CS300\_MODULE\_6

6-2\_PROJECT ONE   
  
Date: 02/10/2023  
Name: Deonne Ludwig

SECTION 1.

CS300\_MODULE\_3\_3-3\_MILESTONE\_DLUDWIG  
VECTOR DATA STRUCTURE\*\*PSEUDOCODE\*\* *(REVISED)*

LOAD csv file via path to read

PARSE file check for

- at least two parameters on each line

- prereq must be a courseNumber in csvFile

IF csv file is good

return

ELSE return error

FORWARD DECLARATIONS

DEFINE STRUCTURE TO HOLD COURSES

IMPLEMENT QUICKSORT LOGIC

PARTITION VECTOR INTO LOW AND HIGH

COURSE LOW = BEGIN

COURSE HIGH = END

ESTABLISH MIDDLE ELEMENT AS PIVOT POINT

WHILE NOT DONE

WHILE COURSE LOW < COURSE PIVOT

++ COURSE LOW

WHILE COURSE PIVOT < COURSE HIGH

IF COURSE LOW >= COURSE HIGH

DONE = TRUE

ELSE SWAP COURSE LOW, COURSE HIGH

++ COURSE LOW

-- COURSE HIGH

RETURN COURSE HIGH

CLASS QUICKSORT

SET MIDPOINT EQUAL TO ZERO

IF COURSE BEGIN >= END

RETURN

MIDPOINT = PARTITION(COURSE, BEGIN, END)

RECURSIVELY SORT BEGIN TO MID

RECURSIVELY SORT MID+1 TO END

CLASS SELECTIONSORT

DEFINE MIN

CHECK SIZE OF COURSES VECTOR

FOR POSITION = 0; POSITION < SIZE - 1; ++ POSITION

MIN = POSITION

LOOP OVER REMAINING TO RIGHT OF POSITION

FOR I = POSITION + 1; I < SIZE; ++I

IF COURSE < MINIMUM

MINIMUM = I

IF MINIMUM != POSITION

SWAP COURSE POSITION, COUSE MINIMUM

PRINT COURSES VECTOR

FOR ALL COURSE

IF COURSE = COURSE NUMBER

PRINT COURSE NUMBER AND NAME

IF PREREQUISITE EXISTS

PRINT COURSE NUMBER

PRINT SAMPLESCHEDULE VECTOR

IF COURSE = COURSE NUMBER AND STUDENT ID

PRINT COURSE NUMBER AND NAME

IF PREREQUISITE EXISTS

PRINT COURSE NUMBER

CS300\_MODULE 4 | 4-3\_MILESTONE\_DLUDWIG  
HASHTABLE DATA STRUCTURE\*\*PSEUDOCODE\*\*

LOAD csv file via path to read

PARSE file check for

- at least two parameters on each line

- prereq must be a courseNumber in csvFile

IF csv file is good

return

ELSE return error

SET DEFAULT SIZE

FORWARD DECLARATIONS

DEFINE STRUCTURE

CLASS HASHTABLE

DEFINE STRUCTURE TO HOLD COURSES

DEFAULT CONSTRUCTOR

INITIALIZE WITH COURSE

INITIALIZE COURSE WITH KEY

DEFAULT CONSTRUCTOR HASHTABLE

RESIZE TABLE SIZE

DESTRUCTOR HASHTABLE

ERASE TO FREE STORAGE

CALCULATE HASH VALUE

RETURN

ADD NEW COURSE HASHTABLE

CREATE NEW KEY

RETRIEVE NODE USING KEY

IF NO ENTRY FOR KEY

ASSIGN NEW NOD TO THE KEY

ELSE

IF OLD NODE KEY == MAX

OLDNODE->KEY = KEY

OLDNODE->COURSE = COURSE

OLDNODE->NEXT = NULL POINTER

ELSE

WHILE OLDNODE->NEXT != NULL POINTER

FIND THE NEXT OPEN NODE

OLDNODE = OLDNODE->NEXT

ADD NEW NODE TO END

OLDNODE->NEXT = NEW NODE(COURSE, KEY)

PRINT COURSES HASHTABLE

FOR ALL COURSE

IF KEY != MAX

PRINT COURSE NUMBER AND NAME

NODE = NEXT

WHILE NODE != NULL

PRINT COURSE NUMBER AND NAME

NODE = NODE->NEXT

IF PREREQUISITE EXISTS

PRINT COURSE NUMBER

NODE = NEXT

WHILE NODE != NULL

PRINT COURSE NUMBER

NODE = NODE->NEXT

PRINT SAMPLESCHEDULE HASHTABLE

IF COURSE = COURSE NUMBER AND STUDENT ID

PRINT COURSE NUMBER AND NAME

NODE = NEXT

WHILE NODE != NULL

PRINT COURSE NUMBER AND NAME

NODE = NODE->NEXT

IF PREREQUISITE EXISTS

PRINT COURSE NUMBER

NODE = NEXT

WHILE NODE != NULL

PRINT COURSE NUMBER

NODE = NODE->NEXT

CS300\_MODULE 5 | 5-3\_MILESTONE\_DLUDWIG  
TREE DATA STRUCTURE\*\*PSEUDOCODE\*\*

LOAD csv file via path to read

PARSE file check for

- at least two parameters on each line

- prereq must be a courseNumber in csvFile

IF csv file is good

return

ELSE return error

FORWARD DECLARATIONS

DEFINE STRUCTURE TO HOLD COURSES

DEFINE INTERNAL STRUCTURE FOR TREE NODES (KEY, LEFT, RIGHT)

DEFAULT CONSTRUCTOR (LEFT AND RIGHT POINT TO NULL)

INITIALIZE COURSE WITH KEY

CLASS BINARY SEARCH TREE

DEFAULT CONSTRUCTOR

DESTRUCTOR

INORDER TRAVERSAL

POSTORDER TRAVERSAL

PREORDER TRAVERSAL

ADD NEW COURSE BINARY SEARCH TREE

CREATE NEW KEY

IF COMPARE KEY >0 FOR LEFT

IF NO ENTRY FOR KEY

ASSIGN NEW NODE TO THE KEY

ELSE

RECURSE DOWN LEFT NODE

ELSE

IF NO RIGHT NODE

NODE BECOMES RIGHT

ELSE

RECURSE DOWN RIGHT NODE

PRINT COURSES BINARY SEARCH TREE

FOR ALL COURSE INORDER

IF COURSE = COURSE NUMBER

INORDER NODE-> LEFT

PRINT COURSE NUMBER AND NAME

INORDER NODE RIGHT

IF PREREQUISITE EXISTS

INORDER NODE-> LEFT

PRINT COURSE NUMBER

INORDER NODE RIGHT

PRINT SAMPLESCHEDULE BINARY SEARCH TREE

IF COURSE = COURSE NUMBER AND STUDENT ID

INORDER NODE-> LEFT

PRINT COURSE NUMBER AND NAME

INORDER NODE RIGHT

IF PREREQUISITE EXISTS

INORDER NODE-> LEFT

PRINT COURSE NUMBER

INORDER NODE RIGHT

SECTION 2.

CS300\_MODULE 6 | 6-2\_PROJECT\_ONE\_DLUDWIG  
MENU\*\*PSEUDOCODE\*\*

int choice = 0;

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:

LOAD CSV FILE

CASE 2:

Sort courseS HIGH TO LOW by alphanumeric COURSE number

PRINT COURSES

CASE 3:

PRINT COURSE TITLE AND PREREQUISITE

COUT << “GOOD BYE” << ENDL;

SECTION 3.

CS300\_MODULE 6 | 6-2\_PROJECT\_ONE\_DLUDWIG  
SORT AND PRINT\*\*PSEUDOCODE\*\*

sort(courses begin, courses end)

cout << sorted alphanumeric course list << endl;

SECTION 4.

CS300\_MODULE 6 | 6-2\_PROJECT\_ONE\_DLUDWIG  
RUN-TIME AND MEMORY

**VECTOR:**

| **CODE** | **LINE COST** | **# TIMES EXECUTES** | **TOTAL COST** |
| --- | --- | --- | --- |
| LOAD csv file via path to read | 1 | n | n |
| PARSE file - at least two parameters on each line | 1 | n | n |
| PARSE file - prereq must be a courseNumber in csvFile | 1 | n | n |
| IF csv file is good - return | 1 | 1 | 1 |
| ELSE return error | 1 | 1 | 1 |
| FORWARD DECLARATIONS | 1 | n | n |
| DEFINE STRUCTURE TO HOLD COURSES | 1 | n | n |
| PARTITION VECTOR INTO LOW AND HIGH | 1 | n | n |
| COURSE LOW = BEGIN | 1 | n | n |
| COURSE HIGH = END | 1 | n | n |
| ESTABLISH MIDDLE ELEMENT AS PIVOT POINT | 1 | n | n |
| WHILE NOT DONE | 1 | n | n |
| WHILE COURSE LOW < COURSE PIVOT | 1 | n | n |
| ++ COURSE LOW | 1 | n | n |
| WHILE COURSE PIVOT < COURSE HIGH | 1 | n | n |
| IF COURSE LOW >= COURSE HIGH | 1 | n | n |
| DONE = TRUE | 1 | n | n |
| ELSE SWAP COURSE LOW, COURSE HIGH | 1 | n | n |
| ++ COURSE LOW | 1 | n | n |
| -- COURSE HIGH | 1 | n | n |
| RETURN COURSE HIGH | 1 | 1 | 1 |
| Total Cost | | | 18n + 3 |
| Runtime | | | O(n) |

**HASHTABLE:**

| **CODE** | **LINE COST** | **# TIMES EXECUTES** | **TOTAL COST** |
| --- | --- | --- | --- |
| LOAD csv file via path to read | 1 | n | n |
| PARSE file - at least two parameters on each line | 1 | n | n |
| PARSE file - prereq must be a courseNumber in csvFile | 1 | n | n |
| IF csv file is good - return | 1 | 1 | 1 |
| ELSE return error | 1 | 1 | 1 |
| SET DEFAULT SIZE | 1 | n | n |
| FORWARD DECLARATIONS | 1 | n | n |
| DEFINE STRUCTURE | 1 | n | n |
| CLASS HASHTABLE | 1 | n | n |
| DEFINE STRUCTURE TO HOLD COURSES | 1 | n | n |
| DEFAULT CONSTRUCTOR | 1 | n | n |
| INITIALIZE WITH COURSE | 1 | n | n |
| INITIALIZE COURSE WITH KEY | 1 | n | n |
| Total Cost | | | 11n + 2 |
| Runtime | | | O(n) |

**TREE:**

| **CODE** | **LINE COST** | **# TIMES EXECUTES** | **TOTAL COST** |
| --- | --- | --- | --- |
| LOAD csv file via path to read | 1 | n | n |
| PARSE file - at least two parameters on each line | 1 | n | n |
| PARSE file - prereq must be a courseNumber in csvFile | 1 | n | n |
| IF csv file is good - return | 1 | 1 | 1 |
| ELSE return error | 1 | 1 | 1 |
| FORWARD DECLARATIONS | 1 | n | n |
| DEFINE STRUCTURE TO HOLD COURSES | 1 | n | n |
| DEFINE INTERNAL STRUCTURE FOR TREE NODES (KEY, LEFT, RIGHT) | 1 | n | n |
| DEFAULT CONSTRUCTOR (LEFT AND RIGHT POINT TO NULL) | 1 | n | n |
| INITIALIZE COURSE WITH KEY | 1 | n | n |
| Total Cost | | | 8n + 2 |
| Runtime | | | O(n) |

SECTION 5.

CS300\_MODULE 6 | 6-2\_PROJECT\_ONE\_DLUDWIG  
ANALYSIS

abc university WANTS SOFTWARE THAT WILL HELP THEIR COMPUTER SCIENCE ADVISORS ACCESS COURSE INFORMATION AND THEIR PREREQUISITES FOR THEIR STUDENTS.

CSCI100,INTRODUCTION TO COMPUTER SCIENCE

CSCI101,INTRODUCTION TO PROGRAMMING IN C++,CSCI100

CSCI200,DATA STRUCTURES,CSCI101

MATH201,DISCRETE MATHEMATICS

CSCI300,INTRODUCTION TO ALGORITHMS,CSCI200,MATH201

CSCI301,ADVANCED PROGRAMMING IN C++,CSCI101

CSCI350,OPERATING SYSTEMS,CSCI300

CSCI400,LARGE SOFTWARE DEVELOPMENT,CSCI301,CSCI350

Using a vector data structure allows you to keep a dynamically sized, ordered list of items in a linear arrangement that can be searched quickly for any element. Memory may suffer and it may be more difficult to change.

using a hashtable data structure allows data to be stored with a key, is very efficient, and handles searching, insertion, and deletion equally as well. it is space efficient and resizable. Collisions need to be avoided, capacity is limited and order of elements is not maintained.

using a treedata structure allows fast retrieval and allows insertion, deletion, and searching although deleting may be more complex. the elements are stored in a hierarchical format and allows for storage of unlimited nodes.

SECTION 6.

CS300\_MODULE 6 | 6-2\_PROJECT\_ONE\_DLUDWIG  
RECOMMENDATION

the way that abc university wants to utilize the software I believe a vector data structure would be the simplest application. if theY wanted to retrieve more detailed INFORMATION, i would POSSIBLY move to a tree structure.