Lab 8 — Markov Chains: Written Answers

Cole Akers

November 5, 2025

Lab 8 — Written Answers

Matrix Multiplication and Threads

In the product $C = A \times B$, each entry is:

$$C_{i,j} = \sum_k A_{i,k} \cdot B_{k,j}$$

Each $C_{i,j}$ can be computed independently since it only depends on one row of A and one column of B. That means the whole matrix multiply can be split up among threads with no need for synchronization (as long as A and B aren't being modified).

If I were extending this into a multithreaded version, I'd make each thread responsible for **one row of** C, or even one **single entry** (i, j) if I wanted maximum parallelism. Every thread would just compute its piece of C, and then we'd join them all at the end. It's the same basic idea as any other divide-and-conquer parallel loop, just applied to the matrix math.

Walks and Matrix Multiplication

We can represent a directed graph as an adjacency matrix A, where:

- `A[i,j] = 1` if there's an edge from node i → j
- `A[i,j] = 0` otherwise

What does A^2 represent?

The (i, j) entry of A^2 is:

$$(A^2)_{i,j} = \sum_k A_{i,k} \cdot A_{k,j}$$

That's literally counting all "two-step" paths from i to j — paths that go from i to some middle node k, and then from k to j. So the (i, j) entry of A^2 is the **number of 2-length walks** from node i to node j.

What about A^k?

By the same logic, $(A^k)_{i,j}$ counts the number of **walks of length** k from node i to node j. Each time we multiply by A, we're adding one more step to every possible walk.

So A, A^2, A^3, … all encode how the connectivity expands as you allow longer and longer walks through the graph.

Markov Chains

A Markov chain is just like an adjacency matrix, except instead of 0s and 1s, each entry is a **probability** — the chance of going from state i to state j in a single step.

For a matrix M to be a Markov chain:

- All entries must be non-negative (probabilities can't be negative)
- Each row must sum to 1.0 (the total probability of leaving state i is 100%)

Is M^k a Markov chain?

Yes, for any positive integer k, M^k is also a Markov chain.

When you multiply two row-stochastic matrices (where each row sums to 1), the result is still row-stochastic — so the property is preserved. And since all entries are non-negative, that stays true too.

Basically, if M describes 1-step transitions, then M^k describes k-step transitions, and both are valid Markov chains.

What does M^k[i,j] mean?

If $M_{i,j}$ is the probability of going from i $\rightarrow$ j in one step, then $(M^k)_{i,j}$ is the probability of going from i $\rightarrow$ j in **exactly** k **steps**. So M^2 is two steps, M^3 is three steps, etc. It's the same "walks in a graph" idea, just weighted by probabilities instead of simple counts.

Predictive Paging with Markov Chains

If we treat each **page number** in memory as a **state**, we can record how often we move from one page to another as the program runs. That gives us the

transition probabilities to build a matrix M, where:

$$M_{i,j} = P(\text{next page is } j \mid \text{current page is } i)$$

We can estimate these by looking at a trace of page accesses. For each page i, count how many times it's followed by j, then normalize that row so it sums to 1. Now we've got a simple Markov chain representing how our program usually moves between pages.

When a page fault happens and we need to **evict** something, we can look at the row of M that corresponds to the current page (or even powers like M^2 or M^3 to look a few steps ahead). The pages with low probability of being used again soon are good eviction candidates.

This gives us a **predictive paging** strategy: instead of reacting after the fact, we use the transition probabilities to guess which pages we'll need next and keep those around. It's not perfect, but it's a big step up from purely reactive algorithms like FIFO or even LRU.