Data Structure Run Time and Memory

Vector

Struct Course {

String courseNumber

String name

Vector<string> prerequisites

}

Vector<Course> readFile(string fileName) { Cost Times run

Vector<Course> courseList | 1 | 1

Ifstream file(fileName) | 1 | 1

If file didn’t open { | 1 | 1

output “could not open file” | 1 | 1

Return courseList | 1 | 1

}

String lineInFile | 1 | 1

While getline(file, lineInFile) { | 1 | n + 1

Course course | 1 | n

stringstream ss(lineInFile) | 1 | n

getline(ss, course.courseNumber, ‘,’) | 1 | n

if course.courseNumber is empty { | 1 | n

output invalid coursenumber | 1 | 1

Return courseList | 1 | 1

}

getline(ss, course.name, ‘,’) | 1 | n

if course.name is empty { | 1 | n

output invalid course name | 1 | 1

return courseList | 1 | 1

}

string prereq | 1 | n

while getline(ss, prereq, ‘,’) { | 1 + n – 1 (could technically have all courses before as prerequisites | n

if prereq in courseList.courseNumber { | n^2 (last in list) | n^3

course.prerequisite.push\_back(prereq) | 1 | n^3

}

Else {

output invalid prerequisite | 1 | 1

return courseList | 1 | 1

}

}

courseList.push\_back(course) | 1 | n

}

File.close() | 1 | 1

Return courseList | 1 | 1

}

Hash Table

Class HashTable {

Structure Node to hold a course, a key, and a pointer to the next node

A vector of nodes to hold the hashtable

Node\* InsertCourse(Course course, Node\* previousNodePtr) Function to insert a course into the hashtable: at the start of the key if the list is empty or at the end of the key if it isn’t. {

hash the course id to make a key

create a node and assign its key to the hashed key and course

if the vector of nodes at the key is empty {

assign the node to the vector at the key

}

else {

add the node to the end of the vector

}

previousNodePtr points to this node

return this nodes pointer

}

}

Struct Course {

String courseNumber

String name

Vector<string> prerequisites

}

HashTable readFile(string fileName) { Cost Times run

HashTable courseList | 1 | 1

Ifstream file(fileName) | 1 | 1

If file didn’t open { | 1 | 1

output “could not open file” | 1 | 1

Return courseList | 1 | 1

}

String lineInFile | 1 | 1

While getline(file, lineInFile) { | 1 | n + 1

Course course | 1 | n

stringstream ss(lineInFile) | 1 | n

getline(ss, course.courseNumber, ‘,’) | 1 | n

if course.courseNumber is empty { | 1 | n

output invalid coursenumber | 1 | 1

Return courseList | 1 | 1

}

getline(ss, course.name, ‘,’) | 1 | n

if course.name is empty { | 1 | n

output invalid course name | 1 | 1

return courseList | 1 | 1

}

string prereq | 1 | n

while getline(ss, prereq, ‘,’) { | 1 + n – 1 (could technically have all courses before as prerequisites | n^2

if prereq in courseList.courseNumber { | n^2 (all courses are in same key) | n^3

course.prerequisite.push\_back(prereq) | 1 | n^3

}

Else {

output invalid prerequisite | 1 | 1

return courseList | 1 | 1

}

}

InsertCourse(course, previousNodePtr) | n (all courses are in same key) | n

}

File.close() | 1 | 1

Return courseList | 1 | 1

}

Binary Search Tree

Class BST {

Tree root pointer

Structure Node to hold a course and a pointer to the left and right node

Void InsertCourse(Course course) Function to insert a course into the tree {

Set node pointer current to tree root

If current is null {

set root to new node(course)

return

}

While current is not null {

If courseId < current courseId {

If current left = null {

Current left = new node(course)

Current = null

}

Else {

Current = current left

}

}

Else {

If current right = null {

Current right = new node(course)

Current = null

}

Else {

Current = current right

}

}

}

}

}

Struct Course {

String courseNumber

String name

Vector<string> prerequisites

}

BST readFile(string fileName) { Cost Times run

BST courseList | 1 | 1

Ifstream file(fileName) | 1 | 1

If file didn’t open { | 1 | 1

output “could not open file” | 1 | 1

Return courseList | 1 | 1

}

String lineInFile | 1 | 1

While getline(file, lineInFile) { | 1 | n + 1

Course course | 1 | n

Stringstream ss(lineInFile) | 1 | n

getline(ss, course.courseNumber, ‘,’) | 1 | n

If course.courseNumber is empty { | 1 | n

output invalid coursenumber | 1 | 1

Return courseList | 1 | 1

}

getline(ss, course.name, ‘,’) | 1 | n

If course.name is empty { | 1 | n

output invalid course name | 1 | 1

return courseList | 1 | 1

}

string prereq | 1 | n

while getline(ss, prereq, ‘,’) { | 1 + n – 1 (could technically have all courses before as prerequisites | n^2

if prereq in courseList.courseNumber { | n^2 (if list is already sorted) | n^3

course.prerequisite.push\_back(prereq) | 1 | n^3

}

else {

output invalid prerequisite | 1 | 1

return courseList | 1 | 1

}

}

courseList.InsertCourse(course) | n (if list is already sorted) | n

}

File.close() | 1 | 1

Return courseList | 1 | 1

}