Assignment 1: To-Do Calendar Sparse Table

COS 212

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

Department of Computer Science
Deadline: 06/03/2020 at 18:00

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 1-dimensional 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 zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- 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 considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.library.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then choose the *Plagiarism* option under the *Services* menu). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

After completing this assignment:

Upon successful completion of this assignment you will have implemented your own dynamic mini-planner application that can hold prioritised lists of to-do items for an entire year.

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. A sparse table need not be only two dimensional, they can be extended to any dimension.

Refer to Section 3.6 in the textbook. For this assignment you will have to implement and extend a sparse table from two dimensions to three. Your task will be to create a mini-diary which you can use to schedule to-do items.

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 Item object

Download the archive `assignment1Code.tar.gz`. You will have to complete the `Item` 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 ($1^{st}, 2^{nd}, ..., 31^{st}$) of the months, link down in term of the months (January to December) and link backwards for Items on the same date. `Item` objects will be the **nodes** in your sparse table. These represent the items on your to-do list, where every day of the year will have its own to-do list.

Each `Item` object will have a description, a duration, and a priority. These values do not have to be unique.

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

Step 2: Creating the Calendar

Insertion

The `Calendar` 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 months of the year (i.e. January to December) and the days of the months (i.e. 1 to 31).
- To-do items on the same day should be stored in a linked list, sorted according to the assigned priorities. The lowest priority is zero (0). The higher the priority, the closer the `Item` object should be to the head of the list. If some or all items on a particular date have the same priority, then the newly inserted `Item` should be added at the end of the list or sub-list of the matching priority. In other words, if priorities are the same, use insertion order instead.

Your Sparse table should allow for the insertion of `Item` objects, constructed with given parameters (day, month, description, duration, priority). If there is already an `Item` object at a particular date, then the object should be inserted into the correct position (according to priority) of the corresponding list of items for that particular day, by using the "back" references. Only the first item on the list should be linked with right/down nodes to describe the relationship between days (right) and months (down). This node serves as the head for the to-do list of items for the entire day (back). All lists in the sparse table must be kept in a chronological or a prioritised order at all times. You might consider using an organized list-like structure for this. You may assume all months of the year have 31 days and you do not have to check for valid dates in this regard.

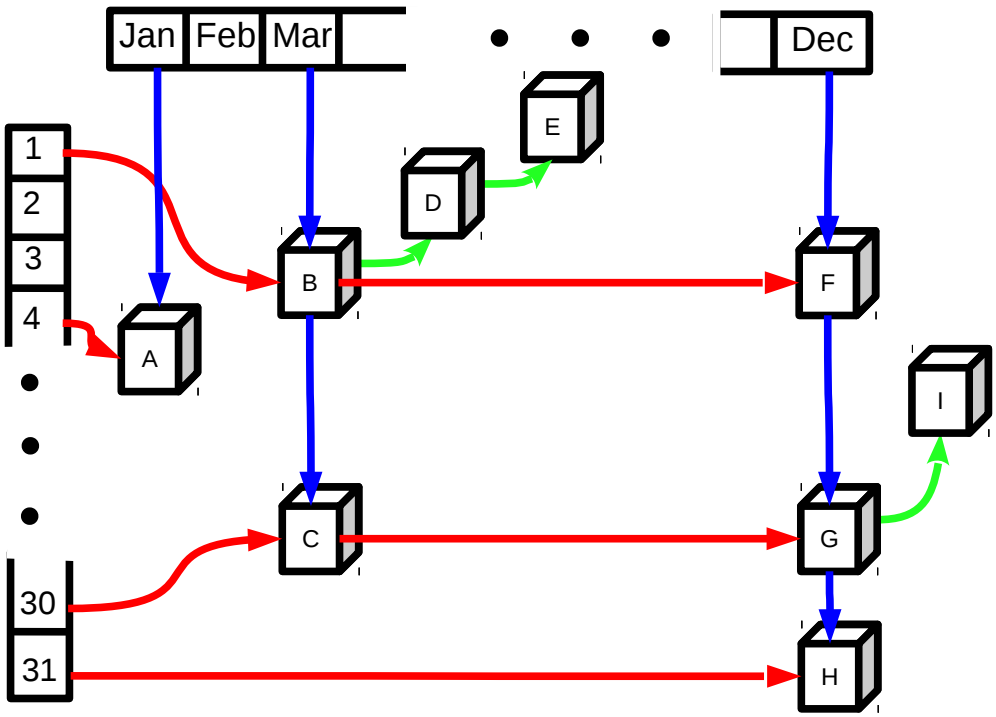

Figure 1: A calendar implemented as a sparse table

Some examples:

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

- There is one item `A` on 4 January.
- There are three items `B, D` and `E` on 1 March. B is the first item, D the second, and E the last. Assume they all have a priority of zero.
- There is one item `C` on 30 March.
- There is one item `F` on 1 December.
- There are two items `G` and `I` on 30 December. item `I` is scheduled after item `G`. Assume `G` has a priority of 2, and `I` has a priority of 0.
- There is one item `H` on 31 December.

In *Figure 1*, red arrows correspond to `right`, blue arrows to `down`, and green arrows to `back` references. New items on the same day should be inserted according to the assigned priorities, using order of insertion in case duplicate priorities are encountered. Consider that a new item `J` is to be added:

- Adding `J` on 2 January: Create a new node `J`. The index `2` is set to point to this new node and `J` is now the head at `2`. `J`'s down reference should be set to `A`, and the original `Jan` index should be set to `J`. `J` should now be the head of the list at index `Jan`.
- Adding `J` on 1 March: Navigate to `B`. `B` is the head of the sorted list to which `J` has to be added. Compare `J`'s priority to the priorities of other items in the list, and update `back` references accordingly. If `J`'s priority is 0, add it an the end of the list (after `E`). If `J`'s priority is 1 or any other value above 0, make it the head of the list (insert in front of `B`).
- Adding `J` on 30 January: `A`'s `down` reference should be set to `J`. `J`'s right reference should point to `C`, and the reference at index 30 should now point to `J`. `J` has replaced `C` as the head of the list at index 30.

Valid input values

The days are integer values from 1 to 31. Strings with the abbreviations of the months are used as valid inputs for the months. Use the following strings to represent each month:

- `Jan` for January.
- `Feb` for February.
- `Mar` for March.
- `Apr` for April.
- `May` for May.
- `Jun` for June.
- `Jul` for July.
- `Aug` for August.
- `Sep` for September.
- `Oct` for October.
- `Nov` for November.
- `Dec` for December.

`Item` durations are stored in the following string format: `hh:mm`. Valid values for hours (`hh`) are in the range `00` to `24`. Valid values for minutes (`mm`) are in the range `00` to `59`. Examples of valid durations: `00:30` (half an hour), `01:00` (one hour), `06:00` (six hours), `24:00` (whole day - can't be longer!). If an invalid duration is provided (for example, `24:59`), set the duration to `00:00` (placeholder for "unknown").

Item descriptions can be any string, eg. "Study for COS212", "Hang out with friends", etc. Priorities can only be non-negative integers. If a negative priority is supplied, set it to zero.

Your implementation must **not** be case-sensitive, eg. it should accept both `JAN` and `jan` as valid inputs. You may assume that only valid month and day values will be provided. Write code to test your implementation in `ToDo.java`.

Deletion

There are mtuliple deletion methods to complete.

1. The first one accepts the month, day and description as parameters, and deletes the corresponding `Item` from the list. If there are multiple items with the same description, delete the first one encountered in the list.
2. The second one accepts the month, day and priority as parameters, and deletes all items of the given priority from the given date. This would allow you to clear the list of unimportant items.
3. The third one accepts the month and day as parameters, and deletes **all** items on that specific day.

Some examples:

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

- Deleting `B`: `D` should take its place in the table.
- Deleting `C`: 30 should now reference `G`, and `B`'s `down` reference becomes `null`.
- Deleting `A`: The `down` reference of `Jan` and the `right` reference of 4 should be set to `null`.

Write code to test your implementation in `ToDo.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 January, in which case `all` items are removed for January. Be careful not to delete items from other months. The same holds for days, e.g. clear the 2nd for all months for the year. You should also provide functionality to clear the entire table upon request.

Write code to test your implementation in `ToDo.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 queries:

- Given the day and month, your table should return the `head` node for this combination (first Item). If there is no such node (no Item), you must return `null`.
- Given only the month, for example March, you must return the `head` node for the month.
- Given only the day, for example 4, you must return the `head` node for the day.

Write code to test your implementation in `ToDo.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 commandline 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, where XXX is your student/staff number. Submit your code for marking under the appropriate link (Assignment 1) before the deadline.