Anthony Natale

Summer 2020

Data Structures & Algorithms

# Approach

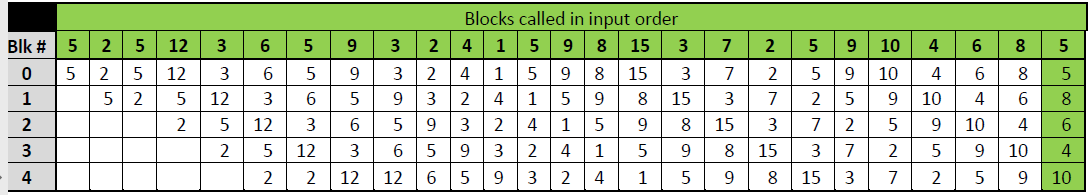
Note: Citations to resources that may have been used in this project will have been added both to functions and this document.

# At a High Level

We are creating a buffer pool of a fixed size that accomplishes buffer block replacement using the LRU heuristic.

The LRU heuristic describes how the buffer blocks are replaced when new data blocks from the disk are read. LRU, or “Least Recently Used” indicates that the buffer block that has been least recently used (“used” meaning written to or read from) by the program ought to be the one discarded or written back to the disk when a new block is read into the buffer pool from the disk.

This heuristic was the one on which we were evaluated in the Module 5 Homework 3 assignment. A manual assignment of blocks was accomplished using a table, which is pictured here for visualization purposes:



As seen in the 1st and 2nd columns from left (which have headers “5” and “2”) new items were added to the buffer pool blocks when blocks were called. The 3rd column from left (header “5”) deviates from this pattern and simply pulls the “5” data item to block 0 while pushing the other data items down in the blocks. Continuing to the right shows that items are “forced out” once they are assigned to block 4 and then not called by the program in the next step. Therefore, this pattern holds the most recently used items in RAM to speed up access times, and pushes out the least recently used items to efficiently use storage space. This is the LRU heuristic pattern.

# Process

1. Add all the existing code into a Visual Studio C++ console application project.
2. Create a new private GitHub repository with which to monitor changes.
3. Create LRUBufferPool.h (implementation of the Buffer Pool ADT) and LRUBufferBlock.h (implementation of the Buffer Block ADT).
4. Ensure inheritance is correct:
   1. Main.cpp inherits LRUBufferPool.h & constants.h
   2. LRUBufferPool.h inherits BufferPoolADT.h & LRUBufferBlock.h
   3. LRUBufferBlock.h inherits BuffBlockADT.h
5. Begin implementation.

# Implementation

1. LRUBufferBlock.h
   1. Implement all virtual functions from BufferBlockADT
      1. Constructor
         1. Inserts info and sets blockStart/blockEnd variables
      2. Destructor
      3. setID
      4. setBlockStart
         1. Sets the starting position of the block
      5. setBlockEnd
         1. Sets the ending position of the block
      6. setBlock
         1. Sets the actual char array associated with the block
      7. getData
         1. Returns the requested data from the block to LRUBufferPool.h
      8. getID
      9. getBlockSize
      10. getBlock
          1. Returns the entire char array associated with the block
      11. getBlockStart
      12. getBlockEnd
   2. Ensure block size is 4096 bytes in all cases except last block of file
   3. Add blockID integer instance variable (a private variable) and other variables as needed.
2. LRUBufferPool.h
   1. Pool should hold 5 blocks
   2. Pool should manage blocks with LRU strategy
      1. Need function to reorder blocks without pushing the last block off
      2. Need function to reorder blocks AND push the last block off
   3. Implement all functions from BufferPoolADT
      1. Constructor to read the file and create the 5 initial blocks
      2. Destructor
      3. getBytes
         1. Checks if the requested bytes are in the buffer
            1. If so, gets them and calls reorderBlocksWithoutPushingLast
            2. If not, reads them from the file and calls reorderBlocksWithPush
      4. printBufferBlockOrder
         1. Prints IDs from the first block to the LRU block
      5. getLRUBlockID
         1. Returns the ID of the LRU block (last block)
      6. reorderBlocksWithoutPushingLast
         1. Performs the LRU heuristic reorder when the requested bytes are in the buffer pool
      7. reorderBlocksWithPush
         1. Performs the LRU heuristic reorder when the requested bytes are not in the buffer pool
3. Main.cpp
   1. Make sure to open file in binary mode for read/seekg methods to work
   2. Add call to show use of getLRUBlockID

# Integrity Statements

1. I have not shared the source code in my program with anyone other than my instructor’s approved human sources.
2. I have not used source code obtained from another student, or any other unauthorized source, either modified or unmodified.
3. If any source code or documentation used in my program was obtained from another source, such as a text book or course notes, that has been clearly noted with a proper citation in the comments of my program.
4. I have not knowingly designed this program in such a way as to defeat or interfere with the normal operation of any machine it is graded on or to produce apparently correct results when in fact it does not.