Vector

Vector<Course> loadCourses (string filePath) {

**Init vector of type Course to hold courses**

**Open file**

**While not end of file:**

**Save line info in vector**

**If line size less than 2**

**Skip line**

**Create new course with valid id and title**

**Start loop at I = 2, loop for course line size:**

**For course in courses**

**If course at I equals course id**

**Push back course prereq with prereq id**

**Else invalid prereq**

**Push back course on courses vector**

**Close file.**

}

Struct Course {

**String courseID;**

**String title;**

**Vector<string> prereqIDs;**

**Course constructor ();**

}

void searchCourse (Vector<Course> courses, String courseNumber) {

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

Void printAllCourses(){

run ascending quick sort algorithm on vector

for I while less than vector size:

print course at i

}

//the quick sort is the same as in our vector sorting but instead of title it used the id

|  |  |  |  |
| --- | --- | --- | --- |
| Vector Data Structure | Line Cost | # Times Executes | Total Cost |
| Init vector of type course to hold courses | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| While not end of file | 1 | n | n |
| Save line info to vector | 1 | n | n |
| If line size less than 2 | 1 | n | n |
| Skip line | 1 | 1 | 1 |
| Create new course with valid id and title | 1 | n | n |
| Start loop at I = 2 loop for course line size | 1 | n | n |
| For courses in courses | 1 | n | n |
| If course at I equal course id | 1 | n | n |
| Push back course at prereq with prereq id | 1 | 1 | 1 |
| Else invalid prereq | 1 | n | n |
| Push back course on course vector | 1 | n | n |
| Close file | 1 | 1 | 1 |
|  |  | Total Cost | 9n + 4 |
|  |  | Runtime | O(n) |

Hash Table

HashTable<Course> loadCourses (string filePath) {

**Init HashTable to store courses**

**Open file**

**While not end of file:**

**Save line info in vector**

**If line size less than 2**

**Skip line**

**Create new course with valid id and title**

**Start loop at I = 2, loop for course line size:**

**For course in courses**

**If course at I equals course id**

**Push back course prereq with prereq id**

**Else invalid prereq**

**Insert complete course item into HashTable**

**Close file.**

**Return HashTable;**

}

Struct Node {

**Course course**

**Unsigned key**

**Node pointer next;**

**Node () constructor and overloaded constructors.**

**Vector<Node> nodes;**

**Int tablesize**

}

void searchCourse (HashTable<Course> courses, String courseNumber) {

**while int I less than vector size:**

**if vector at[i].key not default:**

**print out node at vector[i]**

**set node pointer equal to node at vector[i]. next**

**while node pointer not nullptr;**

**print node pointer**

**go to next node pointer**

}

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table Data Structure | Line Cost | # Times Executes | Total Cost |
| **Init HashTable to store courses** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **While not end of file:** | 1 | n | n |
| **Save line info in vector** | 1 | n | n |
| If line size less than 2 | 1 | n | n |
| Skip line | 1 | 1 | 1 |
| Create new course with valid id and title | 1 | n | n |
| Start loop at I = 2, loop for course line size | 1 | n | N |
| For course in course | 1 | n | n |
| If course at I equals course id | 1 | n | n |
| Push back course prereq with prereq id | 1 | 1 | 1 |
| Else invalid prereq | 1 | n | n |
| Insert complete course item into HashTable | O(n) | n | n |
| Close file | 1 | 1 | 1 |
| Return HashTable | 1 | 1 | 1 |
|  |  | Total Cost | 9n + 5 |
|  |  | Runtime | O(n) |

Binary Search Tree

BST<Course> loadCourses (string filePath) {

**Init BST to store courses**

**Open file**

**While not end of file:**

**Save line info in vector**

**If line size less than 2**

**Skip line**

**Create new course with valid id and title**

**Start loop at I = 2, loop for course line size:**

**For course in courses**

**If course at I equals course id**

**Push back course prereq with prereq id**

**Else invalid prereq**

**Insert complete course item into BST**

**Close file.**

**Return BST;**

}

Struct Node {

**Course course**

**Node left;**

**Node right;**

**Node() constructor and overloaded constructors.**

}

void searchCourse(Tree<Course> courses, String courseNumber) {

**Node curr = root;**

**While curr not null:**

**If curr id equal passed id:**

**print out course**

**if curr id less than passed id:**

**curr = curr left  
 else:**

**curr = curr right**

}

void printCourses(){

//recursive calls to which is doing the following

inOrder left side

print out the course info

inOrder right side

}

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table Data Structure | Line Cost | # Times Executes | Total Cost |
| **Init HashTable to store courses** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **While not end of file:** | 1 | n | n |
| **Save line info in vector** | 1 | n | n |
| If line size less than 2 | 1 | n | n |
| Skip line | 1 | 1 | 1 |
| Create new course with valid id and title | 1 | n | n |
| Start loop at I = 2, loop for course line size | 1 | n | n |
| For course in course | 1 | n | n |
| If course at I equals course id | 1 | n | n |
| Push back course prereq with prereq id | 1 | 1 | 1 |
| Else invalid prereq | 1 | n | n |
| **Insert complete course item into BST** | O(n) | n | n |
| Close file | 1 | 1 | 1 |
| Return BST | 1 | 1 | 1 |
|  |  | Total Cost | 9n + 5 |
|  |  | Runtime | O(n) |

Menu Pseudocode

Main(){

Print out menu options

Take in response

Switch on response

Case 1:

Load data into data structure

Case 2:

Print in ascending alphanumeric order

Case 3:

Take in course id

Print title and prerequisites for given course   
 Case 4:

Exit program

}

Conclusion

Each of the data structures that we have analyzed in this class have their own advantages and disadvantages. The vector data structure is dynamic and simple to insert or remove elements, but it can be slow when looking for a specific element having to go over possibly all entries until it finds a match. The hash table data structure that allows for very fast look ups being able to directly access the desired data with the use of buckets, but the downside of this structure is that it is an unordered structure meaning it is impossible to sort the data with a HashTable alone. The Binary Search Tree structure can be searched extremely fast and can be dynamically sized but can fair worse than other structures if you come across an unbalanced tree or your data set is too large which would cause the tree to lose performance with the number of levels.

Finally, in my opinion the correct data structure for this assignment would be the binary search tree. It solves some of the problems that the other two data structures had, namely the slow search capabilities of the vector and the unordered nature of the hash table. And based on the BigO analysis, our main points of contention when deciding on the correct structure would be its ability to sort easily and the time it takes to look up an entry, thus BST is our solution.