YSC4231: Parallel, Concurrent and Distributed Programming

Programming Assignment 6

In this assignment, you will be practicing your understanding of the idea of *splitting* in parallel collections. To do so, you will have to implement a familiar binary heap structure and enhance it with the possibility to perfrom non-modifying operations in parallel, by means of "heap-respectful" splitting.

The template repository for this assignment is available on Canvas. Once you clone the template repository, you'll be ready to go! The files you need to modify are:

- `ParBinHeap.scala`
- `ParBinHeapSplitter.scala`
- `BinHeapTests.scala`
- `ParBinHeapSplitterTests.scala`

Start by implementing a textbook version of a correct sequential binary min-heap by developing the corresponding methods of `ParBinHeap`. Whenever necessary, feel free to add your own (private) fields and methods. Next, test your implementation by implementing and extending the test suite in the `BinHeapTests` class. The test skeletons have descriptive names, which should hint your implementation. One test is provided; it requires some Scala trickery to work, but you most probably won't need it for other tests.

For the main part of this assignment, implement a splitting discipline by filling the gaps in the class `ParBinHeapSplitter`. The trickiest parts of this tasks are

(a) to ensure that the key splitter operations (*i.e.*, `split`, `dup`, `hasNext` etc) all work in $O(1)$ (*i.e.*, take constant time), which is crucial for the parallel heap's performance, and
(b) to implement splitting discipline in a way that it *respects the heap structure*. In other words, the fragments of the heap managed by splitters should be heaps themselves. Splitting a heap array into arbitrary array chunks that are not heaps won't cut it!

The assignment will only be awarded a full score if both parts (a) and (b) are done right—this will be ensured by the provided tests and benchmarks as described below.

Implement tests in `ParBinHeapSplitterTests` to validate that your parallel heap behaves correctly with standard split-based operations (`reduce`, `aggregate`, `fold`, etc), and that they don't affect the structure of the heap. You are also encouraged to develop any additional tests you deem necessary. The test suite also features a test that checks that your implementation respects the property (b) above. Feel free to study its implementation.

Finally, use the provided `ParBinHeapBenchmarks` to benchmark your implementation of the parallel heap. On the provided case studies, if implemented correctly (*i.e.*, splitting takes constant time), your parallel heap veresion should achieve 2-3 time speed-up compared to the sequential heap version.

To hand in your code for this assignment, submit the link to the tagged release in your repository on GitHub. Feel free to provide additional comments about your implementation and discoveries to the `README.md` file in the root of the project.

Here are some tips to get you started on the assignment:

- You can find an implementation of a binary max-heap in OCaml in the lecture notes for YSC2229.[1]
- The type parameter annotations `T: Ordering: ClassTag` generate the *implicit* constructor parameters of types `Ordering[T]` and `ClassTag[T]` that are passed to the constructors of `ParBinHeap` class. The former one allows one to deduce the priority discipline for the insertions. It will be inferred automatically when working with standard data types, such as `Int` and `String`, but you will need to define it yourself for custom classes, or in the case if the default ordering is not suitable for the problem. The class `BinHeapTests` provides examples on how to do this.
- The parameters of `ParBinHeapSplitter` and their descriptions should provide some idea on how *constant-time* splitting of the heap into sub-heaps should work.
- When implementing splitting, don't worry about the iterator's methods (`hasNext`, `next`, and `remaining`) running before `split`. They are always run on the chunks of the collection *after* all splitting has been performed. That is, splitting and iteration do not interleave.

[1] https://ilyasergey.net/YSC2229/week-05.html