

Assignment #4 (practice only – not graded)

1: Huffman and Arithmetic Coding.

- (a) Suppose we have six letters A, B, C, D, E, F as our symbols. Their letter-probability pairs are given as follows:

$$(A, \frac{9}{32}), (B, \frac{3}{32}), (C, \frac{5}{64}), (D, \frac{5}{16}), (E, \frac{3}{16}), (F, \frac{3}{64})$$

- (i) Construct a Huffman tree for the above letter-probability tuples. Please label the edges of the tree so that each left edge receives the value “0” and each right edge receives the value “1”. Use the convention that for any node in the tree, its left child is labeled with a smaller occurrence probability than its right child. In case of a tie in probabilities, the letter occurring earlier in the alphabet is selected first.
- (ii) Encode the message “FACE” with the Huffman tree you have built.
- (iii) Calculate the average number of bits per letter required in general for the Huffman code derived in (i), assuming that letters appear according to the above probabilities. Also calculate the alphabet’s entropy limit. Is the average number of bits per letter the same as the alphabet’s entropy limit? Explain.
- (b) Recall that in arithmetic coding, messages are represented by an interval of real numbers between 0 and 1.
- (i) With the letter-probability tuples given in 1(a), encode the message “FACE” with Arithmetic coding. As your solution, show the resulting intervals.
- (ii) How would you exactly represent the message “FACE” under arithmetic coding? How many bits are needed? (There are several reasonable choices of representations that you could use here, but try to follow what we did in class. Use fractional binary numbers.) Justify your choice.
- (c) Suppose we do not know the occurrence probabilities of the letters given in 1(a) beforehand. In that case, we might use adaptive Huffman Coding for data compression. Assume the occurrence frequencies of the letters A to H above are initially set to 1. Show how the frequencies and the probabilities of the alphabet are updated when the message “FECEB” is encoded with Adaptive Huffman Coding. Show the Huffman tree after every step, and the resulting encoded bit stream for “FECEB”.

2: Index Compression. Suppose the format of the inverted list postings is of the form:

$$\langle d, f_{d,t}, [p_{1,d,t}, \dots, p_{f_{d,t},d,t}] \rangle$$

where d is the document ID, $f_{d,t}$ is the frequency f of term t in the document d , and $p_{1,d,t}, \dots, p_{f_{d,t},d,t}$ are the $f_{d,t}$ positions where the term t appears in the document d .

Below is an inverted list for the term “Pangolin”:

$\langle 7, 2, [56, 301] \rangle \langle 33, 3, [7, 147, 224] \rangle \langle 59, 1, [79] \rangle \langle 87, 3, [25, 133, 445] \rangle$

Thus, the word “Pangolin” occurs two times in document 7, in positions 56 and 301, three times in document 33, in positions 7, 147, and 224, and so on. Assume that the total number of documents N is 100, and that documents 7, 33, 59, 87 have lengths (in words) of 400, 300, 200, and 600, respectively.

- (a) Rewrite the above inverted list so that the document IDs are stored with offsets instead of their raw values. (Show the list of document ID, frequencies, and position offsets.)
- (b) Encode the document ID offsets using the following methods:
 - (i) Rice coding
 - (ii) Golomb coding
 - (iii) Gamma coding
 - (iv) Delta coding
 - (v) Variable-Byte coding
- (c) Encode the entire inverted list using what you think is the best method for each of the components, that is, document IDs, frequencies, and positions. There are many possible solutions here, but please justify your design in a few sentences.

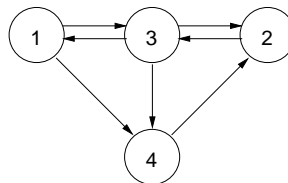
3: Pagerank. Recall that Google’s Pagerank is defined as:

$$PR(A) = \frac{(1 - d)}{n} + d \times \left(\frac{PR(T_1)}{C(T_1)} + \dots + \frac{PR(T_n)}{C(T_n)} \right)$$

where $PR(A)$ is the normalized Pagerank of page A , the T_i are the pages pointing to page A , $C(T_i)$ is the out-degree of T_i , d is the damping factor which can be set between 0 and 1, and n is the total number of pages.

Assume that each node initially gets a Pagerank of $1/n$, so that throughout the process, the sum of the Pageranks of the n pages is always 1.

Assume a damping factor $d = 0.85$ and the following graph G_1 with 4 nodes:



- (a) Show the Pagerank of every page after the first and the second iterations of the iterative process (with random jumps as defined above).
- (b) Show the Pagerank transitional matrix A . Your matrix should incorporate the random jump.
- (c) Will the pagerank of this graph eventually converge? Why or why not?

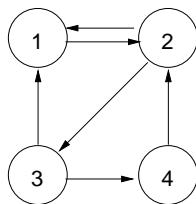
4: Hubs and Authorities.

- (a) The terms “hub” and “authority” are defined in a mutually recursive way: a good hub is a page that points to many good authorities, and a good authority is page that is pointed to by many good hubs. Each node v has a non-negative “hub score” $h(v)$ and a non-negative “authority score” $a(v)$. According to the definitions of hub and authority, we have the following equations.

$$a(v) = \sum_{u:(u,v) \in E} h(u)$$

$$h(u) = \sum_{v:(v,u) \in E} a(v)$$

You are given with the following directed graph G resulting from a query to a text-based search engine. (Note that this is already the expanded graph that includes the search results as well as the neighbors of the search results.)



- (i) Construct a bipartite graph of hubs and authorities from the directed graph G . (Note that G itself is not bipartite.)
- (ii) Initially, set $h(v) = 0.25$ for all v . Compute the authority scores of all authorities from the initial hub scores.
- (iii) Compute the hub scores of all hubs from the authority scores found in (ii), then normalize the computed hub scores (so they sum up to 1).
- (b) The formulas for computing hub and authority scores given in 4(a) can be represented in matrix form. Given a matrix A with one row and one column for each page, let entry $A_{i,j} = 1$ if page i points to page j , and 0 if not. (Note that A^T (the transpose of A) is similar to the matrix used for computing Pagerank (without random jumps), but A^T has 1's where the Pagerank matrix has fractions.)

Let \vec{h} and \vec{a} be two column vectors whose i th element corresponds to the hub score and the authority score of page i , respectively, and λ and μ are scaling factors to be determined later. Then,

$$\vec{h} = \lambda A \vec{a}$$

$$\vec{a} = \mu A^T \vec{h}$$

- (i) Rewrite \vec{h} and \vec{a} in terms of themselves.
- (ii) What are A , A^T , AA^T , and $A^T A$ for the graph G in 4(a)?
- (iii) Initialize \vec{h} by setting each value to $1/n = 0.25$. Now compute \vec{a} and \vec{h} after one, two, and three iterations, and also the converged values after a larger number of iterations. Make sure to normalize after each iteration. (It is recommended to write a small program to perform this computation.)