CE204 Lab 1: Stacks and Queues

David Richerby

Week 17, 2020–21

The lab exercises are not assessed and you are not required to complete all of them, though I recommend that you attempt them all. Feel free to work with others and talk about your answers, especially for the discussion questions I've included in some of the exercises.

Solutions will be released on Moodle, on the Friday after the labs.

1 Implementing stacks with references

a) Write a class called `LinkedStack` that implements the stack ADT using references. Implement the `push()` and `pop` methods for this setting – they will be similar to the implementation of the queue ADT shown in the lectures.

b) Add a method `void multiPush(String[] strings)` that pushes all the strings in its array argument onto the stack.

c) Add a method `String[] multiPop(int n)` that returns an array of `n` elements popped from the stack. If there are fewer than `n` elements in the stack, it should return all the items.

d) Add a method `void merge (LinkedStack that)` that corresponds to placing the stack `that` on top of the stack `this`. That is, repeatedly calling `pop()` on the merged stack until it is empty should give the same sequence of `String`s that you would have got by calling `that.pop()` until it is empty, followed by calling `this.pop()` until it is empty.

2 Implementing stacks with arrays

Make a copy of the `Stack` class from the lectures, which you can download from Moodle.

a) Modify the `pop()` method so it decreases the size of the array by 50% when it is less than 25% full.

b) Why not decrease by 50% when the stack is less than 50% full?

c) Modify the `push()` method so the class keeps track of how many array elements have been copied. Verify the claim in the lectures that growing the array by adding a fixed number of cells each time it fills requires much more copying than growing it by multiplying the size by a constant. For example, compare growing by adding 100 cells with growing by doubling, when you do 10 000 pushes. What happens when you push even more items?

3 Queueing theory

This exercise simulates a queue of people waiting to be served at a business such as a shop or bank. On average, a new customer arrives once every t_a minutes and joins the queue to wait to be served. When a customer reaches the front of the queue, the server takes, on average, t_s minutes to serve them. (t_a and t_s don't have to be integers.)

We can simulate the queue like this. Initially, the queue is empty and the server is idle. Every minute, we do the following things.

- With probability $1/t_a$, a new customer arrives and joins the queue. Represent this by adding the current time to the queue.
- If the server is serving somebody then, with probability $1/t_s$, they finish and become idle.
- If the server is idle, they begin to serve the first customer in the queue, if there is one.

In Java, you can do something with probability `p` using

```
if (rand.nextDouble () < p) {...},
```

where `rand` is an object of type `util.Random`.

a) Download `IntQueue.java` from Moodle and use it to write a program to simulate the system described above.

b) Investigate the behaviour of the queue as you vary t_a and t_s. Simulate, say, 10 000 minutes and compute the average waiting time. If t_s is much less than t_a, you should find that the average waiting time is about t_s. Why is this? What happens when $t_s > t_a$? Why? What happens when $t_s = t_a$? Did that surprise you? Can you explain it?

c) Larger businesses have $n > 1$ servers. Supermarkets tend to arrange one queue per server, whereas banks typically have a single queue and the customer at the front of the queue goes to the next server who becomes free. Use your simulator to compare these two systems. In the case of

multiple queues, each server has their own queue and each new customer joins one of the queues uniformly at random; customers cannot change queues. When there is just one queue, each server independently finishes serving a customer with probability $1/t_s$ each minute. Which system is more efficient? Why?

If you have time, you could consider other queueing systems, such as smarter supermarket shoppers who join the shortest queue instead of a random one and/or smarter servers who call over a customer from an adjacent queue if they're idle and their own queue is empty. You could also track things like the average queue length and the number of customers who were still queueing at the end of the simulation.