Cristian Gallegos

Oct 12, 2025

Project One

**Vector**

* Initialize course vector<courses>
  + Call and open file
  + If file not found return -1
    - Else
    - While not at eof
      * Read each line
        + If less than two values

Return error

* + - * + Else read parameters
        + Loop while additional parameters

Checks for additional parameters and if found in elsewhere

If found current.prerequesets.pushback

* + - * + Else pushback current couse into courses vector
  + Closes file

Search and print vector

* Prompt user for input
* Call search and print
  + Loop through courses vector
    - If input equal course number
      * Print course information
      * If course has prerequisites
        + Call search and print function with prerequisites as user input

**Hash Table**

* Create hash table
* Create vector of courses
* Open file at file name
* If (file fails to open)
  + Return error file not found exit program
* Else
  + While end of file
    - Each line is read
      * Items in line separated by commas
      * If items is less than two
        + Print item error
        + CONTUINE
      * Set course into vector of courses
      * If more than two items
        + Add to course prerequisites list
    - Close file
* Loop through courses vector
  + If course has prerequisites
    - loop through courses vector
      * Check to see if prerequisite is in vector
      * If prerequisite is in vector
        + Add to hash table using course id as key
  + Else
    - Add to hash table using course id as key

Insert hash table using course

* Create int with key using course id
* Create node called ooldnode with the data of courses nodes at key
* If oldnode is null
  + Create new node with nodes vector and key as paramaters
* Else
  + If oldnode key equals uint max
    - Set oldnode key to key
    - Set oldnodes course to course
    - Set oldnodes next pointer to null
  + Else
    - While oldnode next node is not equal to null
      * Oldnode is set to the oldnode’s next node
    - Oldnode next node is equal to new node with (courses vector and course passed as parameter)

search courses function with vector courses and course id

* Assign key as course id
  + Loop through hash table finding key
    - If key is found
      * print course info
      * if prerequisite found
        + call searchfunction prerequisite as course id parameters
      * else
        + print course not found
* return

**Binary Search Tree**

* **Struct course** 
  + String course id
  + String course name
  + List prerequisites
* struct node
  + Left node
  + Right node
  + Course data
* Class
  + Node root =null
  + Insert (course)
  + addNode(node, course)
* Initialize tree of courses
* Set Root equal to null
* Using fstream to open file
  + if file not found
    - return error
* Create vector tokens value
* Create course temp
* Else if file is found
  + While not EOF
    - For each line Read each line
      * If line has two or fewer tokens
        + Return error
      * Else read and split line
        + store tokens to course id and title of temp
      * If three or more tokens

Check to see if the third parameter and onward are found in courses tree

If not found

Return error

Else

store in temp list of prerequisites

Call insert and temp as parameters

* + Close file

INSERT FUNCTION (course)

* If root is equal to null
  + Create new node with the data of course and set it to root
* Else
  + Call function Add node( root, course)

Insert node function with courses nodes and course as paramaters

* + If root is null
    - Course becomes root
  + Else if course is less than root add left
    - If left node is null
      * Add course to left node
    - else
      * function call insert node with left node of root and course passed
  + else
    - If right node is null
      * Add course to right node
    - Else
      * function call insert node with left node of root and course passed as parameters

gets course id from user

Search and print function (root,course id)

* create temp node with copy of root nodes info
* while temp is not equal to null
  + create new tempcourse called current with temp nodes course info
  + if current’s course id is equal to temp’s course id
    - print currents course info(course id, title)
    - if current course has prerequisites
      * loop for each prerequisite
        + call search and print function (with prerequisite as parameter)
    - else return
  + else
    - if course id is less than current course id
      * current node is shifted to current left node
    - else
      * current node is shifted to current right node

**Pseudocode for menu**

* While choice not equal to 9 loop
  + Print menu options function
  + Ask user for choice
  + Switch user choice
    - Case 1:
      * Load data into data structure
      * Break;
    - Case 2:
      * If data not loaded to data structure
        + Throw error
      * Else Call function to sort the courses into alphanumeric order
        + Prints sorted courses
      * Break;
    - Case 3:
      * Print Ask user for course id
      * Input added course search
      * Function search and print function called
        + If not fount

Print not found

* + - * + Else print course info

If course has prerequisites

Loop for each prerequisite

Function call search and print, prerequisite as parameter

* + - * Break;
    - Case9
      * Print goodbye
      * Exit program
    - Default
      * Unknown error
      * Print unknown input

**Runtime Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Vector** | **Cost Per line** | **# of times executed** | | **Big O Value** |
| Initialized courses vector | 1 | 1 | | 1 |
| Read each line in file | 1 | N | | n |
| Check each line for two values | 1 | N | | N |
| add course to vector of courses | 1 | 1 | | 1 |
| Check for additional values(prerequisites) | 1 | N | | N |
| If it does have prerequisites search in course vector for course | 1 | n | | n |
| Search for course | 1 | n | | n |
| Total cost | | | 5n+2 | |
| Runtime | | | O(n) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hash Table** | **Cost Per line** | **# of times executed** | | **Big O Value** |
| Create hash table | 1 | 1 | | 1 |
| Open file | 1 | 1 | | 1 |
| Read each line in file | 1 | N | | N |
| Split string | 1 | 1 | | 1 |
| Set course id and title | 1 | 1 | | 1 |
| Check for additional values | 1 | N | | N |
| If prerequisite found  Seach in hash for course | 1 | N | | n |
| Convert course id to key | 1 | 1 | | 1 |
| Search for key in hash | 1 | n | | n |
| Key not found add course |  |  | |  |
| Find next open node | 1 | N | | N |
| Set new node to the end of string of nodes | 1 | N | | N |
| Search for courses | 1 | N | | n |
| Total cost | | | 7n+5 | |
| Runetime | | | O(n) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Binary Tree** | **Cost Per line** | **# of times executed** | | **Big O Value** |
| Create tree | 1 | 1 | | 1 |
| Read each line of file | 1 | n | | n |
| Add node | 1 | 1 | | 1 |
| Root =null | 1 | 1 | | 1 |
| Add root | 1 | 1 | | 1 |
| IF Node<root SET LEFTNODE | 1 | 1 | | 1 |
| If no left node set as left node | 1 | 1 | | 1 |
| Else transverse left | 1 | N | | n |
| Else check right node | 1 | n | | n |
| Node empty place right | 1 | 1 | | 1 |
| Transverse right node till empty node | 1 | n | | n |
| Search for a course | 1 | n | | O(log n) |
| Total Cost | | | 3n+9 | |
| Runtime | | | O log n | |

**Advantages and Disadvantages**

There are many advantages and disadvantages for each of these methods when it comes to creating a list of courses and then searching these courses and printing their information. Take for example, Vectors, out of the three it is the easiest to implement for both adding items to the vector, but the problem arises if you want to search if your course is at the end of a extremely long vector it would have to search though each item in the list before getting to the course it was looking for.

The advantages of the hash table is that it is far more easier to look for a course due to the fact that a hash tree looks by obtaining a key and then using that key to find the location of the course, so unlike the vector which looks through each item the hash table method check for the key and the corresponding key location. The disadvantages of a hash table is compared to the vector method it is harder to implement, and if the key has multiple items in its location then the program has to look through each item in that location to find the course that the user is looking for.

Now for the advantages of the binary search tree, one such advantage is that it automatically sorts the courses as its being added. And because it each node only has child nodes it is easier to code/ implement, so compared to the other two it is easier to implement than the hash table but more complex than the vector. A disadvantage to the binary tree is if the tree is off balance then when it comes to searching it could lead to worst case scenario for runtime.

In conclusion for this project, the binary search tree would be the best option of the three, because if the courses increase as the years go by it would be easier for a for them to be added to tree. And because it is presorted it easier to print all courses in alphanumerical order without the extra step of sorting the list first, whereas it becomes quite complex to sort a hash table to be alphanumerical. If the number of courses is small and is not likely to change I would recommend the vector method, but since we don’t know how many courses there are and if the amount could change, the binary tree has the ability to be flexible with bigger amount of courses and can easily be added and found.