Algorithms & Data Structures: Lab 18

week of 11th March 2019

1 Setup

1.1 Saving your work from last week

By now you should be familiar with the operations needed to save your work. Make sure you commit your work to version control often, and always have a backup copy, ideally remotely (for example in your own account on the department's gitlab installation.)

1.2 Downloading this week's distribution

Once you have successfully saved your changes from last week, you can get my updates by doing

```
git pull
```

which *should* automatically merge in new content. After the `git pull` command, you should have a new directory containing this week's material (named `18/`) alongside your existing directories.

2 Tries

This lab exercise involves implementing a trie data structure to store sets of strings. To simplify things, we will restrict our alphabet to 26 lower-case letters, and only implement insertion and exact querying (not prefix querying).

2.1 Standard Trie data structure

Our implementation of the standard trie data structure will represent each node as a 27-element array of `Trie` objects (Java) / `Trie *` pointers (C++). If the set of strings contained in the trie contains a string whose next character at this point is `c`, then the corresponding element of the array should be the child Trie; if it does not contain any such string, the corresponding element of the array should be `null` (Java) / `NULL` (C++).

As a special case, you will need to handle the end-of-word character, which we will take as being `{` (for reasons which might become obvious if you consider ASCII offsets from `a`). You will need some kind of non-null Trie to store in the end-of-word (27th) position, but you should never need to look inside that Trie. (Does this remind you of anything, for example in last term's work on linked lists?)

2.1.1 Insert

Implement the `insert` method for Trie objects, which should:

- if necessary, construct a `children` array for the Trie;

- if necessary, construct a `Trie` object for the current character of the string being inserted;
- insert the substring of all but the first character into the Trie for the first characters.

Don't forget to include an end-of-word marker in the Trie at the end of the insertion!

2.1.2 Query

Once you have implemented `insert`, you should be able to implement the MEMBER operation from the lectures (here called `query`). This should return `true` if the query string is exactly present in the Trie, and `false` otherwise.

2.2 Compressed Trie data structure

A compressed trie is like a standard trie, except that if any node other than the root node has only one descendant, the path to that descendant is collapsed into its parent node.

We will represent each node as a `Map` (Java) / `map` (C++), mapping `String` to `CompressedTrie` (Java) / `string` to `CompressedTrie *` (C++). If the compressed trie contains an edge to a node, then the map contains an entry whose key is the label of the node and whose value is the child node.

2.2.1 compressTrie

Implement `compressTrie`, which should take a standard trie as input and produce the corresponding compressed trie.

2.2.2 Query

Implement `query` for compressed tries, which should have exactly the same results as `query` on the equivalent standard trie.

2.2.3 [Optional] Insert

Inserting into a compressed trie will in general require splitting existing edges in two, if a string is inserted which matches the beginning but not the end of any given label. Implement `insert` for compressed tries; you will need to write your own tests for this method.

3 Code submission

Submit your work on tries, along with answers to the optional extra questions, to the submission area for this activity on the VLE by **Friday 29th March** at **16:00**. As usual, you may submit more than once, and the highest-achieved score is retained.

4 Peer-assessment

4.1 Submission

Your draft summaries, for peer-assessment, are due on the VLE by **16:00** on **Friday 15th March** (submissions are open from **16:00** on **Wednesday 6th March**). Failure to submit a draft will lead to an automatic mark of zero for both the submission and the assessment part of this activity.

4.2 Peer-assessment

Your assessments of other people's draft summaries are due on the VLE by **16:00** on **Friday 22nd March** (the activity is open for peer-assessment from **16:00** on **Friday 15th March**). Failure to submit peer-assessments will lead to an automatic mark of zero for the assessment part of the activity.