Assignment 1

COS 212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Deadline: 08/03/2019 at 12:00

General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of 0. Only the output of your program will be considered for marks, but your code may be inspected.
- **All submissions will be checked for plagiarism.**
- Read the entire assignment before you start coding.
- You will be afforded three upload opportunities.

Plagiarism:

The Department of Computer Science regards plagiarism as a serious offence. Your code will be subject to plagiarism checks and appropriate action will be taken against offending parties. You may also refer to the the Library's website at `www.library.up.ac.za/plagiarism/index.htm` for more information.

After completing this assignment:

Upon successful completion of this assignment you will have implemented your own dynamic mini-planner application that can hold your weekly schedule.

Your task

The relationship between sparse tables and matrices is similar to the relationship between linked lists and arrays. Sparse tables can be seen as linked lists in more dimensions. Refer to section 3.6 in the textbook. Your task will be to create a mini diary which you can use to schedule weekly events.

This assignment is divided into a number of steps. You must implement all of the steps. It is strongly advised but not required that you implement this assignment in the order in which the steps are given.

Step 1: Creating an Event object

Download the archive `assignment1Code.zip`. You will have to complete the `Event` class which was given to you. Objects of this class should be linkable to three objects of the same type. The idea is that the objects link to the right according to the days of the week (Monday through Sunday), and link down according to the time of the day (06:00, 06:30,. . ., 21:30, 22:00). For the sake of convenience and efficiency, the objects should also link up according to the time of the day, but in reverse order. Further, the latest object should store a reference to the first object, and the first object should store a reference to the last object. `Event` objects will be the nodes in your sparse table.

You are also given the file `Main.java`. Write your test code in this file. It will be overwritten for marking purposes.

Step 2: Creating the Schedule

Insertion

The `Schedule` class represents the sparse table. You have to complete this class by implementing all of the methods.

- The sparse table should be indexed by the days of the week (i.e. Monday to Friday) and the times of the day in half-hour intervals, starting from 6:00, and ending at 22:00 (i.e. 6:00, 6:30, . . ., 22:00).
- Events that are longer than half an hour should be stored as multiple half-hour events that add up to the total duration of the event.

Your sparse table should allow for the insertion of `Event` objects, constructed with given parameters (time, day, description, duration). If there is already an `Event` object at a particular day/time, then the event should be inserted into the next (later) available position. If no later available time slot of correct length can be found on the same day, the following day has to be considered. For Sunday, consider Monday to be the following day.

Some examples:

Consider *Figure 1.* The schedule depicts the following scenario:

- There is one event `A` at 06:30 on Monday.
- There are two events on Wednesday, `B` and `C`. B is the first event, at 07:00, C the second, at 21:30.
- There are three events `D, E` and `F` on Sunday, at 07:00, 21:30, and 22:00, respectively.

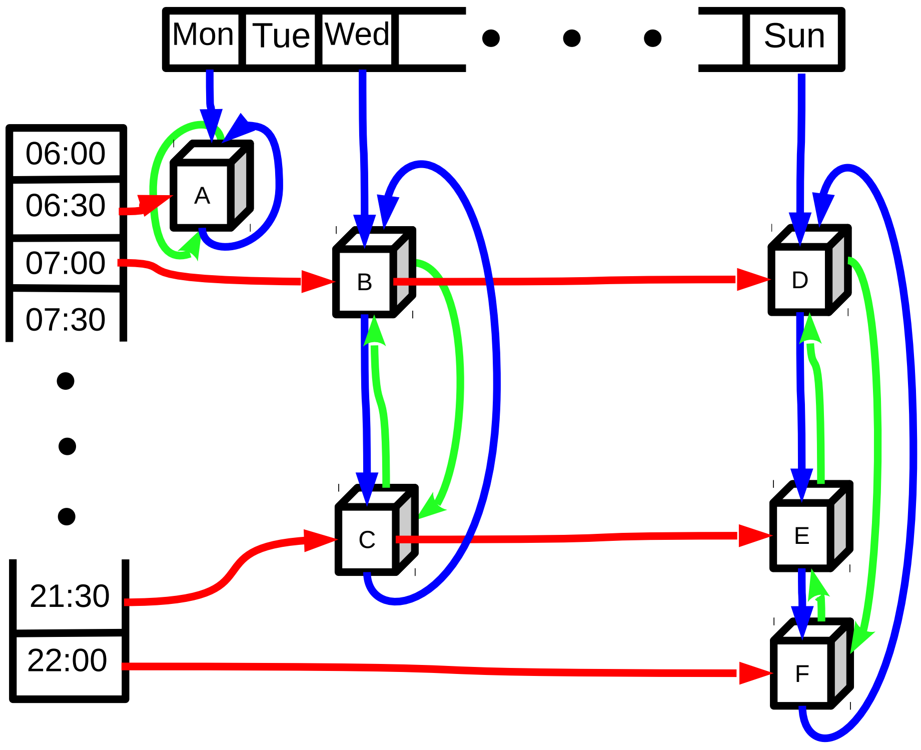

Figure 1: A weekly schedule implemented as a sparse table

In *Figure 1*, red arrows correspond to `right`, blue arrows to `down`, and green arrows to `up` references. Consider that a new event `G` is to be added:

- Adding `G` on Monday at 07:00: Create a new node `G`. The index `07:00` is set to point to this new node, and `G` is now the head at `07:00`. Thus, the `right` reference of `G` should point to `B`. Because `G` is not the first event on Monday, navigate the Monday list to the place where `G` belongs – after `A`. Thus, `A`'s down reference should be set to `G`, and `G`'s up reference should be set to `A`. Because the list is circular, `G`'s down reference should also point to `A`, `A`'s up reference should now point to `G`.
- Adding `G` on Wednesday at 07:00: Navigate to `B`. Because the 07:00 slot is taken by `B`, find the next open slot (07:30). Now, `G` will become the head of the 07:30 list (right references). `B`'s down reference should point to `G`, and `G`'s down reference should point to `C`. `C`'s up reference should point to `G`, and `G`'s up reference should point to `B`.
- Adding `G` on Sunday at 21:00, where `G` is 1 hour long: We have to insert two consecutive `G` objects, half-hour each, at 21:00 and 21:30. We cannot insert two `G` objects before `E`, because then the second object will overlap with `E`. Search the rest of Sunday (going down, not up) for two consecutive empty slots. Since none can be found, consider Monday. On Monday, two consecutive slots can be foud after `A`. Thus, insert two `G` objects on Monday after `A`, at 07:00 and at 07:30. Update the corresponding lists (Monday, 07:00, 07:30).

Valid days and times

The times and days are String values. Strings with the abbreviations of the days are used as valid inputs for the days of the week. Use the following strings to represent each day:

- `Mon` for Monday.

- `Tue` for Tuesday.
- `Wed` for Wednesday.
- `Thu` for Thursday.
- `Fri` for Friday.
- `Sat` for Saturday.
- `Sun` for Sunday.

Event times are stored in the following string format: `hh:mm`. Valid values for hours (`hh`) are in the range `06` to `22`. Valid values for minutes (`mm`) are either `00` or `30`. First available time is `06:00`, last available time is `22:00`. If an invalid time is provided, set the time to `06:00`.

Your implementation must not be case-sensitive. You may assume that only valid day values will be provided. You may assume that the schedule will have enough time slots for all events, provided that conflict resolution is implemented as described above. Write code to test your implementation in `Main.java`.

Deletion

There are two deletion methods to complete.

1. The first one accepts the day and time as parameters, and deletes the corresponding event from the list. Note: all adjacent events on the same day with the same description should also be deleted.
2. The second one accepts an event description, searches the entire sparse table, and deletes all events that match the provided description.

Some examples:

Once again, consider *Figure 1.* The following are some example delete operations:

- Deleting `B`: `07:00` should now point to `D`, and `Wed` should point to `C`. `C`'s up and down references should both point back to `C`.
- Deleting `E`: `D`'s down reference should reference `F`, `F`'s up reference should reference `D`, and `C`'s `right` reference becomes `null`.
- Deleting `A`: The `down` reference of `Mon` and the `right` reference of `06:30` should be set to `null`.

Write code to test your implementation in `Main.java`.

Clearing

In addition to deleting a single item you must also provide functionality to clear the table for a particular index. For example, the request could be made to clear Monday, in which case `all` events are removed for Monday. Be careful not to delete items from other days. The same holds for times, e.g. clear the 07:00 events for all days of the week. You should also provide functionality to clear the entire sparse table upon request.

Write code to test your implementation in `Main.java`.

Step 3: Querying the sparse table

Create a mechanism for your table in order for it to be queried. There are three types of node access queries:

- Given the day and time, your table should return the event node for this combination. If there is no such node (no event), you must return `null`.
- Given only the day, for example Monday, you must return the `head` node for the day.
- Given only the time, for example 07:00, you must return the `head` node for that time.
- Given the event description, you must find the first event that matches the description, and return the corresponding node.

Write code to test your implementation in `Main.java`.

Submission instructions

You must create your own makefile and submit it along with your Java code. Your code should be compiled with the following command:

```
javac *.java
```

Your makefile should also include a `run` rule which will execute your code if typed on the command line as in `make run`.

Once you are satisfied that everything is working, you must tar all of your Java code, including any additional files which you've created, into one archive called sXXX.tar.gz, or sXXX.zip, where XXX is your student/staff number. Submit your code for marking under the appropriate link (Assignment 1) to the assignments portal before the deadline.