**File and parsing**

**Open** the file that was passed

**while** EOF is not true

**while** a new line character or comma is not reached

**read** the file into the buffer

**end** **while**

**if** the character is a comma

**store** information into the csCourse object

**change** the position that the information will be stored in the object

**end** **if**

**else**

**store** information into the csCourse object

**if** the size of the object is less than 2

**throw** an error

**end** **if**

**else**

**push** the object <*to the proper data structure>*

**create** a new object

**end** **else**

**end** **else**

**throw** out the comma or new line character

**end** **while**

**close** **file**

**call** **verify** **prereqs** <*for the appropriate data structure>*

**hash table insert**

**push** courseId of the object in a linked list called courses

**hash** the courseId

**store** the hashed courseId as key

**store** the csCourse object into the right bucket

**if** there is a collision

**store** the object in the last space in a linked list in the bucket

**end** **if**

**binary tree insert**

**If** there is no root

**set** the current node to the root

**end if**

**If** the current node’s courseId is larger than the stored one

**if** the current node’s left is a nullptr

**set** the node’s left to a new node with the data given

**end** **if**

**else**

**recursively** move down the left side of the tree

**end** **else**

**end** **if**

**else**

**if** the current node’s right is a nullptr

**set** the node’s right to a new node with the data given

**end** **if**

**else**

**recursively** move down the right side of the tree

**end** **else**

**end** **else**

**Verify prereqs (hash table)**

**for** **loop** through courses

**if** csCourse object prereqs list is not empty

**for** **loop** through the prereq list

**if** prereq is in the hash table

**move** to next item in list

**end** **if**

**else**

**throw** error

**end** **else**

**end** **for** **loop**

**end** **if**

**end** **for** **loop**

**Verify prereqs (binary tree)**

**Use** **inorder** tree traversal to move through the tree

**if** prereqs list is not empty at current node

**for** **loop** through the prereq list

**call** **search** function to find the proper courseId

**if** null is returned

**throw** an error

**end** **if**

**end** **for** **loop**

**end** **if**

**Verify prereqs (vector)**

**For** **loop** through the vector containing the csCourse objects

**If** the prereqs list at the current object is not empty

**for** **loop** through the prereqs list

**search** for the courseId in the vector containing csCourse objects

**if** null is returned

**throw** an error

**end** **if**

**end** **for**

**end** **if**

**end** **for**

**Search (hash table)**

**Hash** the passed course number and store as key

**Get** the bucket for the key

**If** the bucket is empty

**Print** course not found

**Return** null/empty object

**end if**

**If** the course number matches the first item

**Return** csCourse object

**End if**

**Else If** there are more items in the linked list

**Set** the current node to the next item

**While** the current node is not a null pointer

**If** the course number matches

**Return** csCourse object

**end if**

**Set** the current node to the next item

**End** **while**

**Print** course not found (if the program got here the course is not at the location)

**End** **if**

**Search (binary tree)**

**set** the current node to the root

**while** the current node is not a nullptr

**if** the current node’s courseID matches

**return** the node

**end** **if**

**if** the courseId is larger than the current node’s Id

**traverse** the tree to the left

**end** **if**

**else**

**traverse** the tree to the right

**end** **else**

**end** **while**

**Search (vector)**

**For loop** through the vector storing the csCourse objects

**if** courseId matches

**return** csCourse object

**end** **if**

**end** **for** **loop**

**print** error course not found

**print course info (hash table)**

**call** **search** function

**print** out the returned objects information

**print course info (binary tree)**

**call** **search** function

**print** out the returned objects information

**print course info (vector)**

**call** **search** function

**print** out the returned objects information

**print ordered list (hash table)**

**sort** the vector courses

**for** **loop** through courses

**call** **search** for the current object’s Id

**print** out the returned objects information

**end** **for** **loop**

**print ordered list (binary tree)**

**Use** **inorder** tree traversal to move through the tree

**print** each object's information

**print ordered list (vector)**

**order** the vector containing the csCourse objects

**for** **loop** through the vector

**print** each object's information

**end** **for** **loop**

**Menu**

**While** the user selection is not 9

**if** user selection is 1

**call** **file** **and** **parsing** function

**end** **if**

**if** user selection is 2

**call** **print** **ordered** **list** function <*for the proper data structure>*

**end** **if**

**if** user selection is 3

**call** **print** **course** **info** function *<for the proper data structure>*

**end** **if**

**if** the user selection is 9

**break**

**end** **if**

**end** **while**

**runtime analysis for hash table reading file and creating object (worst case)**

*\*n is the number of courses given*

*\*Worst case prereqs is n-1*

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times executed** | **Total cost** |
| **Open** the file that was passed | 1 | 1 | 1 |
| **while** EOF is not true | 1 | n | n |
| **while** a new line character or comma is not reached | 1 | n | n |
| **read** the file into the buffer | 1 | n | n |
| **if** the character is a comma | 1 | n | n |
| **store** information into the csCourse object | 1 | n | n |
| **change** the position that the information will be stored in the object | 1 | n | n |
| **store** information into the csCourse object | 1 | n | n |
| **if** the size of the object is less than 2 | 1 | n | n |
| **throw** an error | 1 | 1 | 1 |
| **push** the object <*to the proper data structure>*  *\*Calls hash table insert\* linked list can append in constant time* | 1 | n | n |
| **create** a new object | 1 | n | n |
| **throw** out the comma or new line character | 1 | n | n |
| **call** **verify** **prereqs** <*for the appropriate data structure>*  *worst case is a collision, changing it from* ***n(n-1)(1)*** *to* ***n(n-1)(n)*** | n2(n-1) | 1 | n3 –n2 |
| **Total Cost** | | n3 – n2 + 11n + 2 | |
| **Runtime** | | O(n3) | |

**runtime analysis for binary tree reading file and creating object (worst case)**

*\*n is the number of courses given*

*\*Worst case prereqs is n-1*

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times executed** | **Total cost** |
| **Open** the file that was passed | 1 | 1 | 1 |
| **while** EOF is not true | 1 | n | n |
| **while** a new line character or comma is not reached | 1 | n | n |
| **read** the file into the buffer | 1 | n | n |
| **if** the character is a comma | 1 | n | n |
| **store** information into the csCourse object | 1 | n | n |
| **change** the position that the information will be stored in the object | 1 | n | n |
| **store** information into the csCourse object | 1 | n | n |
| **if** the size of the object is less than 2 | 1 | n | n |
| **throw** an error | 1 | 1 | 1 |
| **push** the object <*to the proper data structure>*  \*Calls binary tree insert\* | log n | n | n log n |
| **create** a new object | 1 | n | n |
| **throw** out the comma or new line character | 1 | n | n |
| **call** **verify** **prereqs** <*for the appropriate data structure>* | n(n-1) log n | 1 | n2 – n log n |
| **Total Cost** | | n2 + 10n + 2 | |
| **Runtime** | | O(n2) | |

**runtime analysis for vector reading file and creating object (worst case)**

*\*n is the number of courses given*

*\*Worst case prereqs is n-1*

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times executed** | **Total cost** |
| **Open** the file that was passed | 1 | 1 | 1 |
| **while** EOF is not true | 1 | n | n |
| **while** a new line character or comma is not reached | 1 | n | n |
| **read** the file into the buffer | 1 | n | n |
| **if** the character is a comma | 1 | n | n |
| **store** information into the csCourse object | 1 | n | n |
| **change** the position that the information will be stored in the object | 1 | n | n |
| **store** information into the csCourse object | 1 | n | n |
| **if** the size of the object is less than 2 | 1 | n | n |
| **throw** an error | 1 | 1 | 1 |
| **push** the object <*to the proper data structure>*  \*Pushes the object to the end of the vector\* worst case is n for resize | n | n | n2 |
| **create** a new object | 1 | n | n |
| **throw** out the comma or new line character | 1 | n | n |
| **call** **verify** **prereqs** <*for the appropriate data structure>* | n2(n-1) | 1 | n3 –n2 |
| **Total Cost** | | n3 –2n2 + 10n + 2 | |
| **Runtime** | | O(n3) | |

The big O does not present the full picture of the problem we must solve. First, we must understand that the worst big O case is not the only way to examine these data structures. A good example of this is the hash table data structure. Most of the time the hash table will work at constant speed, the worst case is only when there is a collision. A collision may be avoided and mitigated in several ways. A vector may be sped up if we sort the vector and then use binary search. This changes the speed of searching the vector to *log n* time. A binary tree if it is balanced will search at log n time, but if it is improperly balanced then it will be more of a list in tree format and run at linear time.

One of the advantages of the hash table data structure is that the average speed tends to be constant. Running at constant speed means that hash tables search fast. The disadvantage is that a hash table does not have an inherent order. To display a sorted hash table, you must maintain a sorted vector. One of the advantages of a binary tree is that it will run at log n speed, when searching, if it is balanced. It is also easier to print out a sorted list. When the items are put into the tree they are sorted. A binary tree will print out the sorted items at *n* speed consistently without the need to sort the items again. The disadvantage is that a hash table is usually faster for searching. It is also possible to put items in a tree in such a way that it is no better than a list or vector. The advantages of a vector are that it is easy to implement and that there are techniques to speed up sorting and searching. The disadvantages are that a vector must be sorted to achieve good speed, it is slower than a hash table when searching, and resizing can slow down the entire process.

My recommendation is to use a hash table. The average speed for a hash table is constant for lookup. This will make searching for specific courses fast. Sorting the vector needed to display the items in order will only need to be done once and after it will run at linear speed, which is acceptable. The big O for the initial parsing and verification appears to be slower than a binary tree, however that is just the worst case. Utilizing a good hash equation will prevent collisions. It is also possible to reorganize the table to prevent collisions. This will make it the same speed as the tree, with significantly faster lookup speed.