**CS-300 Communication Project One**

Zachary N. Locke

Southern New Hampshire University

Analysis and Design

Jack Lusby

October 20th, 2024

When considering the appropriate data structure for managing a dataset of courses, the distinctions among vectors, hash tables, and binary search trees (BSTs) highlight varied capabilities and constraints relevant to specific data management needs. Vectors fall short for scenarios involving frequent searches or insertions at arbitrary positions due to their O(n) time complexity for these operations. On the other hand, hash tables excel in applications requiring rapid data access, thanks to their average-case time complexity of O(1) for search, insert, and delete operations. However, hash tables encounter challenges with collision management and uneven data distribution, which can potentially degrade performance significantly in the worst cases.

BSTs are particularly advantageous for applications that benefit from ordered data. When balanced, they provide efficient operations at O(log n) time complexity, facilitating effective in-order traversal, range searches, and organized data management. The main drawback of BSTs lies in the need to maintain tree balance to preserve operational efficiency, as unbalanced trees can result in drastically reduced performance, equivalent to O(n) time complexity. Given these insights, the hash table is recommended as the most suitable data structure for managing course data. This choice is justified by the hash table's robust performance in facilitating quick data retrieval, paramount for efficiently accessing course information. The benefits of hash tables, particularly their average-case operational efficiency, outweigh the challenges posed by collision management, primarily since modern strategies like chaining or open addressing can effectively address these issues. The hash table's capacity to handle large datasets with high performance makes it an ideal choice for scenarios where quick data access and frequent updates are crucial.

Analysis Setup

1. Reading and Parsing the CSV File:
   * Open the file.
   * Read each line (assume there are n courses in total).
   * Parse the line to create a course object.
   * Insert the course object into the data structure.
2. Cost Per Line of Code:
   * Simple operations and assignments: Cost = 1
   * Function calls: Cost = Running time of that function
   * Loop over n courses: Executes n times

Data Structures Analysis:

1. Vector

* Reading and Inserting: Each course is appended to the end of the vector.
* Cost Analysis:
  + Open file: 1
  + Loop over lines (n times):
    - Parse line: 1
    - Create course object: 1
    - Append to vector: 1
  + Total Cost: 1 (open file) + 3n (loop costs) = 3n + 1
* Worst-Case Time Complexity: O(n)
* Space Complexity: O(n) — Dynamic array resizing can temporarily double the space required during resizing.
* Advantages: Simple and efficient for linear operations, very fast for adding items at the end.
* Disadvantages: No intrinsic ordering, inefficient searches (linear search), resizing array may be costly.
* Quicksort Application: Particularly useful for vectors, Quicksort can be applied efficiently using the median-of-three method to choose pivots, optimizing the sorting process with a worst-case time complexity of O(n log n).

2. Hash Table

* Reading and Inserting: Courses are inserted into a hash table using a hash function.
* Cost Analysis:
  + Open file: 1
  + Loop over lines (n times):
    - Parse line: 1
    - Create course object: 1
    - Insert into hash table (including hash function computation): 1 (average case if a good hash function and load factor maintained)
  + Total Cost: 1 + 3n = 3n + 1
* Worst-Case Time Complexity: O(n) average; O(n^2) if poor hash function leads to many collisions
* Space Complexity: O(n) — Includes additional overhead for managing collisions and maintaining pointers in chaining.
* Advantages: Fast access if the hash function is practical and suitable for lookup operations.
* Disadvantages: Hash collisions can degrade performance and are more complex to implement correctly.

3. Binary Search Tree (BST)

* Reading and Inserting: Courses are sorted into a BST, sorted by course ID or another key.
* Cost Analysis:
  + Open file: 1
  + Loop over lines (n times):
    - Parse line: 1
    - Create course object: 1
    - Insert into BST (finding the correct position): log n (if the tree is balanced)
  + Total Cost: 1 + (1+1+log n)n = 2n + n log n = n log n (dominant term)
* Worst-Case Time Complexity: O(n log n) if balanced; O(n^2) if not balanced (degenerates to a linked list)
* Space Complexity: O(n) — Each node typically includes data and pointers to two children.
* Advantages: Maintains ordered data and enables efficient searches.
* Disadvantages: Complex implementation requires rebalancing to maintain efficiency.