Calendars Part 1

COSC122 Assignment 2012

August 16, 2012

1 Overview

Calendars are an important part of modern life. These days, it isn't uncommon to have several portable devices with synchronised calendars. This assignment requires you to implement algorithms to search for conflicting events and merging lists of events efficiently, such as might be found in real-world calendaring systems.

This assignment is designed to help you understand both the theoretical and practical concerns related to implementing a variety of algorithms. As a computer scientist, you will often encounter problems for which no pre-existing solution exists. Your knowledge of and experience with algorithms will directly affect the quality and efficiency of systems that you produce, especially when processing problems that involve a large data-sets.

1.1 Due Dates

Part One of this assignment has two tasks which you must complete, each part is worth 20% of the total assignment. The **due date** for part 1 is **5pm, Monday the 17th of September**.

Part Two of this assignment has four tasks which you must complete, each part is worth 15% of the total assignment. The **due date** for part 2 is **5pm, Monday the 8th of October**. This part of the assignment is not included in this document.

The drop dead date is 5pm one week after each due date. Late submissions will be accepted between the due date and the drop dead date, but there will be a penalty of 15% of the maximum possible grade for late submission. No assignments will be accepted after the drop dead date.

1.2 Submission

One week before the due date, an assignment submission quiz will be made available. You must copy and paste your source code as required and answer any questions.

Each incorrect submission will accumulate a penalty. This information will be clearly outlined on the quiz page.

We recommend that you test your code thoroughly using the supplied unit tests before making a submission. Additional unit test may be provided as required if there are any ambiguities with the assignment.

1.3 Implementation

You must write full algorithms by your own hand. Do not import any additional standard libraries unless explicitly given permission within the task outline.

1.4 Getting Help

The work in this assignment is to be carried out individually. You must not discuss code-level details with anyone other than the course tutors and lecturers. You are, however, permitted to discuss high-level details of the program with other students.

If you get stuck on a programming issue, you are encouraged to ask your tutor or the lecturer for help. You may not copy material from books, the internet, or other students. We will be checking carefully for copied work.

If you have a general assignment question or need clarification on how something should work, please use the class forum available at `http://learn.canterbury.ac.nz/mod/forum/view.php?f=375`.

2 Part 1

Sometimes it is useful to compare a personal calendar to a set of event calendars to find conflicts. This might happen when someone wants to find out what events they can attend given their existing schedule.

Task 1 requires that you implement a linear scan to find conflicting events. Task 2 improves this by using a binary search.

2.1 Task 1 (20%)

The file `clashes_linear.py` contains an incomplete implementation of a simple linear scan algorithm that checks for conflicting calendar items using an $O(n^2)$ search.

You must complete this implementation by adding code between the commented segments:

```
# --[ WRITE CODE HERE ]-->
# Collect all events from the event_calendar that don't clash
# with any events in the personal_calendar.
# <--[ WRITE CODE HERE ]--
```

You can run the script from the command line like so:

```
python ./clashes_linear.py data/calendar.txt data/events_*.txt
```

The first file provided is considered to be the user's personal calendar with their schedule. The remaining files are lists of events that may or may not clash with the personal calendar. The output will be a subset of the events which don't clash.

You can assume that the calendar and events are in sorted order and within a specific calendar of events there are no overlaps.

2.1.1 Events

Calendars are lists of events. A Python module `eventlib` is provided that can load events from JSON encoded files. You should familiarise yourself with the code in `eventlib`. You should not reimplement the functions provided by `eventlib`, or change the command line interfaces provided.

The implementation must find conflicts given a personal calendar and one or more event calendars. The linear scan is very simple, in that for each event in the personal calendar, it simply checks every event in the event calendars for conflicts. If there is a conflict, the event is discarded. Given two events, we can check if they conflict like so:

```
if event.clashes_with(other_event): print "Events Clash!"
```

The result of the computation is all events which do not conflict with the personal calendar. These, by default, are printed to `stdout`, along with timing and number of comparisons to `stderr`.

2.1.2 Comparisons

As part of your implementations, you will be asked to record the number of comparisons performed. A Python module `comparelib` is provided with a `class Counter` that is used as a global to track the number of comparisons.

A general rule of thumb is that you must call `counter.increment()` every time you call `event.clashes_with(other_event)`. Specific instructions will be given as required in the comments.

2.1.3 Testing

A unit test `test_clashes_linear.py` is provided which can be used to test the basic functions in `clashes_linear.py`. You can run this file in your preferred IDE or from the command line like so:

```
python ./test_clashes_linear.py
```

The provided unit test covers basic functionality but it doesn't cover all edge cases. You may want to add additional checks to the unit test to verify the correct functioning of your program.

Initially, if you run this, you will get lots of errors. This is normal, as you haven't completed any parts of the implementation. As you complete various parts of the implementation, an increasing number of unit tests should pass successfully.

2.2 Task 2 (20%)

The file `clashes_binary.py` contains an incomplete implementation of a binary search algorithm that checks for conflicting calendar items using an $O(\log n)$ search.

You must complete this implementation similarly to Task 1.

You can run the script from the command line like so:

```
python ./clashes_binary.py data/calendar.txt data/events_*.txt
```

The meaning of the input files is identical to Task 1.

You can assume that the calendar and events are in sorted order and within a specific set of events there are no overlaps.

2.2.1 Binary Search

The implementation requires that you implement a binary search. Essentially we are concerned with finding the first event where the `event.start_timestamp > possible_event.start_timestamp`. Once we know this, we can efficiently check for any clashes. Standard implementation for binary search is sufficient.