Data Structure 14022, A. Jamshidpey

Programming 1

DUE: Khordad 23, 11 PM

1. [20 marks] In this programming question, you are asked to implement a priority queue in 3 ways, using C++17.

   An item in the priority queue will have a non-negative integer priority and a positive integer value. The value is a timestamp recording the order in which items have been inserted; the first item inserted will have value 1, the second item value 2, and so on. The priority will be provided by the user while the timestamp will be computed by your program. The timestamp is initially 1 and is then incremented as each new item is inserted.

   Each implementation of the priority queue must support the insert and deleteMax operations, as well as, findMax which returns the item with the highest priority (but does not remove it from the data structure) and lookup which accesses an internal component of the data structure.

   Implementation 1: A singly linked list that is ordered by priority where the largest priority is in the front of the list. If two items have the same priority, then the tie is broken by ordering the item with the smallest value (timestamp) first.

   Implementation 2: A max-heap implemented with a dynamic array (as described in the lectures). When the first item is inserted, an array of length 1 is dynamically allocated to accommodate the first item. You must then use the doubling strategy to reallocate the dynamic array when more space is required. Similarly, if after removing an item, only 25% or less of the dynamic array contains items, the size of the dynamic array should be reallocated to be twice the number of items in the dynamic array. Note: for this implementation, you may not use a vector. You must implement the reallocation and doubling strategy yourself - use `realloc`.

   Implementation 3: A vector of queues. Each index i of the vector points to a queue of items where each item in the queue has priority i; if no items of priority i exist, this pointer may be left as NULL (i.e. you do not need to create an empty queue). At all times, with one exception, the vector must be appropriately sized to be 1 + largest priority stored in the priority queue. If there are no items in the priority queue, the vector may be of size 0 or 1. You are free to choose the implementation of the queue at each vector index - a circular vector, linked list with back pointer, or std::queue is okay. The insert/delete (enqueue/dequeue, pushback/popfront) queue functions must run in $O(1)$ time.

   You may not use smart pointers, auto or any built in data structures (unless specified) that may trivialize the implementations. You may use std::pair in any of the three implementations and std::vector may only be used in implementation 3. You may not use std:swap or anything from the algorithms library. To compute percentages, such as 25%, you may only use integer division and mod; i.e. you may not include a math library, use a built in floor function, etc. You must also manage the dynamically allocated memory yourself (where applicable) and must free all memory before your program terminates.

In all implementations, findMax must have runtime $O(1)$.

At the top of your program, in comments, include a section for the algorithm analysis of each implementation. You may assume that n is the number of elements currently in the priority queue. For the algorithm analysis:

- define any additional variables that are required for the analysis
- state and briefly justify the running time of insert and deleteMax for each implementation
- state and briefly justify the amount of space required for each implementation

Implement your program in C++ and provide a main function that accepts the following commands from stdin (you may assume that all inputs are valid). You may assume the priority queues described above are numbered 1, 2 and 3, respectively.

- `i num priority` - inserts an item (`priority, timestamp`) into priority queue num.
- `d num` - removes the item with the highest priority from priority queue `num` and prints the priority and timestamp to stdout - the two integers are separated by a space and a newline follows the second integer. If the priority queue is empty, does nothing.
- `f num` - prints (but does not remove) the item with the highest priority from priority queue `num`. Prints the priority and timestamp to stdout - the two integers are separated by a space and a newline follows the second integer. If the priority queue is empty, does nothing.
- `l num i` - performs a lookup of the `i`-th item of the data structure used to implement priority queue `num` and prints the corresponding `timestamp` if `num` is 1 or 2 and the number of items in the queue if `num` is 3. You may assume that i will be valid for the data structure; i.e. do not need to check if it is out-of-bounds. Note: The 0-th, item is the first item in the linked list or the item at index 0 of an array or vector.
- `r` - initializes or resets all priority queues to be empty.
- `x` - terminates the program. Place your entire program in the file `pqueue.cpp`