Shawn Plaisted

CS-300

Southern New Hampshire University

07 Dec 2024

Analysis of Run Time

**Vector**

| **Line Description** | **Line Cost** | **Times Executed** | **Total Cost** |
| --- | --- | --- | --- |
| Create Vector | 1 | 1 | 1 |
| For Each line in File | 1 | N | N |
| Create Vector Course Item | 1 | n | N |
| With Prerequisite Exists | 1 | n | N |
| Append Prerequisite | 1 | n | N |
| Pushback Course Item | 1 | n | n |

**Total Cost: 5n + 1**

**Runtime: Runtime: O(n)**

**Hash Table**

| **Line Description** | **Line Cost** | **Times Executed** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Hash Table** | 1 | 1 | 1 |
| **Insert Method** | 0 | 0 | 0 |
| **Create Key** | 1 | N | N |
| **If no entry found for key** | 1 | N | N |
| **Assign node to key** | 1 | N | N |
| **Else** | 4 | N | 4n |
| **Assign old node Key to unit\_max** | 1 | N | N |
| **Find next open node** | 1 | N | N |
| **Add new newNode** | 1 | N | N |
| **Each new Line** | 1 | N | N |
| **Create Vector Course Item** | 1 | N | N |
| **While prereq exists** | 1 | N | N |
| **Append prereq** | 1 | N | N |
| **Insert Course Item** | 1 | N | N |

**Total Cost: 16n + 1**

**Runtime: O(n)**

**Binary Tree Search**

| **Line Description** | **Line Cost** | **Times Executed** | **Total Cost** |
| --- | --- | --- | --- |
| **Add node method** | 0 | 0 | 0 |
| **If root is null add root** | 1 | 1 | 1 |
| **If node is less than root add left** | 1 | n | N |
| **If no left node** | 1 | N | N |
| **This node becomes left** | 1 | N | N |
| **If node is greater than root add right** | 4 | N | 4n |
| **If no right node** | 1 | N | N |
| **This node becomes right** | 1 | N | N |
| **For each line in file** | 1 | N | N |
| **Create vector course item** | 1 | N | N |
| **While prereq exists** | 1 | N | N |
| **Append prereq** | 1 | N | N |
| **Insert Course Item** | 1 | N | N |

**Total Cost: 11n + 2**

**Runtime: O(n)**

Each data structure comes with its own pros and cons for this program. One downside of using a vector is that searching for a specific course means going through the list one item at a time until you find the right match, which can take longer. Vectors make up for this with their simplicity and speed when reading the file and adding course objects. Of the three options, vectors had the fastest runtime for these tasks, measured at 5n+1. They’re also great for sorting data efficiently, which makes them a strong option for tasks that need ordered output.

Hash tables are incredibly fast when searching for a specific course because they use a constant time lookup (O(1)). This speed comes from assigning keys to each course, making retrieval almost instant even though hash tables don’t keep data in a sorted order. To display the courses in alphanumeric order, extra steps are needed to extract the data, sort it, and then print it. This added complexity makes hash tables less suited for this program.

Binary search trees strike a balance between vectors and hash tables. They keep the data naturally sorted and offer faster searching with a time complexity of O(logn), which depends on the height of the tree. This makes them better than vectors for both searching and sorting. Inserting data into a binary search tree can also be slower, especially if the tree isn’t balanced which can hurt performance.

After weighing the options, vectors stand out as the best choice for this program. Their ability to quickly sort and display the entire course catalog in alphanumeric order is a major benefit. While searching in a vector (O(n)) is slower compared to the other two, the simplicity and efficiency of vectors for reading and sorting data make them the most practical choice for a small to medium-sized dataset. Overall, vectors are a straightforward and reliable solution for what this program needs to accomplish.