6-2 Project One

Tyten Perez

CS 300: Analysis and Design

Saba Jamaliannasrabadi

December 1, 2023

# Vector Pseudocode

Structure Course:

string courseNumber

string name

vector<string> prerequisites

vector<Course> LoadCoursesFromFile(string fileName)

File file = openFile(fileName)

If not file.isOpen():

print("Error opening file")

return vector<Course>()

vector<Course> courses

While not file.endOfFile():

string line = file.readLine()

If line is not empty:

ParseLine(line, courses)

file.close()

Return courses

void ParseLine(string line, vector<Course> &courses)

vector<string> tokens = split(line, ',')

If tokens.size() is less than 2:

print("Invalid line format")

Return

Course newCourse

newCourse.courseNumber = tokens[0]

newCourse.name = tokens[1]

For each token in tokens starting from index 2:

newCourse.prerequisites.push\_back(token)

courses.push\_back(newCourse)

void PrintCourseList(vector<Course> &courses)

vector<Course> sortedCourses = SortCoursesAlphanumerically(courses)

For each course in sortedCourses:

print(course.courseNumber + ": " + course.name)

void PrintCourseInformation(vector<Course> &courses, string courseNumber)

For each course in courses:

If course.courseNumber equals courseNumber:

print("Course Number: " + course.courseNumber)

print("Name: " + course.name)

If course.prerequisites is not empty:

print("Prerequisites: " + join(course.prerequisites, ", "))

Else:

print("No prerequisites")

Return

print("Course not found")

vector<Course> SortCoursesAlphanumerically(vector<Course> &courses)

// simple comparison-based sorting algorithm

int n = courses.size()

For i from 0 to n-1:

For j from 0 to n-i-1:

If courses[j].courseNumber > courses[j+1].courseNumber:

// Swap courses[j] and courses[j+1]

Course temp = courses[j]

courses[j] = courses[j+1]

courses[j+1] = temp

Return courses

void Menu()

vector<Course> courses

boolean running = true

While running:

print("1. Load Data Structure")

print("2. Print Course List")

print("3. Print Course")

print("4. Exit")

string choice = getUserInput()

Switch choice:

Case "1":

string fileName = "course\_data.txt"

courses = LoadCoursesFromFile(fileName)

Case "2":

If courses is empty:

print("Load data first")

Else:

PrintCourseList(courses)

Case "3":

If courses is empty:

print("Load data first")

Else:

string courseNumber = getUserInput("Enter course number: ")

PrintCourseInformation(courses, courseNumber)

Case "4":

running = false

Default:

print("Invalid choice")

int main()

Menu()

# HashMap Pseudocode

Structure Course:

string courseNumber

string name

vector<string> prerequisites

HashMap<string, Course> LoadCoursesFromFile(string fileName)

File file = openFile(fileName)

If not file.isOpen():

print("Error opening file")

return HashMap<string, Course>()

HashMap<string, Course> courses

While not file.endOfFile():

string line = file.readLine()

If line is not empty:

ParseLine(line, courses)

file.close()

Return courses

void ParseLine(string line, HashMap<string, Course> &courses)

vector<string> tokens = split(line, ',')

If tokens.size() is less than 2:

print("Invalid line format")

Return

Course newCourse

newCourse.courseNumber = tokens[0]

newCourse.name = tokens[1]

For each token in tokens starting from index 2:

newCourse.prerequisites.push\_back(token)

courses[newCourse.courseNumber] = newCourse

vector<string> SortCourseNumbersAlphanumerically(HashMap<string, Course> &courses)

vector<string> courseNumbers

// Extracting keys (course numbers) from hashmap

For each course in courses:

courseNumbers.push\_back(course.key)

// simple comparison-based sorting algorithm

int n = courseNumbers.size()

For i from 0 to n-1:

For j from 0 to n-i-1:

If courseNumbers[j] > courseNumbers[j+1]:

// Swap courseNumbers[j] and courseNumbers[j+1]

string temp = courseNumbers[j]

courseNumbers[j] = courseNumbers[j+1]

courseNumbers[j+1] = temp

Return courseNumbers

void PrintCourseList(HashMap<string, Course> &courses)

vector<string> sortedCourseNumbers = SortCourseNumbersAlphanumerically(courses)

For each courseNumber in sortedCourseNumbers:

Course course = courses[courseNumber]

print(course.courseNumber + ": " + course.name)

void PrintCourseInformation(HashMap<string, Course> &courses, string courseNumber)

If courses.contains(courseNumber):

Course course = courses[courseNumber]

print("Course Number: " + course.courseNumber)

print("Name: " + course.name)

If course.prerequisites is not empty:

print("Prerequisites: " + join(course.prerequisites, ", "))

Else:

print("No prerequisites")

Else:

print("Course not found")

void Menu(HashMap<string, Course> &courses)

boolean running = true

While running:

print("1. Load Data Structure")

print("2. Print Course List")

print("3. Print Course")

print("4. Exit")

string choice = getUserInput()

Switch choice:

Case "1":

string fileName = "course\_data.txt"

courses = LoadCoursesFromFile(fileName)

print("Data loaded successfully.")

Case "2":

If courses.isEmpty():

print("Load data first")

Else:

PrintCourseList(courses)

Case "3":

If courses.isEmpty():

print("Load data first")

Else:

string courseNumber = getUserInput("Enter course number: ")

PrintCourseInformation(courses, courseNumber)

Case "4":

running = false

Default:

print("Invalid choice")

int main()

HashMap<string, Course> courses

Menu(courses)

# BST Pseudocode

Structure Course:

string courseNumber

string name

vector<string> prerequisites

Structure TreeNode:

Course course

TreeNode left

TreeNode right

Structure BST:

TreeNode root

Procedure InsertCourse(BST &tree, Course course)

TreeNode newNode = new TreeNode(course)

If tree.root is null:

tree.root = newNode

Return

TreeNode current = tree.root

TreeNode parent = null

While current is not null:

parent = current

If course.courseNumber < current.course.courseNumber:

current = current.left

Else:

current = current.right

If course.courseNumber < parent.course.courseNumber:

parent.left = newNode

Else:

parent.right = newNode

BST LoadCoursesFromFile(string fileName)

File file = openFile(fileName)

BST coursesTree

If not file.isOpen():

print("Error opening file")

Return coursesTree

While not file.endOfFile():

string line = file.readLine()

If line is not empty:

Course course = ParseLine(line)

InsertCourse(coursesTree, course)

file.close()

Return coursesTree

Course ParseLine(string line)

vector<string> tokens = split(line, ',')

Course newCourse

If tokens.size() >= 2:

newCourse.courseNumber = tokens[0]

newCourse.name = tokens[1]

For i from 2 to tokens.size() - 1:

newCourse.prerequisites.push\_back(tokens[i])

Return newCourse

void InOrderTraversal(TreeNode node, Procedure action)

If node is not null:

InOrderTraversal(node.left, action)

action(node.course)

InOrderTraversal(node.right, action)

void PrintCourseList(BST &coursesTree)

InOrderTraversal(coursesTree.root, (Course course) => {

print(course.courseNumber + ": " + course.name)

})

void PrintCourseInformation(BST &coursesTree, string courseNumber)

TreeNode node = SearchCourse(coursesTree.root, courseNumber)

If node is not null:

Course course = node.course

print("Course Number: " + course.courseNumber)

print("Name: " + course.name)

If course.prerequisites is not empty:

print("Prerequisites: " + join(course.prerequisites, ", "))

Else:

print("No prerequisites")

Else:

print("Course not found")

TreeNode SearchCourse(TreeNode node, string courseNumber)

While node is not null:

If courseNumber equals node.course.courseNumber:

Return node

ElseIf courseNumber < node.course.courseNumber:

node = node.left

Else:

node = node.right

Return null

void Menu()

BST coursesTree

boolean running = true

While running:

print("1. Load Data Structure")

print("2. Print Course List")

print("3. Print Course")

print("4. Exit")

string choice = getUserInput()

Switch choice:

Case "1":

string fileName = "course\_data.txt"

coursesTree = LoadCoursesFromFile(fileName)

Case "2":

If coursesTree.root is null:

print("Load data first")

Else:

PrintCourseList(coursesTree)

Case "3":

If coursesTree.root is null:

print("Load data first")

Else:

string courseNumber = getUserInput("Enter course number: ")

PrintCourseInformation(coursesTree, courseNumber)

Case "4":

running = false

Default:

print("Invalid choice")

int main()

Menu()

# Complexity

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | HashMap | BST |
| Load Courses From File | O(n2) | Average O(n) Worst O(n2) | Average O(n log n)  Worst O(n2) |
| Print Course List | O(n2) | O(n2) | O(n) |
| Print Course Information | O(n) | Average O(1)  Worst O(n) | Average O(log n)  Worst O(n) |

# Recommendation

Each data structure has its own strengths in specific use cases. The vector-based implementation suffers from inefficiency in sorting and searching, making it less suitable for larger datasets. The hash map offers fast lookups but still struggles with sorting the courses alphanumerically because it currently uses a similar bubble sort algorithm that the vector data structure uses. The BST provides a balanced solution with efficient traversal and relatively good performance for search and insertion, particularly for small to medium-sized datasets. This makes BSTs a more suitable choice for managing a list of Computer Science courses, especially when considering the courses need to be sorted and the relatively small size of the data. Going forward this may be the best data structure to use in this project if the dataset remains small. However, If the dataset continues to scale, the HashMap data structure with an improved sorting algorithm would be the next recommendation.