A doubly linked list

Algorithms and data structures ID1021

Johan Montelius

Fall term 2023

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 doubly linked list

Start by implementing a double linked list much in the same way as you implemented the single linked list. The difference is that each cell in the sequence now has two references (besides the head), a `next` cell in the list and the `previous` cell in the list.

Implement the same functions as before:

- `void add(int item)` : add the item as the first cell in the sequence.
- `int length()` : return the length of the sequence.
- `boolean find(int item)` : return true or false depending on if the item can be found in the sequence.
- `void remove(int item)` : remove the item if it exists in the sequence.

You will see that they are almost the same as before, the remove might be tricky but should not impose a problem.

The true advantage of a doubly linked list is when you want to remove a specific cell i.e. not a cell with a specific value but a cell identified by a reference. In order to do this you will have to make the cell structure visible out side of the doubly class. The `unlink` method should thus be given a cell and unlink it from the doubly linked list. The advantage is that we do not

have to search for its position in the list, the cell holds all information that we need.

Also implement the same method for your single linked data structure. In the single linked structure it does not matter if we have a reference to the cell we want to unlink since we do not know its previous cell; you need to start form the beginning and search for the cell in order to unlink it.

For both data structures also implement a method that `insert` a cell as the first cell in the list i.e. similar to `add` but now we have the cell in question.

A benchmark

You should benchmark how long time it takes to unlink a cell and then insert it and do this for k randomly selected cells (k being for example 1000). You should run the benchmark using both a doubly and single linked list. The length of the lists should increase to see how the two data structures behave as the lists become longer.

In order to randomly select a cell you could try something like follows. Create a list of n cells but when you create the cells you not only add them to the list but also keep a reference to them in an array of length n (let's call this the *cell array*).

Create an array of k random numbers between 0 and $n - 1$; this is the array of indices of cells that you should unlink. In the benchmark you measure the time it takes to run through all k indices and for each index find the cell in the cell array. Given the cell reference you first unlink the cell and then add it to beginning of 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.