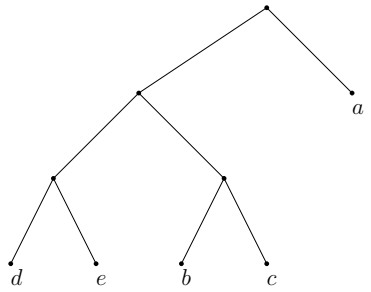
- This problemset has *four* questions.
- For other questions, either send the solutions to `gawry1+aos@gmail.com`, or leave them in the envelope attached to the doors of my office (room 321).

1. Huffman code is defined as follows: given the probability $p(c)$ of each character $c \in \Sigma$ we want to select all $\text{code}(c) \in \{0,1\}^+$ so that they are prefix-free, meaning that no $\text{code}(c)$ is a proper prefix of $\text{code}(c')$, and that the expected cost $\sum_{c\in\Sigma} p(c)|\text{code}(c)|$ is minimized.

   (a) Construct the Huffman code for $\Sigma = \{a, b, c, d, e\}$ and the probabilities $p(a) = \frac{12}{31}$, $p(b) = \frac{6}{31}$, $p(c) = \frac{5}{31}$, $p(d) = \frac{4}{31}$, $p(e) = \frac{4}{31}$. Compute the corresponding expected cost.

   Solution: To construct the code we can use a greedy method: iteratively combine two symbols with the smallest probability. The probability of the combination is simply the sum of the corresponding two probabilities. So, we first combine d and e to get symbol α with $p(\alpha) = \frac{8}{31}$, then combine b and c to get β with $p(\beta) = \frac{11}{31}$, then we combine α and β to get γ with $p(\gamma) = \frac{19}{31}$, and finally we combine γ and a. The resulting code is shown below, it's cost is $\frac{69}{31}$.

   (b) Alphabetical Huffman code have the additional property that $\text{code}(c) < \text{code}(c')$ if $c < c'$, where the first inequality is understood as the lexicographical comparison. Show an example where this additional restriction increases the expected cost.

   Solution: Let's try to find such an example of three elements $\{a, b, c\}$. We should choose the probabilities in such a way that the "usual" Huffman tree must orders them differently. So, let's check $p(a) = 0.25$, $p(b) = 0.5$ and $p(c) = 0.25$. Then the optimal Huffman tree first merges a and c, so its cost is $0.5 * 1 + 0.25 * 2 + 0.25 * 2 = 1.5$. If we want the tree to be alphabetical, there are just two possibilities: first merge a and b, or first merge b and c. They are symmetric, so it's enough to look at just one. Its cost is $0.25 * 2 + 0.5 * 2 + 0.25 * 1 = 1.75 > 1.5$.

   Even though the alphabetical cost is larger, being able to compare the characters by looking at their codes is useful. So the question is how to compute the optimal alphabetical Huffman code quickly? It turns out that it can be done in $\mathcal{O}(n \log n)$, so the same complexity as for the "usual" variant by the Hu-Tucker algorithm, but it's quite

nontrivial to prove its correctness. There is a simpler $\mathcal{O}(n^2)$ dynamic programming solution that can be found in the Knuth's TAOCP.

2. Recall that the zeroth order entropy was defined as $H(w[1..n]) = \sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n}{n_c}$, where n_c is the number of occurrences of the letter c in $w[1..n]$.

   (a) Compute the entropy for $w = a^{12}b^6c^5d^4e^4$.

   Solution: The answer rounded up is 2.18.

   (b) Prove that if the number $\ell_1, \ell_2, \ldots, \ell_k$ are chosen so that $\sum_i 2^{-\ell_i} \leq 1$, it is possible to construct a binary tree on k leaves, where each ℓ_i is a depth of one of the leaves.

   Hint: This is known as (one half) of the Kraft's inequality. Use induction on k.

   Solution: For $k = 1$, the claim is obvious. So assume that it holds for k, and look at the situation for $k+1$. Choose two largest depths ℓ_i and ℓ_j, where $i < j$. We want to construct the tree on $k-1$ leaves on depths $\ell_1, \ldots, \ell_{i-1}, \ell_{i+1}, \ldots, \ell_{j-1}, \ell_{j+1}, \ldots, \ell_k, \min(\ell_i, \ell_j) - 1$. If we can do so, then we can simply attach two new leaves to the one corresponding to depth $\min(\ell_i, \ell_j) - 1$, they will be at depth $\min(\ell_i, \ell_j) \leq \ell_i, \ell_j$. So, we only have to check if the assumption holds for this new smaller set of depths.

   Observe that the sum corresponding to the smaller set is the old sum minus $2^{-\ell_i} + 2^{-\ell_j}$ plus $2^{-(\min(\ell_i, \ell_j)-1)}$, so the difference is

   $$\frac{1}{2^{\ell_i}} + \frac{1}{2^{\ell_j}} - \frac{2}{2^{\min(\ell_i, \ell_j)}}$$

   and we should show that this value is at least 0. Then the new sum will be at most 1, because the old sum was at most 1.

   If $\ell_i = \ell_j$, the sum is exactly 0. So what can be done if $\ell_j > \ell_i$? Remember that we have chosen the two largest depths, which means that we have ℓ_j and a bunch of strictly smaller depths. Now, we know that $\sum_i 2^{-\ell_i} \leq 1$. If we look at $2^{-\ell_j} + \sum_{i \neq j} 2^{-\ell_i}$, we can observe that the sum is actually at most $1 - 2^{-\ell_j}$! This is because the binary expansion of the second part cannot contain ones at positions $\ell_j, \ell_j + 1, \ldots$, as all fractions in this sum are at least $2^{-\ell_j+1}$. Hence we can safely decrease ℓ_j by one, and still have $\sum_i 2^{-\ell_i} \leq 1$. By repeating this we reduce the general case to the one with $\ell_i = \ell_j$.

   (c) Show that the Huffman cost is less than $H(w[1..n]) + 1$. You might find the previous subquestion useful.

   Hint: show that some code with the expected cost not exceeding $H(w[1..n]) + 1$ exists, then argue that the Huffman code is the best possible, so can't be worse.

Solution: Set $\ell_i = \left\lceil \log \frac{n}{n_i} \right\rceil$, where n_i is the number of occurrences of the i-th letter. We want to show that there exists a binary tree where each ℓ_i is a depth of one of the leaves. For this we first compute $\sum_i 2^{-\ell_i}$.

$$\sum_i 2^{-\ell_i} = \sum_i \frac{1}{2^{\left\lceil \log \frac{n}{n_i} \right\rceil}} \leq \sum_i \frac{1}{2^{\log \frac{n}{n_i}}} = \sum_i \frac{1}{\frac{n}{n_i}} = \sum_i \frac{n_i}{n} = \frac{n}{n} = 1$$

So from the previous part, such binary tree exists. Now let's compute its expected cost:

$$\sum_i \frac{n_i}{n} \ell_i = \sum_i \frac{n_i}{n} \left\lceil \log \frac{n}{n_i} \right\rceil \leq \sum_i \frac{n_i}{n} + \sum_i \frac{n_i}{n} \log \frac{n}{n_i} = 1 + H(w[1..n])$$

so the cost of our code is at most $1 + H(w[1..n])$. The Huffman code has the smallest possible cost, so might be only better (or the same).

(SPOJ) 3. Solve the BWHEELER problem, which asks you to reverse the Burrows-Wheeler transform.

Solution: We discussed this during the lecture.