Vector

Struct Course{

String courseNumber

String courseName

Vector<string> prerequisites

}

LoadCourseData(string csvPath)

try:

File = open(csvPath)

Except:

print("Error: File not found")

Return

Vector<Course>courses= []

for line in file:

line = line.strip()

Fields = line.split(“,”)

Course course;

If len(fields) <= 1

Print Error

Else if len(fields) > 2

If field[3] is in courses-> courseNumber

add course vales

Else

Print Error

Break

Else

Add course values

Courses.pushback(course)

printCourseInformation(vector<Course> courses, string courseNumber)

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

PrintCourses(vector<Course> courses)

sortCourses(vector<Course> courses,int begin,int end)

for all courses

print out the course information

for each prerequisite of the course

print the prerequisite course information

Menu

vector<Course> courses;

LoadCourseData(string csvPath)

int choice = 0;

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Load Courses" << endl;

cout << " 2. Display All Courses" << endl;

cout << " 3. Display Course << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

courses = loadCourseData(csvPath);

print << courses.size() << " courses read" << endl;

break;

case 2:

printCourses( vector<Course> courses)

break;

case 3:

Print Input course number:

String courseNumber = input;

printCourseInformation(vector<Course> courses, string courseNumber)

break;

}

}

cout << "Good bye." << endl;

return 0;

}

Hash Table

Struct Course{

String courseNumber

String courseName

Vector<string> preRequisites

}

Const unsigned int DEFAULT\_SIZE = 8;

Class HashTable

Struct Node

Course\* course;

Node\*next;

Unsigned int key;

Node() ///default

Node(Course course, unsigned int key)

Vector<Node> nodes;

unsigned int tableSize = DEFAULT\_SIZE;

unsigned int hash(int key);

LoadCourseData(string csvPath)

try:

File = open(csvPath)

Except:

print("Error: File not found")

Return

for line in file:

line = line.strip()

Fields = line.split(“,”)

If len(fields) <= 1

Print Error

Else if len(fields) > 2

Vector<string> preReq

For (i=2;i < len(fields);++I)

If field[i] is in courses-> courseNumber

add pre req to preReq

Course newCourse

Add fields[0] to courseNumber

Add fields[1] to courseName

PreRequisites = preReq

Return newCourse

Else

Print Error

Break

Else

Course newCourse

Add course values

Return newCourse

PrintCourseInformation(Hashtable<Course> courses, String courseNumber) {

Key = hash(courseNumber)

searchCourse = courses.at(key)

While(searchCourse != nullptr)

IF searchCourse->courseNumber == courseNumber

print out course information

FOR each prerequisite of the course

print the prerequisite course information

ELSE

searchCourse = searchCourse ->next

PrintCourses(Hashtable<Course> courses){

sortCourses(Hashtable<Course> courses)

for (unsigned int i = 0; i < nodes.size(); i++) {

Node\* node = &nodes[i];

while (node != nullptr) {

Print courseName

Print courseNumber

FOR each prerequisite of the course

print the prerequisite course information

}

Node = node.next;

}

}

Menu

int main(int argc, char\* argv[]) {

string csvPath, bidKey;

HashTable\* courseTable;

Course course;

courseTable = new HashTable();

int choice = 0;

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Load Course" << endl;

cout << " 2. Display All Courses" << endl;

cout << " 3. Display Course" << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

loadBids(csvPath, courseTable);

break;

case 2:

courseTable->printCourses();

break;

case 3:

Print input course number

courseNumber = input

printCourseInformation(Hashtable<Course> courseTable, String courseNumber)

break;

}

}

cout << "Good bye." << endl;

return 0;

}

Tree

struct Course {

String courseNumber

String courseName

Vector<string> preRequisites

}

};

// Internal structure for tree node

struct Node {

Course course;

Node \*left;

Node \*right;

// default constructor

Node() {

left = nullptr;

right = nullptr;

}

// initialize with a bid

Node(Course aCourse) :

Node() {

course = aCourse;

}

LoadCourseData(string csvPath, BinarySearchTree bst)

try:

File = open(csvPath)

Except:

print("Error: File not found")

Return

for line in file:

line = line.strip()

Fields = line.split(“,”)

If len(fields) <= 1

Print Error

Else if len(fields) > 2

Vector<string> preReq

For (i=2;i < len(fields);++I)

If field[i] is in courses-> courseNumber

add pre req to preReq

Course newCourse

Add fields[0] to courseNumber

Add fields[1] to courseName

PreRequisites = preReq

Return newCourse

Else

Print Error

Break

Else

Course newCourse

Add course values

Return newCourse

bst -> Insert(newCourse)

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

currNode = root;

while (currNode != nullptr) {

if (currNode->course.courseNumber == courseNumber) {

return currNode->course;

}

else if (currNode->course.courseNumber > courseNumber) {

currNode = currNode->left;

}

else {

currNode = currNode->right;

}

}

Course course;

return course;

}

PrintCourses(Node\* node) {

if (node != nullptr) {

PrintCourse(node->left);

cout << node->course.courseNumber << "|" << node->course.courseName << "|"

FOR each prerequisite of the course

print the prerequisite course information

PrintCourse(node->right);

}

}

Menu

binarySearchTree\* bst;

bst = new BinarySearchTree();

Course course;

int choice = 0;

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Load Courses" << endl;

cout << " 2. Display All Courses" << endl;

cout << " 3. Display Course" << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

loadBids(csvPath, bst);

cout << bst->Size() << " bids read" << endl;

case 2:

bst->printCourses();

break;

case 3:

Print input course number;

courseNumber = input;

printCourseInformation(Tree<Course> courses , String courseNumber)

break;

}

}

cout << "Good bye." << endl;

return 0;

}

Runtime Evaluation

1. Vector

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **try: File = open(csvPath)** | 1 | 1 | 1 |
| **print("Error: File not found")**  **Return** | 1 | 1 | 1 |
| **for line in file:** | 1 | n | n |
| **line = line.strip()** | 1 | n | n |
| **Fields = line.split(“,”)** | 1 | n | n |
| **Course course;** | 1 | 1 | 1 |
| **If len(fields) <= 1** | 1 | n | n |
| **Print Error** | 1 | 1 | 1 |
| **Else if len(fields) > 2** | 1 | n | n |
| **If field[3] is in courses-> courseNumber** | 1 | n | n |
| **add course vales** | 1 | n | n |
| **Else**  **Print Error** | 1 | 1 | 1 |
| **Courses.pushback(course)** | 1 | 1 | 1 |
|  |  |  |  |
| **Total Cost** | | | 7n + 6 |
| **Runtime** | | | O(n) |

1. Hash Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **try: File = open(csvPath)** | 1 | 1 | 1 |
| **print("Error: File not found")**  **Return** | 1 | 1 | 1 |
| **for line in file:** | 1 | n | n |
| **line = line.strip()** | 1 | n | n |
| **Fields = line.split(“,”)** | 1 | n | n |
| **If len(fields) <= 1** | 1 | n | n |
| **Print Error** | 1 | 1 | 1 |
| **Else if len(fields) > 2** | 1 | n | n |
| **Vector<string> preReq** | 1 | 1 | 1 |
| **For (i=2;i < len(fields);++I)** | 1 | n-1 | n-1 |
| **If field[i] is in courses-> courseNumber** | 1 | n | n |
| **add pre req to preReq** | 1 | n | n |
| **Course newCourse** | 1 | 1 | 1 |
| **Add fields[0] to courseNumber** | 1 | n | n |
| **Add fields[1] to courseName** | 1 | n | n |
| **PreRequisites = preReq** | 1 | n | n |
| **Return newCourse** | 1 | n | n |
| **Else Print Error** | 1 | 1 | 1 |
| **Break** | 1 | 1 | 1 |
| **Else Course newCourse** | 1 | 1 | 1 |
| **Add course values** | 1 | n | n |
| **Return newCourse** | 1 | n | n |
|  |  |  |  |
| **Total Cost** | | | n-1....13n+8 |
| **Runtime** | | | O(N^2) |

1. Binary Tree

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **try: File = open(csvPath)** | 1 | 1 | 1 |
| **print("Error: File not found")**  **Return** | 1 | 1 | 1 |
| **for line in file:** | 1 | n | n |
| **line = line.strip()** | 1 | n | n |
| **Fields = line.split(“,”)** | 1 | n | n |
| **If len(fields) <= 1** | 1 | n | n |
| **Print Error** | 1 | 1 | 1 |
| **Else if len(fields) > 2** | 1 | n | n |
| **Vector<string> preReq** | 1 | 1 | 1 |
| **For (i=2;i < len(fields);++I)** | 1 | n-1 | n-1 |
| **If field[i] is in courses-> courseNumber** | 1 | n | n |
| **add pre req to preReq** | 1 | n | n |
| **Course newCourse** | 1 | 1 | 1 |
| **Add fields[0] to courseNumber** | 1 | n | n |
| **Add fields[1] to courseName** | 1 | n | n |
| **PreRequisites = preReq** | 1 | n | n |
| **Return newCourse** | 1 | n | n |
| **Else Print Error** | 1 | 1 | 1 |
| **Break** | 1 | 1 | 1 |
| **Else Course newCourse** | 1 | 1 | 1 |
| **Add course values** | 1 | n | n |
| **Return newCourse** | 1 | n | n |
| **Insert(newCourse)** | 1 | 1 | 1 |
| **Total Cost** | | | n-1....13n+9 |
| **Runtime** | | | O(N^2) |

Pro/Cons

1. Vector
   1. Pro
      1. Simple implementation
      2. Dynamic Resizing
      3. Constant Time Access
      4. Fast Append Operations
   2. Cons
      1. Slower Insertions and Deletions
      2. Linear Time Searching
      3. Increased Memory Usage
2. Hash Table
   1. Pros
      1. Quick search
      2. Quick Insertions and Deletions
      3. Efficient Memory Utilization
   2. Cons
      1. Non-Linear Ordering
      2. Increased Memory Usage
      3. The quality of hash table is influenced by hash function
3. Tree
   1. Pros
      1. Logarithmic Time Complexity
      2. Hierarchical structure
   2. Cons
      1. Slow compared to hash table
      2. Complex structure

Recommendation

My recommendation for the data structure to hold course information would be to use a vector. Vectors allow for constant time access to elements which would be low considering the number of courses that would need to be added. The vectors can also dynamically grow or be shortened depending on if any courses will need to be added or removed. It’s the simplest method as the file is inputted and the courses are appended to the end of the vector. This choice would be better than a binary tree because binary trees have logarithmic time complexity which can be slower for small data sets. It is also a complex structure that may not be needed for storing these courses. A vector may be more efficient than a hash table as well because when printing course in alphabetic order the process requires more operations.