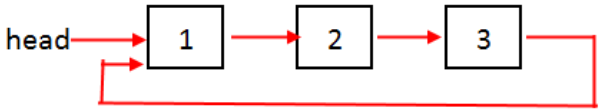
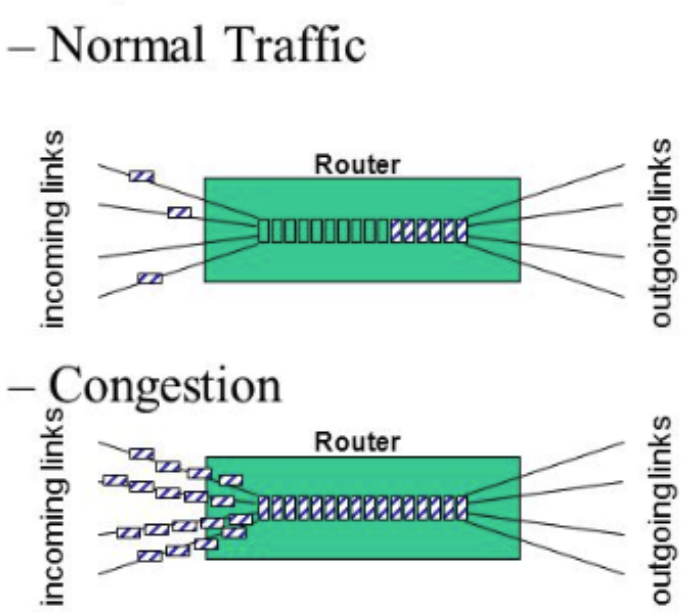
Computational Problem Solving CSCI-603
Ring Buffer Madness Lab 6

Singly Linked List



In this lab, we will develop and use a ring buffer as an underlying data structure for an implementation of a stack and queue. Sometimes when developing software solutions, you may be faced with constraints. One such constraint, especially when working with embedded systems, is a limitation of memory.

Consider a router which receives data on one end and then sends the data out the other based on a routing table that determines the next hop that a data packet needs to take in order to reach its ultimate destination. Some routers are very costly because they have a higher than average amount of memory to reduce packet loss if its received bandwidth is very high. However, that is not practical for all routers since not all consumers are willing to pay extra money for the additional memory. Therefore, based on a limited amount of memory, when the incoming bandwidth is too high, a router will drop packets. In that scenario, the dropped packets would most likely be the most recent packets that arrive.



Looking at the above image we can see that when normal volumes of traffic are present the internal queue of the router stays relatively empty. Conversely, the congested traffic flow has large volumes of incoming packets which overflows the internal queue. When there is no place for the packets to go, they are dropped until there is room in the queue for another packet.

Another use case with a different priority of data flow is flight information for an unmanned aerial vehicle. In this scenario, the most recent information is more important than information that is no longer relevant. Clearly the pilot of a UAV would not want to be making decisions based upon flight information that is not current. So it makes sense that the oldest information would need to be dropped in favor of more current information.

Problem Solving

1. A singly linked list can grow without bound. If we have a memory constraint, that won't work. What modifications can we make to our ring buffer to meet this constraint?

2. Consider all of the useful attributes (class member variables) that a ring buffer would need in order to work correctly. Write a class declaration, declare class member variables as necessary in __slots__, and a constructor for your implementation.

3. Now, write a list of all of the useful methods that will be needed for a a ring buffer.

4. Using valid Python code, write two functions in subsections *a and b* to add a new value to your ring buffer. Be mindful of the different use cases that are present when doing so with each different type of implementation.
   (a) Name this function insert_keep_new in which any new values added to the ring buffer will **overwrite** the oldest data in the ring buffer.

   (b) Name this function insert_keep_old in which any new values added to the ring buffer will **keep** the oldest data in the ring buffer and drop the new.

5. Since the structure is circular, how would you write the __str__ function to show the contents of your ring buffer?

6. You may recall in our lecture code that we used a test function for our classes to make sure that they worked on their own before we imported them into another module. Write 3 test cases that would go in your ring buffer test function. Remember that the goal of a test function is to catch edge cases that might potential break your code.

Implementation

For the implementation portion of this lab, you will write three classes. The first will be called `RingBuffer` in a file named ring_buffer.py, the second will be called `Stack` in a file named stack.py, and the third will be called `Queue` in a file named queue.py. You will also write a test file, router_test.py that will use ListStack and ListQueue classes provided for you on MyCourses. Each class should have its own test function which is called when the file is run as the primary file passed to the Python 3 binary.

Your RingBuffer class should be a linked structure and implement the following functions:

- `__init__` - Constructor
- `__str__` - String representation of you RingBuffer
- `insert_keep_new` - Add item at next location, keeping new data
- `insert_keep_old` - Add item at next location, keeping old data
- `find(value)` - Find item if in the list, return a cursor
- `capacity` - Max size of RingBuffer
- `size` - Current size of RingBuffer
- `replace(cursor, val)` - Replace this location - a use case might be using this after calling find
- `remove_oldest` - Remove oldest item in RingBuffer
- `remove_newest` - Remove newest item in RingBuffer

Your `Stack` and `Queue` classes will use your `RingBuffer` class as their underlying data structure. This means that you will create an instance of your RingBuffer class and make the needed calls on that instance to implement stack and queue functionality. You will need to implement all of the methods for a stack and queue that were covered in the lecture notes in each of their respective classes. An example of this style of design can be seen in the `ListStack` and `ListQueue` classes.

Remember a stack is Last In First Out which means that it should always preserve the most recent data. Therefore, if you hit a memory constraint, you must keep the most recent data.

Conversely, a queue is First In First Out which means that it should preserve the oldest data. Therefore, if you hit a memory constraint, you must keep the oldest data.

As mentioned above, with all data structures, comprehensive testing is also important. A significant portion of your grade will be given for a good test suite. Each class must provide it's own testing and each test function should be commented thoroughly to explain what it is you are testing. You must also develop a separate `router_test.py` that runs tests

using both your implementations of a stack and queue and the array list implementations provided to you in `ListStack.py` and `ListQueue.py`. The provided files should serve as a comparison for your own classes to make sure they are working as expected. You are encouraged to examine the code and test cases to answer any questions about how your own classes should work and what your structures should contain at any given point. Keep in mind that the router test is meant to be a stress test so that means you should try to *break* your code. The more creative you are in the scenarios that you test your code against, the more likely it is that you will catch hidden bugs such as a misplaced pointer, so test thoroughly!

Submission

Transfer your program to the CS machines. Submit your program before the deadline using try:

```
try grd-603 lab6-1 ring_buffer.py stack.py queue.py router_test.py
```

Grading

- Problem solving: 20%
- `File: RingBuffer`: 20%
- `File: Stack`: 20%
- `File: Queue`: 20%
- `File: Router testing`: 20%
  For each file:
  Documentation: 5%
  Style: 5%
  Design: 10%