Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

Project One

I have included all the pseudocode first, I have used red comments for runtime notes (line time & function synopsis). The runtime analysis table, evaluation and recommendation are placed at the bottom of this document.

```
/* Vector */
struct Course{
	std::string CourseNo;
	std::string CourseDescription;
	Vector<std::string> PreReqs;
};

Course CreateCourseFromTokens(Vector<string> tokens){
	//Initialize the CourseNo and CourseDescription from the first two
tokens
	// .. The remaining tokens are pushed into a vector of PreReqs
	// C
	return Course {
		CourseNo = tokens[0]
		CourseDesciption = tokens[1]
		PreReqs = Vector<std::string>

		ForEach token[i] in tokens where i >= 2:
			Prereqs.append(token)
	};

}

void ParseFile(const CSV & File){
	// C
	Create CoursesVector to hold courses

	// Process each line of file
	ForEach line in File: // N
		Tokenize line as tokens vector // C

		// If there is not at least a course number and description
		// .. Then we do not add this line (Invalid format)
		// C
		If tokens.size < 2:
			Invalid entry, continue
		End if

		// Since we know this course at least has the required elements
		// .. We can create the course structure instance
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
            // .. Initialize the CourseNo and CourseDescription from the first
      two tokens
            // C
            CreateCourseFromTokens(Tokens)

            // Add the created Course to the CoursesVector
            // C
            CoursesVector.append (New Course)

      End ForEach // line in file

      // We need to check that all the pre-req classes exist
      // So we iterate every course, and we iterate every pre-req for every
course
      // .. If the course has a pre-req that does not exist in the
CourseVector
      // .. We remove the course.
      // In practice, we should notify the user that there is a missing class
      // It would also be a good idea to use a hashtable as a look up table
      // .. Rather than looping every course for every pre-req

      ForEach Course in CoursesVector: // N
            ForEach PreReq in Course.PreReqs: // N
                  if PreReq is not in CoursesVector: // N
                  Remove Course From CoursesVector
                  End if
            End ForEach // PreReq in Course.PreReqs
      End ForEach courseNumberourse in CourseVector

      // (5C + N * N^3) = N^3
}

void PrintCourse(Vector<Course> all_courses, std::string query){

      // Iterate every course and check if the CourseNo
      // .. matches the given query
      // This could be simpler with a Hashtable
      For Course & course in all_courses: // N
            if course.CourseNo == query:  // C
                  output CourseNo, Description, Pre-Reqs // C

      // 2C + N = N
}

// (Not part of the template)
Course GetCourseFromVec(Vector<Course> courses, string courseNumber){
      // Lookup course
      Course course = null // C
      For Course c in courses: // N
            If c.courseNumber == courseNumber // C
                  Return c
      Return null
      // 2C + N = N
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
}

// Returns the number of pre-requisite courses for a given course
int numPrerequisiteCourses(Vector<Course> courses, String courseNumber) {

      // Lookup course
       Course course = GetCourseFromVec(courses, courseNumber)

      // If course does not exist, return -1
      if course == null
      return -1

      // return number of pre-reqs
      return course.PreReqs.size()

}

// (Not part of the template, used for printSampleSchedule())
// Print re-reqs of a course recursively
void printPreReqsRecursive(Vector<Course> courses, Course course){

      // If this course has no pre-reqs
      // .. return
      if course.prereqs.size() <= 0
      return

      // For each pre-req this course has:
      //  - print it
      //  - call this function recursively
      ForEach String pre-req in prereqs
      print pre-req
      printPreReqsRecursive(GetCourseFromVec(courses, pre-req))
      end ForEach
}

// Given a course number, print a list of all courses that are needed
void printSampleSchedule(Hashtable<Course> courses, String courseNumber) {

      // lookup course
   Course course = GetCourseFromVec(courses, courseNumber)

      // if course does not exist, print error and return
   if course == null
      print error message
      return
      end if

      // recursively print course pre-reqs
      printPreReqsRecursive(course)
}

// print out the list of the courses Alphanumeric
Void PrintAllCoursesAlphaNumeric(Vector<Course> courses){
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
	// Start by sorting all the courses
	// I use the quicksort algorithm like we learned in Module 2

	// N Log N
	QuickSort(courses)
	// Then simply print the list
	// N
	For c in courses
		output CourseNo, Description, Pre-Reqs // C
	// C + N + N Log N = N Log N
}
/* Hash Table */

struct Course{
	std::string CourseNo;
	std::string CourseDescription;
	Vector<std::string> PreReqs;
};

Course CreateCourseFromTokens(Vector<string> tokens){
	//Initialize the CourseNo and CourseDescription from the first two
tokens
	// .. The remaining tokens are pushed into a vector of PreReqs
	// C
	return Course {
	CourseNo = tokens[0]
	CourseDesciption = tokens[1]
	PreReqs = Vector<std::string>

	ForEach token[i] in tokens where i >= 2:
		Prereqs.append(token)
	};
}

void ParseFile(const CSV & File, HashTable * hashTable){

	// Process each line of file
	// N
	ForEach line in File:
		Tokenize line as tokens vector // C

		// If there is not at least a course number and description
		// .. Then we do not add this line (Invalid format)
		// C
		If tokens.size < 2:
			Invalid entry, continue
		End if

		// Since we know this course at least has the required elements
		// .. We can create the course structure instance
		// C
		newCourse = CreateCourseFromTokens(tokens);
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
            // Insert course into hashtable
            // C
            hashTable.Insert(newCourse)

      End ForEach // line in file

      // We need to check that all the pre-req classes exist
      // .. So we iterate every course in the hashtable
      // .. For each course, we check each pre-req
      // If the pre-req does not exist as a key in the table
      // .. We then remove the course from the table
      ForEach Course course in hashTable: // N
            ForEach PreReq in course.PreReqs: // C
                  // If course has pre-req missing
                  if hashTable.Find(PreReq) == null // C

                  // Remove this course from the hash table
                  hashTable.Remove(course) // C

                  // break out of PreReq loop
                  // .. and proceed tot he next course
                  break

                  End if

            End ForEach // PreReq in course.PreReqs
      End ForEach courseNumberourse in hashTable
      // 6C + 2N = N
}

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

      // Get course from hashTable via Lookup
      // C
      Course course = courses.Search(courseNumber)

      // In case the query returns no value from the hashtable
      // .. print an error message and return
      if course == null // C
      print error message (missing key-value pair) // C
      return

      // Print the information from the course
      print course.CourseNo // C
      print course.CourseDesciption // C

      // Print each of the pre-reqs in the list for the course
      ForEach string & prereq in course.PreReqs // C
            print prereq // C
      // 7C = C

}
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
// Returns the number of pre-requisite courses for a given course
int numPrerequisiteCourses(Hashtable<Course> courses, String courseNumber) {

      // Lookup course
      Course course = courses.Search(courseNumber)

      // If course does not exist, return -1
      if course == null
      return -1

      // return number of pre-reqs
      return course.PreReqs.size()

}

// (Not part of the template, used for printSampleSchedule())
// Print re-reqs of a course recursively
void printPreReqsRecursive(Hashtable<Course> courses, Course course){

      // If this course has no pre-reqs
      // .. return
      if course.prereqs.size() <= 0
      return

      // For each pre-req this course has:
      //  - print it
      //  - call this functioon recursively
      ForEach String pre-req in prereqs
      print pre-req
      printPreReqsRecursive(course.Search(pre-req))
      end ForEach
}


// Given a course number, print a list of all courses that will needed
// .. based on the pre-reqs of each course
void printSampleSchedule(Hashtable<Course> courses, String courseNumber) {

      // lookup course
   Course course = courses.Search(courseNumber)

      // if course does not exist, print error and return
   if course == null
      print error message
      return
      end if

      // recursively print course pre-reqs
      printPreReqsRecursive(course)
}

// print out the list of the courses Alphanumeric
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
Void PrintAllCoursesAlphaNumeric(HashTable<Course> courses){

	// Start by sorting all the courses
	// I use the quicksort algorithm like we learned in Module 2
	// N Log N
	QuickSort(courses)

	// Then simply print the list
	// N
	For c in courses
		output CourseNo, Description, Pre-Reqs // C
	// C + N + N Log N = N Log N
}
```

```
/* Binary Tree */
struct Course{
	std::string CourseNo;
	std::string CourseDescription;
	Vector<std::string> PreReqs;
};


// These first two methods are for iterating through the tree
// .. this is extremely beneficial for when some operation needs
// .. to be performed on every node in the tree

// Recursive function for Tree::ForEach
// .. based on inOrder printing
void Tree::inOrderCall(Lambda<Course, void> func, Node node)
{
	// If we reach a null node, do nothing
	// This means we stop recursive calls from this node
	if (node == nullptr)
	return;

	// Recursive call left
	inOrder(node->left);

	courseNumberall function with current node's course as input
	func(node->course);

	// Recursive call right
	inOrder(node->right);
}
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
// Iterates each item in tree and calls some function
void Tree::ForEach(Lambda<Course, void> func)
{
	inOrerCall(func, tree->root);
}


Course CreateCourseFromTokens(Vector<string> tokens){
	//Initialize the CourseNo and CourseDescription from the first two
tokens
	// .. The remaining tokens are pushed into a vector of PreReqs
	// C
	return Course {
	CourseNo = tokens[0]
	CourseDesciption = tokens[1]
	PreReqs = Vector<std::string>

	ForEach token[i] in tokens where i >= 2:
		Prereqs.append(token)
	};
}

void ParseFile(const CSV & File, Tree tree){

	// Process each line of file
	ForEach line in File: // N
		Tokenize line as tokens vector // C

		// If there is not at least a course number and description
		// .. Then we do not add this line (Invalid format)
		// C
		If tokens.size < 2:
			Invalid entry, continue
		End if

		// Since we know this course at least has the required elements
		// .. We can create the course structure instance
		// C
		newCourse = CreateCourseFromTokens(tokens);

		// Insert course into Tree
		// C
		tree.Insert(newCourse)

	End ForEach // line in file

	// We need to check that all the pre-req classes exist
	// .. So we iterate every course in the tree (using the above ForEach
Method)
	// .. For each course, we check each pre-req
	// If the pre-req does not exist as a key in the tree
	// .. We then remove the course
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
	// Provide an anonymous lambda to the tree's foreach method
	// Log N
	Tree.ForEach({
		ForEach prereq in course.prereqs: // C
		if (tree.search(prereq) == null) // Log N
			Tree.remove(course) // C
	})
	// 6C + 2Log N + N = N
}

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

	// Get course from tree via Lookup
	// Log N
	Course course = courses.Search(courseNumber)

	// In case the query returns no value from the tree
	// .. print an error message and return
	// C
	if course == null
		print error message (missing key-value pair)
	return

	// Print the information from the course
	// C
	print course.CourseNo
	print course.CourseDesciption

	// Print each of the pre-reqs in the list for the course
	// C
	ForEach string & prereq in course.PreReqs
		print prereq
	// 3C + Log N = Log N
}



// Returns the number of pre-requisite courses for a given course
int numPrerequisiteCourses(Tree<Course> courses, String courseNumber) {

	// Lookup course
	Course course = courses.Search(courseNumber)

	// If course does not exist, return -1
	if course == null
	return -1

	// return number of pre-reqs
	return course.PreReqs.size()

}
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
// (Not part of the template, used for printSampleSchedule())
// Print re-reqs of a course recursively
void printPreReqsRecursive(Tree<Course> courses, Course course){

      // If this course has no pre-reqs
      // .. return
      if course.prereqs.size() <= 0
      return

      // For each pre-req this course has:
      //  - print it
      //  - call this functioon recursively
      ForEach String pre-req in prereqs
      print pre-req
      printPreReqsRecursive(course.Search(pre-req))
      end ForEach
}


// Given a course number, print a list of all courses that will needed
// .. based on the pre-reqs of each course
void printSampleSchedule(Tree<Course> courses, String courseNumber) {

      // lookup course
   Course course = courses.Search(courseNumber)

      // if course does not exist, print error and return
   if course == null
      print error message
      return
      end if

      // recursively print course pre-reqs
      printPreReqsRecursive(course)
}

// print out the list of the courses Alphanumeric
Void PrintAllCoursesAlphaNumeric(Tree<Course> courses){
      // Using the ForEach Method Defined above for the Tree
      // Since it already calls inOrder(...)
      // No sorting needed
      // N
      courses.ForEach([]=>
            {
            Output Course data // C
            })
      // C + N = N
}


/* General Code */

//Menu
```

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

```
Enum options{
      Load Data Structure = 0
      Print Course List = 1
      Print Course = 2
      Exit = -1
}

int main(...){

      While true{

            // Display options to user
            Output Options as strings

            // Get user selected option
            Option = user input

            // Switch based on user's choice
            Switch option{
                  // Exit while loop, exits program
                  Case exit:
                        Break;
                  // Loads data from csv file
                  Case Load Data:
                        ParseFile(...)
                  // Prints all courses
                  Case Print Course List:
                        PrintAllCoursesAlphaNumeric(...)
                  // Prints a single course
                  Case Print Course:
                        PrintCourse(...)
            }
      }
      Return 0;
}
```

Runtime Evaluation

Function	Vector	HashTable	BinaryTree
ParseFile	O(n^3)	O(n)	O(n)
PrintCourse	O(n)	O(1)	O(Log n)
PrintAllCoursesAlphaNumeric	O(n log n)	O(n Log n)	O(n)

Dante A Trisciuzzi
10.5.2022
CS-300-T1157 SNHU
Prof. Dewin Andujar

Evaluation & Recommendation

All three of these data structures have their advantages, though it's clear that the hash table and binary tree pull ahead in the runtime speed department. The vector data structure is the simplest to implement, as it is essentially a managed array; often called a list. Accessing individual elements is straightforward, and looping through all the elements is very simple. The HashTable is significantly more complex since the developer will be required to write an algorithm, however, it is unmatched for speed of accessing a single element; as long as the hashing algorithm has a time complexity of $< O(\log n)$. The hash table is nearly useless for iterating all elements though, as it provides no benefit over the vector for this purpose. Lastly, the Binary Tree has a level of complexity similar to the hash table, but provides great benefits in speed for both single-element access and complete iteration.

I recommend the Binary Search Tree for ABCU's computer department. The tree is fast and efficient, requires no sorting, and with some extension methods (like the ForEach(...) method defined above) very flexible.