Brad Jackson

CS300

August 8, 2021

Project One Pseudocode and Evaluation

Vector Pseudocode

```
int numPrerequisiteCourses(Vector<Course> courses, Course c) {
    totalPrerequisites = prerequisites of course c
    for each prerequisite p in totalPrerequisites
        add prerequisites of p to totalPrerequisites
    return number of totalPrerequisites
}

void printSampleSchedule(Vector<Course> courses) {
    coursesWithPrerequisites = empty list
    for each course c in courses
        if numPrerequisiteCourses of course c is zero
            print course c
        else
            append course c to coursesWithPrerequisites
    print coursesWithPrerequisites
}

void 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
}
```

Tree Pseudocode

```
int numPrerequisiteCourses(Tree<Course> courses) {
    int totalPrerequisites
    for each course c in courses keys
        if property value of course is prerequisite
            add prerequisites to totalPrerequisites
    print number of totalPrerequisites
}

void printSampleSchedule(Tree<Course> courses) {
    coursesWithPrerequisites = empty list
    for each course c in courses
        if numPrerequisiteCourses of course c is zero
            print course c
        else
            append course c to coursesWithPrerequisites
    print coursesWithPrerequisites
}
void printCourseInformation(Tree<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
}
```

Hashtable Pseudocode

```
int numPrerequisiteCourses(Hashtable<Course> courses) {
      int totalPrerequisites
      for each course c in courses keys
            if property value of course is prerequisite
                  add prerequisites to totalPrerequisites
      print number of totalPrerequisites
}

void printSampleSchedule(Hashtable<Course> courses) {
      create schedule hashtable = courses
      for all schedule
            if course has prerequisites
                  move prerequisites to begin of schedule
      for all schedule
            print keys of schedule
}

void printCourseInformation(Hashtable<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
```

Menu Pseudocode

```
int main(int argc, char *argv[]) {
    string csvPath, courseKey
    DataStruct *chosenDataStructObject;
    int choice = 0
    string csvPath, bidKey;
    switch (argc)
    {
        case 2:
            csvPath = argv[1];
            bidKey = "CSCI100";
            break;
        case 3:
            csvPath = argv[1];
            bidKey = argv[2];
            break;
        default:
            csvPath = "courseData.csv";
            bidKey = "CSCI100";
    }

    while (choice != 9)
    {
        cout << "Menu:" << endl;
        cout << "  1. Load Data Structure" << endl
        cout << "  2. Print Course List" << endl
        cout << "  3. Print Course" << endl
        cout << "  9. Exit" << endl
        cout << "Enter choice: "
        cin >> choice

        switch (choice)
        {
            case 1:
                courseFile = new DataStruct
                loadCourses(csvPath, courseFile)
            case 2:
                DataStruct->printInAlpha()
            case 3:
                printCourseInformation(courseKey)
        }
    }
    return 0
}
```

Print in Alphabetical Order Pseudocode

```
void printInAlpha(Vector<Course> courses) {
      for all courses
            if course number is lower than previous
                  swap current course with previous
      print newly ordered vector
}

void printInAlpha(Tree<Course> courses) {
      inorder = in-order traversal of courses
      print inorder
}

void printInAlpha(Hashtable<Course> courses) {
      Vector alphaOrder
      for all courses
            push current course to alphaOrder
      for all courses
            if course number is lower than previous
                  swap current course with previous
      print alphaOrder
}
```

Evaluation

Based on the advisor's requirements, each data structure has an advantage and disadvantage. Vectors keep track of their size and are resizable. This will be helpful when doing real-time updates to the data from a programming perspective. Accessing these objects is also easy because it acts like an array. The disadvantage of vectors is that they are slow to update. Hash tables are much more complicated but can be faster. In my example I converted my hash table to a vector in order to sort alphabetically. In my opinion this is not a very efficient implementation. Binary search trees are faster if we have multiple updates but sorting and moving objects within the BST consumes more time than a vector. BST traversals take O(n) time because every element is accessed. Lookups take O(log n) time. Updates/inserts take O(1) time. For ease of implementation and a balance between performance and speed, I would recommend using vectors for this application.

Example Runtime Analysis

Code	Line Cost	# Times Executes	Total Cost
`for all courses`	1	n	n
`if the course is the same as courseNumber`	1	n	n
`print out the course information`	1	1	1
`for each prerequisite of the course`	1	n	n
`print the prerequisite course information`	1	n	n
		Total Cost	4n + 1
		Runtime	O(n)