# CS 300 6-2 Project One: Pseudocode

Robert Umland

Robert.Umland@SNHU.EDU

Southern New Hampshire University

**Pseudocode**

**Vector**

// Define a structure to hold course information

structure Course

string courseNumber

string title

vector<string> prerequisites

class CourseVector:

vector<Course> courses

void AddCourse(Course course):

courses.push\_back(course)

void PrintCourse(Course course):

Print "Course Number: " + course.courseNumber

Print "Title: " + course.title

Print "Prerequisites: "

for each prereq in course.prerequisites:

Print prereq

void SearchCourse(string courseNumber):

for each course in courses:

if course.courseNumber == courseNumber:

PrintCourse(course)

return

Print "Course not found"

void LoadCourses(string filename):

Open file for reading

if file is not open:

Print "Error opening file"

return

while not end of file:

Read line from file

if line is not valid format:

Print "Error: Invalid format"

continue

Parse line to extract course number, title, and prerequisites

Create a new Course object

Set course.courseNumber, course.title, and course.prerequisites

AddCourse(course)

Close file

void PrintAllCourses():

Sort courses by courseNumber

for each course in courses:

PrintCourse(course)

// Menu options for vector-based program

void Menu():

Initialize CourseVector vector

while true:

Print "1. Load courses"

Print "2. Print all courses"

Print "3. Print course details"

Print "9. Exit"

Read option

if option == 1:

Print "Enter filename: "

Read filename

vector.LoadCourses(filename)

else if option == 2:

vector.PrintAllCourses()

else if option == 3:

Print "Enter course number: "

Read courseNumber

vector.SearchCourse(courseNumber)

else if option == 9:

break

int main():

Menu()

return 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Line of Pseudocode** | **Line Cost** | **# Times Executes** | **Total Cost** | **Runtime** |
| **LoadCourses** | Open file for reading | 1 | 1 | 1 |  |
|  | While not end of file | 1 | n | n |  |
|  | Read line from file | 1 | n | n |  |
|  | Parse line to extract course number, title, and prerequisites | 1 | n | n |  |
|  | Create a new Course object | 1 | n | n |  |
|  | AddCourse(course) | 1 | n | n |  |
| **Total** |  |  |  | **5n + 1** | ***O*(n)** |
|  |  |  |  |  |  |
| **SearchCourse** | For each course in courses | 1 | n | n |  |
|  | If course.courseNumber == courseNumber | 1 | n | n |  |
|  | PrintCourse(course) | 1 | n | n |  |
|  | Return | 1 | n | n |  |
| **Total** |  |  |  | **4n** | ***O*(n)** |
|  |  |  |  |  |  |
| **PrintCourses** | Sort courses by course number | nlogn | 1 | nlogn |  |
|  | For each course in sorted courses | 1 | n | n |  |
|  | Print course information | 1 | n | n |  |
| **Total** |  |  |  | **nlogn+2n** | ***O*(nlogn)** |

**Hash Table**

// Define a structure to hold course information

structure Course

string courseNumber

string title

vector<string> prerequisites

class CourseHashTable:

hash\_table<string, Course> courseTable

void AddCourse(Course course):

unsigned int key = Hash(course.courseNumber)

if courseTable[key] is empty:

courseTable[key] = course

else:

node\* current = courseTable[key]

while current.next is not null:

current = current.next

current.next = new node(course)

void PrintCourse(Course course):

Print "Course Number: " + course.courseNumber

Print "Title: " + course.title

Print "Prerequisites: "

for each prereq in course.prerequisites:

Print prereq

void SearchCourse(string courseNumber):

unsigned int key = Hash(courseNumber)

node\* current = courseTable[key]

while current is not null:

if current.course.courseNumber == courseNumber:

PrintCourse(current.course)

return

current = current.next

Print "Course not found"

void LoadCourses(string filename):

Open file for reading

if file is not open:

Print "Error opening file"

return

while not end of file:

Read line from file

if line is not valid format:

Print "Error: Invalid format"

continue

Parse line to extract course number, title, and prerequisites

Create a new Course object

Set course.courseNumber, course.title, and course.prerequisites

AddCourse(course)

Close file

void PrintAllCourses():

vector<Course> courseVector

for each entry in courseTable:

if entry is not empty:

node\* current = entry

while current is not null:

courseVector.push\_back(current.course)

current = current.next

Sort courseVector by courseNumber

for each course in courseVector:

PrintCourse(course)

// Menu options for hash table-based program

void Menu():

Initialize CourseHashTable hashTable

while true:

Print "1. Load courses"

Print "2. Print all courses"

Print "3. Print course details"

Print "9. Exit"

Read option

if option == 1:

Print "Enter filename: "

Read filename

hashTable.LoadCourses(filename)

else if option == 2:

hashTable.PrintAllCourses()

else if option == 3:

Print "Enter course number: "

Read courseNumber

hashTable.SearchCourse(courseNumber)

else if option == 9:

break

int main():

Menu()

return 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Line of Pseudocode** | **Line Cost** | **# Times Executes** | **Total Cost** | **Runtime** |
| **LoadCourses** | Open file for reading | 1 | 1 | 1 |  |
|  | While not end of file | 1 | n | n |  |
|  | Read line from file | 1 | n | n |  |
|  | Parse line to extract course number, title, and prerequisites | 1 | n | n |  |
|  | Create a new Course object | 1 | n | n |  |
|  | InsertCourse(course) | 1 | n | n |  |
| **Total** |  |  |  | **5n + 1** | ***O*(n)** |
|  |  |  |  |  |  |
| **SearchCourse** | unsigned int key = Hash(courseNumber) | 1 | 1 | 1 |  |
|  | if courseTable[key] is empty | 1 | 1 | 1 |  |
|  | while current.next is not null | 1 | n | n |  |
|  | current = current.next | 1 | n | n |  |
| **Total** |  |  |  | **2 + 2n** | ***O*(n)** |
|  |  |  |  |  |  |
| **PrintCourses** | Create a list from hash table entries | 1 | n | n |  |
|  | Sort the list by course number | nlogn | 1 | nlogn |  |
|  | For each course in sorted list | 1 | n | n |  |
|  | Print course information | 1 | n | n |  |
| **Total** |  |  |  | **nlogn+2** | ***O*(nlogn)** |

**Binary Search Tree**

// Define a structure to hold course information

structure Course

string courseNumber

string title

vector<string> prerequisites

class Node:

Course course

Node left

Node right

class BinarySearchTree:

Node root

void Insert(Course course):

if root is null:

root = new Node(course)

else:

addNode(root, course)

void addNode(Node node, Course course):

if course.courseNumber < node.course.courseNumber:

if node.left is null:

node.left = new Node(course)

else:

addNode(node.left, course)

else:

if node.right is null:

node.right = new Node(course)

else:

addNode(node.right, course)

void PrintCourse(Course course):

Print "Course Number: " + course.courseNumber

Print "Title: " + course.title

Print "Prerequisites: "

for each prereq in course.prerequisites:

Print prereq

void SearchCourse(Node node, string courseNumber):

if node is null:

Print "Course not found"

return

if node.course.courseNumber == courseNumber:

PrintCourse(node.course)

else if courseNumber < node.course.courseNumber:

SearchCourse(node.left, courseNumber)

else:

SearchCourse(node.right, courseNumber)

void LoadCourses(string filename):

Open file for reading

if file is not open:

Print "Error opening file"

return

while not end of file:

Read line from file

if line is not valid format:

Print "Error: Invalid format"

continue

Parse line to extract course number, title, and prerequisites

Create a new Course object

Set course.courseNumber, course.title, and course.prerequisites

Insert(course)

Close file

void InOrderTraversal(Node node):

if node is not null:

InOrderTraversal(node.left)

PrintCourse(node.course)

InOrderTraversal(node.right)

void PrintAllCourses():

InOrderTraversal(root)

// Menu options for binary search tree-based program

void Menu():

Initialize BinarySearchTree tree

while true:

Print "1. Load courses"

Print "2. Print all courses"

Print "3. Print course details"

Print "9. Exit"

Read option

if option == 1:

Print "Enter filename: "

Read filename

tree.LoadCourses(filename)

else if option == 2:

tree.PrintAllCourses()

else if option == 3:

Print "Enter course number: "

Read courseNumber

tree.SearchCourse(tree.root, courseNumber)

else if option == 9:

break

int main():

Menu()

return 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Line of Pseudocode** | **Line Cost** | **# Times Executes** | **Total Cost** | **Runtime** |
| **LoadCourses** | Open file for reading | 1 | 1 | 1 |  |
|  | While not end of file | 1 | n | n |  |
|  | Read line from file | 1 | n | n |  |
|  | Parse line to extract course number, title, and prerequisites | 1 | n | n |  |
|  | Create a new Course object | 1 | n | n |  |
|  | bst.Insert(Course) | logn | n | nlogn |  |
| **Total** |  |  |  | **5n + nlogn + 1** | ***O*(nlogn)** |
|  |  |  |  |  |  |
| **SearchCourse** | bst.Find(courseNumber) | logn | 1 | logn |  |
|  | PrintCourseInfo(course) | 1 | 1 | 1 |  |
|  | Return | 1 | 1 | 1 |  |
| **Total** |  |  |  | **logn + 2n + 1** | ***O*(n)** |
|  |  |  |  |  |  |
| **PrintCourses** | InOrderTraversal(bst.root) | 1 | n | n |  |
|  | For each course in traversal | 1 | n | n |  |
|  | Print course information | 1 | n | n |  |
| **Total** |  |  |  | **3n** | ***O*(n)** |

**Discussion of Advantages and Disadvantages**

Each sort method has its use case where it’s the better choice and use case where it’s the worst choice. If we look at vectors, they’re easy to implement, easy for insertion of an item at the end of the list, and accessing data is fast. But vectors search sequentially, creating slow returns especially on large data sets. Also, adding data items in between other items requires shifting of the elements. Hash tables are very efficient for large data sets where quick access is required. Additionally, the key system allows for fast searching and printing. On the downside, hash tables do not allow for sorting of the table. For sorting as we’re doing, the elements must be pulled, stored into another data structure and then sorted. Also, the hash function must be robust when created to minimize collisions. Binary search trees are great for in-order traversal of the data set as by design, the elements are stored in sorted order. This allows for quick search and print of the data. Problems arise if the tree becomes unbalanced with degrading performance. Planning and initial work also needs to be done to help maintain the balance of the tree, possibly creating more complexity on the front-end

**Recommendation:** Based on the pros and cons listed in the last section and the specific requirements of the project, the Binary Search Tree (BST) is recommended for this application.