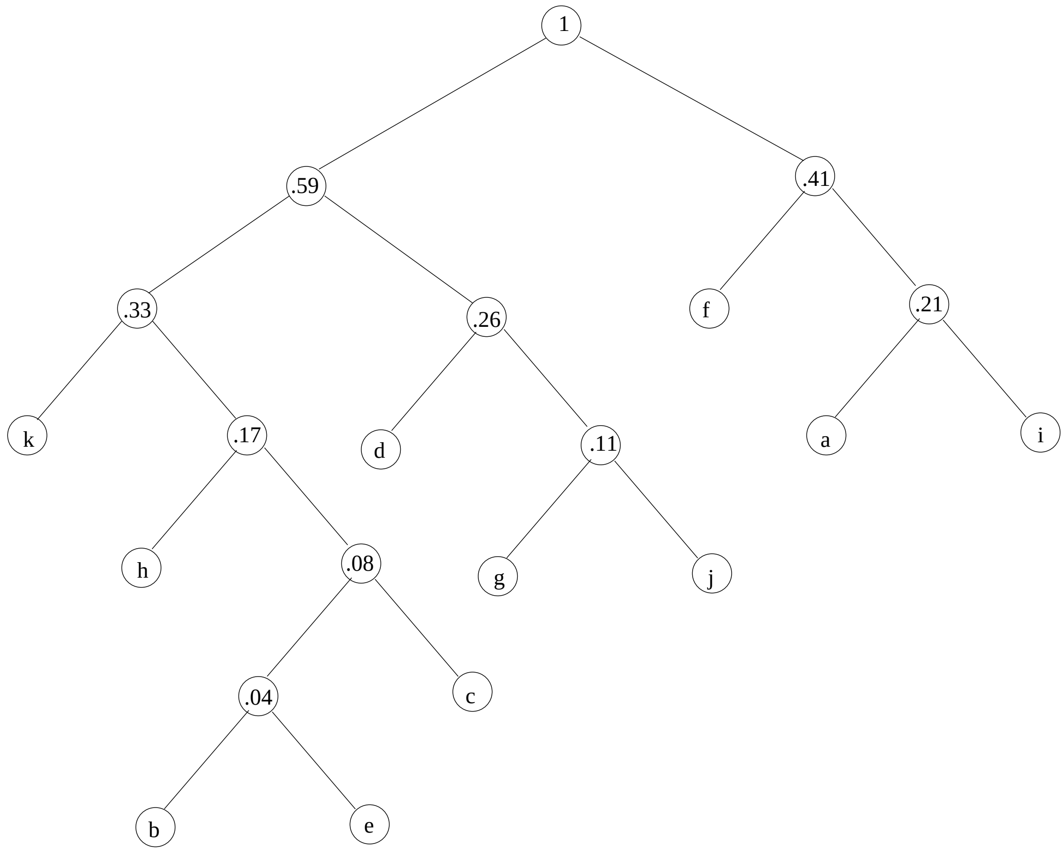
22C:44 Homework 8 Solution

1. The binary tree that represents the Huffman codes for the input is shown below.



The Huffman codes themselves can be read off from this tree and are given in the table below.

a	110
b	001100
c	00111
d	010
e	001101
f	10
g	0110
h	0010
i	111
j	0111
k	000

The average number of bits per symbol is $43/11 = 3.90909$.

2. The key idea is to choose the next stop greedily by choosing the stop that is farthest from among all stops that are within 50 miles of the current stop. If we let `currentStop` and `nextStop` denote the current and the next stops respectively, then the "greedy choice" transalates to choosing `nextStop` such that $A[\texttt{nextStop}] - A[\texttt{currentStop}] \leq 50$ and $A[\texttt{nextStop}+1] - A[\texttt{currentStop}] > 50$.

```
GumpStops(A[1..n] ) {
      A[0] = 0;
      currentStop = nextStop = 0;
      stops = empty set;
      while (nextStop < n) do
            if (A[nextStop+1] - A[currentStop] > 50) then {
                  stops = stops ∪ {nextStop};
                  currentStop = nextStop;
            }
            else nextStop++;
      return stops;
}
```

Two comments about the algorithm are in order: (i) The end point of the route will not be reported as a stop by this algorithm and F. Gump should have enough sense to stop and eat a box of choclates there as well. (ii) It is assumed that there is no portion of the route longer than 50 miles that is deprieved of stores that sell choclates.

Correctness of this algorithm is established using the following two claims.

Claim 1: There is an optimal solution to the problem in which the first stop is i, where $i = \max\{j \mid A[j] \leq 50\}$.
Proof: Suppose there is no such optimal solution. Let J^* be some optimal solution and let j be the first stop in J^*. Clearly, $j \leq i$ since the first stop in any solution has to be within 50 miles of the start-point. Replacing j by i in J^* gives us yet another optimal solution contradicting our assumption that there is no optimal solution containing i.

Claim 2: Let i be the first greedy choice, that is, $i = \max\{j \mid A[j] \leq 50\}$. Let J be an optimal solution to Gump's problem in which he starts at stop i rather than at the start point of the route. In other words, let J be an optimal solution to the problem with input array of distances B, where $B[j] = A[j+i]$ for all $j = 1, 2, \ldots, n-i$. Then $J \cup \{i\}$ is an optimal solution to the orginal problem containing the stop i.
Proof: Let J' be an optimal solution to the original problem, containing stop i. Then $J' - \{i\}$ is a solution to the problem with input B. Therefore, $|J| \leq |J' - \{i\}|$ implying that $|J| + 1 \leq |J'|$ and therefore implying that $|J \cup \{i\}| \leq |J'|$. This makes $J \cup \{i\}$ an optimal solution to the original problem containing i.

Using induction along with Claims 1 and 2 we immediately see the correctness of the algorithm.

3. For any positive integer k, call the fraction $1/k$ an *Egyptian fraction*. The key idea here is to choose the next Egyptian fraction $1/k$ greedily by making it the largest Egyptian fraction

no greater than m/n. In other words, $1/k$ is the largest such fraction satisfying $1/k \leq m/n$. Therefore, k is the smallest positive integer satisfying $k \geq n/m$. By definition of the ceiling function, this means that $k = \lceil m/n \rceil$.

The pseudocode for the algorithm is given below:

```
EgyptianNumber(m, n) {
      solution = empty set;
      while (m > 0) do {
            k = ⌈ m/n ⌉;
            solution = solution ∪ {k};
            m = mk - n;
            n = nk;
      }
      return solution;
}
```

To see that the algorithm eventually terminates note that since $1/k$ is the largest such fraction no greater than m/n, we have

$$\frac{1}{k} \leq \frac{m}{n} < \frac{1}{(k-1)}.$$

This simplifies to

$$0 \leq mk - n < m.$$

At each stage in the algorithm the old dnominmator m is replaced by a *smaller non-negative* denominator $mk - n$. Therefore the denominator eventually shrinks to 0 and the algorithm terminates. Furthermore, since the denominator decreases by 1 in each execution of the while-loop and it can only decrease m times, the while-loop executes at most m times. The amount of work inside the while-loop takes $O(1)$ time and therefore the running time of the algorithm is $O(m)$.

To show that the algorithm is correct, it is sufficient to show that it terminates, because each fraction chosen by the algorithm is an Egyptian fraction and when the algorithm terminates the chosen Egyptian fractions add up to m/n.

4. Let $A_{n \times n} = (m_{ij})$ be an adjacency matrix. If $m_{ij} = 1$ then there is an edge (i, j) in the graph and this means that i is not a sink. If $m_{ij} = 0$ then the edge (i, j) is not present in the graph and therefore j is not a sink. Testing the value of m_{ij} can be thought of as a comparison between two potential sinks i and j which eliminates one of them, while the other continues to be a potential sink. This suggests the following "generic" algorithm to solve the problem.

   Start with the pool of all potential sinks. This is of course, all of the n vertices in the graph. Repeatedly make a comparison between a pair of potential sinks, eliminating a vertex with each comparison. At the end of $n - 1$ comparisons $(n - 1)$ vertices have been eliminated and we have one vertex, say i, left as a potential sink. To determine if i is really a sink we check row i and column i in A and if row i has only 0's and column i has only 1's (excepting at m_{ii}) then i is a sink.