Analysis Of Run-Time And Memory

Antoine Boylston

Evaluation

**Vector Implementation:**

vector<Course> loadCourses(string fileName)

vector<Course> courses **// Cost = 1 (once)**

open file **// Cost = 1 (once)**

if file not opened **// Cost = 1 (once)**

return empty vector **// Not included if successful**

while not end of file **// Loop runs n times (n lines/courses)**

read line **// Cost = 1**

split line by commas **// Cost = 1**

if tokens < 2 **// Cost = 1**

continue **// Cost = 1 if invalid line**

create Course object **// Cost = 1**

add Course to vector **// Cost = 1**

for each Course in courses **// Loop runs n times**

for each prerequisite **// Inner loop depends on prerequisites**

check prerequisite in courses **// Worst case O(n)**

close file **// Cost = 1 (once)**

return courses **// Cost = 1 (once**)

**Hash Table Implementation:**

HashTable<string, Course> loadCourses(string fileName)

vector<Course> courses **// Same as vector: O(n)**

... **// Same reading logic as vector**

for each Course in courses **// Outer loop: n**

for each prerequisite **// Inner loop: up to n**

check prerequisite in courses **// Linear search: up to n**

insert each course into hash table **// O(1) average**

**// O(n) worst-case for collision resolution**

**BST Implementation:**

VOID loadCourseData(string filename, BST<Course>& courseTree)

while not end of file **// n iterations**

read line **// 1**

split line **// 1**

create course object **// 1**

insert into BST **// O(log n) for balanced BST**

for each course **// n**

for each prerequisite **// up to n**

check prerequisite in map **// O(1) average**

**Big O Runtime Analysis:**

|  |  |  |
| --- | --- | --- |
| Data Structure | Load Data Big O (worst-case )  (due to prerequisite check) | Load Data Big O  (w/o prerequisite check) |
| Vector | O(n^2) | O(n) |
| Hash Table | O(n^2) | O(n) average (O(n^2) worst-case collisions) |
| BST | O(n^2) | O(n log n) |

* Although insertion into the data structure has different costs, the overall worst-case time is controlled by the nested prerequisite validation loop.

**Advantages and Disadvantages:**

|  |  |  |
| --- | --- | --- |
| Data Structure | Advantages | Disadvantages |
| Vector | Fast to load, low memory, easy to implement | Linear search for finding a course (O(n)) |
| Hash Table | Fast course lookup (O(1) average) | High memory usage due to bucket, collisions |
| BST | Natural order of courses, efficient load time (O(n log n)) | Higher memory with pointers |

**Final Recommendation:**

I would recommend **Hash Table**. The advisor’s key requirement is to quickly search for any course. Hash table provides O(1) lookup time for searching courses, which is significantly faster than vector’s O(n). Although printing sorted course lists requires converting to a vector and sorting (O(n log n)), this isn’t the main concern.

The Vector is easy to implement and low memory, but slower for searching.

The BST naturally orders data but can degrade without self-balancing. Has good load time but not excellent.

The Hash Table is best suited for the advisor’s need for quick lookup. It satisfies performance needs and ease of use.