1.a.

{ A, B } appears in 4 baskets. The absolute support is 4.

1.b.

There are 6 baskets total. The relative support is 4/6 = 2/3

1.c

Baskets that have { A } = 6

Baskets that have { A, B } = 4

Confidence = 4/6 = 2/3

2.a.

I = 7, J = 8

(7-1)(20 – (7/2)) + 8 – 7 = 100

2.b.

Tabular approach beats triangular when at most 33% (1/3) of the data has a non-zero count. Since only 10% has a non-zero count, triangular matrix will perform better.

3.a.

|  |  |  |
| --- | --- | --- |
| Item | Absolute support | Notes |
| 1 | 4 | 1 appears 4 times in all of the baskets |
| 2 | 6 | 2 appears 6 times |
| 3 | 8 |  |
| 4 | 8 |  |
| 5 | 6 |  |
| 6 | 4 |  |

Since all of the values appear at least 4 times, they all meet the support threshold.

3.b.

Finding all the pairs and their support values

|  |  |  |  |
| --- | --- | --- | --- |
| Basket | Pairs | Support of each Pair | Notes |
| 1 = {1,2,3} | { (1,2), (1,3), (2,3) } | (2,3,3) | (1,2) occurs 2 times in all the buckets, (1,3) occurs 3 times |
| 2 = {2,3,4} | { (2,4), (3,4) } | (3, 4) | We don’t include (2,3) in the pairs since it is in the previous line |
| 3 = { 3,4,5} | { (3,5), (4,5) } | (4, 3) |  |
| 4 = {4,5,6} | { (4,6), (5,6) } | (3, 2) |  |
| 5 = { 1,3,5 } | { (1,5) } | (1) |  |
| 6 = { 2,4,6 } | { (2,6) } | (1) |  |
| 7 = { 1,3,4 } | { (1,4) } | (2) |  |
| 8 = { 2,4,5 } | { (2,5) } | (2) |  |
| 9 = { 3,5,6 } | { (3,6) } | (2) |  |
| 10 = {1,2,4} | No new |  |  |
| 11 = {2,3,5} | No new |  |  |
| 12 = {3,4,6} | No new |  |  |

Reorganizing the table because I do not like that format for calculating the hash values

|  |  |
| --- | --- |
| Pairs | Bucket Value |
| (1,2) | 1\*2 mod 11 = 2 |
| (1,3) | 1\*3 mod 11 = 3 |
| (1,4) | 1\*4 mod 11 = 4 |
| (1,5) | 1\*5 mod 11 = 5 |
| (2,3) | 2\*3 mod 11 = 6 |
| (2,4) | 2\*4 mod 11 = 8 |
| (2,5) | 2\*5 mod 11 = 10 |
| (2,6) | 2\*6 mod 11 = 1 |
| (3,4) | 3\*4 mod 11 = 1 |
| (3,5) | 3\*5 mod 11 = 4 |
| (3,6) | 3\*6 mod 11 = 7 |
| (4,5) | 4\*5 mod 11 = 9 |
| (4,6) | 4\*6 mod 11 = 2 |
| (5,6) | 5\*6 mod 11 = 8 |

|  |  |
| --- | --- |
| Bucket | Pairs |
| 0 |  |
| 1 | (2,6), (3,4) |
| 2 | (1,2), (4,6) |
| 3 | (1,3) |
| 4 | (1,4) , (3,5) |
| 5 | (1,5) |
| 6 | (2,3) |
| 7 | (3,6) |
| 8 | (2,4), (5,6) |
| 9 | (4,5) |
| 10 | (2,5) |

3.c.

None of the buckets meet the support threshold of 4, so none of them are frequent?

3.d.

In order to be a candidate pair, the pair must be made up of frequent values, and also the pair must hash to a frequent bucket. I am not sure I am understanding how to tell if a bucket is frequent correctly, since it seems odd that none of the buckets are frequent in this example. Assuming I did this correctly, I don’t think any pairs are counted on to the second pass.

4.

This article is exploring the different way to detect digital copying in its various forms. The article begins by defining digital copying and narrows in on the concept of plagiarism. Plagiarism poses an interesting challenge because it implies that people are attempting to make it difficult to detect. In sections 2.1, three main criteria are outlined for what makes copy detection successful. These criteria are whitespace insensitivity, noise suppression, and position independence. These three points are what lead to the need for more robust methods of selecting fingerprints.

The processes of generating fingerprints starts by cleaning up the text. The text is stripped of capitalization, punctuation, and whitespace. Once the data has become a big, long string of characters, it is then chunked into k-grams. The K value can be modified depending on the desired implementation. Each k gram is then hashed to generate the hash value. Then, from this sequence of hashes, a select few are chosen to be the fingerprint.

There are several variables in this setup. The K value, the hash function, and the method of selecting the hash values can be changed, and they lead to different methods.

The Karp-Rabin String Matching technique relies on the idea of a rolling hash function to save resources when hashing a string with a large K value. This has a weakness because the hash function doesn’t generate a wide range of outputs.

Section 3 of the document discusses Winnowing. In order to avoid noise, the noise-threshold is introduced. This makes sure there aren’t arbitrary fingerprints compared for things that don’t matter. In contrast, the guarantee-threshold makes sure the algorithm isn’t looking for something that is unlikely to be found. In my brain, I think of a person scanning papers for plagiarism. If the criteria for cheating was to find a whole paragraph copied character for character, it would be very easy for students to avoid being caught. The guarantee-threshold is where the professor would look for smaller pieces of language. However, if the professor swung the other direction, and looked for super common words, everyone would be accused of cheating. Obviously, these algorithms aren’t looking for words or phrases, they are looking for fingerprints, but the concept is the same. If we make the criteria too specific, it will be inaccurate. If we make it not specific enough, it will be inaccurate.

Winnowing protects the guarantee by creating windows where a hash value must be selected from. When the hash is being chosen, only the values from that window are considered. A weakness of winnowing is that low-entropy strings generate the same hash values. This leads to robust winnowing, where ties between windows are broken by leaning towards a hash that has already been chosen in an adjacent window.

Moss is an application that hits close to home, since I know lots of professors at WSU use it.