Paul Mauviel

Professor Amy Burns

CS-300

June 12th 2022

Project 1

Contents

Pseudocode for Menu

```
FUNCTION main

    SET courseList TO NULL
    SET coursesLoaded TO FALSE

    SET running TO TRUE

    WHILE running
        DISPLAY “Select option. 1. Load. 2. Print Course List. 3. Print Course. 9. Exit.”

        GET INPUT AS selection

        IF selection IS NOT 1,2,3, or 9

            PRINT “Invalid Selection”
            CONTINUE
        END IF

        IF selection IS 1:

            SET courseList TO CALL ReadCoursesFromFile(filePath)

        ELSE IF selection IS 2:

            CALL printCourseList(courseList)

        ELSE IF selection IS 3:

            CALL printCourseInformation(courseList, coursed)

        ELSE

            SET running TO FALSE

    END LOOP

END FUNCTION
```

Sorting Lists

NOTE TO INSTRUCTOR: I don’t see the value in re-iterating the actual sorting algorithms in these submissions. Therefore, I will be calling an assumed-to-be-existing sort function in each of these methods in order to demonstrate sorting. When this happens, I will indicate the O(N) runtime next to that particular line to demonstrate my understanding that this particular line is not a single operation.

Note that I will be providing a comparison function that will determine the order of the sort.

Vector

```
# Returns TRUE if course 1 is LESS THAN course 2

FUNCTION comparisonFunction PARAMETERS course1, course2

    RETURN course1.ID < course2.ID
END FUNCTION


FUNCTION PrintCourses PARAMETERS courseList

    SORT courseList USING FUNCTION quickSort(courseList, comparisonFunction) # Average: O(NlogN); worst case: O(N^2)

    FOR EACH course IN courseList

        PRINT course.ID

    END LOOP

END FUNCTION
```

Hash Table

```
FUNCTION comparisonFunction PARAMETERS key1, key2
    RETURN key1 < key2
END FUNCTION

FUNCTION PrintCourses PARAMETERS courseList
    GET keys IN courseList AS Vector # This is likely going to be implemented as a O(N) function unless keys are cached in the datastructure
    SORT keys USING FUNCTION quicksort(courseLIst, comparisonFunction) # Average: O(NlogN); Worst case: O(N^2)
    FOR EACH key IN keys
        SET course TO courseList.Get(key)
        PRINT course.ID
    END LOOP
END FUNCTION
```

Binary Tree

```
FUNCTION PrintCourses PARAMETERS courseList
    # It is assumed that the sorting of a binary tree will be performed upon insertion
    # Therefore we can just perform an in order walk of the tree using an InOrderIterator

    SET iterator TO courseList.GetInOrderIterator()
    WHILE iterator IS NOT EMPTY
        SET course TO iterator.GetItem()
        PRINT course.ID
        SET iterator TO iterator.Next()
    END LOOP
END FUNCTION
```

Resubmission and Runtime Analysis

NOTE TO INSTRUCTOR:

Rather than try to coral my code into an ugly table, I will be adding numbers to the end of each line in the format: “# LINE_COST NUM_EXECUTED TOTAL_COST”. At the end of each data structure, I will add everything together and give a highlighted total cost.

```
STRUCTURE Course
    String ID
    String Name
    Vector<String> prereqs
END STRUCTURE
```

Vector

# Print requested course information

```
FUNCTION printCourseInformation PARAMETERS coursesVector, courseId
    LOOP FOR EACH course IN coursesVector                   # 1 n n
        IF course.ID IS EQUAL TO coursed                    # 1 n n
            Print Course ID and Course Name                 # 1 1 1
            LOOP FOR EACH prereq IN course.prereqs          # 1 m m
                Print prereq                                # 1 m m
            END LOOP
        END IF
    END LOOP
END FUNCTION
```

Total Cost: 2n + 2m

Runtime: O(n + m) == O(n)

# Reading courselist from file into vector

```
FUNCTION ReadCoursesFromFile PARAMETERS filePath RETURNS Vector<Course>
    OPEN filePath AS file                                                    # 1 1 1

    IF file IS NOT VALID                                                     # 1 1 1
        THROW FileNotFoundError(filePath)                                    # 1 1 1
    END IF

    CREATE courses AS Vector<Course>  # courses read from file               # 1 1 1
    CREATE courseIdSet AS Set<String>  # set of course ids for validation    # 1 1 1
    SET currentLine = 1                                                      # 1 1 1

    LOOP FOR EACH line IN file                                               # 1 n n
        SET tokens TO line split by space (‘ ‘)                              # 1 n n
        IF tokens.size SMALLER THAN 2                                        # 1 n n
            THROW FileSyntaxError(currentLine)                               # 1 1 1
        END IF

        CREATE NEW Course AS course                                          # 1 n n
        course.ID = tokens[0]                                                # 1 n n
        course.Name = tokens[1]                                              # 1 n n

        LOOP FOR iterator START AT 2 WHILE iterator < tokens.size           # 1 m m
            ADD tokens[iterator] TO course.prereq Vector.                   # 1 m m
        END LOOP

        ADD course.ID TO courseIdSet                                         # 1 n n
        ADD course TO courses                                                # 1 n n
        INCREASE currentLine BY 1                                            # 1 n n

    END LOOP

    CLOSE file                                                               # 1 1 1
    LOOP FOR EACH course IN courses                                          # 1 n n
        LOOP FOR EACH prereq IN course.prereqs                              # 1 m m
            IF prereq NOT IN courseIdSet                                    # 1 m m
                THROW CoursePrerequisteError(course.ID)                      # 1 1 1
            END IF
        END LOOP
    END LOOP

    RETURN courses                                                           # 1 1 1
END FUNCTION        # Total Cost: 10n + 4m + 9 | Runtime: O(N)
```

Hash Table

# Print requested course information

```
FUNCTION printCourseInformation PARAMETERS coursesTable, courseId

    SET courseToPrint TO coursesTable[courseId]                       # 1 1 1

    IF courseToPrint IS NOT empty                                     # 1 1 1
        PRINT courseId and courseToPrint.name                         # 1 1 1

        LOOP FOR EACH prereq IN courseToPrint.prereqs                 # 1 m m
            Print prereq                                              # 1 m m
        END LOOP
    END IF
END FUNCTION
```

Total Time: 2m + 3
Runtime: O(N)

# Reading courselist from file into hash table

```
FUNCTION ReadCoursesFromFile PARAMETERS filePath RETURNS HashTable<Course>
    OPEN filePath AS file                                                # 1 1 1

    IF file IS NOT VALID                                                 # 1 1 1
        THROW FileNotFoundError(filePath)                                # 1 1 1
    END IF

    CREATE courses AS HashTable<String, Course>                          # 1 1 1
    SET currentLine = 1                                                  # 1 1 1

    LOOP FOR EACH line IN file                                           # 1 n n
        SET tokens TO line split by space (' ')                          # 1 n n
        IF tokens.size SMALLER THAN 2                                    # 1 n n
            THROW FileSyntaxError(currentLine)                           # 1 1 1
        END IF

        CREATE NEW Course AS course                                      # 1 n n
        course.ID = tokens[0]                                            # 1 n n
        course.Name = tokens[1]                                          # 1 n n

        LOOP FOR iterator START AT 2 WHILE iterator < tokens.size        # 1 m m
            ADD tokens[iterator] TO course.prereq Vector.                # 1 m m
        END LOOP

        ADD course TO courses WITH KEY course.ID                         # 1 n n
        INCREASE currentLine BY 1                                        # 1 n n

    END LOOP

    CLOSE file                                                           # 1 1 1
    LOOP FOR EACH key, course IN courses                                 # 1 n n
        LOOP FOR EACH prereq IN course.prereqs                           # 1 m m
            IF prereq NOT IN courses.GetKeys()                           # 1 m m
                THROW CoursePrerequisteError(course.ID)                  # 1 1 1
            END IF
        END LOOP
    END LOOP

    RETURN courses
END FUNCTION                # Total Cost: 9N + 4M + 8 | Runtime: O(N)
```

Binary Tree

# Print requested course information

```
FUNCTION printCourseInformation PARAMETERS coursesTree, coursed
    GET InOrder Iterator FROM coursesTree AS iter                          # 1 1 1

    WHILE iter IS VALID                                                    # 1 n n
        SET course to iter->GetItem()                                      # 1 n n

        IF course.ID IS EQUAL TO courseId                                  # 1 n n
            Print Course ID and Course Name                                # 1 1 1
            LOOP FOR EACH prereq IN course.prereqs                         # 1 m m
                Print prereq                                               # 1 m m
            END LOOP
            RETURN                                                         # 1 1 1
        END IF
        SET iter to iter->Next()                                           # 1 n n
    END LOOP
END FUNCTION
```

Total Cost: 4N + 2M + 3
Runtime: O(N)

(font size reduced to fit page)

```
FUNCTION ReadCoursesFromFile PARAMETERS filePath RETURNS BinaryTree<Course>
    OPEN filePath AS file                                                       # 1 1 1

    IF file IS NOT VALID                                                        # 1 1 1
        THROW FileNotFoundError(filePath)                                       # 1 1 1
    END IF

    CREATE courses AS BinaryTree <Course>     # courses read from file          # 1 1 1
    CREATE courseIdSet AS Set<String># set of course ids for validation         # 1 1 1
    SET currentLine = 1                                                         # 1 1 1

    LOOP FOR EACH line IN file                                                  # 1 n n
        SET tokens TO line split by space (' ')                                 # m n n
        IF tokens.size SMALLER THAN 2                                           # 1 n n
            THROW FileSyntaxError(currentLine)                                  # 1 1 1
        END IF

        CREATE NEW Course AS course                                             # 1 n n
        course.ID = tokens[0]                                                   # 1 n n
        course.Name = tokens[1]                                                 # 1 n n

        LOOP FOR iterator START AT 2 WHILE iterator < tokens.size               # 1 m m
            ADD tokens[iterator] TO course.prereq Vector.                       # 1 m m
        END LOOP

        ADD course.ID TO courseIdSet                                            # 1 n n
        INSERT course INTO courses                                           # logN n n
        INCREASE currentLine BY 1                                               # 1 n n

    END LOOP

    CLOSE file                                                                  # 1 1 1

    GET InOrder Iterator FROM coursesTree AS iter                               # 1 1 1

    WHILE iter IS VALID                                                         # 1 n n
        SET course to iter->GetItem()                                           # 1 n n

        LOOP FOR EACH prereq IN course.prereqs                                  # 1 m m
            IF prereq NOT IN courseIdSet                                        # 1 m m
                THROW CoursePrerequisteError(course.ID)                         # 1 1 1
            END IF
        END LOOP
        SET iter to iter->Next()                                                # 1 n n
    END LOOP

    RETURN courses                                                              # 1 1 1
END FUNCTION
```

Total Time: NlogN + 10N + 4M + 10 | Runtime: O(NlogN)

Advantages and Disadvantages

Data Structure	Advantages	Disadvantages
Vector	- Constant Time Appending - Constant time index-accessible - More easily searchable using common search algorithms that use indices to traverse - Contiguous memory which is more easily cached by the CPU	- Not sorted by default - No constant-time access with data-specific key
Hash Table	- Constant Time lookup by key (good for search) - Near-constant time insertion if properly balanced	- Unsortable - Performance degradation when full. - Not easily cached with chaining
Binary Tree	- Sorted by default - Search algorithms can be efficient due to default sorting - Ease of retrieving partitioned data. For example: all values less than X.	- No constant time insertion (price of sort on insertion) - No constant time lookup - When inserting sorted data, all operations degrade to O(N) worst-case time.

Recommendation

Why no BST?

The application is not doing any searching of data, which is the major strength of a Binary Search Tree (NlogN.) This being the case, the increased cost of insertion and inability to perform a constant-time lookup excludes the BST.

Why no vector?

For purposes of this application, a vector has no distinct advantages over a hash table. Insertion is constant O(1) time, and sorting is roughly (NlogN) – however looking up a course by ID is O(N) time as the data is stored unsorted. While an optimization could be performed to store the course list sorted, it would require resorting the list any time the list changed which would be costly in the long-term.

Hash Table!

All that being said, it is my recommendation that the application be implemented using hash tables as its core data structure. It matches vectors with it's (near) O(1) insertion time. In order to sort, it is only required to get the set of keys and sort those. Assuming we sort in place and don't require a copy of the data structure, this would give us a sort time of O(NlogN), matching vector. Finally, hash table has the key advantage of being able to query by a given key (courseID in our case) – which gives it the win in this analysis.