Suraj Hamal

Date: 06/14/2025

# CS 300 Data Structures and Algorithms

**Project One: Evaluation of run time, memory, and evaluation of Data Structures**

**Vector Data Structure**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | 1 | 1 |
| **Validate file opened** | 1 | 1 | 1 |
| **WHILE loop to collect course numbers** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **IF check for valid line format** | 1 | n | n |
| **Add courseNumber to set** | 1 | n | n |
| **Reset file pointer** | 1 | 1 | 1 |
| **WHILE loop to read file again** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **Create and set course fields** | 1 | n | n |
| **FOR each prerequisite** | 1 | n × m | nm |
| **Validate each perquisite** | 1 | n × m | nm |
| **Append each prerequisite to list** | 1 | n × m | nm |
| **Append course to vector** | 1 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 3nm + 4 |
| **Runtime** | | | O(nm) |
| **Total Memory Usage** | | | O(n + m) |

The advantages of the vector data structure are that it is simple and straightforward to implement, making it easier to insert and print, which can be good for small datasets where performance isn’t critical.

The disadvantages of vector data structure are that it takes O(n) time in worst case and must loop through each element in the vector until finding a match by course number which is inefficient as the datasets grow or for large datasets. To maintain the alphanumeric order in vector dataset, inserting a course into the correct position needs shifting all elements after that position making insertion O(n) time in a worst case.

**Hash Table Data Structure**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | 1 | 1 |
| **Validate file opened** | 1 | 1 | 1 |
| **WHILE loop to collect course numbers** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **IF check for valid line format** | 1 | n | n |
| **Add courseNumber to set** | 1 | n | n |
| **Reset file pointer** | 1 | 1 | 1 |
| **Clear hashTable** | 1 | 1 | 1 |
| **WHILE loop to read file again** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **Create and set course fields** | 1 | n | n |
| **FOR each prerequisite** | 1 | n × m | nm |
| **Validate each perquisite** | 1 | n × m | nm |
| **Append each prerequisite to list** | 1 | n × m | nm |
| **Insert course into hash table** | 1 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 3nm + 5 |
| **Runtime** | | | O(nm) |
| **Total Memory Usage** | | | O(n + m + k) |

The advantages of the hash table data structure are that it has an average case of O(1) performance for insert, search and delete which makes it easier for the frequent course lookup. The key-based access of hash table data structures makes it efficient to find the unique course numbers without scanning all the data, simplifies the duplicate detection by either overwriting or showing errors, and handles large amounts of data by using the hash function to distribute keys evenly.

The disadvantages of hash table data structure are that it has the worst-case time complexity of O(n) due to collisions or hash functions. It is difficult to traverse through sorted data in hash tables as it does not preserve any order of the keys. The hash table uses additional memory to manage its structure which can be high for unused buckets and the performance can be slowed because of its poor hash function.

**Binary Search Tree Data Structure**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | 1 | 1 |
| **Validate file opened** | 1 | 1 | 1 |
| **WHILE loop to collect course numbers** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **IF check for valid line format** | 1 | n | n |
| **Add courseNumber to set** | 1 | n | n |
| **Reset file pointer** | 1 | 1 | 1 |
| **Clear BST** | 1 | 1 | 1 |
| **WHILE loop to read file again** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **Create and set course fields** | 1 | n | n |
| **FOR each prerequisite** | 1 | n × m | nm |
| **Validate each perquisite** | 1 | n × m | nm |
| **Append each prerequisite to list** | 1 | n × m | nm |
| **Insert course into BST** | Log n or n | n | O(n log n)/O(n2) |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 3nm + 5 |
| **Runtime** | | | O(nm) |
| **Total Memory Usage** | | | O(n + m + k) |

The advantages of the BST data structure are that it has sorted traversal support with inorder traversal which returns all the courses to display in alphanumeric order. The average case of O(log n) speed allows fast search, insert and deletion, the dynamic data handling allows insertion of data without reallocating of space making it suitable for growing datasets, and structured trees with hierarchical relations makes it easier to add extended features in the dependency trees in the future.

The disadvantages of BST data structure are that it can have poor worst-case performance with O(n) if the tree is unbalanced and uses high memory overhead per node compared to vectors and hash tables. The BST data structures have more complex code logic while inserting, deleting, and traversing and require balancing to achieve higher efficiency.

**Recommendation for the ABCU University:**

After analyzing all three data structures, I would recommend the Binary Search Tree as the best choice for our project. The BST has the in-order traversal which can maintain the alphanumeric order automatically ideal for our project to display course lists. Even though BST has higher memory overhead than a vector, it is efficient to store, search and display the course data in an alphanumeric order without needing another sperate sort step. The Binary Search tree has an average O(log n) for the prerequisite search which can quickly validate if perquisite course exists or not, maintains sorted order for presentation, and is scalable as the courses increase. Its hierarchical form helps represent the prerequisites naturally. Therefore, BST data structure is appropriate for our project because it has fast average insert, maintains sorted order, avoids and rejects duplication with the same course number, scalable to add more courses for the future and helps keep the data organized even after appending or deleting preserving the BST ordering property.