Solution sketch
IN3120/IN4120
Final exam, fall 2020

Link analysis [20%]

a) See also the 2018 exam.

$$\begin{bmatrix} [1/8; 7/24; 7/24; 7/24] \\ [1/8\ ; 1/8; 3/8; 3/8] \\ [5/8; 1/8; 1/8; 1/8] \\ [3/8; 1/8; 3/8; 1/8] \end{bmatrix} = \begin{bmatrix} [0.1250; 0.2917; 0.2917; 0.2917] \\ [0.1250; 0.1250; 0.3750; 0.3750] \\ [0.6250; 0.1250; 0.1250; 0.1250] \\ [0.3750; 0.1250; 0.3750; 0.1250] \end{bmatrix}$$

Comment: We just ask for the matrix, but in the face of a wrong answer it is a lot easier to give partial credit if the student makes it clear how they arrived at their answer. Both P and P^T give full marks.

b) By iterating matrix multiplication until convergence (i.e, the dynamical systems approach), or by computing the principal left eigenvector directly by solving the characteristic equation (i.e., the linear algebra approach). See here for details. A surfer simulation approach would work, but be very inefficient.

c) We could then for some graphs have dead ends, or disconnected islands. Why is this a problem? Because the Markov chain would then no longer be ergodic, and the desired steady-state solution might no longer exist.

Edit distance [20%]

a) Look at the entry for (gobb[e], gobbleh[o]) and the next one down, it skips directly from 3 to 5. That's an impossible gap, since we have unit edit costs. With unit edit costs, neighbouring entries in an edit table can maximally differ by 1 per the edit equations, so this must be an error. This is the only error in the table, everything else (including the final edit distance of 3) is correct.
Comment: Solvable not by having to do the tedious computation of the table yourself, but by looking for table properties you know must be true but that are violated.

b) See, e.g., this paper distributed as part of the course. Organize your dictionary in a trie. Exploit that strings that share prefixes also share columns in the edit table up to the prefix length, so that when you go down a level in the trie you only have to recompute one column in the edit table. Abort traversing the branch you are on, and thus prune away all strings under that branch of the trie, as soon as the edit distance exceeds k.

Quiz-o-rama [20%]

a) ld$d

b) E.g., “moonbath” would generate a false positive.
c) $127 = 2^7 - 1$, $16383 = 2^{14} - 1$, 13.
   Comment: Since gaps cannot be 0 a plausible answer is also 128/16384/13.
d) Yes. A simple example is sufficient justification, but note that the log function is monotonically increasing and grows to infinity.
e) We can simply omit this term from the query and proceed, since its contribution to the dot product with any document vector will be zero.
   Comment: The question is arguably ambiguous, so solutions that discuss, e.g., smoothing could also trigger points.
f) John’s posting lists do not have a common ordering, so he’ll have to implement term-at-a-time scoring instead.
g) False. At the break-even point we have P = R by definition, so $F_1 = P = R$.
h) A “find pages like this one” search ignores the query (alpha), no negative judgments are used (gamma), and we have a single positive judgment (beta). So alpha = gamma = 0 and beta > 0.
i) False. This is a strange, made-up claim. A trivial counter-example will suffice.
j) True. See the 2015 exam for a short proof. Also, the textbook states this as a fact in an exercise. Simply referring to either will suffice.

Large-scale systems [30%]

Comment: Alternative answers might also trigger points.

a) Bloom filters to the rescue! Basically, keep a Bloom filter per partition. The Bloom filter for a partition is constructed so that all user identifiers hosted on that particular partition are added to the Bloom filter. We check if the incoming query (user identifier) is a member of that partition’s Bloom filter. If it is not, then we can avoid dispatching the query to that partition since we can be sure that we’d get an empty result set. If it is, then we dispatch the query to that partition. This might result in an empty result set from that partition, but it probably won’t. We can control the false positive rate by selecting suitable parameters for the Bloom filters (number of bits and number of hashes.) See here for a description of how Bloom filters work. If we need to delete entries, either Bloom filter variants that support deletion will have to be used or some external bookkeeping mechanism is needed.
b) The strategy (that is covered by this course) that seems appropriate here is that of dynamic indexing and logarithmic merging, as described here: We keep around multiple indices of varying size, one per “level” where typically one level is double the size of the previous level, and new documents are added to the smallest (i.e., lowest-level) index which typically fits in memory and is quick and easy to reindex on demand with low latency as new documents are added. As this lowest-level index grows, documents spill over to the next level and cause that to be reindexed, and so on. To search, we dispatch the query to all levels at the same time and merge the results. Deletions are handled by invalidation bit vectors instead of physically purging the deleted documents from the index, which complicates giving a correct number of hits for a term. Corpus-wide statistics (used for,

e.g., relevancy and spell checking purposes) become complicated to keep track of since we have multiple indices, merges can become heavy and cause delays, etc. Lots of other complications possible.

c) (i) Note that *algo1* is basically a buggy version of the naive Bayes testing phase for text classification from Figure 13.2 in the textbook. The algorithm should be summing the log probabilities (which could/should be precomputed) to compute the scores instead of multiplying them, the smoothing logic is off, and the return statement won't work as intended. As it stands, *algo1* would always return the lexicographically largest class name instead of the class having the highest score. (ii) Note that *algo2* is basically Figure 14.4 in the textbook for text classification, collapsed into a single phase. It is very inefficient since it repeats the training phase every time, and should be decomposed accordingly. As it stands it would slow down the system. It also returns argmax instead of argmin and returns an integer value instead of the class key. (iii) Note that *algo3* is basically a buggy version of Figure 6.14 in the textbook, for computing vector space scores using the cosine metric. The return statement is buggy, the indexing won't work, the length normalization is off, and the result set has not been sorted. This cannot run, and if it did it would not have returned the *n* entries with the highest score.

Evaluation [10%]

a) See, e.g., here or here or here. Possible discussion points include binary versus graded item relevance, how they account for the positions in the result set, and more.

b) Possible issues to discuss include: Human judgments can be costly to produce, relevance judgments get stale as new content becomes available or time passes, query coverage will typically be very low, humans might tend to disagree and/or change their minds. The kappa statistic can be used to quantify such disagreements. It's appealing to consider proxies for explicit judgments, e.g., by looking at clickthrough logs and other user behavioral data.