CS 261 Data Structures

Assignment 3 -- Linked List Variations

Problem 1: Linked List Deque and Bag implementation

First, complete the linked list implementation of the deque (as in Worksheet 19) and bag ADTs (Worksheet 22). To do this, implement all functions with the `// FIXME...` comments in `linkedList.c`.

Grading (36 pts):

- `init` -- 2 pts
- `addLinkBefore` -- 4 pts
- `removeLink` -- 4 pts
- `linkedListAddFront` -- 2 pts
- `linkedListAddBack` -- 2 pts
- `linkedListFront` -- 2 pts
- `linkedListBack` -- 2 pts
- `linkedListRemoveFront` -- 2 pts
- `linkedListRemoveBack` -- 2 pts
- `linkedListIsEmpty` -- 2 pts
- `linkedListPrint` -- 4 pts
- `linkedListContains` -- 4 pts
- `linkedListRemove` -- 4 pts

Problem 2: Linked List vs Dynamic Array performance comparison

The files `linkedListMain.c` and `dynamicArrayMain.c` each contain code which does the following:

1. Takes an integer argument (say n) from the command line and adds that many elements to the bag.

2. Prints the memory in KB used by the bag.
3. Calls the contains function for each element in the bag.
4. Prints the time taken by these contains calls in milliseconds.

Your job is to compare the performance of the linked list implementation vs the dynamic array implementation over various numbers of elements. Create a plot comparing performance over element counts from $n=2^{10}$ to $n=2^{18}$, doubling n for each data point, for each of the following metrics:

1. Memory usage -- linked list vs dynamic array for n in $[2^{10}, 2^{18}]$.
2. Running time -- linked list vs dynamic array for n in $[2^{10}, 2^{18}]$.

Important: Please run all memory usage and timing tests on flip for consistency. The memory calculation code runs on flip only. Please follow the instructions in `linkedListMain.c` and `dynamicArrayMain.c` to comment out the memory code if you develop your programs in Visual Studio.

You must also implement the dynamic array. Use the implementation from Assignment 2 or the previous week's worksheets. Note that in `dynamicArrayMain.c` the dynamic array must have a capacity of n^{10} to start with.

After creating the two plots, answer the following questions:

1. Which of the implementations uses more memory? Explain why.
2. Which of the implementations is the fastest? Explain why.
3. Would you expect anything to change if the loop performed remove() instead of contains()? If so, why?

Grading (24 pts):

- Timing plot -- 10 pts
- Memory plot -- 10 pts
- Answers to the 3 questions -- 4 pts

Problem 3: Circularly Linked List Deque implementation

For this problem, you will implement the Deque ADT with a Circularly-Doubly-Linked List with a Sentinel. As you know, the sentinel is a special link, does not contain a meaningful value, and should not be removed. Using a sentinel makes some linked list operations easier and cleaner in implementation. This list is circular, meaning the end points back to the beginning, thus one sentinel suffices. Implement all functions with the `// FIXME...` comments in `circularList.c`.

Grading (50pts):

- `init` -- 4 pts

- `createLink` -- 4 pts
- `addLinkAfter` -- 4 pts
- `removeLink` -- 4 pts
- `circularListAddBack` -- 3 pts
- `circularListAddFront` -- 3 pts
- `circularListFront` -- 3 pts
- `circularListBack` -- 3 pts
- `circularListRemoveFront` -- 3 pts
- `circularListRemoveBack` -- 3 pts
- `circularListDestroy` -- 4 pts
- `circularListIsEmpty` -- 2 pts
- `circularListPrint` -- 4 pts
- `circularListReverse` -- 6 pts

Submission

As usual do not make any modifications to the header files or include any additional headers, and make sure everything compiles and runs on flip. Submit the following files:

- `linkedList.c` -- your linked list deque and bag implementation.
- `circularList.c` -- your circularly linked list deque implementation.
- `problem2.pdf` -- a pdf containing your plots and answers to the questions in Problem 2.