.

Warm-Up Problems (do not hand in for grading)

Work out the problems below as practice problems before going on to the next section. Note that warm-up problems might sometimes show up on an exam!

1. DPV Problem 5.9 (all parts) – **SOLUTIONS:**

   (a) False. Consider the case in which the heaviest edge is a bridge, i.e., the heaviest edge is the only edge connecting two connected components of G.

   (b) True. Prove this claim using contradiction (i.e., assume edge e is part of an MST).

   (c) True. Use Kruskal's algorithm and remember that graph G might have multiple MSTs.

   (d) True. Prove this claim using contradiction.

   (e) True. Consider the cut that has vertex u in one side and vertex v in the other, with edge $e = (u, v)$.

   (f) False. Construct a graph to show a counterexample.

   (g) False. Construct a graph to show a counterexample (e.g., graph $G = (V, E)$ with $V = \{u, v, w\}$, $E = \{e_1 = (u, v), e_2 = (u, w), e_3 = (v, w)\}$, and weights $\mathrm{w}(e_1) = 11$, $\mathrm{w}(e_2) = 10$, and $\mathrm{w}(e_3) = 2$.

   (h) False. Construct a graph to show a counterexample (e.g., in the graph from part (g) above, the shortest path between vertices u and v is simply edge (u, v), but the only MST has edges (u, w) and (v, w)).

   (i) True. Prove this claim by proving that each step of the algorithm works for an edge of negative edge weight.

   (j) True. Prove this claim using contradiction.

2. DPV Problem 5.11 – Also do this problem without using path compression.

3. DPV Problem 5.13 – What happens if the four symbols are equally likely (i.e., appear with 25% frequency)?

   SOLUTIONS: For characters A, C, G, T with frequencies 31%, 20%, 9%, and 40%, respectively, multiple answers are possible. Regardless, the Huffman encoding algorithm will assign codewords of length 1 to T, length 2 to A, and length 3 to both C and G. One possible encoding is 0 for T, 10 for A, 110 for C, and 111 for G. (Draw the tree to confirm.)

   If the four symbols are equally likely, we will have the same length for the four codewords as above, but we can arbitrarily assign the codewords to A, C, G, T because each is equally likely to occur.

4. DPV Problem 2.4 – **SOLUTIONS:**

   - For Algorithm A, use the Master Theorem with $a = 5$, $b = 2$, and $d = 1$. Since $a > b^d$, the runtime is $O(n^{\log_b a}) = O(n^{\log_2 5}) = O(n^{2.33})$.
   - For Algorithm B, the recurrence is $T(n) = 2T(n-1) + C$ for some constant C. We can expand $T(n)$ a few terms and detect a pattern, which we can expand to $T(n) = C\sum_{i=0}^{n-1} 2^i + 2^n T(0)$; this simplifies to $O(2^n)$ since the latter term dominates.
   - For Algorithm C, use the Master Theorem with $a = 9$, $b = 3$, and $d = 2$. Since $a = b^d$, the runtime is $O(n^d \log n) = O(n^2 \log n)$.

     Given the above, we conclude that Algorithm C is asymptotically the best for large n.

5. DPV Problem 2.5 (especially b, d, f, g, and h) – **SOLUTIONS:**

   (a) Use the Master Theorem....

   (b) $T(n) = 5T(n/4) + n = \Theta(n^{\log_4 5})$ by the Master Theorem.

   (c) Use the Master Theorem....

   (d) $T(n) = 9T(n/3) + n^2 = \Theta(n^2 \log_3 n)$ by the Master Theorem.

   (e) Use the Master Theorem....

   (f) Use the Master Theorem....

   (g) $T(n) = T(n-1) + 2 = \Theta(n)$ by expanding out a few terms and observing the pattern.

Required Problems (hand in for grading)

Work out the problems below and submit for grading before the deadline.

1. DPV Problem 5.15 (all parts) – Also add (d) Code: $\{0, 100, 1010, 1011, 11\}$. **(v1.1)** And extend the alphabet as necessary by using the next alphabetical symbols d, e, etc.

   SOLUTIONS AND GRADING GUIDELINES:

   $\Longrightarrow$ 3 points for each part below

   (a) To yield the code $\{0, 10, 11\}$, we have frequencies $(f_a, f_b, f_c) = (2/3, 1/6, 1/6)$. Other frequencies are possible here as long as $f_a + f_b + f_c = 1$, $f_a > f_b$ and $f_a > f_c$.

   (b) Encoding $\{0, 1, 00\}$ is not possible since the code for a (0) is a prefix of the code for c (00).

   (c) Encoding $\{10, 01, 00\}$ is not possible since it is not optimal and does not correspond to a *full* binary tree.

   (d) Encoding $\{0, 100, 1010, 1011, 11\}$ is valid, with one set of possible frequencies $(f_a, f_b, f_c, f_d, f_e) = (0.5, 0.15, 0.05, 0.05, 0.25)$. Other frequencies are possible here as long as $f_a + f_b + f_c + f_d + f_e = 1$, f_a is greater than all other frequencies, f_b is greater than f_c and f_d, and f_e is greater than f_b, f_c, and f_d.

2. DPV Problem 5.16 (all parts)

SOLUTIONS AND GRADING GUIDELINES:

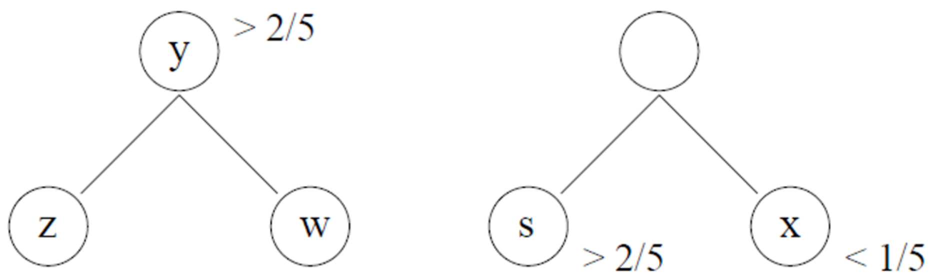
For each part below,
$\Longrightarrow$ 2 points for setting up the proof
$\Longrightarrow$ 3 points for an accurate and convincing proof

(a) If some character occurs with frequency more than 2/5, then there is guaranteed to be a codeword of length 1. We prove this by contradiction.

First, let s be the symbol with the highest frequency (probability) $p(s) > 2/5$ and suppose that it merges with some other symbol during the process of constructing the tree; therefore, s does not correspond to a codeword of length 1.

To be merged with some other node, node s and some other node x must be the two with minimum frequencies. This implies that there was at least one other node y (formed by other mergings) with $p(y) > p(s)$ and $p(y) > p(x)$. Thus, $p(y) > 2/5$ and (halving 2/5), $p(x) < 1/5$.

Given the above, y must also have been formed by merging some two nodes z and w with at least one of them having probability greater than 1/5 since they add up to more than 2/5. This is a contradiction, i.e., $p(z)$ and $p(w)$ could not have been the minimum since $p(x) < 1/5$. This is further illustrated in the diagram below:



(b) If all characters occur with frequency less than 1/3, then there is guaranteed to be no codeword of length 1. We prove this by contradiction.

Suppose the claim is not true. Let x be a node corresponding to a single character with $p(x) < 1/3$ such that the encoding of x is of length 1. Then x must not merge with any other node until the end. Consider the stage of the algorithm when there are only three leaf nodes left in the tree, i.e., nodes x, y, and z. At the last stage, y and z must merge to form another node so that x still corresponds to a codeword of length 1. But $p(x) + p(y) + p(z) = 1$ and $p(x) < 1/3$ combine to imply that $p(y) + p(z) > 2/3$. Therefore, at least one of $p(y)$ or $p(z)$ must be greater than 1/3; if we choose $p(z) > 1/3$ (without loss of generality), then these two cannot merge since $p(x)$ and $p(y)$ would be the minimum. Therein lies the contradiction.

3. DPV Problem 2.17 – Show that your algorithm is $O(\log n)$.

SOLUTIONS AND GRADING GUIDELINES:

$\Longrightarrow$ 2 points for clearly describing a divide-and-conquer algorithm
$\Longrightarrow$ 2 points for an accurate and correct algorithm
$\Longrightarrow$ 2 points for showing that the algorithm is $O(\log n)$.

First examine middle element $A[\frac{n}{2}]$; if $A[\frac{n}{2}] = \frac{n}{2}$, we are done (in constant $O(1)$ time).

If instead $A[\frac{n}{2}] > \frac{n}{2}$, then we can ignore the latter half of the array since the given array is sorted and contains distinct elements (both of these constraints are required here). Similarly, if $A[\frac{n}{2}] < \frac{n}{2}$, we can ignore the first half of the array.

After the comparison, we recurse on the appropriate half of the array, continuing this division until we get down to a single element x. If $A[x] = x$ then we have found an index i for which $A[i] = i$; otherwise, no such element exists.

Each recursive call does a constant amount of work. Therefore, our recurrence relation is as follows:

$$T(n) = T(n/2) + O(1)$$

Using the Master Theorem, we have $T(n) = O(\log n)$.

4. DPV Problem 2.19 (all parts) – Assume elements in the arrays are integers. For part (b), show the runtime complexity of your algorithm; note that it must be faster than part (a) and also faster than that of just combining all k arrays into one large array and sorting that from scratch.

SOLUTIONS AND GRADING GUIDELINES:

$\Longrightarrow$ 4 points for part (a) for showing the time complexity
$\Longrightarrow$ 2 points for part (b) for describing a divide-and-conquer algorithm
$\Longrightarrow$ 2 points for part (b) for an accurate and correct algorithm
$\Longrightarrow$ 2 points for part (b) for showing the time complexity

(a) Let $T(i)$ be the time taken to merge arrays 1 to i. This essentially consists of the time taken to merge arrays 1 to $i-1$ plus the time taken to merge the resulting array of size $n \times (i-1)$ with array i to produce an array of size $n \times i$. Therefore, for some constant c, $T(i) \leq T(i-1) + cni$, from which we obtain:

$$T(k) \leq T_1 + cn \sum_{i=2}^{k} i = O(nk^2)$$

(b) Using a divide-and-conquer approach, we first divide the arrays into two sets, each with $k/2$ arrays, continuing to divide until each subproblem consists of just one array (i.e., $k/2 = 1$). Recursively merge the arrays.

The runtime is given by the following recurrence:

$$T(k) = 2T(k/2) + O(kn)$$

By the Master Theorem, we have $T(k) = O(kn \log k)$.

5. DPV Problem 2.23 (all parts)

SOLUTIONS AND GRADING GUIDELINES:

$\Longrightarrow$ 2 points for part (a) for describing a divide-and-conquer algorithm
$\Longrightarrow$ 2 points for part (a) for showing the time complexity
$\Longrightarrow$ 2 points for part (b) for describing a divide-and-conquer algorithm
$\Longrightarrow$ 2 points for part (b) for an accurate and correct algorithm
$\Longrightarrow$ 2 points for part (b) for showing the time complexity

(a) To find v, we repeatedly divide the problem into subproblems with subarrays A_1 and A_2 that split array A into two halves. If A has majority element v, then v must also be a majority element of A_1 or A_2 or both. We can recursively compute the majority elements of these subarrays. The runtime is given by $T(n) = 2T(\frac{n}{2}) + O(n)$, which by the Master Theorem gives us $T(n) = O(n \log n)$.

(b) Using a divide-and-conquer approach, pair up (i.e., divide) elements of A arbitrarily to yield $n/2$ pairs. Then look at each pair; if the two elements are different, discard them both, but if the two elements are the same, deduplicate and keep just one of them.

After this divide-and-conquer approach is applied, there are at most $n/2$ elements left since at least one element in each pair is discarded. If these remaining elements have a majority, then there exists a v among them appearing at least $n/4$ times. Therefore, v must have been paired up with itself in at least $n/4$ pairs, showing that A contains at least $n/2$ copies of v.

The runtime of this algorithm is linear $O(n)$, as shown from the Master Theorem applied to recurrence $T(n) = T(n/2) + O(n)$.