UNIVERSITY OF MISSOURI-COLUMBIA
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

CS 8050– DESIGN AND ANALYSIS OF ALGORITHMS II

Fall 2025

ASSIGNMENT 2: Advanced Dictionaries and Hashing – Theory & Practice

Due Date: ~~Friday October 17th, 2025, at 5:00 pm~~

Extended to Friday October 24th at 5:00 pm SHARP NO LATE WILL BE ACCEPTED
(100 points)

1 Goals

- Interleave theoretical analysis with comprehensive engineering/application work on dictionaries (associative arrays) based on hashing.
- Demonstrate algorithmic depth while addressing real-world impact.
- Extend learning beyond standard coursework through both theory and application.

2 Description and Requirements

This assignment has five distinct components:

2.1 Dictionary Implementation & Extensions

- Implement a dictionary ADT supporting standard operations (insert, find, delete, update, size of the collection).
- Two independent implementations required:
  - **Hash Table with Chaining**: Use either simple linked lists or balanced BSTs in each bucket.
  - **Hash Table with Open Addressing**: Implement both linear probing and quadratic probing variants.
- Design with dynamic resizing and adjustable load factor thresholds.
- Integrate at least two string hash functions; compare a classical method (e.g., polynomial rolling hash) to a cryptographically strong or universal hash.
- **Extension**: Allow users to "plug in" custom collision resolution strategies or hash functions at runtime for benchmarking.

2.2 Theoretical Deep Dive

- **Formal Analysis**:
  - Give precise, formal definitions of the dictionary ADT and all implemented operations.

- Prove* the expected amortized complexity of insert, find, and delete for both chaining and open addressing under the simple uniform hashing assumption.
- Derive and prove** upper bounds for worst-case (adversarial) performance.
- Compare the effects of variable load factors, table resizing policies, and selected hash functions.
- For open addressing, analyze primary/secondary clustering and expected probe sequence lengths, justifying with theorems where possible.

- **Research-Backed Discussion**: Summarize one major modern theoretical advance in dictionary hashing (e.g., Cuckoo Hashing, dynamic perfect hashing, or recent lower bounds).

2.3 Controlled Experimental Study

- **Benchmark Suite**:
  - Construct reproducible experiments to measure per-operation runtimes, average probe counts, memory overhead, and collision rates under varying:
    - Load factors (0.25 – 0.95+)
    - Key distributions (uniform random, power-law, and adversarially designed duplicates/clusters)
    - Hash functions (at least two)
    - Table sizes (including prime vs power-of-two capacities)
  - Automate tests with large-scale datasets (real-world, e.g., books or logs, and synthetically generated).
- **Data Visualization**:
  - Produce graphs—such as time/op vs load factor, memory utilization, collision frequency—clearly relating empirical to theoretical findings.
- **Empirical vs Theory Comparison**:
  - Discuss alignment and divergence between observed and predicted complexity.
  - Analyze sources of discrepancies: system characteristics (caching, branch behavior, pointer indirection), memory layout, and non-random hash map inputs.

2.4 Substantial Real-World Application

Choose and implement one (or more) of the following applications:

- **Word Frequency Counter** on multi-gigabyte text datasets (concept drift, rare events, scaling behavior).
- **Real-time Duplicate Detection** in massive event streams (think fraud/abuse filtering or spam logging).

- **Symbol Table/Name Resolution** in a toy compiler that supports nested scoping or function overloading, exploring impact of hash strategies on real parsing workloads.
- **User-Defined**: Propose your own application with instructor approval, subject to high data scale and practical complexity (e.g., fast genome k-mer lookup, knowledge graph property table, etc).

2.5 Innovation/Extension Track

Go deeper in any one direction (choose or propose an approved option):

- Implement & analyze an advanced dictionary structure (e.g., Cuckoo Hashing, Dynamic Perfect Hashing, Robin Hood Hashing, minimal perfect hash).
- Integrate a probabilistic membership filter (Bloom or Cuckoo filter) and analyze its false positive/negative rates compared with your dictionary.
- Study the consistency and security issues in hashing—e.g., attacks based on hash collisions or DoS against poorly chosen hash functions; explore mitigations used in production systems.

Written Dictionaries Report Requirement

Your report (10–15 pages, excluding code, IEEE format one column) should include the following components:

1. **Formal Foundations**
   - Provide precise definitions of key concepts.
   - Include complexity proofs supported by rigorous argumentation.
2. **System Architecture & Design**
   - Present the implementation architecture clearly.
   - Discuss design trade-offs, including why certain approaches were chosen over others.
3. **Experimental Evaluation**
   - Describe experimental methods in detail (datasets, parameters, setup).
   - Present results using well-annotated graphs and tables.
   - Ensure reproducibility and clarity in methodology.
4. **Comparative Analysis**
   - Critically compare empirical findings with theoretical expectations.
   - Highlight discrepancies and analyze underlying causes.
5. **Real-World Case Study**
   - Conduct an in-depth case study of a real-world application of hashing-based dictionaries.
   - Show how theory translates into practical use and impact.
6. **Extensions & Reflections**
   - Discuss extension findings beyond the core assignment.
   - Reflect on the limits of hashing-based dictionaries.
   - Suggest directions for future work or open research questions.

3 Grading Rubric

Category	Description	Points	Report Tie-in
Implementation Quality	Correctness, modularity, flexibility of dictionary code (chaining, open addressing, dynamic resizing, hash functions)	20	*System Architecture & Design*
Theoretical Depth	Precise definitions, rigorous complexity proofs, and formal analysis of clustering/load factor effects	22	*Formal Foundations*
Experimental Rigor	Benchmarks, graphs/tables, reproducibility, and insightful analysis of experimental results	18	*Experimental Evaluation + Comparative Analysis*
Real-World Application	Sophistication, relevance, and scale of chosen application (e.g., word counter, duplicate detection, compiler table)	20	*Real-World Case Study*
Extension / Innovation	Advanced methods (e.g., cuckoo hashing, Bloom filters, security analysis) and forward-looking insights	10	*Extensions & Reflections*
Report Clarity	Cohesiveness, formatting (IEEE style), readability, and professional technical presentation	10	*Entire Report (Presentation & Formatting)*
Total		**100**	

4 Hand in Instructions

- Use **IntelliJ IDEA** to create a **Java Project** named **“group#_cs8050_assignment#”**.
    - For example, if your group number is *Group1* and you are submitting *Assignment2*, the project name should be **“group1_cs8050_assignment2”**.
    - Within this project, create a **Java package** using the same name as the project (e.g., `group1_cs8050_assignment2`).
    - All your **Java classes, interfaces, and documentation** should be placed under this package.
- Your **report document** can be in **MS Word** or **PDF** format. Place a copy of this report inside the project folder (i.e., `group#_cs8050_assignment#`).
- **ZIP** the entire project folder and **submit it electronically through Canvas**.