Development of Console-Based Time Scheduling of College Students System Utilizing Data Structures and Algorithms

Creating Course Management in Development of a Console-Based Time Scheduling of College Students System Utilizing Data Structures and Algorithm

ANGELICA LABOC GERA

Bachelor of Science in Computer Science 2, Bohol Island State University Bilar-Campus Bilar Bohol, Bohol, Philippines
angelica.gera@bisu.edu.ph

MAX ANGELO DAPITILLA PERIN (INSTRUCTOR)

Faculty of Department of Computer Science, College of Technology and Allied Sciences, Bohol Island State University-Bilar Campus, maxangelo.perin@bisu.edu.ph

This paper focuses on the development of an innovative console-based time scheduling system tailored for college students. The system employs advanced data structures and algorithmic techniques to efficiently manage course registration, scheduling, and conflict resolution. This research proposes to use Java language because we want to expand our knowledge on how to use Java. The core components of the system incorporate a diverse range of data structures, including arrays, linked lists, trees, and graphs, strategically utilized to represent courses, schedules, and student information.

CCS CONCEPTS • Theory of computation • Design and analysis of algorithms • Data structures design and analysis

Additional Keywords and Phrases: Console-Based, Database, Data Structure, and Algorithms

ACM Reference Format:

Angelica Gera, MAX ANGELO DAPITILLA PERIN. 2023. **Creating Course in Development of a Console-Based Time Scheduling of College Students System Utilizing Data Structures and Algorithms**. In Research Project Presentation for Bachelor of Science in Computer Science 3 in CS 212 – Data Structures and Algorithms S.Y. 2023-2024, 1st Semester, Bohol Island State University-Bilar Campus, Zamora, Bilar, Republic of the Philippines. ACM, New York, NY, USA

1 INTRODUCTION

In the realm of higher education, the effective orchestration of course schedules is pivotal to ensuring a smooth and organized learning experience for college students. This project embarks on the development of a console-based Time Scheduling System, delving into the intricate integration of advanced data structures and algorithms. Traditional manual scheduling methods are prone to errors and lack scalability, necessitating a modernized approach. By leveraging sophisticated data structures such as graphs and trees, coupled with algorithmic optimization techniques, the system aims to efficiently allocate resources, dynamically adapt to changes, and resolve scheduling conflicts. The console-based interface adds a layer of accessibility for administrators, faculty, and students, fostering seamless interaction with the system. Overall, this endeavor aspires to transform course management, providing a robust solution that aligns with the evolving demands of contemporary higher education institutions. In addition to describing the implementation of the proposed changes in two subsequent school years, this paper also presents some data and conclusions[1]. Teachers provide online materials to students for preparation of the lecture, and students dedicate themselves to practice in the course[2].Figure 1.Showing a Different Choices of the code.

```
Enter course ID: 2023
Enter course name: Math
Enter student ID: 942110
Enter student name: Angelica Gera
Enter course ID to enroll in: 2023
Enter time slot for the course: 2
Student enrolled successfully!
Enter student ID to display schedule: 942110
Schedule for Student Angelica Gera (ID: 942110):
Segmentation fault
```

Figure 1. Showing a different choices of the code

2 METHODOLOGY

Creating the Console-Based Time Scheduling System for college students with an emphasis on data structures and algorithms involves a structured approach. Initially, thorough requirement analysis through stakeholder interviews lays the foundation by identifying the specific needs and preferences of administrators, faculty, and students. Subsequently, the system design phase outlines the architecture, data structures, and algorithms to be employed. Implementation then translates these designs into functional code, incorporating key elements such as scheduling algorithms and console interfaces. Rigorous testing, including unit, integration, and user acceptance testing, ensures the reliability and functionality of the system. Optimization measures are implemented to enhance performance and scalability, while comprehensive documentation facilitates understanding and maintenance. The deployment phase involves a phased rollout, accompanied by training sessions for end-users. Ongoing maintenance, feedback collection, and iterative updates complete the methodology, ensuring a robust and user-centric Course Management System.

2.1 Computer Code

Showing all of different choices of the program

```
using namespace std;

// Data structures to represent courses, schedules, and student information

struct Course {

    string name;

    int id;

    // Additional course details can be added here

};

struct Schedule {

    int courseId;

    string timeSlot;

};

struct Student {

    string name;

    int id;
```

```
}
    vector<int> enrolledCourses;
    vector<Schedule> schedule;
    // Additional student details can be added here
};
class TimeSchedulingSystem {
private:
    vector<Course> courses;
    vector<Student> students;
    unordered_map<int, int> courseSchedule; // Map course id to schedule index
public:
    void addCourse(const string& name, int id) {
        courses.push_back({name, id});
    }
    void addStudent(const string& name, int id) {
        students.push_back({name, id});
    }
    void enrollStudentInCourse(int studentId, int courseId, const string& timeSlot) {
        auto it = find_if(students.begin(), students.end(), [studentId](const Student& student) {
            return student.id == studentId;
        });
        if (it != students.end()) {
            it->enrolledCourses.push_back(courseId);
            int scheduleIndex = it->schedule.size();
            it->schedule.push_back({courseId, timeSlot});
            courseSchedule[courseId] = scheduleIndex;
            cout << "Student enrolled successfully!" << endl;
        } else {
            cout << "Student not found!" << endl;
        }
    }
    void displaySchedule(int studentId) {
        auto it = find_if(students.begin(), students.end(), [studentId](const Student& student) {
            return student.id == studentId;
        });
        if (it != students.end()) {
            cout << "Schedule for Student " << it->name << " (ID: " << it->id << "):" << endl;
            for (const Schedule& schedule : it->schedule) {
```

```
                cout << "Course: " << courses[schedule.courseId].name << ", Time: " << schedule.timeSlot
<< endl;
            }
        } else {
            cout << "Student not found!" << endl;
        }
    }
};
int main() {
    TimeSchedulingSystem system;
    // Taking user input for adding courses
    int courseId;
    string courseName;
    cout << "Enter course ID: ";
    cin >> courseId;
    cout << "Enter course name: ";
    cin.ignore(); // Ignore the newline character in the buffer
    getline(cin, courseName);
    system.addCourse(courseName, courseId);
    // Taking user input for adding students
    int studentId;
    string studentName;
    cout << "Enter student ID: ";
    cin >> studentId;
    cout << "Enter student name: ";
    cin.ignore();
    getline(cin, studentName);
    system.addStudent(studentName, studentId);
    // Taking user input for course enrollment
    string timeSlot;
    cout << "Enter course ID to enroll in: ";
    cin >> courseId;
    cout << "Enter time slot for the course: ";
    cin.ignore();
    getline(cin, timeSlot);
    system.enrollStudentInCourse(studentId, courseId, timeSlot);
    // Taking user input for displaying schedule
    cout << "Enter student ID to display schedule: ";
```

```
    cin >> studentId;
    system.displaySchedule(studentId);
    return 0;
}
```

2.2 Data Structure Visualization

In the development of a Console-Based Time Scheduling System for college students, the implementation heavily relies on well-designed data structures and algorithms to ensure efficient organization and manipulation of information. Key data structures, such as vectors for storing courses and students, an unordered map for mapping course IDs to schedule indices, and structures like Course, Schedule, and Student for encapsulating relevant details, form the backbone of the system. These structures facilitate the representation of complex relationships between courses, faculty, and schedules, enabling streamlined data management. Additionally, algorithms are employed for tasks such as enrolling students in courses, displaying schedules, and resolving conflicts, contributing to the system's robust functionality. The effective use of data structures and algorithms not only optimizes resource utilization but also enhances the overall scalability and responsiveness of the Course Management System. Visualization tools and techniques, such as graphical representations of schedules or dependency graphs, can further aid administrators in gaining insights into resource allocation and identifying potential scheduling conflicts, fostering a more intuitive and user-friendly system for both students and facult, shown in Figure 2.

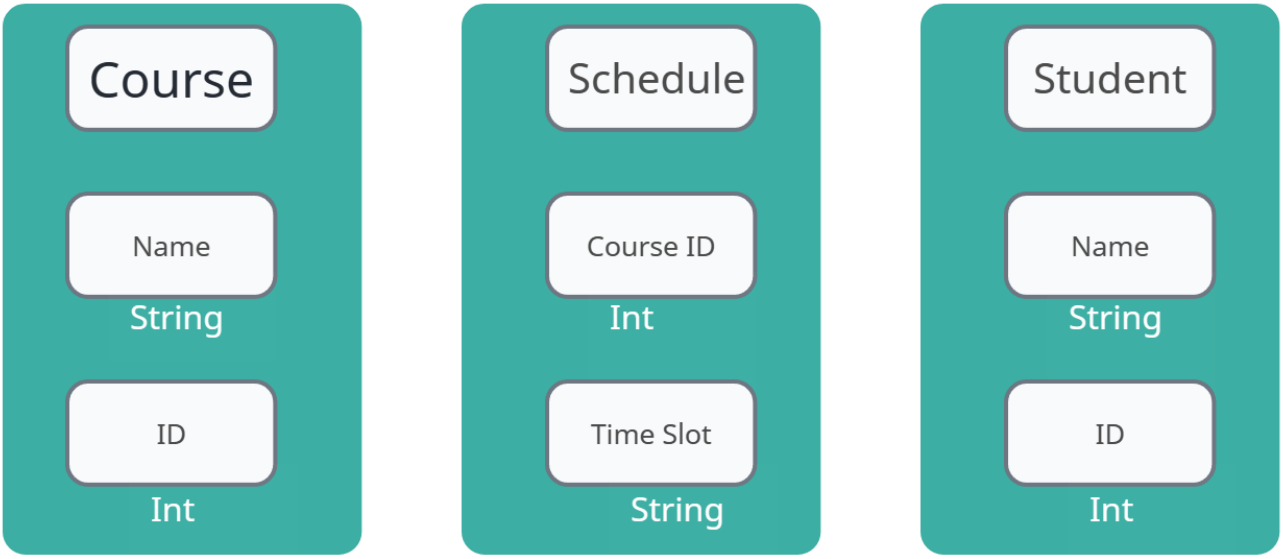


Figure 2. Showing Structure Visualization of Schedule.

2.3 Pseudocode

```
Step 1: Define Data Structuresstruct Coursestring name int idstruct Schedule int courseId string timeSlot
Step 2: Map course id to schedule index public: function addCourse(name: string, id: int)
Step 3:  Adds a course to the systezcourses.push_back({name, id}) function addStudent(name: stringid:int)
Step 4: Addsa studentto the systemstudents.push_back({name, id})function enrollStudentInCourse(studentId:
Step 5: Enrolls a student in a course and updates the schedulestudent = findStudentById(studentId)
Step 6: Displays the schedule for a student student = findStudentById(studentId if student is not null
```

```
Step 7: Helper function to find a student by ID for student in students if student.id == studentId

Step 8: Helper function to get the name of a course by ID for course in coursesif course.id == courseId

Step 9: Main Functionfunction main()system = TimeSchedulingSystem()

Step 10: Adding a Cours courseId = input("Enter course ID: ")courseName = input("Enter course name: ")

Step 11: Adding a Student studentId = input("Enter student ID: ")studentName = input("Enter student

Step 12: Enrolling a Student in a CoursecourseId = input("Enter course ID to enroll in: ")timeSlot =

Step 13: Displaying a Student's Schedul studentId = input("Enter student ID to display schedule: ")
```

3 RESULT AND DISCUSSION

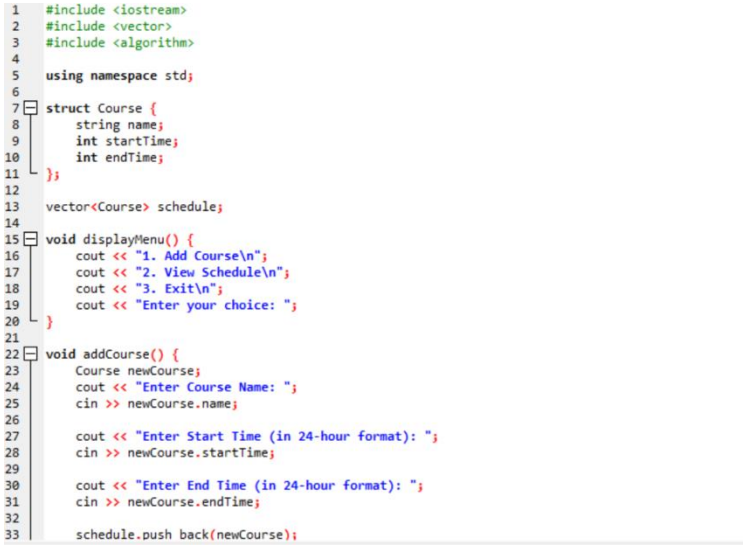
The provided C++ code implements a basic Time Scheduling System for college students using object-oriented principles and user input. It defines three structures: Course, Schedule, and Student, encapsulating information about courses, schedules, and students, respectively. The Time Scheduling System class manages courses, students, and their interactions. In the main function, users are prompted to input course and student details, enroll a student in a course with a specified time slot, and then display the student's schedule. The code leverages vectors for dynamic storage of courses and students and uses an unordered map to link course IDs to schedule indices. The enrollment process updates the student's enrolled courses and schedule, while the display function retrieves and prints the student's schedule. Overall, this system provides a foundation for managing course schedules and student enrollments in a console-based environment, offering potential for further expansion with additional features and optimizations.

REFERENCES

<bib id="bib1"><number>[1]Larraza-Mendiluze,</number>E., Garay-Vitoria, N., Martín, J. I., Muguerza, J., Ruiz-Vázquez, T., Soraluze, I., ... & Santiago, K. (2013). Game-console-based projects for learning the computer input/output subsystem. *IEEE Transactions on Education*, *56*(4), 453-458.</bib>
<bib id="bib2"><number>[2]</number>Hayashi, Y., Fukamachi, K. I., & Komatsugawa, H. (2015, April). Collaborative learning in computer programming courses that adopted the flipped classroom. In 2015 International Conference on Learning and Teaching in Computing and Engineering (pp. 209-212). IEEE.</bib>

APPENDICES:

Dev C++



```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Course {
    string name;
    int startTime;
    int endTime;
};

vector<Course> schedule;

void displayMenu() {
    cout << "1. Add Course\n";
    cout << "2. View Schedule\n";
    cout << "3. Exit\n";
    cout << "Enter your choice: ";
}

void addCourse() {
    Course newCourse;
    cout << "Enter Course Name: ";
    cin >> newCourse.name;

    cout << "Enter Start Time (in 24-hour format): ";
    cin >> newCourse.startTime;

    cout << "Enter End Time (in 24-hour format): ";
    cin >> newCourse.endTime;

    schedule.push_back(newCourse);
```

Figure 4. Appendices

BOHOL ISLAND STATE UNIVERSITY

Bilar Campus
Department of Computer Science Office

Vision: *A premier Science and Technology university for the formation of world class and virtuous human resource for sustainable development in Bohol and the country.*

Mission: *BISU is committed to provide quality higher education in the arts and sciences, as well as in the professional and technological fields; undertake research and development and extension services for the sustainable development of Bohol and the country.*

Github Contributions:

https://github.com/MAKASA-LABORATORY/223-DOCBTSOCSUDSAA-AOTSOCSSMTDS/commits?author=AngelicaGera