Problem 1. Let $G = (V, E)$ be a connected undirected graph where every edge carries a positive integer weight. Divide V into arbitrary disjoint subsets $V_1, V_2, \ldots, V_t$ for some $t \geq 2$, namely, $V_i \cap V_j = \emptyset$ for any $1 \leq i < j \leq t$ and $\bigcup_{i=1}^{t} V_i = V$. Define an edge $\{u, v\}$ in E as a *cross edge* if u and v are in different subsets. Prove: a cross edge with the smallest weight must belong to a minimum spanning tree (MST).

Proof. Let $e = \{u, v\}$ be a cross edge having the smallest weight. Without loss of generality, suppose that $u \in V_i$ and $v \in V_j$ for some $i \neq j$. Consider an arbitrary MST T, where e is not in T.

Add e to T which produces a cycle C. Walk on C in the following manner: start from u, cross e to reach v and continue until crossing e' that takes us back to a vertex in V_i. This edge e' is a cross edge, and thus is at least having a larger weight than e. Deleting e' gives us an MST containing e. □

Problem 2. Let $G = (V, E)$ be a connected undirected graph where every edge carries a positive integer weight. Prove that the following algorithm finds an MST of G correctly:

```
procedure KruskalAlgorithm(S)
    S = ∅
    while |S| < |V| − 1 do
        find the lightest edge e ∈ E that does not introduce any cycle with the edges in S
        add e to S
    end while
    return the tree T = (V, S)
end procedure
```

Proof. We prove the theorem by induction. Let $e_1, e_2, \ldots, e_{n-1}$ be the edges picked by the algorithm. We argue that there is an MST using the first k edges $e_1, e_2, \ldots, e_k$ for any $k \in [1, n-1]$.

Base Case. e_1 is the edge in G with the minimum weight. Therefore, the Kruskal's algorithm must match the choice of Prim's. Thus there must exist an MST with e_1.

Inductive Step. Suppose that there is an MST T that matches with the first k choices of edges by the Kruskal's algorithm. Then, consider the edge e_{k+1}. Clearly, it is a cross-edge from a connected component G_i to another connected component G_j. By the algorithm's selection e_{k+1}, it must be that e_{k+1} is the lightest cross edge between G_i and G_j. By Problem 1, we can replace the cross-edge in T connecting G_i and G_j with e_{k+1}, and thus is no worse. We can conclude that $e_1, e_2, \ldots, e_{k+1}$ must be used by some MST. This finishes the inductive step and concludes the theorem. □

Problem 3. Consider Σ as an alphabet. Recall that a code tree on Σ is a binary tree T satisfying both conditions below:

- Every leaf node of T is labeled with a distinct letter in Σ; conversely, every letter in Σ is the label of a distinct leaf node in T.
- For every internal node of T, its left edge (if exists) is labeled with 0, and its right edge (if exists) with 1.

Define an encoding as a function f that maps each letter $\sigma \in \Sigma$ to a non-empty bit string, which is called the codeword of σ. T produces an encoding where the code word of a letter $\sigma \in \Sigma$ is obtained by concatenating the bit labels of the edges on the path from the root to the leaf σ. Prove:

- The encoding produces by a code tree T is a prefix code.
- Every prefix code f is produced by a code tree T.

Proof. **Proof of the first statement.** Suppose that the codeword of σ_1 is a prefix of the codeword of σ_2, σ_1 must be an ancestor of σ_2. However, this is impossible as σ_1 has to be a leaf of T.

Proof of the second statement. (Grow a trie.) Given any collection of prefix code $\{f(\sigma)|\sigma \in \Sigma\}$ for any alphabet Σ, we can always construct a code tree T in the following manner.

- Initially, T has only a single leaf. Let u be an traversal iterator on the tree T.
- For each letter $\sigma \in \Sigma$, we set u to the root of T, and repeat the following until u is a leaf node:
  - Let ℓ be the level of u. Descend to the left (resp., right) child v of u if the ℓ-th bit of $f(\sigma)$ is 0 (resp., 1). If v does not exist, create it in T, label its edge with u as 0 (resp., 1).
  - Set u to v.
- Mark the leaf node u with the letter σ.

The final T is a code tree that generates the prefix code. □

Problem 4. Let T be an optimal code tree on an alphabet Σ (i.e., T has the smallest average height among all the code trees on Σ). Prove: every internal node of T must have two children.

Proof. Let u be an internal node with a single child v. Let p be the parent of u. Remove u by making v a child of p and label the edge $\{p, v\}$ accordingly. If u is already the root, we make v the new root and delete u. We have then resulted in a code tree with a strictly smaller average height. □

Problem 5. Consider an alphabet Σ of $n \geq 3$ letters with their frequencies given. The prefix code we construct using Huffman's algorithm is binary because each letter $\sigma \in \Sigma$ is mapped to a string that consists of only 0's and 1's. Now, we want the code to be ternary, namely, each letter $\sigma \in \Sigma$ is mapped to a string that consists of three possible characters: 0, 1, or 2. As before, the code must be a prefix code. Assuming n to be an odd number, give an algorithm to find an encoding with the shortest average length.

Solution. We define a code tree on Σ as a ternary tree satisfying the conditions below:

- Every leaf node of T is labeled with a distinct letter in Σ; conversely, every letter in Σ is the label of a distinct leaf node in T;
- For every internal node of T, the edge connecting to the first, second, and third child is labeled with $0, 1, 2$, respectively.

For every letter $\sigma \in \Sigma$, the codeword for σ is obtained by concatenating the edge labels from the root of T to the leaf σ.

Construct a code tree as follows. Initially, for each character $\sigma \in \Sigma$, create a tree that contains only a single node u labeled with σ. Define the frequency of u to be $\text{freq}(\sigma)$. There are n nodes in total, and we put them into a set S. Repeat the following until $|S| = 1$:

- Remove from S the three roots u_1, u_2, and u_3 having the smallest frequencies.
- Create a tree with root u that has u_1, u_2, and u_3 as the child nodes. Define the frequency of u as the frequency sum of u_1, u_2, and u_3. Add u to S.

We note that n is odd and thus the process always terminates as the elements in S decreases by 2 per iteration. When $n = 3$, it is clear that the code tree generated is optimal. Assume that for any $n = k - 2$ where $k \geq 5$ the algorithm gives a correct construction, we show that the algorithm is correct for $n = k$.

Lemma 1. Every internal node in the optimal code tree must have three children.

Proof. (i) Suppose that there exists an internal node u that has only one child, we can always replace u with the child to obtain a smaller average length. (ii) Suppose that there exists an internal node u with two children u_1 and u_2, there must be another internal node u' with two children u'_1 and u'_2 due to the oddity of n. We first move u_1 to be the sibling of u'_1 and u'_2 to obtain a shorter average encoding length, now we are back to the case with one internal node. □

Lemma 2. Let σ_1, σ_2, and σ_3 be three letters in Σ with the lowest frequencies. There exists an optimal code tree where $\sigma_1, \sigma_2, \sigma_3$ have the same parent.

Proof. Without loss of generality, assume that $\text{freq}(\sigma_1) \leq \text{freq}(\sigma_2) \leq \text{freq}(\sigma_3)$. Let T be any optimal code tree. Let p be an arbitrary internal with the largest level in T.By Lemma 1, p must have three leaves. Let x, y, z be letters corresponding to the leaves such that $\text{freq}(x) \leq \text{freq}(y) \leq \text{freq}(z)$. Swap σ_1 with x, σ_2 with y, and σ_3 with z gives us a new code tree T'. Note that both $\sigma_1, \sigma_2, \sigma_3$ are children of p in T'. Since the average length of T' is at most that of T, T' is optimal. □

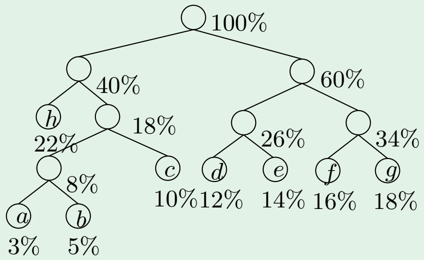
Construct a new alphabet Σ' by removing σ_1, σ_2, σ_3 and add back a letter σ^* with frequency $\text{freq}(\sigma_1) + \text{freq}(\sigma_2) + \text{freq}(\sigma_3)$. Let T' be the tree obtained by removing leaves σ_1, σ_2 and σ_3 from T. T' is a code tree on Σ'. Observe that the average height of T is the sum of average height of T', and the frequency of $\sigma_1, \sigma_2, \sigma_3$.

Let T'_{huff} be the tree obtained by removing the leaves $\sigma_1, \sigma_2, \sigma_3$ from T_{huff}. T'_{huff} is also a code tree on Σ', and the average height of T_{huff} is the sum of average height of T'_{huff}, and the frequency of $\sigma_1, \sigma_2, \sigma_3$.

Since T'_{huff} is optimal on Σ', the average height of T'_{huff} should be no larger than the average height of T'. Hence, the average height of T_{huff} should also be no larger than the average height of T.

Problem 6. Consider the alphabet Σ with letters a, b, c, d, e, f, g, h whose frequencies are 3%, 5%, 10%, 12%, 14% 16%, 18%, and 22%, respectively. Use Huffman's algorithm to find a prefix code on Σ that has the smallest average length.

Solution.



$a = 0100, b = 0101, c = 011, d = 100, e = 101, f = 110, g = 111, h = 00.$

Problem 7. Consider an alphabet Σ that contains n letters with their frequencies given, where Prove: the prefix code constructed using Huffman's algorithm has an average length of at most $\lceil \log_2 n \rceil$.

Proof. A complete binary tree with n leaves produces an encoding for n codewords. Furthermore, this code tree has a height of $\lceil \log_2 n \rceil$. Therefore, the average length

$$\sum_{\sigma \in \Sigma : |\Sigma| = n} \text{freq}(\sigma_i) \cdot \text{len}(\sigma_i) \leq \lceil \log_2 n \rceil \cdot \sum_{\sigma \in \Sigma : |\Sigma| = n} \text{freq}(\sigma_i) = \lceil \log_2 n \rceil \cdot 1 = \lceil \log_2 n \rceil.$$

Since the prefix code generated by Huffman's algorithm must be the optimal, therefore it has a average length no longer than any other prefix encoding of Σ, and now we clearly see by the above that it is bounded by $\lceil \log_2 n \rceil$. $\square$

Problem 8. Describe how to implement Huffman's algorithm to ensure a worst-case time complexity of $O(n \log n)$, where n is the size of the alphabet Σ.

Solution. Create an array for storing the children (at most 2) for each node $v \in \Sigma$. As a Huffman tree generates a binary tree with n leaves, we would result in a tree with at most $2n-1$ nodes.

Also, we maintain a data structure S for (i) insertion of an element e_i with weight w_i and (ii) popping of an element with the minimum weight within the structure, with query time $O(\log|S|)$ associated with the size of structure, $|S|$. This can be done using a heap.

Initially, create n disjoint leaves with no child which corresponds to each $\sigma \in \Sigma$. Insert the n nodes (with index $i = 1, \dots, n$) to S with weights equal to their frequencies. Set i to n. Then, repeat the following until $|S| = 1$:

- Remove from S two nodes u_1 and u_2 with the smallest frequencies in $O(\log|S|)$ time.
- Increment the index i by 1. Create a node v with index i, and set the children of v to u_1 and u_2.
- Add v to S with weight being the frequency sum of u_1 and u_2.

The codeword of every $\sigma \in \Sigma$ can be obtained by a single depth-first traversal of a tree, in $O(n)$ time, by maintaining a string for characters that are on the current path.

Since the elements in S is at most the number of leaves, n. The running time of S for each operation is $O(\log n)$. As there is at most $n-1$ iterations, the total time complexity is $O(n \log n)$.

Problem 9. Consider the alphabet $\Sigma = 1, 2, \dots, n$ for some integer $n \geq 1$. Suppose that the frequency of i is strictly higher than the frequency of $i+1$, for any $i \in [1, n-1]$. Prove: in an optimal prefix code, for any $i \in [1, n-1]$, the codeword of i cannot be longer than that of $i+1$.

Proof. The optimal prefix code consists of codewords that minimizes the average length

$$\sum_{\sigma \in \Sigma} \mathrm{freq}(\sigma) \cdot \mathrm{len}(\sigma).$$

Suppose that the codeword of i is longer than that of $i+1$, then exchanging the codeword of i and j results in a prefix code that has a smaller average length, namely

$$\mathrm{freq}(\sigma_i) \cdot \mathrm{len}(\sigma_i) + \mathrm{freq}(\sigma_{i+1}) \cdot \mathrm{len}(\sigma_{i+1}) > \mathrm{freq}(\sigma_i) \cdot \mathrm{len}(\sigma_{i+1}) + \mathrm{freq}(\sigma_{i+1}) \cdot \mathrm{len}(\sigma_i),$$

due to $\mathrm{freq}(\sigma_i) > \mathrm{freq}(\sigma_{i+1})$ and $\mathrm{len}(\sigma_i) < \mathrm{len}(\sigma_{i+1})$, which is a contradiction. $\square$

Problem 10. Consider an alphabet Σ with n letters, all of which have exactly the same frequency. The value of n is a power of 2. If we use Huffman's algorithm to generate the codewords for all the letters in Σ, how many bits are there in the shortest codeword?

Solution. $\log n$ bits. Since all letters have the same frequency, we denote this frequency as s.

Lemma. Before merging any node with frequency $2s$, all node should have frequency at least $2s$.

Proof. Note that n is even and $s > 0$. Suppose the otherwise that there is a node with frequency $2s$ merges with another with frequency less than $2s$, since $2s > s$, this must indicate that there is only *one* node with frequency s left. However, this is impossible - we know that no nodes with frequencies larger than $2s$ can be merged, and the sum of frequency of all nodes is an even multiple of s.

This allows us to define a *round* for the process of merging all nodes with same frequencies. After the first *round*, we are left with $n/2$ nodes with frequency $2s$, allowing us to apply the lemma again.

Eventually, we will construct a code tree, where the root has frequency $2^{\log_2 n} s = ns = 1$. We should also be able to observe that the height of the tree is associated with the number of rounds, which is $\log_2 n$.