Resit Exam

The Scala part of the exam is worth 50%. It is about 'jumps' within lists.

Important

- Make sure the files you submit can be processed by just calling `scala <<filename.scala>>` on the commandline. Use the template file provided and do not make any changes to arguments of functions or to any types. You are free to implement any auxiliary function you might need.
- Do not use any mutable data structures in your submission! They are not needed. This means you cannot create new `Arrays` or `ListBuffers`, for example.
- Do not use `return` in your code! It has a different meaning in Scala than in Java.
- Do not use `var`! This declares a mutable variable. Only use `val`!
- Do not use any parallel collections! No `.par` therefore!

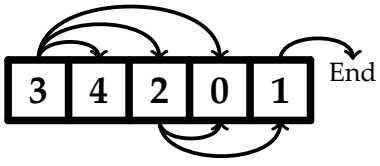
Disclaimer

It should be understood that the work you submit represents your **own** effort! You have not copied from anyone else. An exception is the Scala code I showed during the lectures or uploaded to KEATS, which you can freely use.

During the exam you may **not** communicate with other people: no email, instant messaging, discussion forums, use of mobile phones, etc.

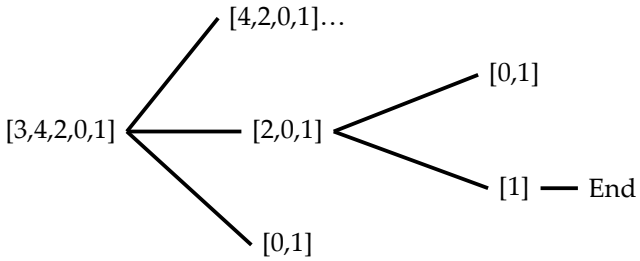
Task

Suppose you are given a list of numbers. Each number indicates how many steps can be taken forward from this element. For example in the list



the first 3 indicates that you can step to the next three elements, that is 4, 2, and 0. The 2 in the middle indicates that you can step to elements 0 and 1. From the final 1 you can step to the End of the list. You can also do this from element 4, since the end of this list is reachable from there. A 0 always indicates that you cannot step any further from this element.

The problem is to calculate a sequence of steps to reach the end of the list by taking only steps indicated by the integers. For the list above, possible sequences of steps are 3 - 2 - 1 - End, but also 3 - 4 - End. This is a recursive problem that can be thought of as a tree where the root is a list and the children are all the lists that are reachable by a single step. For example for the list above this gives a tree like



Tasks

(1) Write a function, called `steps`, that calculates the children of a list. This function takes an integer as one argument indicating how many children should be returned. The other argument is a list of integers. In case of 3 and the list [4,2,0,1], it should produce the list

$$[\ [4,2,0,1],\ \ [2,0,1],\ \ [0,1]\ ]$$

Be careful to account properly for the end of the list. For example for the integer 4 and the list [2,0,1], the function should return the list

$$[\ [2,0,1],\ [0,1],\ \ [1]\ ]$$

[Marks: 12%]

(2) Write a function `search` that tests whether there is a way to reach the end of a list. This is not always the case, for example for the list

[3,5,1,0,0,0,0,0,0,0,0,1]

there is no sequence of steps that can bring you to the end of the list. If there is a way, `search` should return true, otherwise false. In case of the empty list, `search` should return true since the end of the list is already reached.

[Marks: 18%]

(3) Write a function `jumps` that calculates the shortest sequence of steps needed to reach the end of a list. One way to calculate this is to generate *all* sequences to reach the end of a list and then select one that has the shortest length. This function needs to return a value of type `Option[List[Int]]` because for some lists there does not exists a sequence at all. If there exists such a sequence, `jumps` should return `Some(...)`; otherwise `None`. In the special case of the empty list, `jumps` should return `None`

[Marks: 20%]

Hints: useful list functions: `.minBy(..)` searches for the first element in a list that is the minimum according to a given measure; `.length` calculates the length of a list; `.exists(..)` returns true when an element in a list satisfies a given predicate, otherwise returns false; `.map(..)` applies a given function to each element in a list; `.flatten` turns a list of lists into just a list; `_::_` puts an element on the head of the list.