Assignment 4: Priority Queues
Due at 11:59 pm on Sunday, 3/3/2024
Demo due by 11:59 pm on Friday 3/15/2024
(Note: THIS IS A ONE-WEEK ASSIGNMENT!!!)

In this assignment, you will implement a binary heap-based priority queue (PQ). The requirements for the assignment are described below.

Part 0. Download the skeleton code and unzip

For this assignment, you are provided with some starter code that defines the structures you'll be working with and prototypes the functions you'll be writing and also provides some data structures upon which to build a PQ implementation. You may download the skeleton code for this assignment using the wget command:
wget https://classes.engr.oregonstate.edu/eecs/winter2024/cs261-020/assignments/assignment4.zip

To unzip the file, use the following command:
`unzip assignment4.zip`

Part 1. Implement a heap-based priority queue

Your task for this assignment is to implement a priority queue (PQ). A PQ is a structure that orders data elements based on assigned priority values. Specifically, elements can be inserted into a PQ in any order, but when removing an element from a PQ, the highest-priority element is always removed first.

You must build your PQ using a binary heap. A binary heap is a data structure that takes the form of a binary tree. There are two different kinds of binary heaps:

- A minimizing binary heap is organized so that the element with the *lowest* key is always at the root of the tree.
- A maximizing binary heap is organized so that the element with the *highest* key is always at the root of the tree. In this assignment, you will specifically base your PQ implementation on a minimizing binary heap.

The interface for the PQ you'll implement (i.e. the structures and the prototypes of functions a user of the PQ interacts with) is already defined for you in the file `pq.h`. Your job is to implement definitions for the functions that comprise this interface in `pq.c`.

Note that you may not modify the interface definition with which you are provided. Specifically, do not modify any of the already-defined PQ function prototypes. We will use a set of tests to verify your implementation, and if you change the PQ interface, it will break these tests, thereby (negatively) affecting your grade. Beyond these things, though, feel free to add any additional functions or structures your PQ implementation needs. In particular, you'll have to specify a definition of the main PQ structure, `struct pq`, in `pq.c`.

The PQ functions you'll need to implement are outlined briefly below. All of these functions use the type `struct pq`, which represents the PQ itself and which you'll have to define in `pq.c`. For more details, including information on function parameters and expected return values, see the documentation provided in `pq.c`. Here are the functions you'll have to implement:

- `pq_create()` – This function should allocate, initialize, and return a pointer to a new PQ structure.

- `pq_free()` – This function should free all the memory held within a PQ structure created by `pq_create()` without any memory leaks. Note that this function only needs to free the memory held by the PQ itself. It does not need to free the individual elements stored in the PQ. This is the responsibility of the calling function.

- `pq_isempty()` – This function should return 1 if the PQ is empty and 0 otherwise.

- `pq_insert()` – This function should insert a new value with specified priority into the PQ. **This operation must have O(log n) runtime complexity.**

- `pq_first()` – This function should return the first value (i.e. the highest-priority value) from a PQ *without* removing it. **This operation must have O(1) runtime complexity.**

- `pq_first_priority()` – This function should return the *priority value* associated with the first element in a PQ *without* removing that element. **This operation must have O(1) runtime complexity.**

- `pq_remove_first()` – This function should remove the first value (i.e. the highest-priority value) from a PQ and return that value. **This operation must have O(log n) runtime complexity.**

Your priority queue must be implemented using a minimizing binary heap as the underlying data structure. This means that within the priority queue you implement, *lower* priority values should correspond to elements with *higher* priority. In other words, the first element in the priority queue should be the one with the *lowest* priority value among all elements in the collection. For example, your priority queue should return an element with priority value 0 before it returns one with priority value 10.

You are provided with a dynamic array implementation in `dynarray.c` and `dynarray.h` that you can use to implement your heap, if you'd like. In addition to this dynamic array implementation, you may implement any additional helper functions you need to make your priority queue work.

Testing your work

In addition to the skeleton code provided here, you are also provided with some application code in `test_pq.c` to help verify that your PQ implementation, is behaving the way you want it to. In particular, the testing code calls the functions from `pq.c`, passing them appropriate arguments, and then prints the results. You can use the provided `Makefile` to compile all of the code in the project together, and then you can run the testing code as follows:

```
make
./test_pq
```

Example output of the testing program using a correct PQ implementation is provided in the `example_output/` directory.

In order to verify that your memory freeing functions work correctly, it will be helpful to run the testing application through `valgrind`.

Submission

In order to submit your homework assignment, you must create a **zip file** that contains `assignment4/` folder with your implementation. This zip file will be submitted to TEACH . In order to create the zip file, go to the directory where you can access the `assignment4/`, and use the following command:

```
zip assignment4.zip assignment4 -r
```

Remember to sign up with a TA to demo your assignment. The deadline of demoing this assignment without penalties is 3/15/2024.