CS2241 Assignment 1

Nico Fidalgo

March 26, 2025

1 Problem 1: PageRank and HITS Algorithm Analysis

1.1 Problem Statement

We are given a directed graph represented by the following adjacency matrix.

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1}$$

We need to calculate the PageRank scores and Hub and Authority scores. The interpretation of the adjacency matrix is:

- Node a has outgoing edges to nodes c, d, and f
- Node b has outgoing edges to nodes c and f
- Node c has outgoing edges to nodes d and f
- Node d has outgoing edges to nodes b and c
- Node e has an outgoing edge to node a
- Node f has an outgoing edge to node e

1.2 PageRank Algorithm

PageRank was designed to model the behavior of a random surfer on the web. The algorithm assigns a score to each node in a directed graph based on its structural importance. The intuition is that a node is important if many important nodes point to it and the importance of a node is divided among its outgoing links.

$$\mathbf{p} = (1-d) \cdot \frac{\mathbf{1}}{n} + d \cdot M \cdot \mathbf{p} \tag{2}$$

where:

- d is the damping factor (typically 0.85)
- n is the number of nodes
- $\mathbf{1}$ is a vector of all 1's
- M is the transition matrix (columns sum to 1)

For each node i with outgoing links, we set:

$$M_{j,i} = \frac{1}{\text{out-degree}(i)} \tag{3}$$

if there is a link from node i to node j, and 0 otherwise. The algorithm for computing PageRank scores is:

Algorithm 1 PageRank Score Calculation

```
1: Initialize $\mathbf{p}^{(0)} = \frac{1}{n} \cdot \mathbf{1}$
2: for $t = 0, 1, 2, \ldots$ until convergence do
3:     $\mathbf{p}^{(t+1)} = (1-d) \cdot \frac{\mathbf{1}}{n} + d \cdot M \cdot \mathbf{p}^{(t)}$
4:     if $\|\mathbf{p}^{(t+1)} - \mathbf{p}^{(t)}\| <$ tolerance then
5:         break
6:     end if
7: end for
8: Normalize $\mathbf{p}$ to sum to 1
```

1.3 PageRank Implementation

I wrote a JavaScript program to implement the PageRank algorithm:

```javascript
import * as math from 'mathjs';

// Adjacency matrix from the problem
const A = [
    [0, 0, 1, 1, 0, 1],  // a -> *
    [0, 0, 1, 0, 0, 1],  // b -> *
    [0, 0, 0, 1, 0, 1],  // c -> *
    [0, 1, 1, 0, 0, 0],  // d -> *
    [1, 0, 0, 0, 0, 0],  // e -> *
    [0, 0, 0, 0, 1, 0]   // f -> *
];

// Number of nodes
const n = A.length;

// Out-degrees
```

```javascript
const out_degrees = A.map(row => row.reduce((sum, val) => sum + val
    , 0));
console.log("Out-degrees:", out_degrees);

// Create transition matrix (column-stochastic)
const M = Array(n).fill().map(() => Array(n).fill(0));
for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
        if (A[i][j] > 0) {
            M[j][i] = 1.0 / out_degrees[i];
        }
    }
}

// PageRank parameters
const d = 0.85;  // Damping factor
const max_iter = 100;
const tol = 1e-6;

// Initialize PageRank
let pr = Array(n).fill(1/n);

// Algorithm iteration
for (let iter = 0; iter < max_iter; iter++) {
    // Calculate M * pr
    const M_pr = Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            M_pr[i] += M[i][j] * pr[j];
        }
    }

    // Calculate (1-d)/n + d * (M * pr)
    const pr_new = M_pr.map(val => (1-d)/n + d * val);

    // Check convergence
    const diff = math.norm(pr_new.map((val, idx) => val - pr[idx]))
    ;
    if (diff < tol) {
        pr = pr_new;
        console.log(`\nPageRank converged after ${iter+1}
    iterations.`);
        break;
    }

    pr = pr_new;
}

// Normalize to sum to 1
const pr_sum = pr.reduce((sum, val) => sum + val, 0);
pr = pr.map(val => val / pr_sum);
```

1.4 PageRank Results

After running the algorithm, I obtained the following PageRank scores:

Node	PageRank Score
a	0.186551
b	0.091079
c	0.182643
d	0.155479
e	0.190060
f	0.194188

Nodes f, e, and a have the highest PageRank scores, indicating they are structurally important in this network. Node b has the lowest score, suggesting it's less central in the graph's link structure.

1.5 HITS Algorithm

The Hyperlink-Induced Topic Search (HITS) algorithm identifies two types of important nodes in a directed graph: hubs which are nodes that point to many good authorities, and authorities which are nodes that are pointed to by many good hubs. The hub ($\mathbf{h}$) and authority ($\mathbf{a}$) vectors satisfy:

$$\mathbf{a} = A^T\mathbf{h} \tag{4}$$
$$\mathbf{h} = A\mathbf{a} \tag{5}$$

where A is the adjacency matrix of the graph. The algorithm for computing HITS scores is:

Algorithm 2 HITS Score Calculation

```
1: Initialize $\mathbf{h}^{(0)} = \mathbf{1}$ and $\mathbf{a}^{(0)} = \mathbf{1}$
2: for $t = 0, 1, 2, \ldots$ until convergence do
3:     $\mathbf{a}^{(t+1)} = A^T\mathbf{h}^{(t)}$
4:     Normalize $\mathbf{a}^{(t+1)}$
5:     $\mathbf{h}^{(t+1)} = A\mathbf{a}^{(t+1)}$
6:     Normalize $\mathbf{h}^{(t+1)}$
7:     if $\|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}\| <$ tolerance and $\|\mathbf{h}^{(t+1)} - \mathbf{h}^{(t)}\| <$ tolerance then
8:         break
9:     end if
10: end for
```

1.6 HITS Implementation

Again, I wrote a JavaScript program to implement the HITS algorithm:

```
import * as math from 'mathjs';

// Adjacency matrix from the problem
const A = [
    [0, 0, 1, 1, 0, 1],  // a -> *
    [0, 0, 1, 0, 0, 1],  // b -> *
    [0, 0, 0, 1, 0, 1],  // c -> *
    [0, 1, 1, 0, 0, 0],  // d -> *
    [1, 0, 0, 0, 0, 0],  // e -> *
    [0, 0, 0, 0, 1, 0]   // f -> *
];

// Number of nodes
const n = A.length;

// HITS parameters
const max_iter = 100;
const tol = 1e-6;

// Initialize hub and authority scores
let hub = Array(n).fill(1);
let auth = Array(n).fill(1);

// Compute transpose of A
const AT = Array(n).fill().map(() => Array(n).fill(0));
for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
        AT[i][j] = A[j][i];
    }
}

// HITS iteration
for (let iter = 0; iter < max_iter; iter++) {
    // Update authority scores: a = A^T * h
    const auth_new = Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            auth_new[i] += AT[i][j] * hub[j];
        }
    }

    // Normalize authority scores
    const auth_norm = math.norm(auth_new);
    const auth_normalized = auth_new.map(val => val / auth_norm);

    // Update hub scores: h = A * a
    const hub_new = Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            hub_new[i] += A[i][j] * auth_normalized[j];
        }
    }

    // Normalize hub scores
    const hub_norm = math.norm(hub_new);
    const hub_normalized = hub_new.map(val => val / hub_norm);

    // Check convergence
    const auth_diff = math.norm(auth_normalized.map((val, idx) =>
    val - auth[idx]));
    const hub_diff = math.norm(hub_normalized.map((val, idx) => val
     - hub[idx]));

    if (auth_diff < tol && hub_diff < tol) {
        auth = auth_normalized;
        hub = hub_normalized;
        console.log(`HITS converged after ${iter+1} iterations.`);
        break;
    }

    auth = auth_normalized;
    hub = hub_normalized;
}
```

1.7 HITS Results

After running the HITS algorithm, I obtained the following scores:

Node	Hub Score	Authority Score
a	0.684439	0.000000
b	0.501536	0.113935
c	0.446890	0.590796
d	0.283360	0.454889
e	0.000000	0.000000
f	0.000000	0.656548

Nodes a, b, and c have high hub scores, indicating they are good at pointing to authority nodes. This makes sense as a and b point to multiple nodes including high authority nodes c and f. Nodes e and f have zero hub scores because they don't point to any nodes with high authority scores.

Nodes f and c have the highest authority scores, followed by node d. This means they are pointed to by good hub nodes. Again, this makes sense as node f and c are pointed to by nodes a and b which have high hub scores. Nodes a and e have zero authority scores because they aren't pointed to by any good hub nodes.

1.8 Assumptions

For PageRank scores:

- Used a damping factor of 0.85 (standard value)
- Defined convergence as when L2 norm difference $< 10^{-6}$

For HITS:

- Defined convergence as when L2 norm difference $< 10^{-6}$

2 Problem 2: Random Walk Methods for Hub and Authority Scores

2.1 Problem Statement

We need to prove that the Hub score for a page is proportional to the number of outlinks and that the Authority score is proportional to the number of inlinks when using the following random walk approach:

For Authority scores:

- From page p_1, follow back a random inlink to page p_2
- From p_2, follow forward a random outlink to page p_3
- The step takes us from p_1 to p_3

For Hub scores:

- From page p_1, follow forward a random outlink to page p_2
- From p_2, follow backward a random inlink to page p_3
- The step takes us from p_1 to p_3

We assume the Markov chains are finite, irreducible, and aperiodic, ensuring a unique stationary distribution.

2.2 Definitions

- $A[i, j] = 1$ if there's a link from page i to page j, 0 otherwise
- $\text{in}(j)$ = number of inlinks to page $j = \sum_i A[i, j]$
- $\text{out}(i)$ = number of outlinks from page $i = \sum_j A[i, j]$

2.3 Transition Probabilities

For Authority Random Walk:
When starting at page i:

- The probability of following a random inlink back to page k is $\frac{A[k,i]}{\text{in}(i)}$
- The probability of following a random outlink from k to j is $\frac{A[k,j]}{\text{out}(k)}$

Therefore, the transition probability from i to j is:

$$P_a(i, j) = \sum_k \frac{A[k, i]}{\text{in}(i)} \times \frac{A[k, j]}{\text{out}(k)} \tag{6}$$

For Hub Random Walk:
When starting at page i:

- The probability of following a random outlink to page k is $\frac{A[i,k]}{\text{out}(i)}$
- The probability of following a random inlink back from k to j is $\frac{A[j,k]}{\text{in}(k)}$

Therefore, the transition probability from i to j is:

$$P_h(i,j) = \sum_k \frac{A[i,k]}{\text{out}(i)} \times \frac{A[j,k]}{\text{in}(k)} \tag{7}$$

2.4 Authority Score Proof

Let's hypothesize that the stationary distribution $\pi_a(i)$ is proportional to $\text{in}(i)$, i.e., $\pi_a(i) = c \times \text{in}(i)$ for some constant c.

For this to be a stationary distribution, it must satisfy:

$$\pi_a(j) = \sum_i \pi_a(i) P_a(i,j) \tag{8}$$

Substituting our hypothesis:

$$\begin{align} \pi_a(j) &= \sum_i c \times \text{in}(i) \times \sum_k \frac{A[k,i]}{\text{in}(i)} \times \frac{A[k,j]}{\text{out}(k)} \tag{9} \\ &= c \times \sum_i \sum_k \frac{A[k,i] \times A[k,j]}{\text{out}(k)} \tag{10} \\ &= c \times \sum_k \frac{A[k,j]}{\text{out}(k)} \times \sum_i A[k,i] \tag{11} \end{align}$$

Since $\sum_i A[k,i] = \text{out}(k)$ (the number of outlinks from page k):

$$\begin{align} \pi_a(j) &= c \times \sum_k \frac{A[k,j]}{\text{out}(k)} \times \text{out}(k) \tag{12} \\ &= c \times \sum_k A[k,j] \tag{13} \\ &= c \times \text{in}(j) \tag{14} \end{align}$$

This confirms the hypothesis that the Authority score $\pi_a(j)$ is proportional to $\text{in}(j)$, the number of inlinks to page j.

2.5 Hub Score Proof

Similarly, hypothesize that the stationary distribution $\pi_h(i)$ is proportional to $\text{out}(i)$, i.e., $\pi_h(i) = d \times \text{out}(i)$ for some constant d.

For this to be a stationary distribution, it must satisfy:

$$\pi_h(j) = \sum_i \pi_h(i) P_h(i,j) \tag{15}$$

Substituting our hypothesis:

$$\begin{aligned}
\pi_h(j) &= \sum_i d \times \text{out}(i) \times \sum_k \frac{A[i,k]}{\text{out}(i)} \times \frac{A[j,k]}{\text{in}(k)} && (16)\\
&= d \times \sum_i \sum_k \frac{A[i,k] \times A[j,k]}{\text{in}(k)} && (17)\\
&= d \times \sum_k \frac{A[j,k]}{\text{in}(k)} \times \sum_i A[i,k] && (18)
\end{aligned}$$

Since $\sum_i A[i,k] = \text{in}(k)$ (the number of inlinks to page k):

$$\begin{aligned}
\pi_h(j) &= d \times \sum_k \frac{A[j,k]}{\text{in}(k)} \times \text{in}(k) && (19)\\
&= d \times \sum_k A[j,k] && (20)\\
&= d \times \text{out}(j) && (21)
\end{aligned}$$

This confirms the hypothesis that the Hub score $\pi_h(j)$ is proportional to $\text{out}(j)$, the number of outlinks from page j.

3 Problem 3: Reservoir Sampling

3.1 Single-Item Reservoir Sampling

We need to prove that the following algorithm maintains a uniform sample of one item among all items seen so far:

- When the first item appears, store it in memory
- When the k-th item appears, replace the current item with probability $1/k$

I will prove by induction that after processing k items, each item has exactly $1/k$ probability of being in memory. In the base case where $k = 1$, after seeing the 1st item, it is stored with probability 1. Since we've only seen one item, this gives us a uniform distribution. Beyond the base case, assume that after seeing $k-1$ items, each item has a probability of $1/(k-1)$ of being in memory. Now the k-th item arrives. For the k-th item, the probability it replaces the current item is $1/k$, therefore, $P(k\text{-th item in memory}) = 1/k$. For any previous item i (where $1 \leq i < k$):

$$P(i\text{-th item in memory after } k \text{ items}) = P(i\text{-th item was in memory}) \times P(\text{it remains in memory}) \tag{22}$$

$$= \frac{1}{k-1} \times \left(1 - \frac{1}{k}\right) \tag{23}$$

$$= \frac{1}{k-1} \times \frac{k-1}{k} \tag{24}$$

$$= \frac{1}{k} \tag{25}$$

Thus, after processing the k-th item, each of the k items seen so far has exactly $1/k$ probability of being in memory, which confirms the uniform distribution property.

3.2 Sampling s Items Without Replacement

Now we generalize to maintaining a uniform sample of s items without replacement. The algorithm is:

1. Store the first s items as they arrive
2. When the k-th item arrives (where $k > s$):
   - With probability s/k, include it in the sample
   - If including it, replace a randomly chosen item from the current sample

I need to prove that after seeing k items ($k \geq s$), each item has exactly s/k probability of being in the sample. In the base case, after seeing s items, all s items are in the sample with probability 1. Since $s/s = 1$, this is uniform. Beyond the base case, assume that after seeing $k-1$ items ($k-1 \geq s$), each has probability $s/(k-1)$ of being in the sample. Now the k-th item arrives. We need to show that after processing it, each of the k items has probability exactly s/k of being in the sample. For the k-th item, it's included with probability s/k, so $P(k\text{-th item in sample}) = s/k$. For any previous item i (where $1 \leq i < k$):

$$
\begin{aligned}
P(i\text{-th item remains in sample}) &= P(i\text{-th item was in sample}) \\
&\quad \times P(i\text{'s not replaced}) && (26)\\
&= \frac{s}{k-1} \times [1 - P(k\text{-th item is chosen}) \\
&\quad \times P(\text{item } i \text{ is replaced} \mid k\text{-th is chosen})] && (27)\\
&= \frac{s}{k-1} \times \left[1 - \frac{s}{k} \times \frac{1}{s}\right] && (28)\\
&= \frac{s}{k-1} \times \left[1 - \frac{1}{k}\right] && (29)\\
&= \frac{s}{k-1} \times \frac{k-1}{k} && (30)\\
&= \frac{s}{k} && (31)
\end{aligned}
$$

This confirms that after processing the k-th item, each of the k items has probability s/k of being in the sample, maintaining uniformity.

3.3 Implementation

To verify these theoretical results, I implemented both algorithms in JavaScript and ran simulations to check if the sampling is uniform.

```javascript
// Single-item reservoir sampling
function singleItemReservoirSampling(stream) {
    let sample = null;
    let count = 0;

    for (const item of stream) {
        count++;
        if (count === 1) {
            sample = item;
        } else {
            // Replace with probability 1/count
            if (Math.random() < 1/count) {
                sample = item;
            }
        }
    }

    return sample;
}

// Reservoir sampling of s items without replacement
function reservoirSampling(stream, s) {
    let sample = [];
    let count = 0;

    for (const item of stream) {
        count++;

        if (count <= s) {
            // Fill the reservoir with the first s items
            sample.push(item);
        } else {
            // Decide whether to include this item in the sample
            const j = Math.floor(Math.random() * count);
            if (j < s) {
                // Replace the randomly chosen item
                sample[j] = item;
            }
        }
    }

    return sample;
}

// Function to simulate multiple runs and track results
function runSimulations(numRuns, streamSize, sampleSize = 1) {
    const counts = {};

    // Initialize counts for each number
    for (let i = 1; i <= streamSize; i++) {
        counts[i] = 0;
    }

    for (let run = 0; run < numRuns; run++) {
        // Create a stream of numbers from 1 to streamSize
        const stream = Array.from({length: streamSize}, (_, i) => i
     + 1);

        let result;
        if (sampleSize === 1) {
            result = [singleItemReservoirSampling(stream)];
        } else {
            result = reservoirSampling(stream, sampleSize);
        }

        // Update counts
        result.forEach(item => {
            counts[item]++;
        });
    }

    // Convert to frequencies
    const frequencies = {};
    for (const [key, value] of Object.entries(counts)) {
        frequencies[key] = value / numRuns;
    }

    return frequencies;
}
```

3.4 Results

I ran 10,000 simulations with a stream of 10 items for both single-item sampling and sampling 3 items without replacement. The results confirm our theoretical analysis:

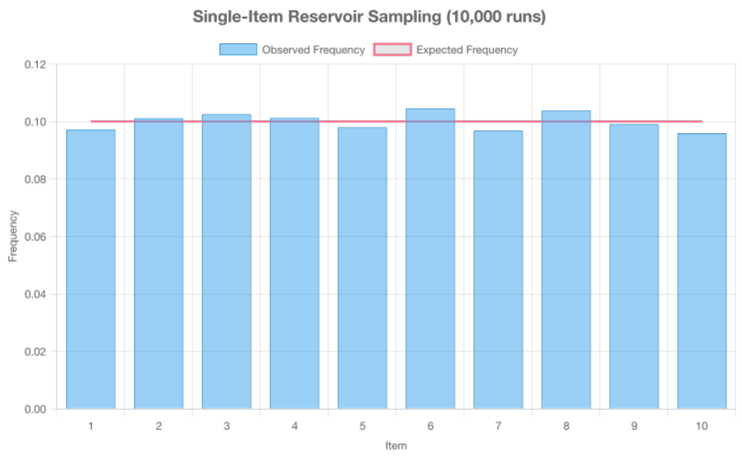

Figure 1: Frequency distribution of single-item reservoir sampling over 10,000 runs. Each item has approximately 0.1 probability of being selected, as expected.

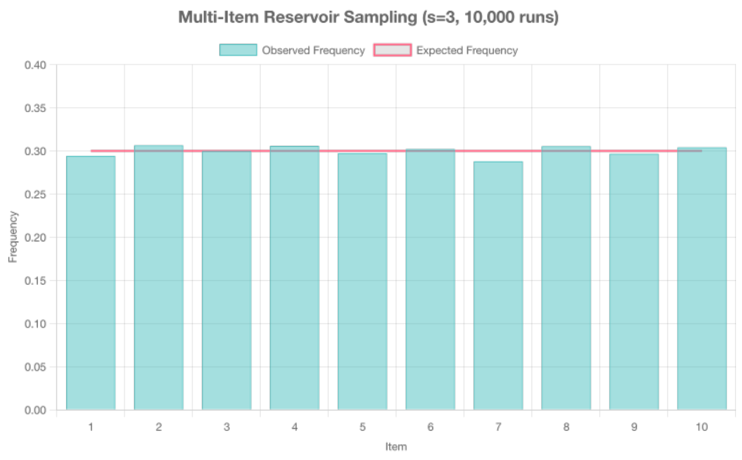

Figure 2: Frequency distribution of 3-item reservoir sampling over 10,000 runs. Each item has approximately 0.3 probability of being selected, as expected.

For the single-item sampling, the expected frequency is $1/10 = 0.1$ for each item. The empirical results in Figure 1 closely match this expectation, with minimal deviation from the expected value. For the 3-item sampling, the expected frequency is $3/10 = 0.3$ for each item. Again, the empirical results in Figure 2 closely match the expectation, confirming our theoretical analysis. These simulations confirm the theoretical correctness of both algorithms, showing that they indeed maintain uniform sampling of the stream at each step.

3.5 Why This Is Useful in Streaming Algorithms

Reservoir sampling is particularly valuable in streaming algorithms because it maintains a representative sample without needing knowledge of the total stream size, it uses constant memory as it stores only s items, it processes each item in constant time, and the sample remains uniform at each step of the process. This makes it ideal for applications like processing large data streams that don't fit in memory, online analytics where a representative sample is needed in real time, distributed systems where data is generated across multiple sources, and monitoring high-volume log data where sampling is necessary for performance.

4 Problem 4: Flipping a Coin Until First Head

4.1 Problem Statement

A fair coin is flipped until the first head occurs. Let X be the number of flips required. I must first find the entropy $H(X)$ in bits and then describe how to ask yes/no questions to determine X adn find the expected number of such questions and compare this result to $H(X)$.

4.2 Entropy Calculation

Since the coin is fair, the probability that $X = k$ (i.e., the first head occurs on the k-th flip) is

$$P(X = k) \ = \ \left(\frac{1}{2}\right)^k, \quad k = 1, 2, 3, \ldots$$

We use the fact that for a geometric random variable with parameter $p = \frac{1}{2}$ (counting the number of trials up to and including the first success), the distribution is:

$$P(X = k) = \frac{1}{2}\left(\frac{1}{2}\right)^{k-1} = \left(\frac{1}{2}\right)^k.$$

The entropy in bits is

$$H(X) \ = \ -\sum_{k=1}^{\infty} P(X = k) \, \log_2\big[P(X = k)\big].$$

Substitute $P(X = k) = (1/2)^k$, and note that $\log_2\big[(1/2)^k\big] = -k$. So

$$H(X) = -\sum_{k=1}^{\infty} \left(\frac{1}{2}\right)^k (-k) = \sum_{k=1}^{\infty} k \left(\frac{1}{2}\right)^k.$$

We now use the series identity:

$$\sum_{k=1}^{\infty} k\,x^k \;=\; \frac{x}{(1-x)^2}, \quad \text{for } -1 < x < 1.$$

With $x = \frac{1}{2}$, we get

$$\sum_{k=1}^{\infty} k \left(\frac{1}{2}\right)^k = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} \;=\; \frac{\frac{1}{2}}{(\frac{1}{2})^2} = \frac{\frac{1}{2}}{\frac{1}{4}} = 2.$$

Thus, the entropy $H(X) = 2$ bits.

4.3 Yes/No Questions and Comparison

To determine X from yes/no questions, we want a strategy that minimizes the expected number of questions. We can think of this as constructing a decision tree for the geometric distribution.

One possible line of reasoning is:

- First question: "Is $X = 1$?"
  - With probability 1/2, the answer is yes, and we are done with just 1 question.
  - With probability 1/2, the answer is no, in which case we continue to ask about $X = 2$, $X = 3$, etc.
- We can create a decision tree that accounts for the probabilities $(1/2)^k$ at each branch.

This geometric distribution achieves an expected number of questions exactly 2, matching the entropy $H(X) = 2$ bits. In other words, on average, we need 2 yes/no questions to determine X. This matches the lower bound given by $H(X)$, illustrating that our questioning strategy is asymptotically optimal.

Hence, the entropy $H(X)$ provides a fundamental limit on the average number of bits (yes/no questions) required, and our constructed question strategy can match that limit.

5 Problem 5: Arithmetic Coding

5.1 Problem Statement

Consider arithmetic coding for the string `abcbaab` given the probabilities:

$$P(a) = 0.2, \quad P(b) = 0.3, \quad P(c) = 0.5.$$

We must (1) show each step of idealized arithmetic coding (with real-number arithmetic) for the string `abcbaab`, and then (2) decode a sequence of length 10 for the real number 0.63215699 using the same model and intervals.

5.2 Encoding `abcbaab`

We begin with the interval $[0, 1)$. At each step:

$$\text{If our current interval is } [L, R),$$

we subdivide it into subintervals based on the probabilities $0.2, 0.3$, and 0.5 for a, b, c:

$$\begin{aligned}&\text{Interval for } a : [\,L,\ L + 0.2(R - L)),\\ &\text{Interval for } b : [\,L + 0.2(R - L),\ L + 0.5(R - L)),\\ &\text{Interval for } c : [\,L + 0.5(R - L),\ R).\end{aligned}$$

We pick the subinterval matching the next character and let that become our new $[L, R)$.

String: `abcbaab`

1. **Step 1 (encode 'a'):**

   $$[L, R) = [0, 1) \ \rightarrow\ \text{subinterval for } a = [0,\ 0.2).$$

   New interval: [0, 0.2).

2. **Step 2 (encode 'b'):**

   $$[L, R) = [0,\ 0.2) \quad \Delta = 0.2.$$

   Subinterval for b is $[\,L + 0.2\Delta,\ L + 0.5\Delta) = [0.04,\ 0.1)$. New interval: [0.04, 0.1).

3. **Step 3 (encode 'c'):**

   $$[L, R) = [0.04,\ 0.1) \quad \Delta = 0.06.$$

   Subinterval for c is $[\,0.04 + 0.5 \cdot 0.06,\ 0.1) = [0.07,\ 0.1)$. New interval: [0.07, 0.1).

4. **Step 4 (encode 'b'):**

   $$[L, R) = [0.07,\ 0.1) \quad \Delta = 0.03.$$

   Subinterval for b is $[\,0.07 + 0.2 \cdot 0.03,\ 0.07 + 0.5 \cdot 0.03) = [0.076,\ 0.0855)$. New interval: [0.076, 0.0855).

5. **Step 5 (encode 'a'):**

$$[L,R) = [0.076,\ 0.0855) \quad \Delta = 0.0095.$$

Subinterval for a is $[\,0.076,\ 0.076 + 0.2 \cdot 0.0095) = [0.076,\ 0.0779)$. New interval: $[0.076, 0.0779)$.

6. **Step 6 (encode 'a'):**

$$[L,R) = [0.076,\ 0.0779) \quad \Delta = 0.0019.$$

Subinterval for a is $[\,0.076,\ 0.076 + 0.2 \cdot 0.0019) = [0.076,\ 0.07638)$. New interval: $[0.076, 0.07638)$.

7. **Step 7 (encode 'b'):**

$$[L,R) = [0.076,\ 0.07638) \quad \Delta = 0.00038.$$

Subinterval for b is $[\,0.076{+}0.2{\cdot}0.00038,\ 0.076{+}0.5{\cdot}0.00038) = [0.076076,\ 0.07619)$. Final interval: $[0.076076, 0.07619)$.

Hence, the arithmetic-coded value for `abcbaab` is any real number in

$$[\,0.076076,\ 0.07619).$$

5.3 Decoding From 0.63215699 for 10 Symbols

We now decode a 10-symbol sequence assuming $P(a) = 0.2, P(b) = 0.3, P(c) = 0.5$. At each step, we split the current interval $[L, R)$ into subintervals for a, b, c and see where 0.63215699 falls.

1. **Step 1:** Start $[0, 1)$.

$$a : [0, 0.2), \quad b : [0.2, 0.5), \quad c : [0.5, 1).$$

Since $0.63215699 \in [0.5, 1)$, the first symbol is c. New interval: $[0.5, 1)$.

2. **Step 2:** Interval $[0.5, 1)$ (length 0.5).

$$a : [0.5, 0.6), \quad b : [0.6, 0.75), \quad c : [0.75, 1).$$

$0.63215699 \in [0.6, 0.75) \implies$ second symbol is b. New interval: $[0.6, 0.75)$.

3. **Step 3:** Interval $[0.6, 0.75)$ (length 0.15).

$$a : [0.6, 0.63), \quad b : [0.63, 0.675), \quad c : [0.675, 0.75).$$

$0.63215699 \in [0.63, 0.675) \implies$ third symbol is b. New interval: $[0.63, 0.675)$.

4. **Step 4:** Interval $[0.63, 0.675)$ (length 0.045).

$$a : [0.63, 0.639), \quad b : [0.639, 0.6525), \quad c : [0.6525, 0.675).$$

$0.63215699 < 0.639 \implies$ fourth symbol is a. New interval: $[0.63, 0.639)$.

5. **Step 5:** Interval $[0.63, 0.639)$ (length 0.009).

$$a : [0.63, 0.6318), \quad b : [0.6318, 0.6345), \quad c : [0.6345, 0.639).$$

$0.63215699 \in [0.6318, 0.6345) \implies$ fifth symbol is b. New interval: $[0.6318, 0.6345)$.

6. **Step 6:** Interval $[0.6318, 0.6345)$ (length 0.0027).

$$a : [0.6318, 0.63234), \quad b : [0.63234, 0.63315), \quad c : [0.63315, 0.6345).$$

$0.63215699 < 0.63234 \implies$ sixth symbol is a. New interval: $[0.6318, 0.63234)$.

7. **Step 7:** Interval $[0.6318, 0.63234)$ (length 0.00054).

$$a : [0.6318, 0.631908), \quad b : [0.631908, 0.63207), \quad c : [0.63207, 0.63234).$$

$0.63215699 > 0.63207 \implies$ seventh symbol is c. New interval: $[0.63207, 0.63234)$.

8. **Step 8:** Interval $[0.63207, 0.63234)$ (length 0.00027).

$$a : [0.63207, 0.632124), \quad b : [0.632124, 0.632205), \quad c : [0.632205, 0.63234).$$

$0.63215699 \in [0.632124, 0.632205) \implies$ eighth symbol is b. New interval: $[0.632124, 0.632205)$.

9. **Step 9:** Interval $[0.632124, 0.632205)$ (length 0.000081).

$$a : [0.632124, 0.6321402), \quad b : [0.6321402, 0.6321645), \quad c : [0.6321645, 0.632205).$$

$0.63215699 < 0.6321645 \implies$ ninth symbol is b. New interval: $[0.6321402, 0.6321645)$.

10. **Step 10:** Interval $[0.6321402, 0.6321645)$ (length 0.0000243).

$$a : [0.6321402, 0.63214506), \quad b : [0.63214506, 0.63215235), \quad c : [0.63215235, 0.6321645).$$

$0.63215699 \in [0.63215235, 0.6321645) \implies$ tenth symbol is c.

Hence, the decoded 10-symbol sequence is

$$c,\ b,\ b,\ a,\ b,\ a,\ c,\ b,\ b,\ c.$$

6 Problem 6: Compression Methods

6.1 Problem Statement

We want to compress the string:

`"abracadabraarbadacarba"`

using three methods:

- LZ77 with a window size of 6 and a lookahead buffer of size 6.
- LZ78.
- LZW (with {`a`,`b`,`c`,`d`,`r`} in the dictionary initially, in alphabetical order).

6.2 LZ77 (Window = 6, Lookahead Buffer = 6)

We scan the string `"abracadabraarbadacarba"` from left to right, always allowing up to 6 characters of search history and up to 6 characters of lookahead. We output triples (distance, length, nextChar) each time. The string, indexed from 0 to 21, is:

Indices: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
Chars: *a b r a c a d a b r a a r b a d a c a r b a*

We encode as follows:

1. **Position 0**, no search window: no match. Output $(0, 0,$ `a`$)$. Move to index 1.
2. **Position 1**, search window = ‘a‘: no match for ‘b‘. Output $(0, 0,$ `b`$)$. Move to index 2.
3. **Position 2**, search = ‘ab‘: no match for ‘r‘. Output $(0, 0,$ `r`$)$. Move to index 3.
4. **Position 3**, search = ‘abr‘: next is ‘a‘; we find a 1-length match for ‘a‘ (found at index 0). Next char after that is ‘c‘ (index 4). Output $(3, 1,$ `c`$)$. We skip the ‘a‘ (index 3) plus that next char (index 4), so jump to index 5.
5. **Position 5**, search (up to length 6) = ‘abrac‘ (indices 0–4). The substring ‘a‘ at index 5 matches ‘a‘ in the window (e.g., at index 3). Next char is ‘d‘ (index 6). Output $(2, 1,$ `d`$)$. Move to index 7.
6. **Position 7**, search = ‘bracad‘ (indices 1–6). The substring ‘a‘ at index 7 matches the ‘a‘ in the window (e.g., index 5). Next char is ‘b‘ (index 8). Output $(2, 1,$ `b`$)$. Move to index 9.
7. **Position 9**, search = ‘acadab‘ (indices 3–8). We see ‘r‘ at index 9 has no direct multi-length match. Output $(0, 0,$ `r`$)$. Move to index 10.
8. **Position 10**, search = ‘cadabr‘ (indices 4–9). We see ‘a‘ at index 10 can match single ‘a‘ in the window. Next char is ‘a‘ (index 11). So output $(1, 1,$ `a`$)$. Move to index 12.
9. **Position 12**, search = ‘dabra a‘ (indices 6–11). Next substring is ‘r‘ at index 12. We match ‘r‘ from index 9, length 1. Next char is ‘b‘ (index 13). Output $(3, 1,$ `b`$)$. Move to index 14.
10. **Position 14**, search = ‘abraar‘ (indices 8–13). The substring ‘a‘ at index 14 matches in the window. Next char is ‘d‘ (15). Output $(3, 1,$ `d`$)$. Move to index 16.

11. **Position 16**, search = ‘raarba‘ (indices 10–15). The substring ‘a‘ at index 16 matches the ‘a‘ at index 14. Next char is ‘c‘ (17). Output $(2, 1, \texttt{c})$. Move to index 18.
12. **Position 18**, search = ‘arbad a c‘ (indices 12–17; effectively up to 6 chars). We see ‘a‘ at 18 matches the ‘a‘ at 16. Next char is ‘r‘ (19). Output $(2, 1, \texttt{r})$. Move to index 20.
13. **Position 20**, search = ‘badacar‘ (indices 14–19). ‘b‘ at 20 is new. Output $(0, 0, \texttt{b})$. Move to index 21.
14. **Position 21**, search = ‘adacarb‘ (indices 15–20). The last character is ‘a‘. No multi-length match; output $(0, 0, \texttt{a})$. Done.

Thus, the final sequence of LZ77 triples is:

$$(0,0,\texttt{a}),\ (0,0,\texttt{b}),\ (0,0,\texttt{r}),\ (3,1,\texttt{c}),\ (2,1,\texttt{d}),\ (2,1,\texttt{b}),$$
$$(0,0,\texttt{r}),\ (1,1,\texttt{a}),\ (3,1,\texttt{b}),\ (3,1,\texttt{d}),\ (2,1,\texttt{c}),\ (2,1,\texttt{r}),$$
$$(0,0,\texttt{b}),\ (0,0,\texttt{a}).$$

6.3 LZ78

For LZ78, we maintain a dictionary of substrings (each with an index). We scan from left to right, always outputting (index, nextChar) where index is the dictionary index of hte longest matching prefix found and nextChar is the next new character that extends that prefix.We then add that new substring (prefix + new character) to the dictionary.

1. Read ‘a‘ (index 0). Not in dictionary, so output $(0, \texttt{a})$. Dictionary: (1) ‘a‘.
2. Read ‘b‘ (index 1). Not in dictionary, output $(0, \texttt{b})$. Dictionary: (1) ‘a‘, (2) ‘b‘.
3. Read ‘r‘ (index 2). Not in dictionary, output $(0, \texttt{r})$. Dictionary: (1) ‘a‘, (2) ‘b‘, (3) ‘r‘.
4. Read ‘a‘ (index 3). That is in dictionary entry 1. Next char is ‘c‘ (index 4). Substring ‘ac‘ not in dictionary. Output $(1, \texttt{c})$. Dictionary: (1) ‘a‘, (2) ‘b‘, (3) ‘r‘, (4) ‘ac‘.
5. Now we move past ‘a,c‘ (indices 3,4). Next index 5 is ‘a‘. That is dictionary entry 1. Next char is ‘d‘ (6). ‘ad‘ not in dictionary, so output $(1, \texttt{d})$. Dictionary: (5) ‘ad‘ added.
6. Move past ‘a,d‘. Next index 7 is ‘a‘. That is entry 1. Next char is ‘b‘ (8). ‘ab‘ not in dictionary. Output $(1, \texttt{b})$. Dictionary: (6) ‘ab‘.
7. Next index 9 is ‘r‘ (dictionary entry 3). Next char is ‘a‘ (10). ‘ra‘ not in dictionary. Output $(3, \texttt{a})$. Dictionary: (7) ‘ra‘.

8. Next index 11 is ‘a‘ (entry 1). Next char is ‘r‘ (12). ‘ar‘ not in dictionary. Output $(1, \texttt{r})$. Dictionary: (8) ‘ar‘.
9. Next index 13 is ‘b‘ (entry 2). Next char is ‘a‘ (14). ‘ba‘ not in dictionary. Output $(2, \texttt{a})$. Dictionary: (9) ‘ba‘.
10. Index 15 is ‘d‘. Not in dictionary, so output $(0, \texttt{d})$. Dictionary: (10) ‘d‘.
11. Index 16 is ‘a‘ (entry 1). Next char is ‘c‘ (17). ‘ac‘ is entry 4, so we have a prefix. Then the next char after that is ‘a‘ (18): ‘aca‘ not in dictionary. So we output $(4, \texttt{a})$. Dictionary: (11) ‘aca‘.
12. Index 19 is ‘r‘ (entry 3). Next char is ‘b‘ (20). ‘rb‘ not in dictionary. Output $(3, \texttt{b})$. Dictionary: (12) ‘rb‘.
13. Final index 21 is ‘a‘ (entry 1), but we have no next character left. Often we do $(1, \mathit{EOF})$ or a special marker. For simplicity, we can just treat it as $(0, \texttt{a})$ if we prefer a fresh literal. Here, let’s do $(0, \texttt{a})$. Dictionary: (13) ‘a‘ (repeated, though duplicates can happen in LZ78).

Therefore, the final LZ78 output pairs are:

$$\begin{gathered}(0,\texttt{a}),\ (0,\texttt{b}),\ (0,\texttt{r}),\ (1,\texttt{c}),\ (1,\texttt{d}),\ (1,\texttt{b}),\\ (3,\texttt{a}),\ (1,\texttt{r}),\ (2,\texttt{a}),\ (0,\texttt{d}),\ (4,\texttt{a}),\ (3,\texttt{b}),\ (0,\texttt{a}).\end{gathered}$$

6.4 LZW

Here, we initialize the dictionary with single characters:

$$1 = \texttt{a}, \quad 2 = \texttt{b}, \quad 3 = \texttt{c}, \quad 4 = \texttt{d}, \quad 5 = \texttt{r}.$$

We parse the string `abracadabraarbadacarba` from left to right, building the code sequence. Let w be the current working prefix.

1. Start $w = \varepsilon$ (empty).
2. Read ‘a‘. $w + \texttt{a} = \texttt{a}$ is in the dictionary (code=1). Keep going.
3. Next char ‘b‘. Now $w + \texttt{b} = \texttt{ab}$ is *not* in the dictionary. Output the code for $w = \texttt{a}$: that is 1. Add ‘ab‘ to dictionary with a new code: $6 = \texttt{ab}$. Set $w = \texttt{b}$.
4. Next char ‘r‘. $w + \texttt{r} = \texttt{br}$ is not in dictionary. Output code of $w = \texttt{b}$: 2. Add ‘br‘ to dictionary: $7 = \texttt{br}$. $w = \texttt{r}$.
5. Next char ‘a‘. $w + \texttt{a} = \texttt{ra}$ is not in dictionary. Output code of $w = \texttt{r}$: 5. Add ‘ra‘: $8 = \texttt{ra}$. $w = \texttt{a}$.
6. Next char ‘c‘. $w + \texttt{c} = \texttt{ac}$ not in dictionary. Output code of $w = \texttt{a}$: 1. Add ‘ac‘: $9 = \texttt{ac}$. $w = \texttt{c}$.

7. Next char ‘a‘. $w + \texttt{a} = \texttt{ca}$ not in dictionary. Output code of $w = \texttt{c}$: 3. Add ‘ca‘: $10 = \texttt{ca}$. $w = \texttt{a}$.

8. Next char ‘d‘. $w + \texttt{d} = \texttt{ad}$ not in dictionary. Output code of $w = \texttt{a}$: 1. Add ‘ad‘: $11 = \texttt{ad}$. $w = \texttt{d}$.

9. Next char ‘a‘. $w + \texttt{a} = \texttt{da}$ not in dictionary. Output $w = \texttt{d}$: code=4. Add ‘da‘: $12 = \texttt{da}$. $w = \texttt{a}$.

10. Next char ‘b‘. $w + \texttt{b} = \texttt{ab}$ *is* dictionary entry 6. So $w = \texttt{ab}$. Next char ‘r‘. $w + \texttt{r} = \texttt{abr}$ not in dictionary. Output code of $w = \texttt{ab}$: 6. Add ‘abr‘: $13 = \texttt{abr}$. $w = \texttt{r}$.

11. Next char ‘a‘. $w + \texttt{a} = \texttt{ra}$ is dictionary entry 8, so $w = \texttt{ra}$. Next char ‘a‘. $w + \texttt{a} = \texttt{raa}$ not in dictionary. Output code(ra)=8. Add ‘raa‘: $14 = \texttt{raa}$. $w = \texttt{a}$.

12. Next char ‘r‘. $w + \texttt{r} = \texttt{ar}$ not in dictionary (we have ‘ac‘, ‘ad‘, but not ‘ar‘). Output code(a)=1. Add ‘ar‘: $15 = \texttt{ar}$. $w = \texttt{r}$.

13. Next char ‘b‘. $w + \texttt{b} = \texttt{rb}$ not in dictionary. Output code(r)=5. Add ‘rb‘: $16 = \texttt{rb}$. $w = \texttt{b}$.

14. Next char ‘a‘. $w + \texttt{a} = \texttt{ba}$ not in dictionary. Output code(b)=2. Add ‘ba‘: $17 = \texttt{ba}$. $w = \texttt{a}$.

15. Next char ‘d‘. $w + \texttt{d} = \texttt{ad}$ is dictionary entry 11, so $w = \texttt{ad}$. Next char ‘a‘. $w + \texttt{a} = \texttt{ada}$ not in dictionary. Output code(ad)=11. Add ‘ada‘: $18 = \texttt{ada}$. $w = \texttt{a}$.

16. Next char ‘c‘. $w + \texttt{c} = \texttt{ac}$ is entry 9, so $w = \texttt{ac}$. Next char ‘a‘. $w + \texttt{a} = \texttt{aca}$ not in dictionary. Output code(ac)=9. Add ‘aca‘: $19 = \texttt{aca}$. $w = \texttt{a}$.

17. Next char ‘r‘. $w + \texttt{r} = \texttt{ar}$ is entry 15, so $w = \texttt{ar}$. Next char ‘b‘. $w + \texttt{b} = \texttt{arb}$ not in dictionary. Output code(ar)=15. Add ‘arb‘: $20 = \texttt{arb}$. $w = \texttt{b}$.

18. Final char ‘a‘. $w + \texttt{a} = \texttt{ba}$ is entry 17, so $w = \texttt{ba}$. String ends, so output code(ba)=17.

Hence the final LZW code stream is:

$$1,\ 2,\ 5,\ 1,\ 3,\ 1,\ 4,\ 6,\ 8,\ 1,\ 5,\ 2,\ 11,\ 9,\ 15,\ 17$$

7 Problem 7: Randomized Hadamard Transform

7.1 Problem Statement

Let $d = 1024$. We:

1. Generate a random point on the d-dimensional unit sphere.

2. Rotate it using a Randomized Hadamard Transform (RHT).
3. Quantize coordinates to ±1 by sign.
4. Invert the transform.
5. Measure mean squared error (MSE) vs. the original.
6. Repeat 100 times. Give min, mean, max MSE.

7.2 Solution and Example Code

Below is a Python script doing exactly that. It uses NumPy for the Fast Walsh-Hadamard Transform (FWHT) and random number generation. Then it prints the min/mean/max MSE from 100 trials.

```python
import numpy as np

def fast_walsh_hadamard_transform(x):
    """
    In-place Fast Walsh-Hadamard Transform of x (length must be a power of 2).
    """
    n = len(x)
    h = 1
    while h < n:
        for i in range(0, n, h*2):
            for j in range(i, i+h):
                a = x[j]
                b = x[j+h]
                x[j]   = a + b
                x[j+h] = a - b
        h *= 2
    return x

def inverse_fast_walsh_hadamard_transform(x):
    """
    Inverse of FWHT up to a 1/n scaling factor. We'll do the same transform again
    and then divide by n to get the true inverse of the orthonormal version.
    """
    fast_walsh_hadamard_transform(x)
    return x

def random_point_on_sphere(d):
    """
    Generate a random point on the d-dimensional unit sphere.
    """
    gauss = np.random.randn(d)
    norm = np.linalg.norm(gauss)
    return gauss / norm

def randomized_hadamard_transform(x):
    """
    Applies random sign flips and the FWHT. Returns (transformed_vector, sign_flips).
    """
    n = len(x)
    signs = np.random.choice([-1, 1], size=n)
    x_flipped = x * signs
    y = fast_walsh_hadamard_transform(x_flipped.copy())
    # no normalization here
    return y, signs

def inverse_randomized_hadamard_transform(y, signs):
    """
    Invert the randomized hadamard transform. After FWHT, scale by 1/len(y) then
    flip signs back.
    """
    n = len(y)
    y_inv = inverse_fast_walsh_hadamard_transform(y.copy())
    y_inv = y_inv / n
    return y_inv * signs

def experiment(d=1024, trials=100):
    errors = []
    for _ in range(trials):
        # 1) random unit sphere point
        x_orig = random_point_on_sphere(d)
        # 2) RHT
        y, flips = randomized_hadamard_transform(x_orig)
        # 3) quantize
        y_quant = np.sign(y)
        # 4) inverse
        x_rec = inverse_randomized_hadamard_transform(y_quant, flips)
        # 5) MSE
        mse = np.mean((x_rec - x_orig)**2)
        errors.append(mse)
    return min(errors), np.mean(errors), max(errors)

if __name__ == "__main__":
    mn, avg, mx = experiment(d=1024, trials=100)
    print(f"Min MSE:  {mn}")
    print(f"Mean MSE: {avg}")
    print(f"Max MSE:  {mx}")
```

7.3 Sample Results

My run gave me these results:

Min MSE $\approx 3.53\times10^{-4}$, Mean MSE $\approx 3.94\times10^{-4}$, Max MSE $\approx 4.28\times10^{-4}$