# CS 300 Project 1

## Structure to Hold Course Information

Struct Course

String course\_number

String course\_title

Vector<Course> prereqs

## Vector Method Pseudocode:

void loadCourses(string csvPath)

PRINT “Loading CSV file” + csvPath

CREATE a vector containing course objects = courses

INITIALIZE CSV parser with the csvPath parameter

TRY

FOR loop to iterate through all the rows in the parser data file

IF file[current row] length >= 2

CREATE Course object = course

SET course.course\_number = file[current row][0]

SET course.course\_title = file[current row][1]

SET j = 2

WHILE file[current row][j] != null

IF file[current row][j] is found in courses

SET course.prereqs = file[current row][j]

SET j += 1

SET new course to the end of the courses

CATCH built in csv method error

PRINT the corresponding error

void printSampleSchedule(Vector<Course> courses)

CALL sortCourses(courses, 0, course\_size – 1)

FOR all courses

PRINT course number, course title, and course prereqs

void printCourseInformation(Vector<Course> courses, String courseNumber)

SET courseFound = False

for all courses

if the course is the same as courseNumber

SET courseFound = True

print out the course information

for each prerequisite of the course

print the prerequisite course information

IF courseFound = False

PRINT “There are no matches with the provided course number.”

int Partition(vector<Course>& courses, int begin, int end)

SET low = begin

SET high = end

SET middle = low + (high – low) / 2

SET pivot = courses[middle]

SET isDone = False

WHILE isDone is False

WHILE courses[low].course\_number < courses[high].course\_number

SET low += 1

WHILE pivot.course\_number < courses[high].course\_number

SET high -= 1

IF low >= high

SET isDone = True

ELSE

CALL swap(courses[low], courses[high])

SET low += 1

SET high -= 1

RETURN high

void sortCourses(vector<Course>& courses, int begin, int end)

SET mid = 0

IF begin >= end

RETURN

SET mid = partition(courses, begin, end)

CALL sortCourses(courses, begin, end)

CALL sortCourses(courses, mid + 1, end)

## Hash Table Method Pseudocode:

void loadCourses(string csvPath)

PRINT “Loading CSV file” + csvPath

INITIALIZE CSV parser with the csvPath parameter

TRY

FOR loop to iterate through all the rows in the parser data file

IF file[current row] length >= 2

CREATE Course object named course

SET course.course\_number = file[current row][0]

SET course.course\_title = file[current row][1]

SET j = 2

WHILE file[current row][j] != null

IF file[current row][j] is found in courses

SET course.prereqs = file[current row][j]

SET j += 1

INSERT course into the hashTable with Insert(course)

CATCH built in csv method error

PRINT the corresponding error

Insert(Course course)

SET key = hash atoi from the parameter course course\_number

SET oldNode == nodes at key

IF oldNode == nullptr

newNode = new Node with the parameter bid and newly created key

SET newNode to the key position

ELSE

IF oldNode key == UINT\_MAX

SET oldNode key == key

SET oldNode course == parameter bid

SET oldNode next == nullptr

ELSE

WHILE oldNode next != nullptr

SET oldNode = oldNode next

SET oldNode next = new node with course parameter and key

int numPrerequisiteCourses(Hashtable<Course> courses)

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

return totalPrerequisites

void printSampleSchedule(Hashtable<Course> courses)

FOR node begin to end in the HashTable

IF key != UINT\_MAX

PRINT course information

SET node = next iter

WHILE node != nullptr

PRINT course information

SET node = next node

Course HashTable Search(parameters: string courseNumber)

SET key = hash atoi for the courseNumber parameter

SET node = nodes at key

IF entry is found for the key

RETURN the nodes course

IF node == nullptr OR node key == UINT\_MAX

RETURN course

WHILE node != nullptr

IF current node matches the courseNumber parameter

RETURN node course

SET node = node next

RETURN course

void printCourseInformation(Hashtable<Course> courses, String courseNumber)

SET course = hashTable->Search(courseNumber)

IF course.course\_number is not empty

PRINT course information

ELSE

PRINT “There are no matches with the provided course number.”

## Tree Method Pseudocode:

void loadCourses(string csvPath)

PRINT “Loading CSV file” + csvPath

INITIALIZE CSV parser with the csvPath parameter

TRY

FOR loop to iterate through all the rows in the parser data file

IF file[current row] length >= 2

CREATE Course object named course

SET course.course\_number = file[current row][0]

SET course.course\_title = file[current row][1]

SET j = 2

WHILE file[current row][j] != null

IF file[current row][j] is found in courses

SET course.prereqs = file[current row][j]

SET j += 1

INSERT new course into the bst with Insert(course)

CATCH built in csv method error

PRINT the corresponding error

Insert(Course course)

IF root = null

SET root = new node with the course parameters

ELSE

CALL the addNode function with the root and course as parameters

addNode(Node node, Course course)

IF node is larger than course

IF node left == null

SET left node = new node with course parameter

RETURN

ELSE

Recurse down the left node

ELSE IF node != null AND node bidID != bid bidID

IF right node is null

SET right node = new node with course parameter

RETURN

ELSE

CALL addNode function to recure down the left node

Course Search(String courseNumber)

SET current node equal to root

WHILE no matching course number is found

IF a course number match is found

RETURN current course

ELSE IF course number < than current course number

SET current = left current node

ELSE

SET current = right current node

CREATE empty Course object

RETURN course

inOrder(Node node)

IF node is not equal to null

CALL inOrder with the left node as the parameter

PRINT course number, course title, and course prereqs

CALL inOrder with the right node as the parameter

int numPrerequisiteCourses(Tree<Course> courses, Course c)

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

return totalPrerequisites

void printSampleSchedule(Tree<Course> courses)

CALL bst->InOrder()

void printCourseInformation(Tree<Course> courses, String courseNumber)

SET course = bst->Search(courseNumber)

IF course is not empty

PRINT course information

ELSE

PRINT “There are no matches with the provided course number.”

## Menu Method Pseudocode:

int main(int argc, char aargv[]) {

INITIALIZE switch with argc parameter

INITIALIZE vector named courses to hold all the Course objects

SWITCH case 2

SET csvPath = argv[1]

BREAK

SWITCH default

SET csvPath = file name that contains course info

SET choice = 0

WHILE choice != 9

PRINT “Menu: “

PRINT “ 1. Load Courses”

PRINT “ 2. Display All Courses”

PRINT “ 3. Search and Display Course and Prerequisites”

PRINT “ 9. EXIT”

PRINT “Enter choice:

SET choice = user input

SWITCH choice

SWITCH case 1

SET courses = loadCourses(cvsPath)

BREAK

SWITCH case 2

SET course\_size = courses.size()

CALL printSampleSchedule with chosen data structure

BREAK

SWITCH case 3

SET courseNumToSearch = user input

CALL printCourseInformation(courses, courseNumber)

BREAK

PRINT “Goodbye.”

RETURN 0

## Example Runtime Analysis

| **Code For Vector Data Structure** | **Line Cost** | | **# Times Executes** | | | **Total Cost** |
| --- | --- | --- | --- | --- | --- | --- |
| **loadCourses Method** | | | | | | | |
| PRINT “Loading CSV file” + csvPath | 1 | | 1 | | | 1 |
| CREATE a vector containing course  objects = courses | 1 | | 1 | | | 1 |
| INITIALIZE CSV parser with the csvPath  parameter | 1 | | 1 | | | 1 |
| TRY | 1 | | 1 | | | 1 |
| FOR loop to iterate through all the  rows in the parser data file | 1 | | n | | | n |
| IF file[current row] length >= 2 | 1 | | n | | | n |
| CREATE Course object =  course | 1 | | n | | | n |
| SET course.course\_number =  file[current row][0] | 1 | | n | | | n |
| SET course.course\_title =  file[current row][1] | 1 | | n | | | n |
| SET j = 2 | 1 | | n | | | n |
| WHILE file[current row][j]  != null | 1 | | n | | | n |
| IF file[current row][j]  is found in courses | 1 | | n | | | n |
| SET course.prereqs =  file[current row][j] | 1 | | n | | | n |
| SET j += 1 | 1 | | n | | | n |
| SET new course to the end of  the courses | 1 | | n | | | n |
| CATCH built in csv method error | 1 | | 1 | | | 1 |
| PRINT the corresponding error | 1 | | 1 | | | 1 |
| **Total For loadCourses Method** | Total Cost: 11n + 6 | | | |  | | |
| **printCourseInformation Method** | | | | | | | |
| SET courseFound = False | 1 | | 1 | | | 1 |
| for all courses | 1 | | n | | | n |
| if the course is the same as  courseNumber | 1 | | n | | | n |
| SET courseFound = True | 1 | | 1 | | | 1 |
| print out the course  information | 1 | | 1 | | | 1 |
| for each prerequisite of the  course | 1 | | n | | | n |
| print the prerequisite  course information | 1 | | 1 | | | 1 |
| IF courseFound = False | 1 | | 1 | | | 1 |
| PRINT “There are no matches with the  provided course number.” | 1 | | 1 | | | 1 |
| **Total For printCourseInformation Method** | Total Cost: 3n + 6 | | |  | | | |
| **printSampleSchedule Method** | | | | | | | |
| FOR all courses | 1 | | n | | | n |
| PRINT course number, course title,  and course prereqs. | 1 | | n | | | n |
| **Total For printSampleSchedule Method** | Total Cost: 2n | | |  | | | |
| **Total Cost** | | 2n + 3n+ 11n + 6 + 6 = 16n + 12 | | | | |
| **Runtime** | | | | | | O(n) |

| **Code For Hash Table Data Structure** | | **Line Cost** | **# Times Executes** | | | | | **Total Cost** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **loadCourses Method** | |  |  | | | | |  |
| PRINT “Loading CSV file” + csvPath | | 1 | 1 | | | | | 1 |
| INITIALIZE CSV parser with the csvPath  parameter | | 1 | 1 | | | | | 1 |
| TRY | | 1 | 1 | | | | | 1 |
| FOR loop to iterate through all the  rows in the parser data file | | 1 | n | | | | | n |
| IF file[current row] length >= 2 | | 1 | n | | | | | n |
| CREATE Course object =  course | | 1 | n | | | | | n |
| SET course.course\_number =  file[current row][0] | | 1 | n | | | | | n |
| SET course.course\_title =  file[current row][1] | | 1 | n | | | | | n |
| SET j = 2 | | 1 | n | | | | | n |
| WHILE file[current row][j]  != null | | 1 | n | | | | | n |
| IF file[current row][j]  is found in courses | | 1 | n | | | | | n |
| SET course.prereqs =  file[current row][j] | | 1 | n | | | | | n |
| SET j += 1 | | 1 | n | | | | | n |
| INSERT course into the  hashTable w/ Insert(course) | | Total Cost: 2n + 12 | | | | | | |
| CATCH built in csv method error | | 1 | 1 | | | | | 1 |
| PRINT the corresponding error | | 1 | 1 | | | | | 1 |
| **Total For loadCourses Method** | | **Total Cost: (11n + 5) + (2n + 12) = 13n + 17** | | | | | | |
| **Insert Method** | | | | | | | | |
| SET key = hash atoi from the parameter  course course\_number | | 1 | 1 | | | | | 1 |
| SET oldNode == nodes at key | | 1 | 1 | | | | | 1 |
| IF oldNode == nullptr | | 1 | 1 | | | | | 1 |
| newNode = new Node with the  parameter bid and newly created key | | 1 | 1 | | | | | 1 |
| SET newNode to the key position | | 1 | 1 | | | | | 1 |
| ELSE | | 1 | 1 | | | | | 1 |
| IF oldNode key == UINT\_MAX | | 1 | 1 | | | | | 1 |
| SET oldNode key == key | | 1 | 1 | | | | | 1 |
| SET oldNode course == parameter  bid | | 1 | 1 | | | | | 1 |
| SET oldNode next == nullptr | | 1 | 1 | | | | | 1 |
| ELSE | | 1 | 1 | | | | | 1 |
| WHILE oldNode next != nullptr | | 1 | n | | | | | n |
| SET oldNode = oldNode next | | 1 | n | | | | | n |
| SET oldNode next = new node with  course parameter and key | | 1 | 1 | | | | | 1 |
| **Total For loadCourses Method** | | Total Cost:  2n + 12 | | | |  | | |
| **printCourseInformation Method** | | | | | | | | |
| SET courseFound = hashTable-  >Search(courseNumber | | 3n + 8 | | | | | | |
| IF course.course\_number is not empty | | 1 | 1 | | | | | 1 |
| PRINT course information | | 1 | 1 | | | | | 1 |
| ELSE | | 1 | 1 | | | | | 1 |
| PRINT “There are no matches with the  provided course number.” | | 1 | 1 | | | | | 1 |
| **Total For printCourseInformation Method** | | Total Cost: 3n + 12 | |  | | | | |
| **HashTable Search Method** | |  |  | | | | |  |
| SET key = hash atoi for the courseNumber  parameter | | 1 | 1 | | | | | 1 |
| SET node = nodes at key | | 1 | 1 | | | | | 1 |
| IF entry is found for the key | | 1 | 1 | | | | | 1 |
| RETURN the nodes course | | 1 | 1 | | | | | 1 |
| IF node == nullptr OR node key ==  UINT\_MAX | | 1 | 1 | | | | | 1 |
| RETURN course | | 1 | 1 | | | | | 1 |
| WHILE node != nullptr | | 1 | n | | | | | n |
| IF current node matches the  courseNumber parameter | | 1 | n | | | | | n |
| RETURN node course | | 1 | 1 | | | | | 1 |
| SET node = node next | | 1 | n | | | | | n |
| RETURN course | | 1 | 1 | | | | | 1 |
| **Total For HashTable Search Method** | | Total Cost: 3n + 8 | | | | |  | |
| **printSampleSchedule Method** | | | | | | | | |
| FOR node begin to end in the HashTable | | 1 | n | | | | | n |
| IF key != UINT\_MAX | | 1 | n | | | | | n |
| PRINT course information | | 1 | n | | | | | n |
| SET node = next iter | | 1 | n | | | | | n |
| WHILE node != nullptr | | 1 | n | | | | | n |
| PRINT course information | | 1 | n | | | | | n |
| SET node = next node | | 1 | n | | | | | n |
| **Total For printSampleSchedule Method** | | Total Cost: 7n | | |  | | | |
| **Total Cost** | 7n + 3n + 8 + 3n + 12 + 2n + 12 = 15n + 32 | | | | | | | |
| **Runtime** | | | | | | | | O(n) |

| **Code For Tree Data Structure** | | **Line Cost** | **# Times Executes** | | | **Total Cost** |
| --- | --- | --- | --- | --- | --- | --- |
| **loadCourses Method** | |  |  | | |  |
| PRINT “Loading CSV file” + csvPath | | 1 | 1 | | | 1 |
| INITIALIZE CSV parser with the csvPath  parameter | | 1 | 1 | | | 1 |
| TRY | | 1 | 1 | | | 1 |
| FOR loop to iterate through all the  rows in the parser data file | | 1 | n | | | n |
| IF file[current row] length >= 2 | | 1 | n | | | n |
| CREATE Course object =  course | | 1 | n | | | n |
| SET course.course\_number =  file[current row][0] | | 1 | n | | | n |
| SET course.course\_title =  file[current row][1] | | 1 | n | | | n |
| SET j = 2 | | 1 | n | | | n |
| WHILE file[current row][j]  != null | | 1 | n | | | n |
| IF file[current row][j]  is found in courses | | 1 | n | | | n |
| SET course.prereqs =  file[current row][j] | | 1 | n | | | n |
| SET j += 1 | | 1 | n | | | n |
| INSERT new course into the  bst with Insert(course) | |  | | | | |
| CATCH built in csv method error | | 1 | 1 | | | 1 |
| PRINT the corresponding error | | 1 | 1 | | | 1 |
| **Total For loadCourses Method** | | Total Cost: n + 5 | | | |  |
| **Tree Insert Method** | | | | | | |
| IF root = null | | 1 | 1 | | | 1 |
| SET root = new node with the course  parameters | | 1 | 1 | | | 1 |
| ELSE | | 1 | 1 | | | 1 |
| CALL the addNode function with the  root and course as parameters | | 22 | | | | |
| **Total For Insert Method** | | Total Cost: 25 | | |  | |
| **Tree addNode Method** | | | | | | |
| IF node is larger than course | | 1 | 1 | | | 1 |
| IF node left == null | | 1 | 1 | | | 1 |
| SET left node = new node with  course parameter | | 1 | 1 | | | 1 |
| RETURN | | 1 | 1 | | | 1 |
| ELSE | | 1 | 1 | | | 1 |
| Recurse down the left node | | 1 | 1 | | | 1 |
| ELSE IF node != null AND node bidID !=  bid bidID | | 1 | 1 | | | 1 |
| IF right node is null | | 1 | 1 | | | 1 |
| SET right node = new node with  course parameter | | 1 | 1 | | | 1 |
| RETURN | | 1 | 1 | | | 1 |
| ELSE | | 1 | 1 | | | 1 |
| CALL addNode function to recure  down the left node | | 11 | | | | |
| **Total for addNode Method** | | Total Cost: 22 | | |  | |
| **inOrder Tree Method** | | | | | | |
| IF node is not equal to null | | 1 | 1 | | | 1 |
| CALL inOrder with the left node as  the parameter | | 2 | | | | |
| PRINT course number, course  title, and course prereqs | | 1 | 1 | | | 1 |
| CALL inOrder with the right node as  the parameter | | 2 | | | | |
| **Total For inOrder Tree Method** | | Total Cost: 6 | |  | | |
| **Search Tree Method** | | | | | | |
| SET current node equal to root | | 1 | 1 | | | 1 |
| WHILE no matching course number is found | | 1 | n | | | n |
| IF a course number match is found | | 1 | n | | | n |
| RETURN current course | | 1 | n | | | n |
| ELSE IF course number < than current  course number | | 1 | n | | | n |
| SET current = left current node | | 1 | n | | | n |
| ELSE | | 1 | n | | | n |
| SET current = right current node | | 1 | n | | | n |
| CREATE empty Course object | | 1 | 1 | | | 1 |
| RETURN course | | 1 | 1 | | | 1 |
| **Total For Search Tree Method** | | Total Cost: 7n + 3 | | | |  |
| **printCourseInformation Method** | | | | | | |
| SET course = bst->Search(courseNumber) | | 7n + 3 | | | | |
| IF course is not empty | | 1 | 1 | | | 1 |
| PRINT course information | | 1 | 1 | | | 1 |
| ELSE | | 1 | 1 | | | 1 |
| PRINT “There are no matches with the  provided course number.” | | 1 | 1 | | | 1 |
| **Total For printCourseInformation Method** | | Total Cost: 7n + 7 | | | |  |
| **printSampleSchedule Method** | | | | | | |
| CALL bst->InOrder() | | 6 | | | | |
| **Total For printSampleSchedule Method** | | Total Cost: 6 | | | |  |
| **Total Cost** | 6 + 7n + 7 + 7n + 3 + 6 + 22 + 25 + n + 5 = 15n + 74 | | | | | |
| **Runtime** | | | | | | O(n) |

## Evaluation

* Define how the program opens the file, reads the data from the file, parses each line, and checks for formatting errors.
* Show how to create course objects, so that one course object holds data from a single line from the input file.
* Using this pseudocode written for the previous assignments, analyze the worst-case running time of each, reading the file and creating course objects, which will be the Big O value. This should not include the pseudocode written for the menu or the sample schedule above. To do this, do the following:
  + Specify the cost per line of code and the number of times the line will execute. Assume there are n courses stored in the data structure.
  + Assume the cost for a line to execute is 1 unless it is calling a function, in which case the cost will be the running time of that function.

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Advantages** | **Disadvantages** |
| Vector | * Vectors can easily be used in conjunction with loops. * The size of a vector is dynamic. Meaning it can be changed in size as needed. * Multiple objects can easily be stored within a vector. * Efficient if the number of total elements needed is not known. * Typically, more efficient with sorting, adding elements, deleting elements, and assigning one vector to another. | * A vector is an object so this can result in more memory consumption. * An element cannot be deleted directly. There must be a method implemented to search for the element and then delete it. * Does not work well for multiple dimensions. * Slower when an element needs to be accessed directly. Similarly, to deleting an element. * Can be difficult to use if trying to use index numbers to access elements. |
| Hash Table | * The main pro here is that searching, inserting, and deleting elements is very fast and efficient. * Regardless of the input size, hash tables will typically have a constant time for searching, inserting, and deleting elements * Great for large and complex data sets * Hash tables are space-efficient * Most hash tables are easy to use | * Can become inefficient if there are too many collisions and there is no method in place to re-hash the table * Hash tables do not accept null as a value since elements are required to have an associated key and value * Hash table collisions are very hard to avoid for a large set of possible keys. * Hash tables have a limited capacity that can eventually reach the limit * Hash tables do not maintain the order of the elements which can make its somewhat difficult to access elements in a specific order. |
| Tree | * Trees can allow us to have a hierarchical representation of data * Trees are dynamic so there is no limit on how big a tree can be * Inserting and deleting nodes within a tree can be done in a moderate amount of time * Elements are automatically sorted as they are inserted into the tree. | * Trees take O(logn) amount of time to modify the list and to find. Making trees not the most efficient compared to other data structures in this aspect. * A balances binary tree is always necessary or else the system will lose speed and efficiency. * Deleting a node within a tree can be somewhat difficult to implement |

**Recommendation:**

My recommendation would have to be the hash table data structure. The reason for this, is that for all the specified methods (I did leave out the search method because it was not specified in the instructions) the total cost was the lowest with my calculations in terms of n. The total cost that I calculated was 15n + 32. Based on some additional knowledge through our readings in zybooks, the average time complexity is O(1) with the worst case time complexity being O(n). This would be very optimal for what our system is trying to achieve which includes loading data into the data structure while having the ability to insert, delete and search for various elements (courses) within the hash table. In comparison a tree when trying to perform these same tasks would be O(logn) which is very usable, but in comparison to a hash table the hash table is superior in this aspect. The only drawback I would have with utilizing a hash table with this system would be the ability to sort the elements in alphanumeric order efficiently. This is important because when the printSampleShecule method is called it needs to have the Courses sorted correctly. This may not be a huge issue for the small data set that is included here but could become more of an issue with a very large data set and in that case something like a tree would be more useful because it sorts the elements as they are inserted into the data structure. With the scope of this project my recommendation would be to go with the hash table.