CSC3002 2023Fall Assignment 5

Due 23:59, Dec 3, 2023

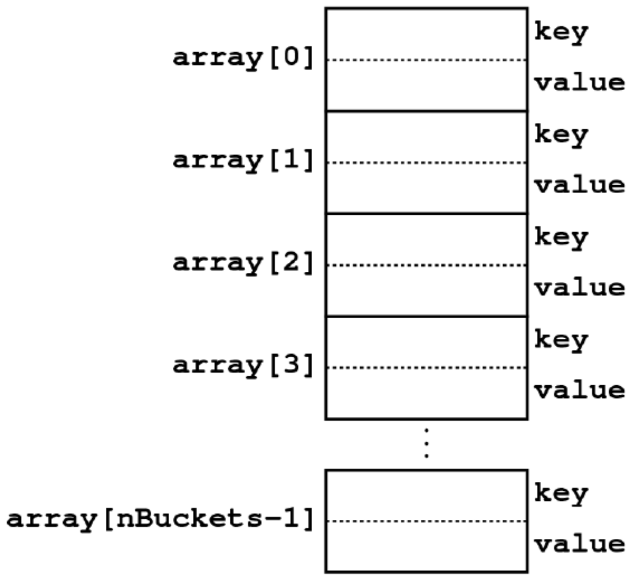
About this assignment

- You don't need to modify or submit any other files other than `stringmap.cpp` and `PriorityQueue.cpp`.
- StanfordCPPLib is not needed in this assignment.

Problem 1 (Points: 50)

Part 1: (Exercise 15.9)

Problem Description Although the bucket-chaining approach described in the text works well in practice, other strategies exist for resolving collisions in hash tables. In the early days of computing—when memories were small enough that the cost of introducing extra pointers was taken seriously—hash tables often used a more memory-efficient strategy called ***open addressing***, in which the key-value pairs are stored directly in the array, like this:



For example, if a key hashes to bucket #2, the open-addressing strategy tries to put that key and its value directly into the entry at `array[2]`. The problem with this approach is that `array[3]` may already be assigned to another key that hashes to the same bucket. The simplest approach to dealing with collisions of this sort is to store each new key in the first free cell at or after its expected hash position. Thus, if a key hashes to bucket #2, the `put` and `get` functions first try to find or insert that key in `array[2]`. If that entry is filled with a different key, however, these functions move on to try `array[3]`, continuing the process until they find an empty entry or an entry with a matching key. As in the ring-buffer implementation of queues in Chapter 14, if the index advances past the end of the array, it should wrap around back to the beginning. This strategy for resolving collisions is called ***linear probing***.

Reimplement the `StringMap` class so that it uses open addressing with linear probing. For this exercise, your implementation should simply signal an error if the client tries to add a key to a hash table that is already full.

Requirements Please complete the **TODO** parts in the `insertKey` and `findKey` functions of the file *stringmap.cpp*. Note that your function should prompt the message "**ERROR: Insufficient space in hash table**" if the space is not enough to insert.

In & Out Statement The input of this problem are command lines. You may check the file *out/p1_1.txt* for standard output file.

Part 2: (Exercise 15.10)

Problem Description Extend your solution to exercise 9 so that it expands the array dynamically whenever the load factor exceeds the constant `REHASH_THRESHOLD`, as defined in exercise 5. As in that exercise, you will need to rebuild the entire table because the bucket numbers for the keys change when you assign a new value to nBuckets.

Requirements Please complete the **TODO** parts in the `rehash` functions of the file *stringmap.cpp*.

In & Out Statement The input of this problem are command lines. You may check the file *out/p1_2.txt* for standard output file.

Problem 2 (Exercise 16.13, Points: 25)

Problem Description Use the algorithm from section 16.5 to implement the `PriorityQueue` class so that it uses a heap as its underlying representation. To eliminate some of the complexity, feel free to use a vector instead of a dynamic array.

The `PriorityQueue` class includes these functions:

- The `enqueue` and `dequeue` functions to add and delete elements to/from the `PriorityQueue`.
- The `peek` and `peekPriority` functions to peek the value and priority, respectively.
- The `toString` method to convert the `PriorityQueue` into a printable `String`.

Requirements Please complete functions in the file *PriorityQueue.cpp*. To implements some of these functions, you may need to complete the **TODO** parts in the functions `enqueueHeap`, `dequeueHeap`, `takesPriority`, `swapHeapEntries` or `operator«`, but not required.

In & Out Statement Your program receives two elements in one line, split by the space: a `string` value to be put into the `PriorityQueue` and the corresponding priority of this element. The priority can be any value that fit the type `double`.You are recommended to test this program using an input file.

Functionalities of your implementation will be tested in the `main()` function, and you may see the file *out/p2.txt* for standard output.