CSC 212 Project
Developing an Application for Manipulating Subtitles
Phase II

College of Computer and Information Sciences
King Saud University

Fall 2017

In this phase, your goal is to speedup the access to the subtitles using a search tree. To this end, you will use a sorted map, which is a data structure that allows for search and traversal of the elements in increasing or decreasing order of the keys. A sorted map can be implemented as a BST with backward and forward links (basically it is a BST + DoubleLinkedList) (see an example in Figure 1).

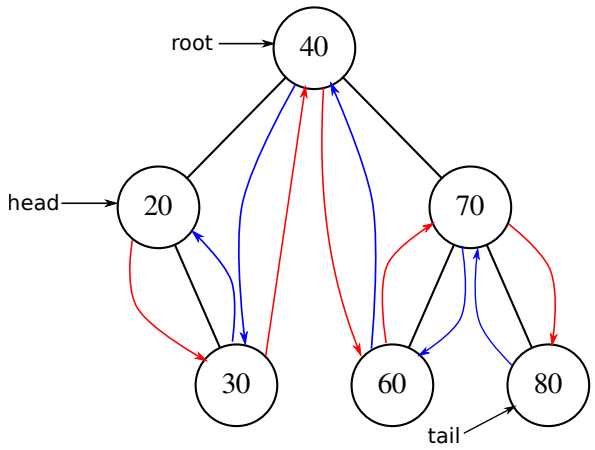

Figure 1: Example of a sorted map implementation. The black edges represent tree edges (left and right children). Red edges represent forward links (next). Blue edges represent backward links (previous).

1 Requirements

1. Use the following data structure to store the subtitles (**no additional data structure can be used**). You are given an implementation of this interface in the class `SortedBST`:

```
// The sorted map interface. K represents the key type, which must extend
    Comparable,
```

```
// and T representds the stored data type.
public interface SortedMap<K extends Comparable<K>, T> {

  // Return the size of the map. Must be O(1).
  int size();

  // Return true if the tree is empty. Must be O(1).
  boolean empty();

  // Return true if the tree is full. Must be O(1).
  boolean full();

  // Move current to the first element (the smallest key). The map must not be
  // empty. Must be O(1).
  void findFirst();

  // Move current to the last element (the largest key). The map must not be
  // empty. Must be O(1).
  void findLast();

  // Move current to the next element (the key immediately greater than the
  // current key). The map must not be empty, and current must not be last. Must
  // be O(1).
  void findNext();

  // Move current to the previous element (the key immediately smaller than the
  // current key). The map must not be empty, and current must not be first.
      Must
  // be O(1).
  void findPrevious();

  // Returns true if current is the first element (the smallest key). The map
      must
  // not be empty. Must be O(1).
  boolean first();

  // Returns true if current is the last element (the largest key). The map must
  // not be empty. Must be O(1).
  boolean last();

  // Return the key and data of the current element
  Pair<K, T> retrieve();

  // Update the data of current element.
  void update(T e);

  // Search for element with key k and make it the current element if it exists.
  // If the element does not exist the current is unchanged and false is
      returned.
  // This method must be O(log(n)) in average.
  boolean find(K key);

  // Return the number of nodes in the search path for key. Must be O(log n).
  int nbNodesInSearchPath(K key);
```

```
  // Update the current key to the new value newKey if possible and returns true
      .
  // If newKey does not respect the order, no change is made and false is
  // returned. Must be O(1).
  boolean updateKey(K newKey);

  // Insert a new element if does not exist and return true. The current points
      to
  // the new element. If the element already exists, current does not change and
  // false is returned. This method must be O(log(n)) in average.
  boolean insert(K key, T data);

  // Remove the element with key k if it exists and return true. If the element
  // does not exist false is returned (the position of current is unspecified
  // after calling this method). This method must be O(log(n)) in average.
  boolean remove(K key);

  // Remove the current element. The next element if it exists is made current,
  // otherwise current moves to the first element. This method must be O(log(n))
  // in average.
  void remove();

  // Return in a list all elements with key k satisfying: k1 <= k <= k2.
  List<Pair<K, T>> inRange(K k1, K k2);
}
```

Remark 1. *The choice of the key is important in this case. Choosing the start or end times as key does not provide the necessary performance. A better solution is to use the subtitle time interval as the key. It is recommended to define a class* `TimeInterval` *which implements the interface* `Comparable` *and use it as a key (the data is the subtitle itself):*

```
public class TimeInterval implements Comparable<TimeInterval> {
...
  @Override
  public int compareTo(TimeInterval that) {
    if (startTime.compareTo(that.endTime) > 0) {
      return 1;
    }
    if (endTime.compareTo(that.startTime) < 0) {
      return -1;
    }
    return 0;
  }
}
```

Notice that this method considers equal any two intervals that overlap.

2. Implement the following interface:

```
// This interface represents a subtitle sequence.
public interface SubtitleSeq {

  // Add a subtitle.
```

```
  void addSubtitle(Subtitle st);

  // Return all subtitles in their chronological order.
  List<Subtitle> getSubtitles();

  // Return the subtitle displayed at the specified time, null if no
  // subtitle is displayed.
  Subtitle getSubtitle(Time time);

  // Return the number of nodes in the search path for finding the subtitle.
  int nbNodesInSearchPath(Time time);

  // Return, in chronological order, all subtitles displayed between the
  // specified start and end times. The first element of this list is the
  // subtitle of which the display interval contains or otherwise comes
  // Immediately after startTime. The last element of this list is the
  // subtitle of which the display interval contains or otherwise comes
  // immediately before endTime.
  List<Subtitle> getSubtitles(Time startTime, Time endTime);

  // Shift the subtitles by offseting their start/end times with the specified
  // offset (in milliseconds). The value offset can be positive or negative.
  // Negative time is not allowed and must be replaced with 0. If the end time
  // becomes 0, the subtitle must be removed.
  void shift(int offset);
}
```

3. Use the class `Perf` to evaluate the performance of the Phase I code and Phase 2 code. Report the results in your report in the form of a table and analyze them.

2 Deliverable and rules

You must deliver:

1. A report written using the provided template.
2. Source code submission to Web-CAT.

The submission **deadline** is: **19/12/2017** (this is a hard deadline and will not be extended).

You have to read and follow the following rules:

1. The specification given in the assignment (**class and interface names, and method signatures**) must not be modified. Any change to the specification results in compilation errors and consequently the mark zero.
2. All data structures used in this project **must be implemented** by the students. The use of Java collections or any other library is strictly forbidden.
3. This project is to be conducted by groups of **four** students. Groups of more than four students are not accepted. Groups of less than four students are strongly discouraged and can only be accepted with a special permission from the course instructor.

4. All the members of a group must have the **same course instructor**.

5. All students must **submit** the list of their **group members** within one week of the announcement of this project. Once the groups are chosen, no student can change the group (even if some group members have dropped the course).

6. **Every member** of the group must participate in **all parts of the project**: designing the software, programming and writing the report. Members of the same group may receive different marks according to their participation in the project.

7. The submitted software will be evaluated automatically (using Web-Cat) and in a demonstration (after phase 3) to which all the group members must attend.

8. Any member of the group who fails to **attend the demonstration** without a proper excuse (consult the university and college regulations) shall receive the **mark 0** in the project.

9. In accordance with the university regulation, **cheating** in the project will be sanctioned by the **grade F**.