Algorithms & Data Structures
Exercise Sheets

Weeks 9-10: Searching dynamic sets efficiently

Submit your solutions via the peergrade page on Brightspace, Week 9-10. Please make sure to submit your solutions **before the deadline** (typically a Friday, but check the deadline in peergrade). If an exercise requires you to write C++ code, then create a new C++ file for each exercise (i.e. do not combine exercises into a single file); if two or more exercisesare about the same ADT, it is OK to use the same file. You MUST hand-in a pdf with the solutions. If code is needed, copy-paste the code into the pdf. Submit the code as a ZIP file, and note in the pdf what file (cpp, h or tpp) that solves the problem. You can hand-in in pairs of two. If an exercise requires other type of meterial (e.g. video), write the name of the video-file in the pdf and include the video in the Zip file.

Note that you also needs to review both other students' and your own solution. If you handed in as a group, you will have to review your submission individually (I kindly ask you to do it individually, that gives you the best learning outcome). The deadline for revievs is typically Thursday after the hand-in at mid-night (BUT check the actual deadline in peergrade).

Exercises

(1) Make a video expleaning how a AVL tree is build with the following elements $[10, 20, 15, 25, 30, 16, 18, 19]$ (i.e.10 is inserted first, then 20 ...). You must illustrate all rotations etc. Then illustrate what happens when 30 is deleted and then 18

(2) A node in a binary tree is an only-child if it has a parent node but no sibling node (Note: The root does **not** qualify as an only child).
The *loneliness-ratio* (LR) of a given binary tree T is defined as the following ratio:

$$LR(T) = \frac{\text{The number of nodes in T that are only children}}{\text{The number of nodes in T}} \tag{1}$$

   (a) Prove that for any nonempty AVL tree T, we have that $LR(T) \leq 1/2$

   (b) Is it true for any binary tree T,that if $LR(T) \leq 1/2$ then $height(T) = log(n)$?

(3) The `Set` is an ordered container that does not allow duplicates. Write an implementation of the `Set` class using the `BinarySearchTree` implementation we saw in class as the internal data structure. Add to each node a link to the parent node to make traversal easier. The supported operations are:

```
iterator insert(const Object& x);
iterator find(const Object& x ) const;
iterator erase(iterator& itr);
```

The `insert` function returns an iterator to that either represents the newly inserted item or the existing duplicate item that is already stored in the container. For

searching, the `find` routine returns an iterator representing the location of the item (or the endmarker if the search fails). The `erase` function removes the object at the position given by the iterator, returns an iterator representing the element that followed `itr` immediately prior to the call to erase, and invalidates `itr`.

(4) Design and implement a linear-time algorithm that verifies that the height information in an AVL tree is correctly maintained and that the balance property is in order for all nodes.

(5) Let T by a hash-table of size 7 with the hash function $h(x) = x \, mod \, 7$. Write down the entries of T after the keys 5, 28, 19, 15, 20, 33, 12, 17, 33 and 10 have been inserted using

   (a) chaining

   (b) linear probing

   (c) quadratic probing

   What is the load-factor(λ) in the three cases

(6) Suppose that a dynamic set S is represented by a `HashTable` T of size m. Describe and implement a procedure that finds the maximum element of S. What is the worst-case complexity of your procedure? Is it possible to modify the data structure to optimize for this case?