CSE250 Fall 2016
Assignment A3 – Key/Value Sequences

Due: 10/23/2016, 11:59PM

Last updated: *2016-10-10 09:01*

Objectives

- Implement more advanced data structure.
- Practice using basic C++ data structures.
- Practice working with C++ classes.
- Practice working with partially provided code.

Introduction

There are times when certain simple functionality is best achieved by a data structure that builds on top of other, standard, data structures. Consider this example scenario. In our code, we are dealing with key/value pairs of integers. These are pairs of integers where the first integer is called key, and the second integer is called value. We want to store all key/value pairs such that for each key we know which values and in what order we have inserted. For example, for the following key/value pairs $(0,2),(-1,3),(3,7),(0,5),(3,2),(3,5),(-1,3),(-1,0)$ we would like to be able to retrieve sequence $(3,3,0)$ for key -1, sequence $(2,5)$ for key 0 and $(7,2,5)$ for key 3.

Your task is to extend partially implemented, and provided to you, `key_value_sequences` class, to provide exactly that functionality. So for example, this simple code (that roughly corresponds to the example above):

```
key_value_sequences A;

A.insert(0, 2);
A.insert(-1, 3);
A.insert(3, 2);
A.insert(0, 5);
A.insert(3, 7);
A.insert(3, 5);
A.insert(-1, 3);
A.insert(-1, 0);

auto p = A.data(3);
for (int i = 0; i < A.size(3); ++i) std::cout << p[i] << " ";
```

when compiled with properly implemented `key_value_sequences` should produce: `2 7 5`.

As usually, your implementation has to be efficient and you have to work under certain constraints. Specifically, you cannot use any external or standard C++ headers except of `<algorithm>`, `<list>` and `<vector>`.

Hints

The simplest and reasonably efficient approach to this assignment is to use a combination of double-linked list (`std::list<>`) and vector (`std::vector<>`). Do not be afraid to make a list of vectors or vector of lists. Do not hesitate to leverage `std::list<>::iterator`, it is a type as any other C++ type. `std::vector<>` provides method `data()` that perfectly meets requirements you will need to implement method `data()` in `key_value_sequences`. This is one of many ways in which this problem can be approached.

Instructions

1. Create directory `A3` where you will place your code.
2. Download `A3-handout.tar` from:
   http://www.jzola.org/courses/2016/Fall/CSE250/A/A3-handout.tar.
3. Untar handout, and move `a3.hpp` and `a3.cpp` to your `A3` directory.
4. Implement missing elements in `a3.hpp` taking into account the following:
   (a) `a3.hpp` is the only file you are allowed to modify. `a3.cpp` is a simple test program demonstrating functionality that your implementation of `key_value_sequences` must support. You must use it to perform the most basic initial testing of your code. If your code fails to work with `a3.cpp` you will not be graded. You are not allowed to make any changes to `a3.cpp`.
   (b) Analyze carefully `a3.hpp` and `a3.cpp` to better understand `key_value_sequences`. This class should implement functionality of multiple sequences, where each sequence stores integers and is "identified" by an integer key.
   (c) Missing parts of the code that you are expected to implement are marked with `IMPLEMENT ME`. In short, you have to implement three critical methods: `insert()`, `size()` and `data()`.
   (d) Method `insert()` is the workhorse of the class. Its purpose is to take a key/value pair, i.e. two integers, and push back the second integer (which is a value) into the sequence identified by the first integer (which is a key). This is the only modifying method.
   (e) Method `data()` is the main method to access the stored values. Its purpose is to take one integer (which is a key), and return a pointer to a sequence of all integers inserted with that key. If no element had been inserted with given key it should return `nullptr`. If a sequence exists for the key, the method should return pointer to the continuous block with that sequence. Elements in the sequence must be in the same order in which they had been inserted (refer to examples above).
   (f) Method `size()` is another access method. Its purpose is to take one integer (which is a key), and return the total number of elements inserted with that key. If no element had been inserted with given key the method should return `-1`.
   (g) **You can add, remove or modify code in `key_value_sequences` without limits as long**

as it correctly compiles and executes with the provided `a3.cpp` and the only included headers are `<algorithm>, <list>, <vector>`. To compile `a3.cpp` with your `a3.hpp` you should use basic compiler setting, e.g.: `g++ -std=c++11 -O2 a3.cpp -o a3`. If your code satisfies the most basic requirements for grading, you should see `pass` printed when executing `a3`.

(h) You are encouraged to use C++11 containers `std::vector` and `std::list` with all their goods to avoid raw pointers.

(i) Your implementation must be efficient. You can expect millions of key/value pairs inserted and your class should be able to handle them in seconds.

(j) You should make no assumptions about the distribution of inserted elements, i.e. keys or values, and the order in which elements will be inserted.

(k) You should be careful with how you utilize memory. If you are too excessive in allocating memory, your code will exhaust available resources and your code will be terminated and not graded.

Submission

1. Remove your binary code and other unrelated files (e.g. your test files).
2. Create a tarball with your `A3` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A3.tar` for grading.
4. You have unlimited number of submissions, however, any submission after the deadline will have 50% points deducted.

Grading

- 10pt: The test `a3.cpp` compiles with your `a3.hpp`, runs and has no memory leaks.
- 90pt: If you pass the initial test, there will be nine benchmark tests. You will get 10pt for each correctly completed test (i.e. your implementation performs as expected, no memory leaks, and runs in reasonable time limits).
- If your code is **extremely** inefficient, autograder will terminate your code and you will receive 0pt.

Remarks

- Make sure that all file and directory names are exactly as instructed. Otherwise the grading system will miss your submission and you will get 0pt.
- Make sure that you start working on the assignment early! If you wait to the very last moment, you may expect delays from autograder (overloaded by other students who like you waited with their submission).
- Focus on finding the best performance algorithm and not low-level optimizations – the autograder architecture is unpredictable :-)