//Pseudocode to open file, read data from file, parse each line, and check for file format

//function Open file

Void OpenReadCloseFile(string filePath, vector<string>& originalCourses) {

Open filePath as ifstream object

If filePath is not open then

Output “could not open file”

Return -1

While file is not at end of file

Get line from the file

Append the row to originalCourses

Close the file

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open filepath as ifstream object | 1 | 1 | 1 |
| If filePath is not open | 1 | 1 | 1 |
| Output “could not open file” | 1 | 1 | 1 |
| Return -1 | 1 | 1 | 1 |
| While file is not at end of file | 1 | N + 1 | N +1 |
| Get line from the file | 1 | N | N |
| Append the row to originalCourses | 1 | N | N |
| Close the file | 1 | 1 | 1 |
| Total Cost | | | 3N + 6 |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis

|  |  |
| --- | --- |
| Code | Total Cost |
| ifstream object | 1 |
| Get line | 1 |
| Append the row to originalCourses | N |
| Total Cost | N + 2 |
| Auxiliary space complexity | O(N) |

//function to parse

Void ParseFile(vector<string>& originalCourses, vector<vector<string>>& parsedCourses, char delimiter) {

For each row in originalCourses

Reset column count to 0

While not at end of line

Get line and split by delimiter

Append to parsedCourses[row][column]

Increment column count by 1

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row in original Courses | 1 | N | N |
| Reset column count to 0 | 1 | N | N |
| While not at end of line | 1 | M +1 | M + 1 |
| Get line and split by delimiter | 1 | M | M |
| Append to parsedCourses[row][column] | 1 | M | M |
| Increment column count by 1 | 1 | M | M |
| Total Cost | | | 2N \* 3M + 1 |
| Runtime | | | O(N\*M) |

Auxiliary Space Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Column count | | 1 |
| Get line | | 1 |
| Append to parsedCourses[row][column] | | N\*M |
| Total Cost | | N\*M + 2 |
| Auxiliary space complexity | O(N\*M) | | |

//function to ensure there are at least two parameters per line

Void ParameterCheck(vector<vector<string>>& parsedCourses) {

For each row in parsedCourses

Get size of parsedCourses[row]

If size is less than 2

Display row that has less than two parameters

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row in parsedCourses | 1 | N | N |
| Get size of parsedCourses[row] | 1 | N | N |
| If size is less than 2 | 1 | N | N |
| Display row that has less than two parameters | 1 | N | N |
| Total Cost | | | 4N |
| Runtime | | | O(N) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Variable to hold size | | 1 |
| Total Cost | 1 | | |
| Auxiliary space complexity | O(1) | | |

//function to make sure each prerequisite has matching course in file

Void PrerequisiteCheck(vector<vector<string>>& parsedCourses) {

For each row in parsedCourses

If size of parsedCourses[row] is greater than 2

For each column in parsed courses greater than 2

Search each row at first column for that course

If a matching course is found

Continue

Else

Display prerequisite course that doesn’t have matching course

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row in parsedCourses | 1 | N | N |
| If size of parsedCourses[row] is greater than 2 | 1 | N | N |
| For each column in parsed courses greater than 2 | 1 | M | M |
| Search each row at first column for that course | 1 | N | N |
| If a matching course is found | 1 | N | N |
| Continue | 1 | N | N |
| Else display prerequisite courses that doesn’t have matching course | 1 | N | N |
| Total Cost | | | N^2 \* M + 4N |
| Runtime | | | O(N^2\*M) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Variable to hold current course | | 1 |
| Total Cost | 1 | | |
| Auxiliary space complexity | O(1) | | |

//create course objects for Binary Search Tree

Course object will have: course ID, name of course, a vector for prerequisites

Node structure will have: course object and left and right pointer

//function to insert a course object into a tree structure

Void Insert(Tree<Course> courses, course object aCourse) {

Set curNode pointer to point to courses root node

If root node is null

Create new node with aCourse and assign as root

While curNode is not null

If aCourse’s courseId is less than curNode’s courseId

If curNode’s left pointer is null

Create new node with aCourse and insert to left of curNode

break

Else

Advance curNode to the left

Else

If curNode’s right pointer is null

Create new node with aCourse and insert to the right of curNode

Break

Else

Advance curNode to the right

}

Runtime Analysis (worst case): maximum tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Set curNode pointer to point to courses root node | 1 | 1 | 1 |
| If root node is null | 1 | 1 | 1 |
| Create new node with aCourse and assign as root | 1 | 1 | 1 |
| While curNode is not null | 1 | N+1 | N+1 |
| If aCourse’s courseId is less than curNode’s courseId | 1 | N | N |
| If curNode’s left pointer is null | 1 | N | N |
| Create new node with aCourse and insert to left of curNode | 1 | 1 | 1 |
| break | 1 | 1 | 1 |
| Else  Advance curNode to the left | 1 | N | N |
| Else  If curNode’s right pointer is null | 1 | N | N |
| Create new node with aCourse and insert to the right of curNode | 1 | 1 | 1 |
| Break | 1 | 1 | 1 |
| Else  Advance curNode to the right | 1 | N | N |
| Total Cost | | | 6N + 8 |
| Runtime | | | O(N) |

Runtime Analysis (best case): minimum tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Set curNode pointer to point to courses root node | 1 | 1 | 1 |
| If root node is null | 1 | 1 | 1 |
| Create new node with aCourse and assign as root | 1 | 1 | 1 |
| While curNode is not null | 1 | logN+1 | logN+1 |
| If aCourse’s courseId is less than curNode’s courseId | 1 | logN | logN |
| If curNode’s left pointer is null | 1 | logN | logN |
| Create new node with aCourse and insert to left of curNode | 1 | 1 | 1 |
| break | 1 | 1 | 1 |
| Else  Advance curNode to the left | 1 | logN | logN |
| Else  If curNode’s right pointer is null | 1 | logN | logN |
| Create new node with aCourse and insert to the right of curNode | 1 | 1 | 1 |
| Break | 1 | 1 | 1 |
| Else  Advance curNode to the right | 1 | logN | logN |
| Total Cost | | | 6logN + 8 |
| Runtime | | | O(logN) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Set curNode pointer to point to courses root node | 1 | |
| Create new node with aCourse | 1 | |
| Total Cost | | 2 | |
| Auxiliary Space Complexity | | O(1) | |

//function to parse through vector created from parsing, create course objects, and insert them into a tree

Void CreateCoursesBST(vector<vector<string>>& parsedCoursesTree, Tree<Course> courses) {

For each row

Create a Course Object courseobj

Course’s ID = parsedCourses[row][0]

Course’s name = parsedCourses[row][1]

If row size is greater than 2

For i = 2 and i < row’s size

Append parsedCourses[row][i] to vector of prerequisites within the courseobj

Insert courseobj to Tree<Courses> courses

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row | 1 | N+1 | N+1 |
| Create a course object course | 1 | N | N |
| Course’s ID = parsedCourses[row][0] | 1 | N | N |
| Course’s name = parsedCourses[row][1] | 1 | N | N |
| If row size is greater than 2 | 1 | N | N |
| For i = 2 and i < row’s size | 1 | 2M | 2M+1 |
| Append parsedCourses[row][i] to vector of prerequisites within the courseobj | 1 | M | M |
| Insert courseobj to Tree<Courses> courses | N | N | N2 |
| Total Cost | | | N2 + 5N \* 3M+2 |
| Runtime | | | O(N2) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Create a course object course | 1 | |
| Insert courseobj to Tree<Courses> courses | N | |
| Total Cost | | N+1 | |
| Auxiliary Space Complexity | | O(N) | |

//determine number of prerequisite courses for a course

Int numPrerequistieCourses(Tree<Courses> courses, string courseId) {

CurNode = course’s root node

While curNode is not null

If curNode->courseId is equal to courseId

Return size of prerequisite vector

Else if courseId is less than curNode->courseId

Advance curNode to curNode’s left node

Else

Advance curNode to curNode’s right node

Return -1

}

Runtime Analysis (worst case): maximum tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| CurNode = course’s root node | 1 | 1 | 1 |
| While curNode is not null | 1 | N+1 | N+1 |
| If curNode->courseId is equal to courseId | 1 | 1 | 1 |
| Return size of prerequisite vector | 1 | 1 | 1 |
| Else if courseId is less than curNode->courseId | 1 | N | N |
| Advance curNode to curNode’s left node | 1 | N | N |
| Else  Advance curNode to curNode’s right node | 1 | N | N |
| Return -1 | 1 | 1 | 1 |
| Total Cost | | | 4N + 4 |
| Runtime | | | O(N) |

Runtime Analysis (best case): minimum tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| CurNode = course’s root node | 1 | 1 | 1 |
| While curNode is not null | 1 | logN+1 | logN +1 |
| If curNode->courseId is equal to courseId | 1 | 1 | 1 |
| Return size of prerequisite vector | 1 | 1 | 1 |
| Else if courseId is less than curNode->courseId | 1 | logN | logN |
| Advance curNode to curNode’s left node | 1 | logN | logN |
| Else  Advance curNode to curNode’s right node | 1 | logN | logN |
| Return -1 | 1 | 1 | 1 |
| Total Cost | | | 4logN + 4 |
| Runtime | | | O(logN) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| CurNode = course’s root node | | 1 |
| Total Cost | 1 | | |
| Auxiliary Space Complexity | O(1) | | |

//search for and print out course information and prerequisites

Void PrintCourseInformationBST(Tree<Course> courses, string courseNumber) {

CurNode = course’s root node

While curNode is not null

If curNode->courseId is equal to courseId

Print curNode’s course information

Else if courseId is less than curNode->courseId

Advance curNode to curNode’s left node

Else

Advance curNode to curNode’s right node

}

Runtime Analysis (worst case): maximum tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| CurNode = course’s root node | 1 | 1 | 1 |
| While curNode is not null | 1 | N+1 | N+1 |
| If curNode->courseId is equal to courseId | 1 | 1 | 1 |
| Print curNode’s course information | 1 | 1 | 1 |
| Else if courseId is less than curNode->courseId | 1 | N | N |
| Advance curNode to curNode’s left node | 1 | N | N |
| Else  Advance curNode to curNode’s right node | 1 | N | N |
| Total Cost | | | 4N+4 |
| Runtime | | | O(N) |

Runtime Analysis (best case): minimum tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| CurNode = course’s root node | 1 | 1 | 1 |
| While curNode is not null | 1 | logN+1 | logN+1 |
| If curNode->courseId is equal to courseId | 1 | 1 | 1 |
| Print curNode’s course information | 1 | 1 | 1 |
| Else if courseId is less than curNode->courseId | 1 | logN | logN |
| Advance curNode to curNode’s left node | 1 | logN | logN |
| Else  Advance curNode to curNode’s right node | 1 | logN | logN |
| Total Cost | | | 4\*logN+4 |
| Runtime | | | O(logN) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| CurNode = course’s root node | 1 | |
| Total Cost | | 1 | |
| Auxiliary Space Complexity | | O(1) | |

//function to print all courses (in-order traversal)

Void PrintAllInOrder(Node\* aNode) {

If aNode is not null

PrintAllInOrder(aNode->left)

Print aNode->course object

PrintAllInOrder(aNode->right)

}

Runtime Analysis: Back-substitution method

T(n) = 2T(n/2) + 1 ----------equation (1)

T(n/2) = 2T(n/) +1 -----------equation (2)

T(n/) = 2T(n/) + 1 ---------equation (3)

T(n) = 2[2T(n/) + 1] + 1 --------equation (2) into equation (1)

T(n) = T(n/) + 2 + 1 ----------equation (4)

T(n) = [2T(n/) + 1] + 2 + 1--------equation (3) into equation (4)

T(n) = T(n/) + + + ------equation (5)

.

.

.

T(n) = T(n/) + + …+ + + --------equation (6)

Assume algorithm ends when n = 1, therefore:

T(n/) = T(1) ------equation (7)

n/ = 1

n = --------equation (8)

k = log2n --------equation (9)

T(n) = T(1) + + …+ + + substituting equation (7) into equation (6)

T(n) = + + …+ + + ---------equation (10)

+ …+ + + is sum of powers of 2 which is -----equation (11)

T(n) = -----substituting equation (11) into equation (10) -------------equation (12)

T(n) = 2\* – 1

T(n) = 2\* – 1

T(n) = 2n – 1 ------equation (13)

Runtime complexity is O(n)

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

The binary search tree was constructed using the course number as the key. Therefore, the implementation of the insert function constructed a binary search tree with the alphanumeric course numbers from lowest to highest. The in-order traversal will therefore print the courses from lowest to highest. The code is shown in the PrintAllInOrder function above.