**ABCU Run-time and Memory Analysis**

Jime Balvin

Southern New Hampshire University

CS-300: Analysis and Design

Sathish Gopalakrishnan

March 2nd, 2025

**Run-time Analysis**

**Vector**

Reading file

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open file filename | 1 | 1 | 1 |
| If file cannot be opened, return error | 1 | 1 | 1 |
| Create Vector to hold courses | 1 | 1 | 1 |
| Read each line of the file | 1 | n | n |
| Split line into parts: courseNumber, courseName, prerequisites | 1 | n | n |
| If the line has fewer than 2 parts, return error | 1 | n | n |
| Create new course object | 1 | n | n |
| Add course to courses vector | 1 | n | n |
| Close file | 1 | 1 | 1 |
| After all lines have been read check for prerequisites | n | 1 | n |
| If any prerequisite is missing, return error | 1 | n | n |
| Return Vector with courses | 1 | 1 | 1 |
| **Total Cost** | | | 4+6n |
| **Runtime** | | | O(n) |

Creating course object

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Use courseNumber, courseName and prerequisites to create new object | 1 | 1 | 1 |
| Return course | 1 | 1 | 1 |
| **Total Cost** | | | 2 |
| **Runtime** | | | O(1) |

**Hash Table**

Reading file

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open file filename | 1 | 1 | 1 |
| If file cannot be opened, return error | 1 | 1 | 1 |
| Create Hash Table to hold courses | 1 | 1 | 1 |
| Read each line of the file | 1 | n | n |
| Split line into parts: courseNumber, courseName, prerequisites | 1 | n | n |
| If the line has fewer than 2 parts, return error | 1 | n | n |
| Create new course object | 1 | n | n |
| If course already in the table, append to the linked list | 1 | n | n |
| If course not in the table, add courseNumber and new course object as key-value pair | 1 | n | n |
| Close file | 1 | 1 | 1 |
| After all lines have been read check for prerequisites | n | 1 | n |
| If any prerequisite is missing, return error | 1 | n | n |
| Return course table | 1 | 1 | 1 |
| **Total Cost** | | | 5+7n |
| **Runtime** | | | O(n) |

Creating course object

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Use courseNumber, courseName and prerequisites to create new object | 1 | 1 | 1 |
| Return course | 1 | 1 | 1 |
| If course already in the hash index, append to the linked list | 1 | n | n |
| If course not in the table, add course to the HashTable using courseNumber as the identifier | 1 | n | n |
| **Total Cost** | | | 2+2n |
| **Runtime** | | | O(n) |

**Binary Tree**

Reading file

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open file filename | 1 | 1 | 1 |
| If file cannot be opened, return error | 1 | 1 | 1 |
| Create an empty tree for courses | 1 | 1 | 1 |
| Read each line of the file | 1 | n | n |
| Split line into parts: courseNumber, courseName, prerequisites | 1 | n | n |
| If the line has fewer than 2 parts, return error | 1 | n | n |
| Create new course object | n | n | n |
| Add course into the tree | 1 | n | n2 |
| Close file | 1 | 1 | 1 |
| After all lines have been read check for prerequisites | n | 1 | n |
| If any prerequisite is missing, return error | 1 | n | n |
| Return Tree with courses | 1 | 1 | 1 |
| **Total Cost** | | | 5+5n+n2 |
| **Runtime** | | | O(n2) |

Creating course object

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Use courseNumber, courseName and prerequisites to create new object | 1 | 1 | 1 |
| Return course | 1 | 1 | 1 |
| Check if the tree is empty | 1 | 1 | 1 |
| Insert course as the root (if empty) | 1 | 1 | 1 |
| Go left if courseNumber smaller than root | 1 | n | n |
| Go right if courseNumber larger than root | 1 | n | n |
| Check if course already in tree | 1 | n | n |
| **Total Cost** | | | 4+3n |
| **Runtime** | | | O(n) |

**Advantages and disadvantages**

**Vector:** Advantages of vectors include accessing data sequentially very fast and iteration done in a meaningful way. However; the disadvantages are that it can work for small datasets and don’t support frequent modifications. Also, the fact that in order to modify, check prerequisites and perform specific searches requires to scan the entire vector makes it expensive.

**Hash Table:** Advantages of hash tables include fast lookup, efficient modification (adding or removing elements) and efficient prerequisite check. However, the requirement of additional memory to handle collision and ineffective sequential traversal are counted as disadvantages.

**Binary Tree:** Advantages of binary trees are maintaining elements in a sorted order, look up time is fast in a balanced tree; therefore, prerequisite check is fast too. However, when a tree is unbalanced, the look up time is disrupted which is a disadvantage as well as memory handling.

**Recommendation**

After the analysis, it is suggested that a hash table is best to use based on the expectations from ABCU. The need for fast lookup, efficient insertion and optimal prerequisite verification can be met with the implementation of the hash table. The hash table offers dynamic course management since the modification is done in constant time. Given the size of a course list, and the possibility of growth, scalability is a must and a hash table provides better performance than a vector or a binary tree.