Hayden Gentry

Dr. Griffith

CS 300

Dec 10, 2023

CS 300 Project One

**Pseudocode for reading and checking a file (for all structures):**

1. Open the Course Information document
2. For every line in the document:
3. Read the line
4. Split the line into its parameters and check for file format errors
5. If there are less than two parameters in a line:

Print an error message

Move to the next line

1. If there is a prerequisite:

Check if the prerequisite is a course in the document

If the prerequisite is not a course in the document:

Print an error message

Move to the next line

1. Close the Course Information document

**Pseudocode for creating course objects and storing them in a vector:**

1. Open the Course Information document
2. Create an empty vector
3. For every line in the document:
4. Read the line
5. Split the line into its parameters
6. Create a new course object
7. Assign the course number parameter as the course number variable in the course object
8. Assign the course title parameter as the course title variable in the course object
9. If there are prerequisites:

For each of the prerequisites:

Assign the prerequisite parameter as a prerequisite variable in the course object

1. Add the course object to the vector
2. Close the Course Information document

**Pseudocode for creating course objects and storing them in a hash table or tree:**

1. Create a hash table or tree to store course information
2. Open the Course Information file
3. If the file is not empty and can be opened:
   1. Create a line variable to read the file
   2. While there is another line in the file to read:
      1. Split the line into its parameters
      2. If there are two parameters:
         1. Assign the course number as the first token
         2. Assign the course title as the second token
         3. Create a set to store prerequisites
         4. If there are more than two parameters:
            1. Split the prerequisites
            2. For every prerequisite that has been split:

If the prerequisite is in the hash table:

Add prerequisite to set of prerequisites

Else:

Print an error message for the prerequisite

* + - 1. Create the course object
      2. Add the course object to the hash table or tree
    1. Else:
       1. Print an error message for line formatting
    2. Read the next line
  1. Close the file

1. Else:
   1. Print an error message for failing to open file

**Pseudocode for printing course information and prerequisites (all structures):**

1. Receive course number as input from user
2. If the data structure contains the course number:
   1. Define a course object as the object retrieved from the data structure
   2. Print the course number of the course object
   3. Print the course title of the course object
   4. Define a prerequisite object as the prerequisites from the course object
   5. If there are prerequisites in the prerequisite object:
      1. For each prerequisite in the prerequisite object:
         1. Print each of the prerequisites
   6. Else:
      1. Print a message that there are no prerequisites for this course
3. Else:
   1. Print an error message that the course is not in the data structure

**Pseudocode for a menu (all structures):**

1. Create a main method to contain the menu logic
2. Create a data structure to hold all of the bids
3. Load the file data into the data structure
4. While the previous input was not exit:
   1. Display the menu options and wait for the user input
   2. Case print course list: calls the print all method to the data structure
   3. Case print course: calls the search method to the data structure
   4. Both of the above cases are timed for the performance and print the results
   5. Case exit: exits user from the program

**Pseudocode for printing courses in alphanumerical order (all structures):**

1. Create a method to sort the data structure alphanumerically
   1. While the list is not sorted:
      1. Begin by finding the course with the next smallest letter and number
      2. For each course within the course letter:
         1. Sort the course by course number
2. Return the sorted list
3. Create a method to print the courses from the sorted data structure
   1. For each course in the course list:
      1. Print the course

**Evaluate the run-time and memory of data structures:**

**Runtime analysis of reading the file and creating course objects for vector data structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total cost** |
| Create an empty vector | 1 | 1 | 1 |
| For every line in the document: | 1 | N | N |
| Read the line | 1 | N | N |
| Split the line into its parameters | 1 | N | N |
| Create a new course object | 1 | N | N |
| Assign the course number parameter to the course number variable in the course object | 1 | N | N |
| Assign the course title parameter to the course title variable in the course object | 1 | N | N |
| If there are prerequisites: | 1 | N | N |
| For each of the prerequisites: | 1 | N | N |
| Assign the prerequisite parameter to the prerequisite variable in the course object | 1 | N | N |
| Add the course object to the vector | 1 | N | N |
| **Total cost** | | | **10n+1** |
| **Run time** | | | **O(n)** |

**Runtime analysis of reading the file and creating course objects for hash table or tree data structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Create a hash table or tree to store course information | 1 | 1 | 1 |
| If the file is not empty and can be opened | 1 | 1 | 1 |
| Create a line variable to read the file | 1 | 1 | 1 |
| While there is another line in the file to read: | 1 | N | N |
| Split the line into its parameters | 1 | N | N |
| If there are two parameters: | 1 | N | N |
| Assign the course number as the first token | 1 | N | N |
| Assign the course title as the second token | 1 | N | N |
| Create a set to store prerequisites | 1 | N | N |
| If there are more than two parameters: | 1 | N | N |
| Split the prerequisites | 1 | N | N |
| For each of the prerequisites: | 1 | N | N |
| If the prerequisite is in the data structure: | 1 | N | N |
| Add prerequisite to set of prerequisites | 1 | N | N |
| Else: | 1 | N | N |
| Print an error message for the prerequisite | 1 | 1 | 1 |
| Create the course object | 1 | N | N |
| Add the course object to the data structure | 1 | N | N |
| Else: | 1 | N | N |
| Print an error message for line formatting | 1 | 1 | 1 |
| Read the next line | 1 | N | N |
| Close the file | 1 | 1 | 1 |
| **Total cost** | | | **16n+6** |
| **Runtime** | | | **O(n)** |

**Advantages and disadvantages to each data structure:**

**Vector:**

**Advantages:** Quick access to elements by using the index. Ability to use different algorithms like quicksort for sorting. Little memory resource usage.

**Disadvantages:** The resource usage for adding, removing, or rearranging elements can be expensive.

**Hash table:**

**Advantages:** Hash tables can store huge amounts of data. Using keys, we can quickly access elements. Built in mechanism to insert and delete elements efficiently.

**Disadvantages:** Needed consideration on how key is chosen to avoid collisions. Large memory resource usage. Elements within the list is not sorted or ordered.

**Binary tree:**

**Advantages:** Trees allow for different types or ordered traversals. Very efficient at searching and adding elements to the tree.

**Disadvantages:** Removing elements from the tree is more complicated than other data structures. Large memory resource usage. Balancing the tree is necessary to maintain performance.

**Data structure recommendation:**

Given the requirements, the runtime analysis, and the time complexity, I would suggest using a hash table as a data structure for this project. All of the important methods needed for the program are achieved by the hash table with minimal time complexity (O(n)). It is very important that our program can add, delete, and search our data structure quickly, all of which is achieved by the hash table. Considering we are making this project for a university course curriculum; it is imperative that our data structure can handle a lot of data too. The drawbacks such as a parse hash table and extra consideration avoiding collisions is offset by all of the advantages the hash table data structure presents. For these reasons, I recommend the hash table data structure for this project.