1. (15 pts total) Acme Corp. now wants you to help them develop a new email application, with a user-specific "blacklist" of n messages that are spam. When a new message arrives in a user's mail queue, the app needs to test whether the message is in the user's blacklist. (If so, then it will mark it as spam.) However, space is at a premium and you don't want to literally store all n spam messages in memory (assume each message is many bits in length), so instead you plan to use a *Bloom filter* with k hash functions. Assume each hash function can map a string of arbitrary length to a uniformly random bit in its range. Show your work.

   (a) (5 pts) For a false positive rate of no more than $f = 1/100$, what is the minimum number of bits of space needed to guarantee that you can test in $O(1)$ time whether a new message is in the blacklist?
   Hint: Recall that $1 - x \approx \mathrm{e}^{-x}$ when $x \ll 1$.

   (b) (3 pts) Assume that you use $k = \Theta(1)$ hash functions. Using asymptotic notation, how many bits do you need if you want the probability of false positives to be no more than $f = 1/n$?

   (c) (3 pts) Assume that you use $k = \Theta(\lg n)$ hash functions. Using asymptotic notation, how many bits do you need if you want the probability of false positives to be no more than $f = 1/n$? Explain why this answer differs from that of 1b.

   (d) (4 pts) Briefly describe a trivial algorithm a spammer could use to defeat this approach to spam filtering. Remember to explain how you define "defeat."

2. (16 pts total) Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a *tiling cover* if the intervals in Y cover the intervals in X, that is, every real value contained within some interval in X is contained in some interval Y. The *size* of a tiling cover is just the number of intervals. The minimum cover $Y_{\min}$ is the tiling cover with the smallest possible size.
   For the following, assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X.

   (a) (1 pts) Identify the input X that yields the largest possible $Y_{\min}$. Explain mathematically why no other input yields as large a $Y_{\min}$.

   (b) (1 pt) Identify the input X that yields the smallest possible $Y_{\min}$. Explain mathematically why no other input yields as small a $Y_{\min}$.

   (c) (4 pts) Under what conditions will a tile $x \in X$ be absent from $Y_{\min}$?

   (d) (5 pts) Describe in words and give pseudo-code for a *greedy* algorithm to compute the smallest Y in $O(n \log n)$ time.
   Hint: Try starting with the left-most tile.

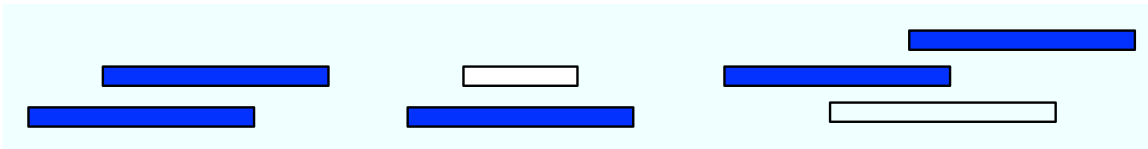

Figure 1: A set of intervals X and a tiling cover Y shown in blue.

(e) (2 pts) Explain why your algorithms runs in $O(n \log n)$ time.

(f) (5 pts) Prove that your algorithm (i) covers each component and (ii) produces the minimal cover for each component. Explain why these conditions are necessary and sufficient for the *correctness* of the algorithm.

3. (9 pts total) When representing a directed graph G in the form given, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degree? Express your answers in asymptotic notation and justify your claim.

   (a) (3 pts) An *edge list* representation.

   (b) (3 pts) An *adjacency list* representation.

   (c) (3 pts) An *adjacency matrix* representation.

4. (10 pts total) A graph (V, E) is bipartite iff the vertices V can be partitioned into two subsets L and R, such that every edge has one end in L and the other in R.

   (a) (5 pts) Prove that every tree is a bipartite graph.
   Hint: Assume a tree is not bipartite and think about odd-length cycles.

   (b) (5 pts) Adapt an algorithm described in class so that it will determine whether a given undirected graph is bipartite. Give and justify its running time.

5. (10 pts) Let $G = (E, V)$ denote a directed multigraph. A "simple" graph is $G' = (V, E')$, such that E' consists of the edges in E where (i) every directed multi-edge, e.g., $\{(u, v), (u, v)\}$, has been replaced by a pair of directed edges $\{(u, v), (v, u)\}$ and (ii) all self-loops (u, u) have been removed. Assume that G is given in an *adjacency list* format and that G' is in the same format.

   (a) (5 pts) Explain why an algorithm based on BFS or DFS for converting a multigraph into a simple graph will fail on some inputs. Include an example G that yields the bad behavior.

(b) (5 pts) Describe in words and provide pseudo-code for an algorithm that takes $O(V+E)$-time to convert G into G'.
Hint: Be sure your solution does not assume adjacencies `Adj[u]` are ordered in any particular way.

6. (13 pts total) When an *adjacency matrix* representation is used, most graph algorithms take $\Omega(V^2)$ time, but there are some exceptions.

   (a) (10 pts) Show that determining whether a directed graph G contains a *universal sink*, which is defined as a vertex with in-degree $|V|-1$ (i.e., every other vertex points to this one) and out-degree 0, can be determined in time $O(V)$, given an adjacency matrix for G.
   Hint: To *show* this, describe the algorithm, provide pseudocode, and prove that it yields the correct answer (yes or no) on every type of input. It will be helpful to identify and analyze the *boundary* cases, i.e., the cases where it is the most difficult to distinguish a "yes" from a "no."

   (b) (3 pts) How much space does this algorithm take?

7. (5 pts) Given an example of a directed graph $G=(V,E)$, a source vertex $s \in V$ and a set of tree edges $E_\pi \subseteq E$ such that for each vertex $v \in V$, the unique path in the graph (V, E_π) from s to v is a shortest path in G, yet the set of edges E_π cannot be produced by running a breadth-first search on G, no matter how the vertices are ordered in each adjacency list. Include a paragraph explaining why your example works.

8. (10 pts) Implement a Huffman encoder using a priority queue data structure. The deliverable here is a function that takes as input a string and returns as output both the binary string that represents its Huffman encoding *and* the Huffman encoding tree. Break ties uniformly at random.

9. (15 pts) Use your implementation from 8. to conduct these numerical experiments.

   (a) (7 pts) Below is the text of a famous poem by Robert Frost. Convert it to lowercase ASCII letters (`a - z`), keep the punctuation marks and the space character, and replace the carriage returns with a single space. This produces a string σ that is $\ell = 761$ symbols long, which are drawn from an alphabet Σ with $|\Sigma| = 31$ symbols (24 alphabetical characters, 6 punctuation marks and 1 space character).

      i. (1 pt) A plaintext ASCII character normally takes 8 bits of space. How many bits does σ require when encoded in ASCII?

ii. (1 pt) The theoretical lower limit for encoding is given by the *entropy* of the frequency of the symbols in the string:

$$H = -\sum_{i=1}^{|\Sigma|} \left(\frac{f_i}{\ell}\right) \log_2 \left(\frac{f_i}{\ell}\right) \quad ,$$

where f_i is the frequency of the ith symbol of the alphabet Σ and ℓ is the length of σ. Because we take $\log_2$, H has units of "bits." What is the theoretical lower limit for the number of bits required to encode σ?

iii. (5 pts) Encode σ using your Huffman encoder and report the number of bits in the encoded string. Compare this to your theoretical calculation above.

The Road Not Taken by Robert Frost

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I-
I took the one less traveled by,
And that has made all the difference.

(b) (8 pts) Show that the asymptotic running time of your Huffman encoder is $O(n \log n)$, where $n = |\Sigma|$ is the size of the input alphabet Σ. The deliverable here is a single figure (like the one below) showing how the number of atomic operations T grows as a function of n. Include a $O(n \log n)$ trend line that tracks your results. Label your axes and trend line. Include a clear and concise description (1-2 paragraphs) of exactly how you ran your experiment.

No credit will be given if you don't label your axes and trend line.

Hint 1: You will need to implement measurement code within your Huffman encoder that counts the number of atomic operations it performs. Here, atomic operations are accessing an element in the priority queue, encoding tree or input.

Hint 2: You'll also need to write a function that generates a sufficiently long random input message with an alphabet containing n symbols. For this experiment, choose any symbol frequencies you like, e.g., $f_i = c/\ell$ for some constant c.

Hint 3: To get a large value of n, you'll need to represent distinct alphabet symbols using multiple ASCII characters. Remember that with a k-bit binary string, you can create 2^k distinct "symbols." To separate the symbols, use a delimiter character that you ignore when you read in the string.

Hint 4: As in PS1, because the inputs are random, the measured running time is a random variable. Thus, for a given value of n, average T over multiple independent measurements in order to produce a smooth trend line.

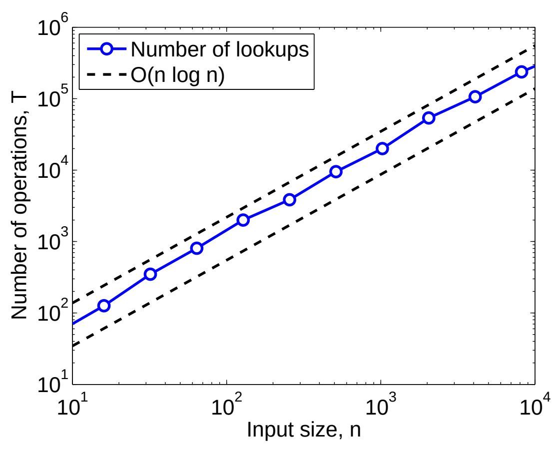

Figure 2: An example of what your Huffman results should look like.