



Design Decisions:

Url Class: I have default constructor, and a constructor that takes in a name and value and sets name and value respectively and next and prev pointers to nullptr. I have a destructor that deletes the next pointer and sets the address to nullptr.

LinkedList Class: I have a constructor that sets head and tail to nullptr. I have a destructor that deletes the head pointer and sets its address to nullptr.

Deque Class: I have a constructor that creates deque of size m, that inherits the doubly linked list class. I have an empty destructor.

* I did not override any operators and I did not have to pass each parameter by reference or use of the keyword “const”.

Test Cases:

We were told that input would be standard and valid. Base case for creating the deque was tested. I tested pushing front and back urls in multiple different orders including when the list was full. I also tested pushing front and back, popping front and back, getting the front url, and clearing the whole deque in multiple different orders. I then verified these inputs by periodically testing the size, printing the deque, checking if it is empty, and searching for specific urls. I tested locally and on eceubuntu.

Time Complexity:

push_front, push_back, pop_front, and pop_back, front, back, Size, and empty are all $O(1)$ constant time. push_front, push_back, pop_front, and pop_back only add to or delete the front or back of the deque, making it constant time. front and back checks if the deque is empty, if it is not, they return the front and back url making it constant time. Size and empty check if the current size of the deque is zero and then returns an appropriate statement, making it constant time.

Find, print, and clear functions are all $O(n)$ time complexity. Find, print, and clear iterate through the deque at most linearly once, making it $O(n)$.