//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 hash table

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

Node structure will have: course object and next pointer

Unsigned int Hash(string courseID) {

Convert string courseID to c string

Convert c string to integer

Return courseID modulo table size

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Convert string courseID to c string | 1 | N | N |
| Convert c string to integer | 1 | N | N |
| Return courseID modulo table size | 1 | 1 | 1 |
| Total Cost | | | 2\* N + 1 |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| C string array | | N |
| Integer for string | | 1 |
| Key | | 1 |
| Total Cost | N | | |
| Auxiliary space | O(N) | | |

Void Insert(Course course) {

Determine the key from Hash() function

Set pointer to point to hashtable vector index at key

If pointer is null

Create new node with course information

Insert new node into the hashtable’s vector

Else

While pointer’s next pointer is not null

Traverse the linked-list

Add new node to the end of the linked-list

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Determine the key from Hash() function | M | 1 | M |
| Set pointer to point to hashtable vector index at key | 1 | 1 | 1 |
| If pointer is null | 1 | 1 | 1 |
| Create new node with course information | 1 | 1 | 1 |
| Insert new node into the hashtable’s vector | 1 | 1 | 1 |
| Else  While pointer’s next pointer is not null | 1 | N | (N+1) |
| Traverse the linked-list | 1 | N | N |
| Add new node to the end of the linked-list | 1 | 1 | 1 |
| Total Cost | | | N+M+6 |
| Runtime | | | O(N) |

Auxiliary space complexity

|  |  |  |
| --- | --- | --- |
| Code | | Total Cost |
| Key | | 1 |
| Set pointer to point to hashtable vector index at key | | 1 |
| Create new node with course information | | 1 |
| Total Cost | 3 | | |
| Auxiliary space | O(1) | | |

Void CreateCoursesHashTable(vector<vector<string>>& parsedCourses, HashTable<Courses> 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 Hashtable courses

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For each row | 1 | N | N |
| 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 | M | M |
| Append parsedCourses[row][i] to vector of prerequisites within the courseobj | 1 | M | M |
| Insert course to Hashtable courseObjects | N | N | N2 |
| Total Cost | | | N2+ 5N \* 2M |
| Runtime | | | O(N2) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Create a course object course | 1 | |
| Insert course to Hashtable courseObjects | N | |
| Total Cost | | N+1 | |
| Auxiliary space | | O(N) | |

//search for and print out course information and prerequisites

Void PrintCourseInformationHashTable(Hashtable<Course> courses, string courseNumber) {

Determine key by calling Hash(courseNumber)

Set pointer to node at hashtable’s vector at key

If pointer is null

Return “no course”

Else if pointer’s courseID matches courseNumber

print course object

Else if pointer’s courseID does not match courseNumber and the next pointer is not null

While pointer’s next pointer is not null

If pointer’s courseID matches courseNumber

Print course object

Else

Advance pointer to next node

If pointer is null

Print “no course”

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Determine key by calling Hash(courseNumber) | M | 1 | M |
| Set pointer to node at hashtable’s vector at key | 1 | 1 | 1 |
| If pointer is null | 1 | 1 | 1 |
| Return “no course” | 1 | 1 | 1 |
| Else if pointer’s courseID matches courseNumber | 1 | 1 | 1 |
| print course object | 1 | 1 | 1 |
| Else if pointer’s courseID does not match courseNumber and the next pointer is not null | 1 | 1 | 1 |
| While pointer’s next pointer is not null | 1 | N+1 | (N+1) |
| If pointer’s courseID matches courseNumber | 1 | 1 | 1 |
| Print course object | 1 | 1 | 1 |
| Else  Advance pointer to next node | 1 | N | N |
| If pointer is null | 1 | 1 | 1 |
| Print “no course” | 1 | 1 | 1 |
| Total Cost | | | 2N+M+11 |
| Runtime | | | O(N) |

Auxiliary Space Complexity

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| key value | 1 | |
| Set pointer to node at hashtable’s vector at key | 1 | |
| Total Cost | | 1 | |
| Auxiliary space complexity | | O(1) | |

//function to print all courses

Void PrintAllHashTable() {

For all rows in Hashtable’s vector

Create pointer to first node

If pointer is not null

Print course object

If pointer’s next pointer is not null

Advance pointer to next node

While pointer’s next pointer is not null

Print the node’s course object

Advance pointer to next node

}

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For all rows in Hashtable’s vector | 1 | 1 | 1 |
| Create pointer to first node | 1 | 1 | 1 |
| If pointer is not null | 1 | 1 | 1 |
| Print course object | 1 | 1 | 1 |
| If pointer’s next pointer is not null | 1 | 1 | 1 |
| Advance pointer to next node | 1 | 1 | 1 |
| While pointer’s next pointer is not null | 1 | N | N |
| Print the node’s course object | 1 | N | N |
| Advance pointer to next node | 1 | N | N |
| Total Cost | | | 3N+6 |
| Runtime | | | O(N) |

Auxiliary Space Complexity

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Total Cost | | |
| Create pointer to first node | 1 | | |
| Total Cost | | 1 |
| Auxiliary Space Complexity | | O(1) |

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

//function to create linked-list data structure from hash table

Void HashTableToVector(HashTable<Courses> courseObjects) {

Create Vector data structure called VectorCourses

Create a counter for vector index

For course objects in courseObjects

If course item exists

Create copy of HashTable’s node and add to vector at index counter

Increment vector index counter

while node’s next pointer is not null

Advance current pointer to node’s next pointer

Create copy of node and add to vector at index counter

}

Runtime Analysis (not required)

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Create Vector data structure called VectorCourses | 1 | 1 | 1 |
| Create a counter for vector index | 1 | 1 | 1 |
| For course objects in courseObjects | 1 | 1 | 1 |
| If course item exists | 1 | 1 | 1 |
| Create copy of HashTable’s node and add to vector at index counter | 1 | 1 | 1 |
| Increment vector index counter | 1 | 1 | 1 |
| while node’s next pointer is not null | 1 | N-1 | N-1 |
| Advance current pointer to node’s next pointer | 1 | N-1 | N-1 |
| Create copy of node and add to vector at index counter | 1 | N-1 | N-1 |
| Total Cost | | | 3N+3 |
| Runtime | | | O(N) |

Auxiliary Space Complexity (Not required)

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Total Cost | | |
| Vector data structure VectorCourses | N | | |
| Counter | 1 | | |
| Copy of node | N | | |
| Total Cost | | 2N + 1 |
| Auxiliary Space Complexity | | O(N) |

//Quicksort function

int Partition(vector<Courses> courseObjects, int low, int high) {

Determine midpoint of the list and set it to mid

Set pivot value to list at mid’s courseID

While not done

While the pivot value is greater than courseID value at low

Increment low

While the pivot value is less than courseID value at high

Decrement high

If low is greater than or equal to high

Set done to true

Else

Swap values at low and high

Increment low

Decrement high

Return high

}

Runtime Analysis (not required)

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Determine midpoint of the list and set it to mid | 1 | 1 | 1 |
| Set pivot value to list at mid’s courseID | 1 | 1 | 1 |
| While not done | 1 | ? | ? |
| While the pivot value is greater than courseID value at low | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| Increment low | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| While the pivot value is less than courseID value at high | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| Decrement high | 1 | Best: 1  Average: N/2  Worst: N | Best: 1  Average: N/2  Worst: N |
| If low is greater than or equal to high | 1 | 1 | 1 |
| Set done to true | 1 | 1 | 1 |
| Else  Swap values at low and high | 1 | 1 | 1 |
| Increment low | 1 | 1 | 1 |
| Decrement high | 1 | 1 | 1 |
| Return high | 1 | 1 | 1 |
| Total Cost | | | 2N+10 |
| Runtime | | | O(N) |

Auxiliary Space Complexity Analysis (not required)

|  |  |  |
| --- | --- | --- |
| Code | Total Cost | |
| Midpoint | | 1 | |
| Pivot | | 1 | |
| Auxiliary Space Complexity | | O(1) | |

Void QuickSort(vector<Courses> courseObjects, int low, int high) {

If low is greater than or equal to high

Return

Set mid equal to index value returned by call to Partition(courseObjects, low, high)

Call QuickSort(courseObjects, low, mid)

Call QuickSort(courseObjects, mid + 1, high)

}

Runtime Analysis (not required)

In the worst case: T(n) = T(n-1) + T(1) + n

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

T(n-2) = T(n-3) + T(1) + n – 1 – 1

Back substitution:

Substituting T(n-1) into T(n)

T(n) = [T(n-2) + T(1) + n-1] + T(1) + n

= T(n-2) + 1 + n – 1 + 1 + n

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

Substituting T(n-2) into T(n)

T(n) = [T(n-3) + T(1) + n – 1 – 1] + 2n +1

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

= T(n-3) + 3n

…

T(n) = T(n-k) + kn

When n = k:

T(n) = T(0) + n2

T(n) = 1 + n2

T(n) = n2

Auxiliary Space Complexity (not required):

In the worst case, quicksort will make n recursive calls, therefore the height of a recursive call tree will be n. Therefore, the auxiliary space complexity is O(n)

Void PrintSortedVector(vector<Courses> courseObjects) {

Call QuickSort(courseObjects, low, high)

For all objects in courseObjects

Print course object

}

Runtime Analysis (not required)

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Call QuickSort(courseObjects, low, high) | O(N2) | 1 | O(N2) |
| For all objects in courseObjects | 1 | N | N |
| Print course object | 1 | N | N |
| Total Cost | | | N2 + 2N |
| Runtime | | | O(N2) |