Kevin Tippin

CS – 300 DSA: Analysis and Design

April 12th  2025

CS 300 Pseudocode Document

1. Vector Implementation

Pseudocode for Vector - Milestone 1

CopyFUNCTION main()  
 DECLARE courses AS VECTOR  
 WHILE TRUE  
 PRINT "1. Load course data"  
 PRINT "2. Print all courses"  
 PRINT "3. Print course info"  
 PRINT "9. Exit"  
 INPUT option  
  
 IF option == 1 THEN  
 LOAD\_COURSES(courses)  
 ELSE IF option == 2 THEN  
 PRINT\_COURSES(courses)  
 ELSE IF option == 3 THEN  
 PRINT\_COURSE\_INFO(courses)  
 ELSE IF option == 9 THEN  
 BREAK  
 ENDIF  
 ENDWHILE  
END FUNCTION  
  
FUNCTION LOAD\_COURSES(courses AS VECTOR)  
 OPEN file "courses.txt"  
 WHILE NOT end of file  
 DECLARE line AS STRING  
 READ line  
 IF NOT VALID\_FORMAT(line) THEN  
 PRINT "Error: Invalid course format"  
 CONTINUE  
 ENDIF  
 DECLARE course AS COURSE  
 course = PARSE\_COURSE(line)  
 APPEND course TO courses  
 ENDWHILE  
 CLOSE file  
END FUNCTION  
  
FUNCTION PRINT\_COURSES(courses AS VECTOR)  
 SORT courses BY course number  
 FOR EACH course IN courses  
 PRINT course.number + ": " + course.title  
 ENDFOR  
END FUNCTION  
  
FUNCTION PRINT\_COURSE\_INFO(courses AS VECTOR)  
 PRINT "Enter course number:"  
 INPUT course\_number  
 FOR EACH course IN courses  
 IF course.number == course\_number THEN  
 PRINT course.title + " Prerequisites: " + course.prerequisites  
 RETURN  
 ENDIF  
 ENDFOR  
 PRINT "Course not found."  
END FUNCTION  
  
FUNCTION VALID\_FORMAT(line AS STRING) AS BOOLEAN  
 // Implementation for format validation.  
END FUNCTION  
  
FUNCTION PARSE\_COURSE(line AS STRING) AS COURSE  
 // Implementation for parsing the line into a COURSE object.  
END FUNCTION  
  
STRUCTURE COURSE  
 STRING number  
 STRING title  
 STRING prerequisites  
END STRUCTURE

2. Hash Table Implementation

Pseudocode for Hash Table –Milesttone 2

CopyFUNCTION main()  
 DECLARE courses AS HASH TABLE  
 WHILE TRUE  
 PRINT "1. Load course data"  
 PRINT "2. Print all courses"  
 PRINT "3. Print course info"  
 PRINT "9. Exit"  
 INPUT option  
  
 IF option == 1 THEN  
 LOAD\_COURSES(courses)  
 ELSE IF option == 2 THEN  
 PRINT\_COURSES(courses)  
 ELSE IF option == 3 THEN  
 PRINT\_COURSE\_INFO(courses)  
 ELSE IF option == 9 THEN  
 BREAK  
 ENDIF  
 ENDWHILE  
END FUNCTION  
  
FUNCTION LOAD\_COURSES(courses AS HASH TABLE)  
 OPEN file "courses.txt"  
 WHILE NOT end of file  
 DECLARE line AS STRING  
 READ line  
 IF NOT VALID\_FORMAT(line) THEN  
 PRINT "Error: Invalid course format"  
 CONTINUE  
 ENDIF  
 DECLARE course AS COURSE  
 course = PARSE\_COURSE(line)  
 INSERT courses WITH key = course.number AND value = course  
 ENDWHILE  
 CLOSE file  
END FUNCTION  
  
FUNCTION PRINT\_COURSES(courses AS HASH TABLE)  
 DECLARE sorted\_courses AS VECTOR  
 FOR EACH key IN courses KEYS  
 APPEND courses[key] TO sorted\_courses  
 ENDFOR  
 SORT sorted\_courses BY course number  
 FOR EACH course IN sorted\_courses  
 PRINT course.number + ": " + course.title  
 ENDFOR  
END FUNCTION  
  
FUNCTION PRINT\_COURSE\_INFO(courses AS HASH TABLE)  
 PRINT "Enter course number:"  
 INPUT course\_number  
 IF course\_number IN courses THEN  
 DECLARE course AS COURSE  
 course = courses[course\_number]  
 PRINT course.title + " Prerequisites: " + course.prerequisites  
 ELSE  
 PRINT "Course not found."  
 ENDIF  
END FUNCTION  
  
FUNCTION VALID\_FORMAT(line AS STRING) AS BOOLEAN  
 // Implementation for format validation.  
END FUNCTION  
  
FUNCTION PARSE\_COURSE(line AS STRING) AS COURSE  
 // Implementation for parsing the line into a COURSE object.  
END FUNCTION  
  
STRUCTURE COURSE  
 STRING number  
 STRING title  
 STRING prerequisites  
END STRUCTURE

3. Tree Implementation

Pseudocode for Tree – Milestone 3

CopyFUNCTION main()  
 DECLARE courses AS BINARY SEARCH TREE  
 WHILE TRUE  
 PRINT "1. Load course data"  
 PRINT "2. Print all courses"  
 PRINT "3. Print course info"  
 PRINT "9. Exit"  
 INPUT option  
  
 IF option == 1 THEN  
 LOAD\_COURSES(courses)  
 ELSE IF option == 2 THEN  
 PRINT\_COURSES(courses)  
 ELSE IF option == 3 THEN  
 PRINT\_COURSE\_INFO(courses)  
 ELSE IF option == 9 THEN  
 BREAK  
 ENDIF  
 ENDWHILE  
END FUNCTION  
  
FUNCTION LOAD\_COURSES(courses AS BINARY SEARCH TREE)  
 OPEN file "courses.txt"  
 WHILE NOT end of file  
 DECLARE line AS STRING  
 READ line  
 IF NOT VALID\_FORMAT(line) THEN  
 PRINT "Error: Invalid course format"  
 CONTINUE  
 ENDIF  
 DECLARE course AS COURSE  
 course = PARSE\_COURSE(line)  
 INSERT courses WITH node = course  
 ENDWHILE  
 CLOSE file  
END FUNCTION  
  
FUNCTION PRINT\_COURSES(courses AS BINARY SEARCH TREE)  
 DECLARE sorted\_courses AS VECTOR  
 IN\_ORDER\_TRAVERSAL(courses, sorted\_courses)  
 FOR EACH course IN sorted\_courses  
 PRINT course.number + ": " + course.title  
 ENDFOR  
END FUNCTION  
  
FUNCTION IN\_ORDER\_TRAVERSAL(node, sorted\_courses AS VECTOR)  
 IF node IS NOT NULL THEN  
 IN\_ORDER\_TRAVERSAL(node.left, sorted\_courses)  
 APPEND node.course TO sorted\_courses  
 IN\_ORDER\_TRAVERSAL(node.right, sorted\_courses)  
 ENDIF  
END FUNCTION  
  
FUNCTION PRINT\_COURSE\_INFO(courses AS BINARY SEARCH TREE)  
 PRINT "Enter course number:"  
 INPUT course\_number  
 DECLARE course AS COURSE  
 course = SEARCH(courses, course\_number)  
 IF course IS NOT NULL THEN  
 PRINT course.title + " Prerequisites: " + course.prerequisites  
 ELSE  
 PRINT "Course not found."  
 ENDIF  
END FUNCTION  
  
FUNCTION SEARCH(node, course\_number) AS COURSE  
 IF node IS NULL OR node.course.number == course\_number THEN  
 RETURN node.course  
 ELSE IF course\_number < node.course.number THEN  
 RETURN SEARCH(node.left, course\_number)  
 ELSE  
 RETURN SEARCH(node.right, course\_number)  
 ENDIF  
END FUNCTION  
  
FUNCTION VALID\_FORMAT(line AS STRING) AS BOOLEAN  
 // Implementation for format validation.  
END FUNCTION  
  
FUNCTION PARSE\_COURSE(line AS STRING) AS COURSE  
 // Implementation for parsing the line into a COURSE object.  
END FUNCTION  
  
STRUCTURE COURSE  
 STRING number  
 STRING title  
 STRING prerequisites  
END STRUCTURE

Runtime Analysis

Big O Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Table** | **Binary Search Tree** |
| Loading Courses from File | O(n) | O(n) | O(n) |
| Creating Course Object | O(1)per course | O(1)per course | O(1)per course |
| Printing Course Info | O(n) | O(1) | O(log(n)) |
| Printing Sorted Course List | O(n log(n)) | O(n) | O(n) |

Advantages and Disadvantages

1. **Vector:**
   1. **Advantages:** Simple to implement; dynamic sizing.
   2. **Disadvantages:** Slower for searching; requires sorting for ordered operations.
2. **Hash Table:**
   1. **Advantages:** Fast access time; O(1) for lookups.
   2. **Disadvantages:** Collisions can slow down performance; requires extra space.
3. **Binary Search Tree:**
   1. **Advantages:** Efficient for dynamic data; sorted data can be printed easily.
   2. **Disadvantages:** Slower in unbalanced trees; requires O(n) time for space rebalancing.

Recommendation

Based on the analysis, I recommend using the **Hash Table** data structure. It provides the fastest lookups for course information, allowing users to retrieve prerequisites in constant time, which aligns well with the advisors’ requirement for quick access to course details. While it requires additional space and can have challenges with collisions, the advantages greatly outweigh them for the functionality needed in this project.