1.

Vectors

1.1.1

Class Course :

String code

String title

Vector<string> pre\_reqs

Vector<Course > courseInfo

F = open(argv[1])

For rec in F:

Line = rec.split(“,”)

If line.size() < 2:

Print(“error in file at line: ” + rec + “ , so skipping it”)

Continue

Code = line[0]

Name = line[1]

pRs = new vector<string>()

for I in range 2 to line.size():

pRs.push\_back(line[i])

courseInfo.push\_back( create\_new\_course( code, name, pRs) )

for Course c in courseInfo:

key = c.code

for pr in c.pre\_reqs :

found = false

for Course c2 in courseInfo:

key2 = c2.code

if key2 == prereq:

found = true;

break

if not found:

print(“Error in file!! Found a prerequisite that is not a listed course”)

1.1.2.

Def create\_new\_course(code, name, prs):

Course c = new Course()

c.code = code

c.name = name

c.pre\_reqs = prs

return c

1.1.3.

Def searchCourse( coursesVec, courseeCode):

For Course c in coursesVec:

If c.code == courseCode:

Return c

Print(“find which course code?”)

courseCode = input()

Course c = searchCourse( courseCode)

Print(c.code + “ “ + c.name + “ “)

For string s in c.pre\_reqs:

Print(s + “ “ )

Hash Tables

1.2.1

Class Course :

String code

String title

Vector<string> pre\_reqs

HashTable<string, Course > courseInfo

F = open(argv[1])

For rec in F:

Line = rec.split(“,”)

If line.size() < 2:

Print(“error in file at line: ” + rec + “ , so skipping it”)

Continue

Code = line[0]

Name = line[1]

pRs = new vector<string>()

for I in range 2 to line.size():

pRs.push\_back(line[i])

courseInfo.add(code , create\_new\_course( code, name, pRs) )

for key in courseInfo.keys():

c = courseInfo.get©

for pr in c.pre\_reqs :

if courseInfo.contains(pr) == false:

print(“Error in file!! Found a prerequisite that is not a listed course”)

1.2.2.

Def create\_new\_course(code, name, prs):

Course c = new Course()

c.code = code

c.name = name

c.pre\_reqs = prs

return c

1.2.3.

Def searchCourse( coursesHashTable, courseeCode):

Return coursesHashTable.get( courseCode)

Print(“find which course code?”)

courseCode = input()

Course c = searchCourse( courseCode)

Print(c.code + “ “ + c.name + “ “)

For string s in c.pre\_reqs:

Print(s + “ “ )

Binary Search Trees

1.3.1

Class Course :

String code

String title

Vector<string> pre\_reqs

Class CourseNode:

String code

Course c

Tree< CourseNode > courseInfo

F = open(argv[1])

For rec in F:

Line = rec.split(“,”)

If line.size() < 2:

Print(“error in file at line: ” + rec + “ , so skipping it”)

Continue

Code = line[0]

Name = line[1]

pRs = new vector<string>()

for I in range 2 to line.size():

pRs.push\_back(line[i])

courseInfo.add( create\_new\_course( code, name, pRs) )

for Course c in courseInfo.values():

for pr in c.pre\_reqs :

found = false

for c2 in courseInfo.values():

for pr in c.pre\_reqs :

if courseInfo.cintains(pr) == False:

print(“Error in file!! Found a prerequisite that is not a listed course”)

1.3.2.

Def create\_new\_course(code, name, prs):

Course c = new Course()

c.code = code

c.name = name

c.pre\_reqs = prs

n = new CourseNode()

n.code = code

n.course = c

return n

1.3.3.

Def searchCourse( courseTree, courseeCode):

Return courseTree.get( courseCode)

Print(“find which course code?”)

courseCode = input()

Course c = searchCourse( courseCode)

Print(c.code + “ “ + c.name + “ “)

For string s in c.pre\_reqs:

Print(s + “ “ )

2.

While True:

Print(“1) Load data structure”)

Print(“2) Print Course List”)

Print(“3) Print Course”)

Print(“4) Exit”)

userAnser = input()

if userAnswer == 1:

// load tree

Bst.loadCourses(“courses.txt”)

// load vector

V = new Vector()

loadCourses(v, “courses.txt”)

// load hash table

hashTable.loadCourses(“courses.txt”)

if userAnswer == 2 or userAnswer == 3:

if userAnswer == 3:

print(“which course?”)

courseChoice = input()

Print(“From which data structure?”)

Print(“a) tree”)

Print(“b) vector”)

Print(“c) hash table”)

userChoice = input()

if userChoice == “a”:

if userAnswer == 2:

bst.printAll()

else:

c = bst.search(courseChoice)

c.printCourse()

if userChoice == “b”:

for (course c in vector):

if userAnswr == 2 or (c.code == courseChoice):

c.printCourse()

if userChoice == “c”:

if userAnswer == 2:

hashTable.printAll()

else:

c = hashTable.search(courseChoice)

c.printCourse()

if userAnswer == 4:

break

3.

3.a

Vector:

For I in range 0 to courseInfo.size():

minCourse = courseInfo.get(i)

minLoc = i

for j in range I + 1 to courseInfo.size():

if minCourse.code < courseInfo.get(j).code

minLoc = j

minCourse = corseInfo.get(j)

courseInfo.set(j, courseInfo.get(i))

courseInfo.set(i, minCourse)

HashTable:

Course[] courses = new course[courseInfo.size]

Int put = 0

While put < courses.size():

lastCourse = null

minCourse = null

if put > 0:

lastCourse = courses[put – 1]

for key in courseInfo:

c = courseInfo.get(key)

if ( minCourse == null or minCourse > c) and(lastCourse == null or c < lastCourse) :

minCourse = c

courses[put] = minCourse

put ++

Trees:

Already sorted

3.b

Vector:

For I in range 0 to courseInfo.size():

courseInfo.get( i ).printCourse()

HastTable:

For I in range 0 to courses.length:

Courses[i].printCourse()

Trees:

coursesInfo.inOrderPrint();

4

4.a See above

4.b See above

4.c & 4.d:

Vector:

Class Course :

String code

String title

Vector<string> pre\_reqs

Vector<Course > courseInfo

//Reading in the courses:

F = open(argv[1]) {1 operation}

For rec in F: {loop repeated n times }

Line = rec.split(“,”) {max n operations}

If line.size() < 2: {1 operation}

Print(“error in file at line: ” + rec + “ , so skipping it”) {1 operation}

Continue {1 operation}

Code = line[0] {1 operation}

Name = line[1] {1 operation}

pRs = new vector<string>() {1 operation}

for I in range 2 to line.size(): {loop repeated max of n times }

pRs.push\_back(line[i]) {1 operation}

courseInfo.push\_back( create\_new\_course( code, name, pRs) ) {5 operations}

// Checking for errors

for Course c in courseInfo: {loop repeated n times }

key = c.code {1 operation}

for pr in c.pre\_reqs : {loop repeated max of n times}

found = false {1 operation}

for Course c2 in courseInfo: {loop repeated max of n times}

key2 = c2.code {1 operation}

if key2 == prereq: {1 operation}

found = true; {1 operation}

break {1 operation}

if not found:

print(“Error in file!! Found a prerequisite that is not a listed course”)

//creating a new course

Def create\_new\_course(code, name, prs):

Course c = new Course() {1 operation}

c.code = code {1 operation}

c.name = name {1 operation}

c.pre\_reqs = prs {1 operation}

return c {1 operation}

// sorting the courses for in order printing

For I in range 0 to courseInfo.size(): {loop repeated n times }

minCourse = courseInfo.get(i) {1 operation}

minLoc = I {1 operation}

for j in range I + 1 to courseInfo.size(): {loop repeated n times}

if minCourse.code < courseInfo.get(j).code {1 operation}

minLoc = j {1 operation}

minCourse = corseInfo.get(j) {1 operation}

courseInfo.set(j, courseInfo.get(i)) {1 operation}

courseInfo.set(i, minCourse) {1 operation}

HashTable:

Class Course :

String code

String title

Vector<string> pre\_reqs

HashTable<string, Course > courseInfo

//Reading in the file

F = open(argv[1])

For rec in F: {loop will repeat n times}

Line = rec.split(“,”) {max n operations}

If line.size() < 2: {1 operation}

Print(“error in file at line: ” + rec + “ , so skipping it”) {1 operation}

Continue {1 operation}

Code = line[0] {1 operation}

Name = line[1] {1 operation}

pRs = new vector<string>() {1 operation}

for I in range 2 to line.size(): {max n operations}

pRs.push\_back(line[i]) {1 operation}

courseInfo.add(code , create\_new\_course( code, name, pRs) ) {5 operations}

//Checking for errors

for key in courseInfo.keys(): {loop will repeat n times}

c = courseInfo.get( c ) {1 operation}

for pr in c.pre\_reqs : {loop will repeat max n times }

if courseInfo.contains(pr) == false: {1 operation }

print(“Error in file!! Found a prerequisite that is not a listed course”) {1 operation}

// creating new course

Def create\_new\_course(code, name, prs):

Course c = new Course() {1 operation}

c.code = code {1 operation}

c.name = name {1 operation}

c.pre\_reqs = prs {1 operation}

return c {1 operation}

// Sorting for printing

Course[] courses = new course[courseInfo.size]

Int put = 0

While put < courses.size(): {loop will repeat max n times }

lastCourse = null {1 operation}

minCourse = null {1 operation}

if put > 0: {1 operation}

lastCourse = courses[put – 1] {1 operation}

for key in courseInfo: {loop will repeat max n times }

c = courseInfo.get(key) {1 operation}

if ( minCourse == null or minCourse > c) and(lastCourse == null or c < lastCourse) :

minCourse = c {1 operation}

courses[put] = minCourse {1 operation}

put ++ {1 operation}

Trees:

Class Course :

String code

String title

Vector<string> pre\_reqs

Class CourseNode:

String code

Course c

Tree< CourseNode > courseInfo

//reading in the file

F = open(argv[1])

For rec in F: {loop will repeat n times}

Line = rec.split(“,”) {1 operation}

If line.size() < 2: {1 operation}

Print(“error in file at line: ” + rec + “ , so skipping it”) {1 operation}

Continue {1 operation}

Code = line[0] {1 operation}

Name = line[1] {1 operation}

pRs = new vector<string>() {1 operation}

for I in range 2 to line.size(): {n operations}

pRs.push\_back(line[i]) {1 operation}

courseInfo.add( create\_new\_course( code, name, pRs) ) {log(n) operations}

//checking for errors

for Course c in courseInfo.values(): {loop will repeat n times}

for pr in c.pre\_reqs : {loop will repeat n times}

if courseInfo.cintains(pr) == False: {log( n )operations}

print(“Error in file!! Found a prerequisite that is not a listed course”) {1 operation}

//creating a new course

Def create\_new\_course(code, name, prs):

Course c = new Course() {1 operation}

c.code = code {1 operation}

c.name = name {1 operation}

c.pre\_reqs = prs {1 operation}

n = new CourseNode() {1 operation}

n.code = code {1 operation}

n.course = c {1 operation}

return n {1 operation}

// sorting courses

--- Already Sorted ---

5.

Vector Analysis:

Reading in the courses: O(n^2)

Checking for errors: O(n^3)

Creating new course: O(1)

Sorting the courses for printing: O(n^2)

Advantages: vector has fast insertions because it uses an array behind the scenes

Disadvantage: Checking for errors is very slow because need to sequentially move through elements.

Hash Table Analysis:

Reading in the courses: O(n^2)

Checking for errors: O(n^2)

Creating new course: O(1)

Sorting the courses for printing: O(n^2)

Advangate: Insertions are very fast because we immediately hash to a specific location given a key to run the hash function on it. Checking for errors is also faster than it is for vectors because of this constant lookup time features.

Disadvantages: Sorting will be as slow as it is for Vectors because of the fact that we still need to visit each element.

Tree Analysis:

Reading in the courses: O(n^2)

Checking for errors: O(log(n)\*n^2)

Creating new course: O(1)

Sorting the courses for printing: O(1)

Advantages: No need for sorting. Due to the way that insertions are implemented, the tree is always in sorted order.

Disadvantage: Checking for errors will be faster than vector (because of the log(n) lookup time but still slower than the Hash Table because the hash table has constant lookup time.

6.

I would plan to use the Tree data structure, as printing in order is highly optimized given that no sorting has to be done. I expect that the print functionality will run much more often than the reading in functionality or the checking for errors functionality, the aforementioned two should only run once at the beginning of the program, however a user may wish to print the courses at any given time.