CSCM712 – Software concepts and efficiency March 13th 2026
Cécilia Pradic

Lab 7: queues, stacks and trees

1. **Implementing queues** Open the file `fsqueue.py` available on canvas. It contains a partial implementation of a queue in a fixed size array[1]. As we discussed in the lecture, the idea is that, on top of the array, you have attributes that will tells you where the dequeue is currently sitting in the array.

   Most operations are implemented aside from `dequeue`. Please have a first read through the file and run it to get a sense of what's going on

   (a) Uncomment the two lines 83 and 84 in `fsqueue`. What happens? Modify the implementation of `enqueue` so that an exception is raised instead of triggering this bug.

   > **Tip.** *You can actually declare exceptions of your own if you like, or reuse one from the standard library. Here is a full declaration of a custom exception.*
   >
   > ```
   > class NoRoom(Exeption):
   >   pass
   > ```

   (b) Implement the method `dequeue` in a consistent way, and write some meaningful test for it in the file instead of the existing tests.

2. **Using stacks** Now, you are asked to use a stack for a basic parsing problem as we discussed in the lectures. Inputs will be strings that can contain

   - Either letters or spaces
   - Or parentheses of several kind:

     `'(' ')' '{' '}' '[' ']' '<' '>'`

   In a new file, write and test a procedure

   ```
   def isWellParenthesized(s):
   ```

   that takes as input a string and outputs a boolean saying whether it is well-parenthesized or not. Here are examples of well-parenthesized strings.

   ```
   "aa( aaa aa aa) { ( adcas ) x (af ( fa )ss )[asf  ]afa }aaa"
   "<(<<[]> [ [] ][]>)>"
   "()()aa(bb)()"
   "aaaa"
   ```

   Here are some strings which are not well-parenthesized:

   ```
   "((())"
   "<)"
   ")(())"
   ```

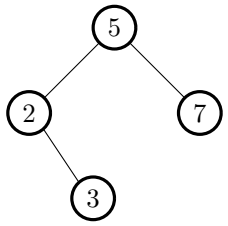
[1] Implemented via a python list attribute for simplicity; but the expectation is that it does not get resized.

For the stack itself, you may simply use python lists for simplicity. The class `list` has a `pop` method, and `push` is essentially the method `append`.

3. **Trees**

   (a) In the file `bTreeIntro.py` file, two examples `example1` and `example2` are defined. Draw the corresponding structures by looking at the code. Check your answer by using the toDot static method and uncommenting the last two lines in the file. Do these structures have sharing? Cycles?

   (b) Write some code that represent the following tree in a variable `example3`:



   (c) Run the program function and try to understand the output of `__str__`. Try to define a variable `example4` such that `str(example4)` is `'(2 (4)) 5 (2)'`.

   (d) **Bonus subtask (not necessary for sign-off):** Define a further `example5` with a cycle.

> **Tip.** *For testing, you may want to use the method* `toDot` *provided in the class.*