A doubly linked list

Algorithms and data structures ID1021

Johan Montelius

Fall term 2022

Introduction

Once you master linked lists it's time to learn about doubly linked lists. A double linked list is a list where you not only have one *forward pointer* but also a *previous pointer*. A reference to the previous element will come in handy when we want to remove an element in the list without traversing the list from the beginning.

A benchmark

A doubly linked list does not provide any new functionality compared to a single linked list. Some operations are however more efficient but we pay a price for this efficiency since we need to update more references. In order to see the difference you should implement a benchmark that can be used with either a single linked list or a doubly linked list and then compare the result.

The benchmark should keep a list of n elements and then perform k remove and add operations. The elements that should be removed should be selected by random so you will need a separate data structure to keep all n elements. Once you have randomly selected one you remove it from the list and then add it back.

In order to compare the two data structures side by side you should set up identical sequences of operations for them to perform. A `sequence` of randomly chosen indexes in the range $0..(n-1)$ might serve as a good solution.

```
int[] sequence = new int[k];
Random rnd = new Random();

for (int i = 0; i < k; i++) {
  sequence[i] = rnd.nextInt(n);
}
```

When this sequence is created before the actual benchmark you will also avoid measuring the time it takes to generate the random numbers.

Double linked list

A doubly linked list is as the name suggests a linked list where each node also has a reference to the previous node in the list. The very last node in the list will as usual have a null *next pointer* but now we will also have special case for the first node in the list.

When you implement the remove method the idea is to change the two adjacent nodes in the list. If we have the nodes E, F and G, and want to remove F, then the next pointer of E should refer to G and the previous pointer of G to E. When you implement this you have to handle the special cases when F happens to be the first or last node in the list.

To make things easy the `add` method will simply add a node to the front of the list. Adding in a doubly linked list becomes slightly more expensive since we have to update the previous pointer of the current first node in the list.

Presentation

Run some benchmarks where you vary n to see how the two data structures perform (try also very low numbers of n). When you present you findings make sure that you present numbers in a sensible form. If you say that a million operations are done in $63452342ns$ then this is a ridiculous statement since the next time you run your benchmark the result might well be $62871342ns$. The number of significant figures that you present should reflect your confidence level. In this example that could mean $60000000ns$ or why not $60ms$. If you have done extensive benchmarks you might write $63ms$ but if you present more than three significant figures you need to justify how accurate the measurements are.