Min-Max-Median Priority Queue Implementation and Performance Analysis

Hyoungshick Kim

October 15, 2024

1 Objective

Your task is to implement a **Min-Max-Median Priority Queue** in C using efficient data structures such as **heaps** to optimize performance. You will then perform an experimental performance analysis of your implementation to demonstrate its efficiency.

2 Required Functions

Implement the following functions for your priority queue:

- `void insert(int element)`: Inserts an integer element into the priority queue.
- `int delete_min()`: Deletes and returns the minimum element.
- `int delete_max()`: Deletes and returns the maximum element.
- `int delete_median()`: Deletes and returns the median element.
- `int find_min()`: Fetches but does not remove the minimum element. The result should be printed to the standard output.
- `int find_max()`: Fetches but does not remove the maximum element. The result should be printed to the standard output.
- `int find_median()`: Fetches but does not remove the median element. The result should be printed to the standard output.

When the number of elements is even, the median is defined as the smaller of the two middle elements.

3 Assumptions and Requirements

1. If there are no elements, the `find` functions should print `NULL` to the standard output.
2. If there are no elements, performing a `delete` operation should result in no action.
3. All element keys in the data structure are guaranteed to be unique.
4. You are required to use efficient data structures (e.g., min-heap, max-heap) to achieve optimal performance.
5. Do not use simple linear searches or sorting algorithms that result in **O(n)** time complexity per operation.

4 Input and Output Format

The input consists of a series of operations. The first integer denotes the number of operations to perform (ranging from 1 up to 500,000). Each subsequent line describes an operation, starting with a character indicating the operation type (`I` for insert, `D` for delete, `F` for find) followed by a character specifying the target (`M` for min, `X` for max, `E` for median), and if inserting, the integer to insert.

Your program must read the input from the file `pq.in` and write the output to `pq.out`.

Example Input (pq.in):

```
8
I 5
I 10
I 20
I 15
D M
F M
F X
F E
```

Expected Output (pq.out):

```
10
20
15
```

5 Grading Criteria

Your work will be assessed based on the following criteria:

1. **Correctness** (30%): The implementation produces correct results for all operations.
2. **Experimental Analysis** (50%):
   - Quality and thoroughness of your performance experiments.
   - Clarity and depth of your analysis and explanations.
3. **Documentation and Code Quality** (20%):
   - Clear and detailed documentation explaining your code and the performance of algorithms.

6 Submission Guidelines

- Your code must be written in ANSI C. The GNU compiler (`gcc`) on Ubuntu Linux will be used for compilation.
- Thoroughly test your code with various input scenarios to ensure correctness.
- Do not use pre-defined algorithms (e.g., `qsort()`) or data structures from libraries, excluding arrays and strings. You must implement the priority queue and any data structures from scratch.
- Basic string functions such as `strcmp`, `strcpy`, and `strlen`, as well as standard I/O functions, are allowed.
- Submit the following files to iCampus:
  1. Your C source code files.
  2. A detailed report (in Korean or English) that includes:
     - An explanation of your code and the data structures used.
     - The design of your algorithms.
     - An analysis discussing the results and relating them to theoretical concepts.
- **Originality is crucial.** Plagiarism checks will be conducted, and any form of code copying will result in penalties.