OBE IMPLEMENTATION:COURSE SETTING

by

KARTHIK [AP23110011065]

RAVI [AP23110011077]

AGASTYA [AP23110011072]

LOKESH [AP23110011075]

NIKHIL [AP23110011070]

A report for the CS204:Design and Analysis of Algorithm project

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY AP::AMARAVATI

INDEX

Introduction

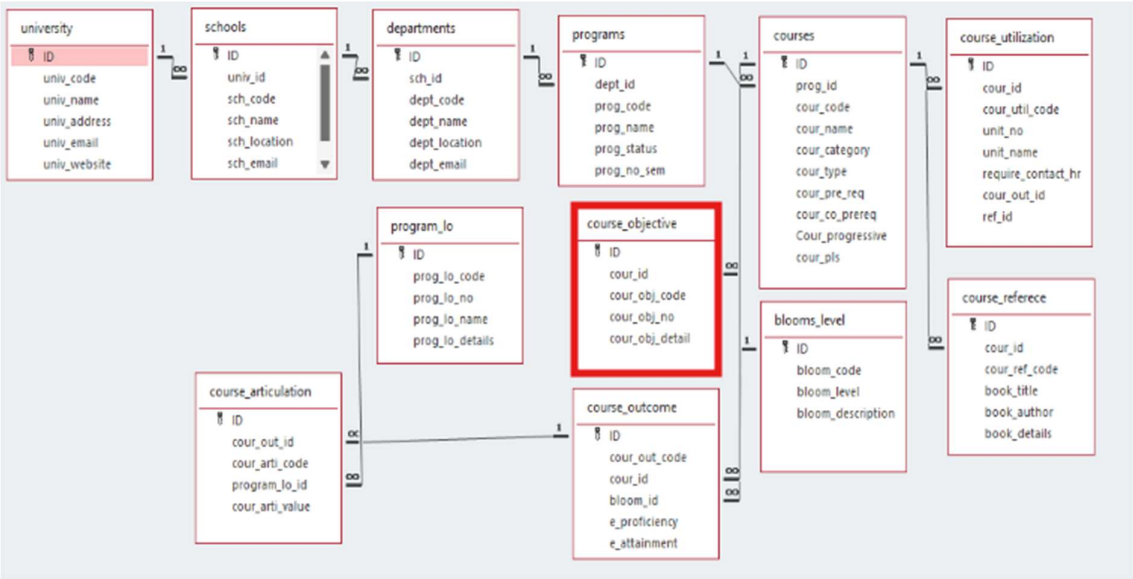
Our University (herewith considered as SRM-AP) is going to implement OBE(Outcome Based Education) in their university and you assigned in the project to develop an application with any programming Language you are well versed and you were supposed to do searching and sorting using learned algorithms,comparing your sorting algorithm with any one of existing algorithm,displaying the time complexity of both algorithms and explaining advantages and disadvantages of the algorithm.

Project Modules:

Various Modules available in the project are

1.Blooms Level setting
2.Program Level Objective Setting
3.University
4.Schools
5.Department
6.Programs
7.Courses
8.Course objective setting
9.Course Outcome Setting
10.Course Articulation matrix Setting
11.Course Utilization Setting
12.Course Reference Setting.

Architecture Diagram



Module Name:COURSE OBJECTIVE SETTING
Module Description:

The Course Objective Management System is a robust program designed to manage course objectives efficiently. It facilitates CRUD (Create, Retrieve, Update, Delete) operations for managing course records and employs advanced algorithms to optimize data handling. All operations are synchronized with a file to ensure data persistence and consistency.

Programming Details naming conventions to be used:

- **File name**:yourregno_course_module
- **Function/method name**
  - **Create**:AP23110011070_courseObjective_create
  - **Update**: AP23110011070_courseObjective_update
  - **Retrieve**: AP23110011070_courseObjective_retrive
  - **Delete**: AP23110011070_courseObjective_delete
  - **Sorting**: AP23110011070_courseObjective_bubbleSort
  - **Searching**: AP23110011070_courseObjective _binarySearch
  - **Storing**: AP23110011070_courseObjective _storing
  - **Comparison**(both searching and Sorting):
    - For Searching-AP23110011070_courseObjective_Compare_Search_binarySearch
    - For Sorting-AP23110011070_courseObjective_Compare_sorting_bubbleSort
  - **Time Complexity**(both searching and Sorting):
    - For Searching- O(log n)
    - For Sorting- $O(n^2)$
  - **Algorithm Details**(pseudocode or steps)(both searching and Sorting):
    - For Searching- AP23110011070_courseObjective_search_binarySearch
    - For Sorting- AP23110011070_courseObjective_sort_bubbleSort
- **File name(for storing the details)**
  - File name to be used is:-courseObjective_setting .txt

Field/table details:

Field Name	**Data type**
course_id	Integer
course_name	String
instructor	String
credits	Integer

Algorithm Details:

(i)Sorting

Bubble sort works by repeatedly swapping adjacent elements if they are in the wrong order. This process continues until no swaps are required, meaning the list is sorted.

Time Complexity of Bubble Sort:

- **Best Case:** O(n) (when the list is already sorted)
- **Average Case:** O(n^2)

- **Worst Case:** O(n^2)

Pseudocode for Bubble Sort:

```
function bubbleSort(arr):
    n = length(arr)
    for i = 0 to n-1:
        for j = 0 to n-i-2:
            if arr[j] > arr[j+1]:
                swap arr[j] and arr[j+1]
```

(ii)Searching

Binary search works by repeatedly dividing the sorted list in half until the target element is found. It requires the list to be sorted before searching.

Time Complexity of Binary Search:

- **Best Case:** O(1)
- **Average Case:** O(log n)
- **Worst Case:** O(log n)

Pseudocode for Binary Search:

```
function binarySearch(arr, target):
    low = 0
    high = length(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid  // Element found
        else if arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1  // Element not found
```

(ii) Storing the details in a text file

- Storing the details in the text file once details are entered.
- Delete the detail from the text file once details are deleted.
- Update the text file once details are updated.

Source Code

```
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm>
#include <string>

using namespace std;

struct Course {
   int course_id;
   string course_name;
   string instructor;
   int credits;
};

vector<Course> courses;

const string DATA_FILE = "courseObjective_setting.txt";

void AP23110011070_courseObjective_create() {
   ifstream file(DATA_FILE);
   string line;

   while (getline(file, line)) {
      size_t pos1 = line.find(",");
      size_t pos2 = line.find(",", pos1 + 1);
      size_t pos3 = line.find(",", pos2 + 1);

      int id = stoi(line.substr(0, pos1));
      string name = line.substr(pos1 + 1, pos2 - pos1 - 1);
      string instructor = line.substr(pos2 + 1, pos3 - pos2 - 1);
      int credits = stoi(line.substr(pos3 + 1));

      courses.push_back({id, name, instructor, credits});
   }
   file.close();
}

void AP23110011070_courseObjective_storing() {
   ofstream file(DATA_FILE);
   for (const auto& course : courses) {
      file << course.course_id << "," << course.course_name << ","
          << course.instructor << "," << course.credits << endl;
   }
   file.close();
}
```

```
void AP23110011070_courseObjective_create(int id, string name, string instructor, int credits) {
   courses.push_back({id, name, instructor, credits});
   AP23110011070_courseObjective_storing();
}

void AP23110011070_courseObjective_update(int id, string newName = "", string newInstructor = "",
int newCredits = -1) {
   for (auto& course : courses) {
      if (course.course_id == id) {
         if (!newName.empty()) course.course_name = newName;
         if (!newInstructor.empty()) course.instructor = newInstructor;
         if (newCredits != -1) course.credits = newCredits;
         AP23110011070_courseObjective_storing();
         return;
      }
   }
   cout << "Course ID not found!" << endl;
}

void AP23110011070_courseObjective_delete(int id) {
   courses.erase(remove_if(courses.begin(), courses.end(),
                     [id](Course& course) { return course.course_id == id; }),
              courses.end());
   AP23110011070_courseObjective_storing();
}

vector<Course> AP23110011070_courseObjective_retrieve(const string& keyword) {
   vector<Course> results;
   for (const auto& course : courses) {
      if (course.course_name.find(keyword) != string::npos || course.instructor.find(keyword) !=
string::npos) {
         results.push_back(course);
      }
   }
   return results;
}

int AP23110011070_courseObjective_Compare_Search_binarySearch(const vector<Course>&
sortedCourses, const string& target, const string& key) {
   int low = 0, high = sortedCourses.size() - 1;

   while (low <= high) {
      int mid = low + (high - low) / 2;

      if ((key == "course_name" && sortedCourses[mid].course_name == target) ||
         (key == "instructor" && sortedCourses[mid].instructor == target)) {
         return mid;
      } else if ((key == "course_name" && sortedCourses[mid].course_name < target) ||
```

```
            (key == "instructor" && sortedCourses[mid].instructor < target)) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1;
}

void AP23110011070_courseObjective_Compare_sorting_bubbleSort(vector<Course>& data, const
string& key) {
    for (size_t i = 0; i < data.size() - 1; i++) {
        for (size_t j = 0; j < data.size() - i - 1; j++) {
            if ((key == "course_name" && data[j].course_name > data[j + 1].course_name) ||
                (key == "credits" && data[j].credits > data[j + 1].credits)) {
                swap(data[j], data[j + 1]);
            }
        }
    }
}

void AP23110011070_courseObjective_sort_bubbleSort(vector<Course>& data, const string& key) {
    for (size_t i = 0; i < data.size() - 1; i++) {
        for (size_t j = 0; j < data.size() - i - 1; j++) {
            if ((key == "course_name" && data[j].course_name > data[j + 1].course_name) ||
                (key == "credits" && data[j].credits > data[j + 1].credits)) {
                swap(data[j], data[j + 1]);
            }
        }
    }
}

void AP23110011070_courseObjective_display(const vector<Course>& data) {
    for (const auto& course : data) {
        cout << "ID: " << course.course_id << ", Name: " << course.course_name
             << ", Instructor: " << course.instructor
             << ", Credits: " << course.credits << endl;
    }
}

int main() {
    AP23110011070_courseObjective_create();

    int choice;
    do {
        cout << "\n=== Course Management System ===\n";
        cout << "1. Add a Course\n";
        cout << "2. Update a Course\n";
```

```
cout << "3. Delete a Course\n";
cout << "4. Display All Courses\n";
cout << "5. Search for a Course (Linear Search)\n";
cout << "6. Search for a Course (Binary Search)\n";
cout << "7. Sort Courses\n";
cout << "0. Exit\n";
cout << "Enter your choice: ";
cin >> choice;

switch (choice) {
   case 1: {
      int id, credits;
      string name, instructor;
      cout << "Enter Course ID: ";
      cin >> id;
      cin.ignore();
      cout << "Enter Course Name: ";
      getline(cin, name);
      cout << "Enter Instructor: ";
      getline(cin, instructor);
      cout << "Enter Credits: ";
      cin >> credits;
      AP23110011070_courseObjective_create(id, name, instructor, credits);
      break;
   }
   case 2: {
      int id, newCredits;
      string newName, newInstructor;
      cout << "Enter Course ID to Update: ";
      cin >> id;
      cin.ignore();
      cout << "Enter New Name (leave blank to skip): ";
      getline(cin, newName);
      cout << "Enter New Instructor (leave blank to skip): ";
      getline(cin, newInstructor);
      cout << "Enter New Credits (-1 to skip): ";
      cin >> newCredits;
      AP23110011070_courseObjective_update(id, newName, newInstructor, newCredits);
      break;
   }
   case 3: {
      int id;
      cout << "Enter Course ID to Delete: ";
      cin >> id;
      AP23110011070_courseObjective_delete(id);
      break;
   }
   case 4:
      AP23110011070_courseObjective_display(courses);
```

```
            break;
        case 5: {
            string keyword;
            cout << "Enter Keyword for Linear Search: ";
            cin.ignore();
            getline(cin, keyword);
            vector<Course> results = AP23110011070_courseObjective_retrieve(keyword);
            if (!results.empty()) {
                AP23110011070_courseObjective_display(results);
            } else {
                cout << "No courses found!" << endl;
            }
            break;
        }
        case 6: {
            string key, target;
            cout << "Search by (course_name/instructor): ";
            cin >> key;
            cin.ignore();
            cout << "Enter Target Value: ";
            getline(cin, target);

            AP23110011070_courseObjective_Compare_sorting_bubbleSort(courses, key);
            int index = AP23110011070_courseObjective_Compare_Search_binarySearch(courses,
target, key);
            if (index != -1) {
                AP23110011070_courseObjective_display({courses[index]});
            } else {
                cout << "Course not found!" << endl;
            }
            break;
        }
        case 7: {
            string key;
            cout << "Sort by (course_name/credits): ";
            cin >> key;
            AP23110011070_courseObjective_sort_bubbleSort(courses, key);
            cout << "\nCourses After Sorting:\n";
            AP23110011070_courseObjective_display(courses);
            break;
        }
        case 0:
            cout << "Exiting program. Goodbye!\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
    }
  } while (choice != 0);
```

```
    return 0;
}
```

Comparison of Sorting Algorithms

Bubble Sort
Time Complexity:

- **Best:** O(n)O(n)O(n) (when the array is already sorted)
- **Worst/Average:** O(n2)O(n^2)O(n2)

How It Works:

- Bubble Sort repeatedly compares adjacent elements in the array and swaps them if they are in the wrong order. This process "bubbles" the largest (or smallest) element to its correct position at the end of the array after each pass. The algorithm continues to do this until the array is fully sorted.

Advantages:

- **Simple and easy to implement:** Bubble Sort is straightforward to understand and implement, making it a good choice for educational purposes or simple use cases.
- **In-place sorting algorithm:** It doesn't require additional memory beyond the input array.

Disadvantages:

- **Inefficient for large datasets:** With a time complexity of O(n2)O(n^2)O(n2), Bubble Sort becomes significantly slower as the size of the dataset increases.
- **Not suitable for performance-critical applications:** The algorithm's worst-case performance makes it impractical for large-scale or complex data sorting tasks.

When to Use?:

- **For small datasets** or when simplicity is more important than performance. It may also be useful when sorting nearly sorted data, as the best-case time complexity is O(n)O(n)O(n), and it can detect early if the data is already in order.

Selection Sort
Time Complexity:

- **Best/Worst/Average:** O(n2)O(n^2)O(n2)

How It Works:

- Selection Sort works by repeatedly finding the minimum (or maximum) element from the unsorted part of the array and swapping it with the element at the current position. This process continues for each position in the array until the entire array is sorted.

Advantages:

- **Simple and easy to implement:** Like Bubble Sort, Selection Sort is easy to understand and implement, making it suitable for educational purposes.
- **In-place sorting algorithm:** It doesn't require extra space beyond the input array, making it memory efficient.

Disadvantages:

- **Inefficient for large datasets:** With a time complexity of O(n2)O(n^2)O(n2), Selection Sort is slow for large arrays or complex data sets.
- **Unstable algorithm:** Selection Sort does not maintain the relative order of equal elements, which can be a disadvantage in certain applications where stability is important.

When to Use?:

- **For small datasets** or when simplicity is prioritized over performance. It can also be useful when memory usage is a concern, as it sorts in-place without additional.

Comparison of Searching Algorithms

1. Binary Search

Time Complexity:
● Best: O(1)
● Worst/Average: O(log n)

How It Works:
● Searches on sorted data by repeatedly halving the search space.

Advantages:
● Efficient for large, sorted datasets.

Disadvantages:
● Data must be sorted before searching.

When to Use?:
● For sorted and static datasets with frequent searches.

2. Linear Search

Time Complexity:
● Best: O(1)
● Worst/Average: O(n)

How It Works:
● Sequentially checks each element until the target is found.

Advantages:
● Simple to implement, and no sorting is required.

Disadvantages:
● Inefficient for large datasets.

When to Use?:
● For small or unsorted datasets with minimal overhead.

Screen Shots

```
=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 1
Enter Course ID: 70
Enter Course Name: Data Structures
Enter Instructor: manoj
Enter Credits: 4

=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 1
Enter Course ID: 71
Enter Course Name: Algorithms
Enter Instructor: sanjay
Enter Credits: 3
```

```
=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 1
Enter Course ID: 72
Enter Course Name: Maths
Enter Instructor: arka
Enter Credits: 2
```

```
=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 2
Enter Course ID to Update: 72
Enter New Name (leave blank to skip): maths
Enter New Instructor (leave blank to skip): arka
Enter New Credits (-1 to skip): -1

=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 4
ID: 70, Name: Data Structures, Instructor: manoj, Credits: 4
ID: 71, Name: Algorithms, Instructor: sanjay, Credits: 3
ID: 72, Name: maths, Instructor: arka, Credits: 2
```

```
=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 6
Search by (course_name/instructor): instructor
Enter Target Value: sanjay
ID: 71, Name: Algorithms, Instructor: sanjay, Credits: 3

=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 7
Sort by (course_name/credits): credits

Courses After Sorting:
ID: 72, Name: maths, Instructor: arka, Credits: 2
ID: 71, Name: Algorithms, Instructor: sanjay, Credits: 3
ID: 70, Name: Data Structures, Instructor: manoj, Credits: 4
```

```
=== Course Management System ===
1. Add a Course
2. Update a Course
3. Delete a Course
4. Display All Courses
5. Search for a Course (Linear Search)
6. Search for a Course (Binary Search)
7. Sort Courses
0. Exit
Enter your choice: 0
Exiting program. Goodbye!


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion

The Course Management System (CMS) enhances course administration through CRUD operations, optimized sorting and searching algorithms, and robust storage mechanisms. Bubble Sort and Selection Sort are employed for sorting, while Linear Search and Binary Search are used to improve search performance. These methods offer efficient solutions for managing course data, with comparisons to alternative techniques demonstrating their effectiveness. This system reflects SRM University's commitment to efficient academic workflows under the OBE framework.