CMPSC 465 Data Structures & Algorithms
Fall 2025 Ke Chen and Yana Safonova

Worksheet 9 Solution

Monday, Nov 03, 2025

1. **Huffman Properties.** Prove the following: if some character occurs with frequency more than 2/5, then there is guaranteed to be a codeword of length 1.

   Solution: 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 and hence does not correspond to a codeword of length 1. To be merged with some node, the node s and some other node x must be the two with minimum frequencies. This means there was at least one other node y (formed by merging of other nodes), with $p(y) > p(s)$ and $p(y) > p(x)$. Thus, $p(y) > 2/5$ and hence $p(x) < 1/5$.

   Now, y must have been formed by merging some two nodes z and w with at least one of them having probability greater than $1/5$ (as they add up to more than 2/5). But this is a contradiction - $p(z)$ and $p(w)$ could not have been the minimum since $p(x) < 1/5$.

2. **Huffman Encoding.** Let $n \geq 2$ and label symbols $s_1, \ldots, s_n$ with frequencies

$$f_i = 2^{n-i} \qquad (i = 1, \ldots, n),$$

   so the frequencies (in descending order) are $2^{n-1}, 2^{n-2}, \ldots, 2, 1$. Construct the Huffman code for these frequencies and determine the codeword lengths $L(s_i)$ for all i.

   Solution: When the frequencies are powers of two as above, the Huffman merges proceed deterministically from the two smallest frequencies upward. First $f_n = 1$ and $f_{n-1} = 2$ are the two smallest, so Huffman merges them into a node of weight $1+2=3$. Since $3 < 4 = f_{n-2}$, the next merge is between the node of weight 3 and $f_{n-2} = 4$, producing weight 7. Inductively, after $k-1$ such merges one obtains a combined node of weight $2^k - 1$ which is smaller than the next untouched original frequency 2^k, so the merged node will next combine with 2^k to form $2^{k+1} - 1$. Continuing this process until all nodes are merged shows that at the final stage the largest original frequency 2^{n-1} is merged with the combined node of weight $2^{n-1} - 1$; hence s_1 (the symbol with frequency 2^{n-1}) becomes a child of the root and has codeword length 1. Consequently the Huffman tree for frequencies $2^{n-1}, 2^{n-2}, \ldots, 1$ yields codeword lengths $L(s_i) = \min(i, n-1)$ where the least frequent two nodes have same code word lenght of $n-1$.

3. **Worst Case for Greedy Set Cover.** Let n be a power of 2. Show that there exists an instance of the set cover problem such that: 1) there are n elements in the base set; 2) the optimal solution uses only two sets; and 3) the greedy algorithm picks at least $\log n$ sets.

   Solution: Let $n = 2^k$. Partition the universe U into disjoint layers $L_1, \ldots, L_k$ with $|L_j| = 2^{k-j}$. For each j let $T_j := L_j$ (the "trap" sets). Split each layer L_j into two equal halves L_j^A, L_j^B and put $A = \bigcup_j L_j^A$, $B = \bigcup_j L_j^B$. Then $A \cup B = U$, so $\{A, B\}$ is a cover of size 2 and hence optimal. Run the greedy algorithm that at each step selects a set covering the largest number of uncovered elements; ties are broken in favor of trap sets. Initially $|T_1| = 2^{k-1} = |A| = |B|$, so greedy may pick T_1. After removing T_1 the remaining universe has size 2^{k-1}, and again $|T_2| = 2^{k-2}$ equals the number of new elements any global set would cover, so greedy (by the tie rule) picks T_2. Iterating this argument shows greedy selects $T_1, T_2, \ldots, T_k$ (and then one more set to finish), hence it picks at least $k = \log_2 n$ sets while the optimum uses only two. $\square$

4. Weighted Set Cover and the Greedy Algorithm

Let us consider the weighted version of the Set Cover problem.

Suppose the universe is

$$U = \{1,2,3,4,5,6,7,8\}.$$

The following sets and weights are given:

$$\begin{aligned} T_1 &= \{1,2,3,4\}, \quad w(T_1) = 9, \\ T_2 &= \{5,6\}, \quad w(T_2) = 7, \\ T_3 &= \{7\}, \quad w(T_3) = 5, \\ A &= \{1,3,5,7\}, \quad w(A) = 10, \\ B &= \{2,4,6,8\}, \quad w(B) = 10. \end{aligned}$$

(a) What can be a possible greedy strategy for solving this weighted set cover problem?

(b) Apply the proposed greedy strategy step by step to find the sets selected by the greedy algorithm and the total cost of the cover it produces.

(c) Determine the optimal solution and its cost. Compare it with the greedy solution.

Solution.

(a) A simple greedy strategy is to repeatedly select the set that minimizes the ratio of weight to the number of uncovered elements it newly covers i.e., minimize $\frac{w_i}{X_i}$ where w_i is the weight of the set S_i and X_i is the number of curretly uncovered element in S_i.

(b) Initially, the cost-per-element ratios are $9/4 = 2.25$ for T_1, $7/2 = 3.5$ for T_2, $5/1 = 5$ for T_3, and $10/4 = 2.5$ for each of A and B. The smallest ratio corresponds to T_1, so it is chosen first, covering $\{1,2,3,4\}$. The remaining uncovered elements are $\{5,6,7,8\}$. The next ratios are $T_2 : 7/2 = 3.5$, $T_3 : 5/1 = 5$, $A : 10/2 = 5$, and $B : 10/2 = 5$. The greedy algorithm next picks T_2, covering $\{5,6\}$. The uncovered elements are now $\{7,8\}$. In the next step, T_3 covers one element with ratio $5/1 = 5$, while A and B each have ratio $10/1 = 10$. Thus the greedy algorithm picks T_3, covering $\{7\}$. Finally, the remaining uncovered element 8 can only be covered by B, so B is added. The greedy solution is therefore $\{T_1, T_2, T_3, B\}$ with total cost $9+7+5+10 = 31$.

(c) The optimal cover uses $\{A, B\}$, which together cover all elements at a total cost of $10+10 = 20$. Hence, the greedy algorithm yields a solution that is $31/20 = 1.55$ times more expensive than the optimal one. The greedy strategy performs worse because it prioritizes sets with low immediate cost-per-element, ignoring that the global sets A and B together provide a cheaper overall cover.