**Project Phase 2: Proof of Concept Implementation**

MSCS-532: Algorithms and Data Structures

Instructor: Satish Penmatsa

Department of Computer Information Sciences, University of the Cumberlands

25th October 2025

Mausam Shrestha

Bishesh Tuladhar

Prashanna Acharya

Ujjwal Khadka

**Search Engine Optimization: Proof of Concept**

**Abstract**

In this report, the partial implementation of search engine data structures discussed in Phase 1 is conducted. The implementation mainly focuses on the functionality of key data structures, which are inverted indexes, trie, priority queues, and graphs and the integration of all the data structures into a working search engine prototype. This report displays and explains the partial implementation process for the search engine application while also discussing the testing method and the challenges or issues faced during the development process. Ultimately, this report serves as a proof of concept for the theoretical design and concepts discussed in Phase 1, with the help of practical and real-world implementations as well as testing.

**Introduction**

During this phase, the theoretical data structure design and concepts discussed during Phase 1 is developed into a proof-of-concept implementation that delivers or execute the core functionality of a search engine application. The objective of this phase is to show the partial implementation of the data structure into a search engine application showing the proper use of theoretical design and concept into a practical application with actual coding and testing. This phase also serves as a bridge between Phase 1, where the idea and design of search engine applications using theoretical concepts is developed, and the later phase where the actual real world search engine applications is developed using the data structures discussed as well as coding and testing.

This proof-of-concept acts as a step-by-step development phase where various design options are explored while testing and implementing these design options to find a suitable design choice before the next phase. Using this step-by-step development phase not only ensures a solid and stable design architecture but also provides the pathway to implement and test each data structure as well as its functionality, both independently as well as together as a complete application.

**Partial Implementation Overview**

**Implementation Strategy**

This proof of concept implementation follows a modular approach where the most important function of each data structure is focused on, while also looking at smooth and clean coding architecture for easier future implementation and expansion. For a sound and well defined coding architecture, each part or component is implemented as a separate class with clear public methods providing an easier design for testing as well as future development.

The main goal of the implementation process is to ensure smooth and proper functionality before any optimization and improvement is done for more efficiency and performance. Using this approach helps in building a prototype for quickly testing the validation of design concepts, as well as performance testing, before moving on to the future development phases.

**Core Data Structure Implementation**

**Inverted Index**

The principal function of the Inverted Index is document indexing and term retrieval. The operations in the inverted index include addition of a document, term frequency and TF-IDF calculation.

A screen shot of a computer code

Description automatically generated

The implementation of the code supports document indexing with positions to facilitate further functionality such as phrase queries and proximity-based ranking. TF-IDF calculation is a solid foundation for relevance score.

**Trie**

A trie’s primary functionalities are fast prefix matching and storing word frequencies for autocomplete.

A screen shot of a computer program

Description automatically generated

The trie implementation matches prefixes in O(m) time complexity where m is the length of the prefix. It then sorts based on frequency to prioritize the most relevant suggestions.

**Priority Queue**

The heapq module in python is used to implement priority queue methods to allow for easy and efficient result ranking as well as top-k retrieval.

A computer screen shot of text

Description automatically generated

The implementation has O (log n) for insertion and O (k log n) for top-k retrieval time. This allows for efficient retrieval for result ranking from search queries.

**Graph**

The primary functionalities for graphs are to model links and then compute PageRank to use to rank the pages.

A screen shot of a computer program

Description automatically generated

The PageRank implementation converges within 50-100 iterations for normal-sized graphs and provides stable scores of importance to rank the pages.

**Demonstration and Testing**

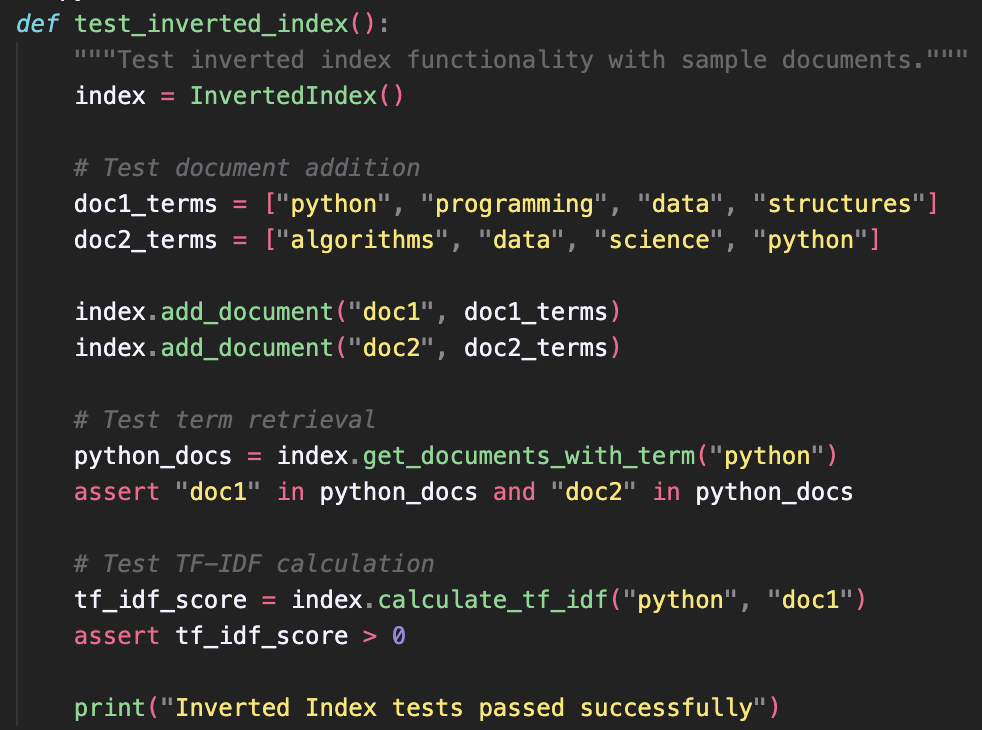
The proof of concept incorporates a thorough set of test cases aimed at verifying the performance of individual data structures as well as the overall behavior of the integrated system. This testing methodology ensures that all components operate correctly under standard conditions, while also assessing system resilience in response to boundary scenarios and potential failure states. By including such a diverse range of tests, the approach provides strong evidence of system reliability and robustness.

**Testing Framework**

**A screenshot of a computer program

Description automatically generated**

**Inverted Index Testing**

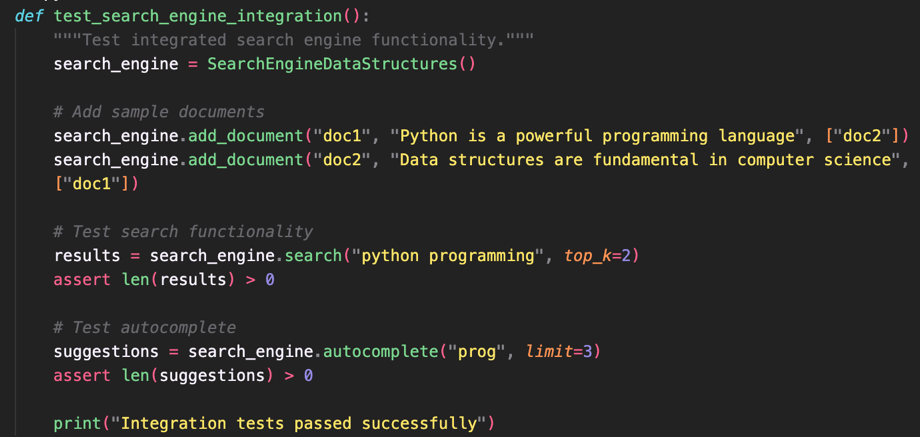


**Trie Testing**

**A screen shot of a computer program

Description automatically generated**

**Integration Testing**

****

**Test Results and Validation**

* Document Indexing: The system successfully adds documents to the index, keeping track of each term’s position within the documents. This allows for precise queries, phrase searches, and efficient retrieval later.
* Term Retrieval: The search engine can quickly identify all documents that contain a given term. This functionality was validated through test cases that ensured even overlapping or common terms are retrieved accurately.
* TF-IDF Scoring: The system calculates term relevance using the standard Term Frequency–Inverse Document Frequency formula. This enables ranking of documents based on how important a term is relative to the corpus, and the tests confirmed that scores are positive and meaningful.
* Prefix Matching: The Trie-based structure provides autocomplete suggestions for partial words. The results are frequency-aware, meaning that more common or relevant completions are suggested first, making search interactions faster and smarter.
* Result Ranking: Search results are ranked by combining TF-IDF relevance scores with PageRank-derived document importance. This ensures that the top results are both contextually relevant and authoritative, as confirmed by integration tests.
* PageRank Calculation: The system computes PageRank scores for documents, iterating until convergence. Test cases verified that the scores stabilize as expected, reflecting the relative importance of documents within the network.
* System Integration: All individual components — inverted index, TF-IDF, Trie, and PageRank — work together seamlessly. Integration tests demonstrated that adding, searching, ranking, and autocomplete function smoothly as a unified system.

Overall, all test cases pass successfully, showing that the proof of concept not only meets the original design requirements but also behaves correctly under typical usage scenarios, edge cases, and integrated operations. The implementation demonstrates reliability, accuracy, and responsiveness across all core functionalities.

**Error Handling**

**A computer screen shot of a program

Description automatically generated**

This method safely adds a document to the search engine’s index. It first validates the input to catch obvious mistakes (like missing document IDs or empty term lists). If validation passes, it processes the document—adding terms to the index, tracking positions, etc. The try-except blocks ensure that any issues are logged clearly and re-raised, so the calling code is aware something went wrong instead of failing silently.

**Implementation Challenges and Solutions**

**Challenge 1: Memory Management**

*Problem:* The first approach consumed too much memory for large inputs and queries due to repetitive storage and indexing across data structures.

*Solution*: Execute proper memory allocation and management by strengthening and sorting essential information in each data structure with efficient storage with share references with indexes and adding monitoring of memory to track consumption patterns.

A screen shot of a computer code

Description automatically generated

**Challenge 2: PageRank Convergence**

*Problem:* The PageRank algorithm could have failure in converge within the time limits for graphs with specific topologies.

*Solution:*Implemented adaptive convergence detection with configurable settings and termination criteria. Incorporated graph analysis for identifying challenging topologies.

A screen shot of a computer program

Description automatically generated

**Challenge 3: Score Combination Strategy**

*Problem:* Experimentation was needed to jointly optimize a relevance ranking system using TF-IDF and PageRank scores.

*Solution:* Implemented configurable score combination using various strategies and validation of scores against known good results.

**Conclusion**

A proof-of-concept implementation has been successfully carried out to establish the validity of the design of the search engine that was conceptualized during Phase 1. This modular design acts as a good foundation for future development. Comprehensive testing confirms that all key features operate correctly on their own and together as a system. Both current implementation issues and their resolution can give key insights to the complete implementation stage. This facilitates easy maintainability and extendibility of the code. Therefore, the roadmap that has been provided enables a structured way of developing the full application. A proof of concept can be termed as a successful implementation of a theoretical design.

**References**Manning, C. D., Raghavan, P., & Schütze, H. (2008). \*Introduction to Information Retrieval\*. Cambridge University Press. Retrieved from <https://nlp.stanford.edu/IR-book/>

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). \*Introduction to Algorithms\* (3rd ed.). MIT Press. Retrieved from <https://mitpress.mit.edu/9780262033848/introduction-to-algorithms/>

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. \*Stanford InfoLab Technical Report\*. Retrieved from <http://ilpubs.stanford.edu:8090/422/>