Karl Goeltner
Joël Porquet
ECS 36C
05/22/2020

Programming #4 – Analysis Report

P1 – LLRB-based Multimap: My program multimap.h implements a multimap using a left-leaning-red-black tree data structure. Its public API includes Size, Get, Contains, Max, Min, Insert, Remove, and Print. The helper methods of Get (iterative), Min, Insert, Remove, Print (recursive), IsRed, FlipColors, RotateRight, RotateLeft, FixUp, MoveRedRight, MoveRedLeft, DeleteMin (self-balancing) aid in their implementation.

(1) I first changed the definition of a node by replacing a value with a deque of values and also rearranged the order, so the color & key come first in the initialization list. I decided to use a deque because the Get() & Remove() methods access the first element while Insert() pushes towards the end, thus, illustrating a queue form factor. I chose a deque over a queue because I iteratively Print() the multimap elements with [] access, which would also be more efficient than a list since nodes are contiguous in memory. (2) In Get(), I altered the line where the key would return the front() value of its list. (3) In Insert(), I first changed it so a new key at a NIL position would insert a new node with a key & deque of values with one value pushed back. I also changed it so when a key already exists, a new value will be pushed back. (4) In Remove(), I revised two locations where a key node is not at bottom & at bottom. For both cases I distinguished between pop_front() of the first element vs removing the entire node. When not at bottom, I made sure I replaced the node with the min node in the right subtree by swapping key-values. (5) In Print(), I printed out all key-value pairs of the internal deque.

In test_multimap.cc, I created several tests to increase the coverage of my implementation. Besides the given tests, OneKey & MultipleKeys which check an insertion of one key and multiple keys with contains & gets, MaxMin & MultipleMaxMin check the max & min of keys / duplicated keys with different values respectively. ExceptionCase checks that get should throw an exception when no key found. DuplicateKeys & DuplicateKeyDiffValues check duplicate insertion, removal, get of keys & keys with different values respectively; the latter also ensures proper print output. GetDuplicateFirst checks the first element obtained has the proper value with duplicated keys. PrintInsertRemoval & RepInsertRemoval insert & remove keys where the former ensures proper printing at the latter alternates between operations.

P2 – Completely Fair Scheduler: My program cfs_sched.cc receives a file of unordered task descriptions and feeds them into the CFS scheduler strategy. It uses the prior implementation of multimap.

The logic of cfs_sched.cc starts at the main method before flowing to respective methods which handle the various operations. The main method first performs error checking to ensure that a data file with tasks is passed in. It first calls checkFileStream() to make sure that the file is opened properly, and then storeData() to store the task data (id, start_time, duration) in a Task* vector. A Task class is made to hold its 3 primary variables along with runtime & vruntime and public accessors. Then, organizeTasks() is called to sort tasks with equal start times by id in alphabetical order. Finally, runCFS() is called which applies the CFS algorithm on tasks.

runCFS() is the primary function which calls the 7 steps of the CFS algorithm in partitioned public methods of the Scheduler class. The Scheduler holds a multimap of vruntime-Task* (key-value) pairs in its timeline. appendTimeline() first sees if any task start_time matches the tick value to be added to the multimap and sets its vruntime to the global value. moveNextTask() checks if the current task should move to the next task. getNextTask() gets the next task if current task is null and removes the next task from timeline. incrementTask() increments the current task's runtime & vruntime. printStatus() prints out the tick counter, total running tasks, and current task's id. purgeCompletion() deletes the current task if it is complete. incrementTick() increments the tick value for the next loop. Finally, done() is continuously checked to see if all tasks have been completed.