Interval Scheduling

Nicholas Mirolli and Morgan Willis

Outline

Introduction

Algorithm

Complexity

Implementation

Test Cases

Conclusion

Activity Interval Scheduling

The hypothetical problem we are facing is we have a client who owns a gym similar to the YMCA, this gym has many rooms to be filled with scheduled activities.

The client is currently choosing the activities by hand and it is time consuming for them because they have so many rooms to fill with the max amount of activities. With a program to finds the optimal solution of scheduled activities for one room at a time would help the client save time and therefore money.

Greedy Algorithm

The greedy version of this algorithm first sorts the events using selection sort by start time. After choosing the first event from the sorted list, we eliminate the conflicts with that event, meaning all the events who's start times come before the end time of the first event. The process is repeated until there are no more events. The events that remain are the events that the gym will schedule if they follow the greedy algorithm.

Optimal Algorithm

The optimal version of this algorithm first sorts the events using selection sort by end time. After choosing the first event from the sorted list, we eliminate the conflicts with that event, meaning all the events who's start times come before the end time of the first event. The process is repeated until there are no more events. The events that remain are the events that the gym will schedule if they follow the optimal algorithm. In most cases the optimal algorithm will schedule more events than the greedy algorithm, but it will never schedule less.

Greedy vs Optimal

Step 1

Sort by start time

(preprocessing)

Step 2

Choose first event

Step 3

Eliminate Conflicts

Step 4

Repeat

Step 1

Sort by end time

(preprocessing)

Step 2

Choose first event

Step 3

Eliminate Conflicts

Step 4

Repeat

Complexity of Greedy and Optimal

```
Sort events by start/finish times so that f1 ≤ f2 ≤ ... ≤ fn.

A ← φ

for e = 1 to n {

        if (event e compatible with A) {

                A ← A ∪ {e}

        }

}

return A
```

Figure 3: Pseudo code for interval scheduling algorithm.

Time complexity is O(nlogn)

Implementation

Event Class

Time Class

Main

Summary of Event Class

```
class Event {
    private:
        string name;
        Time startTime;
        Time endTime;
    public:
        Event() {
            name =
              "Default";
        }
    Event(string n, Time s, Time e, double p) {
            name = n;
            startTime = s;
            endTime = e;
            assert(name != "");
        }
        string getName() const { return name; }
        Time getStartTime() const { return startTime; }
        Time getEndTime() const { return endTime; }
        void setName(const string n) { name = n; }
        void setStartTime(const Time s) { startTime = s; }
        void setStartTime(const short h, const short m) { }
        void setEndTime(const Time e) { endTime = e; }
            void setEndTime(const short h, const short m) { }
            void print() { }
};
bool operator==(Event a, Event b) { }
bool operator!=(Event a, Event b) { }
```

Summary of Time Class

```
class Time {
    private:
        short hour;
        short minute;
    public:
        Time() {
            hour = 0;
            minute = 0;
        }
      Time(short h, short m) {
            hour = h;
            minute = m;
            assert(hour < 24 && hour >= 0);
            assert(minute <= 59 && minute >= 0);
        }
        Time(const Time &t) {
            hour = t.getHour();
            minute = t.getMinute();
            assert(t.getHour() < 24 && t.getHour() >= 0);
            assert(t.getMinute() <= 59 && t.getMinute() >= 0);
        }
```

```
short getHour() const { return hour; }
        short getMinute() const { return minute; }
        void setHour(const short h) { hour = h; }
        void setMinute(const short m) { minute = m; }
        void print() {
            cout << hour << ":" << setw(2) << setfill('0') << minute;
        }
};
bool operator==(Time a, Time b) { }
bool operator!=(Time a, Time b) { }
bool operator<(Time a, Time b) { }
```

Summary of Main

void swapEvent(Event* array, int a, int b);

void greedySort(Event* greedy, int numEvents);

void optimalSort(Event* optimal, int numEvents);

void removeConflicts(Event* array, int numEvents);

void printOutFile(Event* array, Event* greedy, Event* optimal, int numEvents, string outFilename);

int main(int argc, char** argv) { }

Test Case 1

Input:

TestCase1
File for testing Final Project
5
Spinning 8 00 9 00
Dodge_ball 10 30 12 30
Yoga 7 30 8 30
Boxing 15 30 16 30
Boot_camp 11 00 12 00

Output:

Greedy:
Yoga 7:30 8:30
Dodge_ball 10:30 12:30
Boxing 15:30 16:30
Number of Events: 3

Optimal:
Yoga 7:30 8:30
Boot_camp 11:00 12:00
Boxing 15:30 16:30
Number of Events: 3

Test Case 2

Input:

TestCase2
File for testing Final Project
20
Spinning 4 00 23 00
Dodge_ball 10 30 12 30
Yoga 7 30 8 30
Boxing 15 30 16 30
Boot_camp 11 00 12 00
Martial_Arts 17 00 20 00
Basket_Ball 7 00 13 00
Racquet_Ball 10 00 14 00
Weight_Training 18 30 20 30
Aerobics 9 30 13 30
Water_Aerobics 6 00 8 00
Hot_Yoga 7 00 10 00
Jump_Rope 13 00 15 00
Karate 22 00 23 00
Seam_Carving 10 00 10 30
Foot_Skills 9 30 10 30
Zumba 5 00 6 30
Dance_Aerobics 12 00 1 00
Core_Exercises 13 30 14 00
Arm_Sculpting 14 00 14 30

Output:

Greedy:
Spinning 4:00 23:00
Number of Events: 1

Optimal:
Dance_Aerobics 12:00 1:00
Zumba 5:00 6:30
Yoga 7:30 8:30
Seam_Carving 10:00 10:30
Boot_camp 11:00 12:00
Core_Exercises 13:30 14:00
Arm_Sculpting 14:00 14:30
Boxing 15:30 16:30
Martial_Arts 17:00 20:00
Karate 22:00 23:00
Number of Events: 10

Sorting

Efficiency of the algorithm can rely mainly on sorting the dataset.

Weighted Version

Implementation of a weighted version of this algorithm is more difficult than it seems.

Organization

Breaking into classes made implementation harder at first but once implemented it made things easier.

Data Structures

Choosing a specific data structure over another can lessen lines of code and also improve run time of the program.

Q & A