Homework 5
Due: Oct. 27, 2015 at 6:00 PM (early)
Oct. 30, 2015 at 6:00 PM (on time)
Nov 1, 2015 at 6:00 PM (late)

This is a partner homework as usual: work on all problems together; come to all office hours together; you are responsible for everything you and your partner submit and it is an academic code violation to submit something that is not yours. Check for your partner in the Google doc that was sent out. **If you don’t have a partner, please let us know.**

Greedy algorithms

Recall the generic strategy for proving the correctness of a greedy algorithm:

1. Our algorithm makes a series of choices, $c_1, c_2, \ldots, c_k$.
2. Assume for the sake of contradiction that the result of these choices is not optimal.
3. Let $c_i$ be the first choice incompatible with any optimal solution, namely, if choices $c_1, c_2, \ldots, c_i$ are made, there is no way to complete this sequence of choices into an optimal solution. Such an $i$ exists by our assumption.
4. Let $Opt$ be an optimal solution that does use choices $c_1, c_2, \ldots, c_{i-1}$; such an $Opt$ exists because of the word “first” in the above definition of $i$.
5. Now show (this is the meat of the proof!) that there is a way to “swap” choice $c_i$ into $Opt$ without hurting $Opt$.
6. Thus we have contradicted our assumption that there is no way to use choices $c_1, \ldots, c_i$ in an optimal solution, letting us conclude that our greedy algorithm is in fact optimal.

Problem 1

1. (12 points) After discovering his sense of taste while in human morph, Ax the Andalite attends a chili festival, where many chefs are giving tutorial sessions, possibly at overlapping times, and he wants to attend as many as possible (without arriving late or leaving early from any tutorial). If the $i$th tutorial goes from time $s_i$ to time $t_i$, evaluate each of the three scheduling strategies below, and either prove it is correct or find a counterexample to show that it is not optimal.

   (a) Greedily pick tutorials in increasing order of start time. (Sort tutorials in increasing order of start time $s_i$, and repeatedly run the following process: pick the first tutorial in the list, then delete all overlapping tutorials.)

   (b) Greedily pick tutorials in decreasing order of start time.

   (c) Greedily pick tutorials in increasing order of length ($t_i - s_i$).

2. (8 points) K.A. Applegate does not like parties, but unfortunately for her, everyone expects to see her at her new book release party. She knows that the $i$th guest at her party will arrive at time $s_i$ and will leave at time $t_i$. She wants to briefly make an appearance at the party,
as few times as possible, but so that everyone at the party sees her at least once. Design a
greedy algorithm for her to figure out what times she should appear at the party, and prove
its correctness.

Problem 2

K.A. Applegate has a long book tour ahead of her and wants to stay in comfortable hotels along her
route each night, but can only drive 200 miles in a day. Fortunately, she has a guidebook charting
the locations of all the hotels along her route, and thus knows the distance between any two hotels.
(K.A. already knows which route she is taking; she only has to choose where along her route to
stay each night.)

1. (8 points) Find a greedy algorithm for K.A. Applegate to compute how to finish her tour in
the fewest days possible. Points on this problem will only be given for the proof that your
algorithm is optimal; more points will be given for simpler and clearer proofs. (Note that the
“swapping” step of the standard greedy proof approach is a bit unwieldy here, so you may
want to prove the optimality of your algorithm more directly.)

2. (3 points) After studying your algorithm for the previous problem, K.A. realizes that, actually,
the different hotels cost different amounts, and what she actually wants to do is minimize
the total cost of her tour. (Luckily, her guidebook also lists the cost of each hotel.) She
thinks of the following greedy algorithm: wherever she is, for each hotel within a day’s drive
of her (200 miles), she computes the “cost per mile” of staying there, dividing its cost by the
amount of progress she would make by staying there; given this list of costs, she then chooses
to spend her next night at the hotel with the best cost per mile. Demonstrate for her that
being greedy can be costly, that is, describe an example where her algorithm gives suboptimal
performance.

3. (4 points) Find a dynamic programming algorithm for K.A’s problem. Include an explanation
of the meaning of any tables you ask her to construct, and how to generate them. Prove the
correctness of your algorithm (briefly—you should already have practice at getting to the
heart of dynamic programming proofs of correctness in few words.)

Problem 3

(5 points) As opposed to Huffman encoding, here is a proposal for a slightly different algorithm
to arrange a tree to encode a probability distribution (review the problems in the next section,
or Dasgupta et al. to remind yourself of the framework). Construct a “very imbalanced” tree by
repeating the following process: choose the domain element \( p(i) \) of smallest probability that hasn’t
been chosen so far, and join it to the tree \( A \) constructed so far by making \( p(i) \) and \( A \) the two
children of a new node. This will lead to a tree where the \( i \)th–most–frequent element will end up
at depth \( i \) (though the two least frequent elements end up at the same depth).

As precisely as possible, describe why the following sketch of the proof of correctness of Huffman
encoding breaks down for this modified (and bad!) algorithm. This involves going over the original
proof until you understand how it works.
Consider, for the sake of contradiction, the first time our greedy algorithm merges an element $p(i)$ and a tree $A$ to create a tree “$p(i) A$” that never appears in any optimal solution. Since this is the first time, there is an optimal solution $Opt$ that contains tree $A$ and element $p(i)$ but where they are not siblings (not children of the same parent). Move $p(i)$ and $A$ in $Opt$ by swapping them with two siblings at the lowest level of the tree. This can only improve the expected bits per symbol of our code (expected depth in the tree of an element chosen from our probability distribution), because $p(i)$ and $A$ between them contain all the $i$th smallest elements from the probability distribution, and moving them down by swapping them with larger elements can only help. So thus there is an optimal tree containing “$p(i) A$”, violating our assumption, and letting us conclude that our algorithm indeed finds an optimal tree.

### Information compression

**Problem 4**

Because so many computational processes are limited by memory or transfer costs, one of the key computational tasks is *information compression*, where data is *encoded* into a more compact form. In some sense, compression is a way of spending processing resources to save memory or bandwidth resources. Compression algorithms are usually categorized as either *lossless* or *lossy*, where lossless algorithms let one exactly recover the original data (for example “zipping” a file), and therefore are used invisibly in many different hardware and software components (some hard drives, network protocols, document file formats, operating system services); lossy algorithms on the other hand can compress to much smaller sizes because they lose information (examples include most image, audio, and video compression schemes); there is no generic lossy analog of “zipping” a file, because which parts of the information one is willing to lose entirely depend on the meaning of the information.

In this problem, we will examine the lossless compression scheme known as *Huffman coding*.

As covered in class, we will be encoding symbols from an *alphabet* $\Sigma$, from which symbols are drawn according to a probability distribution $p$, where $p(i)$ denotes the probability of drawing the $i$th symbol. We represent an encoding scheme for the alphabet $\Sigma$ by a binary tree, where each leaf is labeled by a distinct element of the alphabet, and the symbol $\sigma$ at a given leaf is represented in the encoding by a sequence of 0’s and 1’s describing the path from the root to the leaf, where 0 means “go left from the parent” and 1 means “go right from the parent”. (See chapter 5.2 of Dasgupta et al. for details.)

1. (2 points) Under the ASCII encoding of text, changing a single bit will end up changing only one letter of the decoded output. Find a binary encoding tree and a message of length at least 5 where changing a single bit in the encoded message will change all the decoded symbols. (This is a general phenomenon: compressed information is more “brittle”; perhaps you have seen compressed video where several seconds have the wrong color and strange ghost-like artifacts—this might have been caused by the corruption of a single bit.)

2. (4 points) Given a probability distribution $p$ each of whose probabilities is a (negative) power of 2 (and which sum to 1 since $p$ is a probability distribution), show that there exists a binary encoding tree where each node $i$ is at depth $|\log_2 p(i)|$. 


3. Huffman codes are constructed as binary trees via the following method. For each symbol $\sigma$, construct the trivial binary tree with just one node $\sigma$, labeled by the probability $p(\sigma)$. Make a list storing each of these objects (binary tree labeled by a probability). Until the list has just one element, repeat the following: find the two elements in the list with the smallest probabilities and “merge” them into one element by 1) replacing their probabilities with the sum of the probabilities, and 2) merging the two binary trees into one by creating a root node that has these two trees as children. The resulting tree represents the Huffman code for the pair $(\Sigma, p)$.

(a) (2 points) Suppose I have a lucky die that, instead of returning $\{1, 2, 3, 4, 5, 6\}$ with equal probability, returns them with probabilities $\{0.01, 0.03, 0.06, 0.3, 0.1, 0.5\}$. What is the Huffman encoding scheme? (As usual, feel free to include a diagram if it would help, and feel free to draw your diagram by hand.)

(b) (1 points) Encode the lucky sequence 665666366466 using this code. How many bits-per-symbol does this use?

(c) (1 point) Encode the unlucky sequence 1112153 using this code. How many bits-per-symbol does this use?

(d) (2 points) What is the expected number of bits–per–symbol of the Huffman code, for sequences produced by rolling your die? (Recall the general expression for expected bits–per–symbol of a code represented as a binary tree.)

(e) (2 points) The entropy of a distribution $p$ is defined as $\sum_{\sigma \in \Sigma} p(\sigma) |\log p(\sigma)|$, where the base of the logarithm is 2 when we are working with bits. The entropy is a lower bound on the efficiency of any coding scheme. What is the entropy of $p$ and how much short of this goal is Huffman coding? Entropy is the universal measure of information, and is our benchmark for whether a compression algorithm is successful.

4. (2 points) How would you describe Huffman encoding as a “greedy algorithm”? (Your explanation should relate the choices made by the greedy algorithm to the goals of the greedy algorithm, and make clear why these choices are a natural way to seek to achieve the goals.)

5. (4 points) We have seen in class that Huffman coding is optimal; but now we would like to know how good it is, how the expected bits-per-symbol compares with the entropy lower bound of $\sum_{\sigma \in \Sigma} p(\sigma) |\log p(\sigma)|$. This is hard to analyze for Huffman coding, but since Huffman coding is the optimal scheme, we can instead analyze any other prefix-free coding scheme, and know that Huffman coding will perform at least as well. Wikipedia suggests \url{http://en.wikipedia.org/wiki/Shannon-Fano_coding}, which is a different attempt at a greedy algorithm for this problem. (Shannon-Fano coding is not a successful greedy algorithm; it often produces codes that are suboptimal; the fact that Claude Shannon, the founder of information theory, is credited with this algorithm is a sign of how hard designing correct greedy algorithms can be).

The second sentence of this article is the crucial point: for each symbol $\sigma$, its depth in the Shannon-Fano coding tree is at most 1 more than $|\log p(\sigma)|$; thus the average length of codewords is at most 1 more than the (weighted) average of this, which is the entropy $\sum_{\sigma \in \Sigma} p(\sigma) |\log p(\sigma)|$. This implies that Shannon-Fano encoding has expected bits-per-symbol within 1 bit of the entropy bound. And Huffman encoding is at least as good, proving that Huffman encoding guarantees expected bits-per-symbol at most $1 + \sum_{\sigma \in \Sigma} p(\sigma) |\log p(\sigma)|$, which is exactly the kind of bound we wanted.
Unfortunately, Wikipedia is wrong! (Or, at least, inconsistent.) Show that the second sentence of the Wikipedia article is not true for Shannon-Fano encoding, as described in the article. Namely, find an alphabet \( \Sigma \) and a probability distribution \( p \) over the alphabet that provides a counterexample.

6. (10 points) To make the Wikipedia article true, we need to modify the Shannon-Fano algorithm. Specifically, replace step 3 of the algorithm with “Let \( s \) be the smallest element of the list, and \( T \) be the total. Divide the list into two parts at the first point which makes the total of the left part greater than \( (T - s)/2 \).” (Remember that the list is sorted in decreasing order, in step 2.)

Prove that this correction makes the second sentence of the Wikipedia article true. This can be stated equivalently as: if a symbol \( \sigma \) ends up at depth \( d \) then its probability must be at most \( 2/2^d \).

**Crucial hint:** for any internal node \( v \) (not a leaf), define the function \( f(v) \) to be the total of all the probabilities of all descendants of \( v \) except the descendant with smallest probability. Show that if \( v \) is a child of \( p \) then \( f(v) \leq \frac{1}{2} f(p) \). Apply this bound repeatedly (induction!) to show that “things deep in the tree are small”, arguing that if a symbol \( \sigma \) ends up at depth \( d \) then its probability must be at most \( 2/2^d \).

7. We now know that the expected number of bits-per-symbol for the optimum prefix-free encoding is between the entropy \( \sum_{\sigma \in \Sigma} p(\sigma) |\log p(\sigma)| \) and the entropy plus 1. But the gap is still a bit disturbing. In fact, Shannon’s source coding theorem provides the answer: one can encode with bits-per-symbol arbitrarily close to the entropy if instead of encoding each symbol separately, we are allowed to encode in larger blocks.

In some sense the worst case for Huffman encoding is with a heavily biased coin: consider the alphabet \{a, b\} where the probability of \( a \) is 0.9 and the probability of \( b \) is 0.1.

(a) (1 point) What is the entropy of this distribution?

(b) (1 point) What is the expected bits-per-symbol for Huffman encoding here?

(c) (3 points) Instead, try considering longer blocks of \( a \)'s and \( b \)'s at a time: redefine the alphabet to consist of all \( \text{triples} \) of \( a \)'s and \( b \)'s, namely an alphabet of size \( 2^3 = 8 \). Write down the probabilities of each of these 8 possibilities (using the probabilities for \( a \) and \( b \) from above), and derive the Huffman encoding for these “supersymbols”. Compare the performance here to the previous case and to the entropy bound.

Comments: extending the approach of the last part is infeasible because the size of the Huffman encoding tree will grow exponentially. For the particular case of the last part, run-length encoding is an easy and effective scheme (which is used successfully in JPEGs, among many other places). More generally, variants of the Lempel-Ziv algorithm are very popular in practice, which aggregate “supersymbols” on the fly in a way that naturally adapts to the data being compressed. Look these up on Wikipedia for more information.

**Problem 5**

This problem presents several scenarios where you must design a good scheme to encode data.
1. (5 points) You want to encode 64 bit integers between 1 and \(2^{64} - 1\), where you want to encode smaller numbers with fewer bits. Can you think of a scheme to encode a sequence of 64 bit integers so that when encoding a number \(i\), if \(i < 2^k\) for some integer \(k\), then you use at most \(k + 5\) bits? (Hint: try \(k + 6\) first.)

2. (5 points) Consider the probability distribution \(P(A) = 0.99, P(B) = .001, P(C) = .001, P(D) = .005, P(E) = .002, P(F) = .001\). Construct an explicit (simple) scheme for compressing long sequences drawn from this distribution, that is within 10% of optimal: compute the bits per symbol of your scheme, and compare with the entropy of \(P\). (Hint: divide the sequence into sections of length 100 first.)

3. (5 points) You want to compress a sequence of 1,000,000 numbers each between 1 and 10,000, with the additional guarantee that: consecutive numbers in the sequence differ by no more than 15. How could you achieve within 10% of the optimal compression? (Do not try to compute the entropy of a random sequence like this, but you should be able to estimate it, by ignoring the bounds 1 and 10,000, and design a scheme accordingly.)

4. (5 points) K. A. Applegate is trying to compress her books, and notices that each of her chapters can be divided into happy and sad sections, which use rather different words. Let \(P\) be the probability distribution of words in a happy section, and let \(Q\) be the distribution of words in a sad section. Suppose both \(P\) and \(Q\) have entropy 8 bits per word. Suppose you want to compress a chapter with 2,000 words, and with 5 happy and 5 sad sections. Describe a compression scheme for such chapters, evaluate its bits per symbol, and argue why it is close to optimal.

5. (5 points) A sequence of 1,000 people are asked which is their favorite Animorphs book. Their answers are stored in a file, with each line having either the form:

   Person 592's favorite book is "The Invasion".

   or the form

   Person 987 thinks Animorphs are no fun at all.

Half of the respondents like the books, and half do not. The lines are sorted by person number (1 through 1,000). Find a compression scheme for this file. You may assume that each of the 54 Animorphs books (listed at https://en.wikipedia.org/wiki/List_of_Animorphs_books) are equally popular. Describe your compression scheme, the total number of bits it will take, and explain why it should be close to optimal.