Exercises 22 : data compression

Questions

1) One natural approach to finding an optimal prefix code, suggested in the lecture, is as follows: given a sample space S and a probability function p(), partition S into two sets (events) whose probabilities are each as close to .5 as possible. Then, repeat recursively. That is, for each of these two events (assuming they consist of more than one element of S), partition the event into two sets whose probabilities are as close as possible to p(E)/2. The base case is that you have just one element.
   For each partition into two sets, define a left (0) and right (1) branch in the code tree.

   Suppose you wanted to implement the above algorithm for creating a code. What techniques have we seen that would be used for this?

2) Consider the example shown in the lecture: p(A) = .32, p(B) = .25, p(C) = .2, p(D) = .18, p(E) = .05. The code that was given in the lecture was c(A) = 00, c(B) = 10, c(C) = 110, c(D) = 01, c(E) = 111. What is the average code length?

3) Take the same probabilities as in the previous question and consider swapping the codewords for C and D. This gives the code, which c(A) = 00, c(B) = 10, c(C) =01, c(D) = 110, c(E) = 111. What is the average code length?

4) Huffman coding requires that we maintain a forest of trees. At each step we merge the two trees that have the lowest probabilities (as represented by the probability at the root). What would be a suitable data structure for this algorithm? What would be the O() runtime of the Huffman algorithm?

5) One of the claims made in motivating Huffman coding is that, for any optimal prefix code, if $p(a) < p(b)$ then $\lambda(a) >= \lambda(b)$. Prove it. (This is really just to check that you understand the definition.)

6) In the lecture, I claimed that for an optimal prefix code, the two least probable elements of S have the same codeword length. The sketch of the proof that I gave was by contradiction. We suppose $\lambda(a) < \lambda(b)$. Fill in the proof.

7) Why do we ignore case $p0 < ½$ in runlength coding ?

8) See Exercises 1 from my Data Compression course. Do Questions 2, 4, 5 only.

Answers

1) Given an event E (starting with the original set S), we want to partition E into two sets each of whose probabilities are as close to p(E)/2 as possible. This is just an instance of subset sum problem. The weights are the probabilities, and the bound W is p(E)/2.

   (more detail added April 8) For example, we start with the whole set S and set W = 0.5. We run subset sum and try to find a subset of S that has probability as close to (or equal to) W = 0.5 as possible. Suppose we find a subset with total probability 0.48. Then its complement has probability 0.52. We now have two sets E1 and E2. For each of these we have the same problem defined in the question, namely partition each of them into two sets whose probabilities are as close to equal as possible. For the set E1 we have p(E1) = 0.48 and so we would define a subset sum problem with W = 0.24. For E2, we have p(E2) = 0.52 so we would have a subset sum problem with W = 0.26. Etc, etc.

2) The average code length is 2*.32 + 2*.25 + .2*3 + .18*2 + .05*3 = 2.25.

3) The average code length is 2*.32 + 2*.25 + .2*2 + .18*3 + .05*3 = 2.23 which is smaller than in the previous question.

4) Use a priority queue, where the priority is the probability of the root node of a tree. Each node of the priority queue stores a priority (the probability) and a reference to a tree. At each step of the code algorithm, remove the two elements with lowest probability (highest priority) from the priority queue, merge them into a new element (merge the trees and add the probabilities) and put the element back into the priority queue.

   The algorithm would run in time O(n log n) since you iterate n-1 times and each time you may need to do O(log n) work when you insert the merged node into the priority queue.

5) Suppose c() is an optimal prefix code. Then the average code length is Σ $\lambda(s)$ $p(s)$ where the sum is over s. Now consider another code in which we swap the codewords of a and b. What happens to the the average code length ? The average code length of the new code is equal to the old average code length, plus the difference that is due to the swap, namely: $\lambda(a)\ p(b) + \lambda(b)\ p(a) - (\lambda(a)\ p(a) + \lambda(b)\ p(b))$ where the $\lambda(a)$ and $\lambda(b)$ refer to the original code. The difference term can be rewritten $(\lambda(a) - \lambda(b))(p(b) – p(a))$. If the original code was an optimal prefix code, then the difference term cannot be less than zero, since the swap would yield a code with shorter average length. Since $p(b) > p(a)$ by assumption, it follows that $\lambda(a)$ must be greater than or equal to $\lambda(b)$.

6) If we're assuming an optimal prefix code, then a's parent must have two children since otherwise we could shorten the code by getting rid of the edge from a's parent to a. Now, what can we saw about a's sibling? It cannot be an internal node, since then this internal node would have two children but these children would have greater probability than a (since by assumption a and b are least probable) and they would have a codeword that is longer than a (which is impossible). Thus, the sibling of a is a leaf. But again, that's impossible since this leaf would have greater probability

than b (by the assumption that a and b are least probable) and the sibling has longer codeword than b.

7) In the case that successes are less more common than failures, we tend to get more 1's than 0s. If successes are MUCH more common than failures, then we'll get long runs of 1 separated (typically) by just one 0. If the goal is compression, then what we really want here is to replace long runs of 1's followed by a 0 by a codeword of that event. So, its really the same as the original problem but now it's a run of successes followed by a failure rather than the other way around.