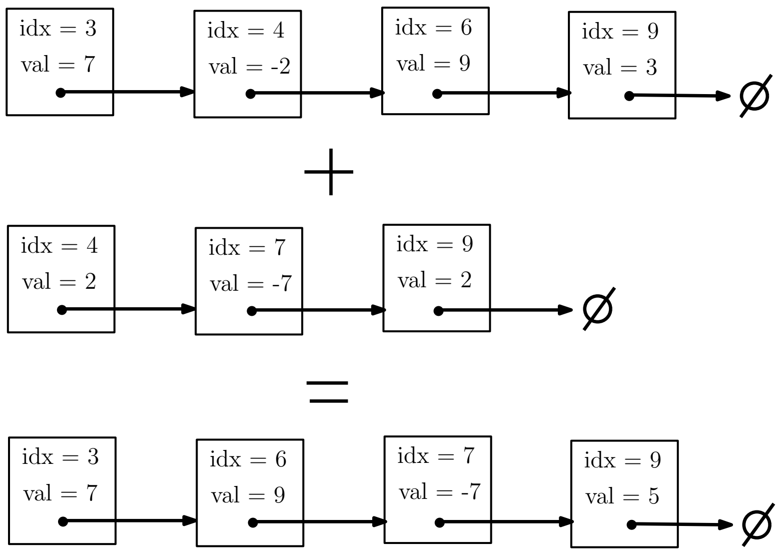
Data Structure Homework 1 (15+5)

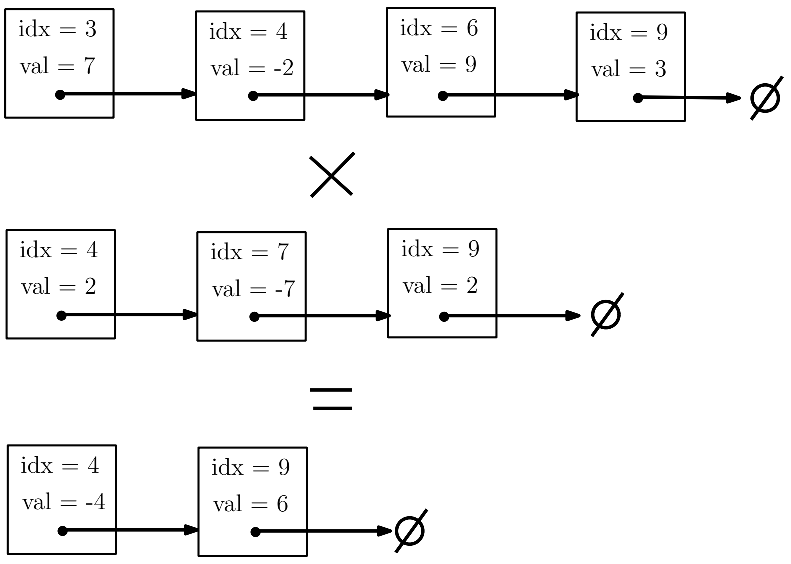
- Due Time: February 27 (Thursday), 23:59PM
- Students may work in pairs.
- Late submission will be heavily penalized, as stated in the syllabus.
- If you cannot finish all, submit solutions for some problems to get partial credits.

Problem 1: Sparse Vector Using Linked List (5 pts)

- Implement the linked list sparse vector class in LLSparseVec.java so that LLMainClass can be executed. The command line arguments are *VEC/MAT File1 A/S/M File2 …*
- Nodes in the linked list are nonzero elements of the vector, sorted according to their indices. The length of a vector is specified at construction.
- When an element is set to zero, the corresponding node should be removed.
- Implement the constructor, accessor methods, getElement, setElement, clearElement, getAllIndices, getAllValues.
- Implement addition, subtraction and element-wise multiplication methods in LLSparseVec. The method subtraction(otherV) means the current vector minus otherV.
- The algorithm has to be O(m), in which m is the maximum number of nonzero elements in a vector (length of the list). **Only algorithms with O(m) complexity will get credits.**
- All operations return a new sparse vector object, storing the result.
- If the two vectors' length do not match, return a null object.
- When LLMainClass is called using VEC argument and with at least one input file, the program should be able to run correctly and give the same output as ArrayMainClass, else a proper error message should be generated.

Examples:

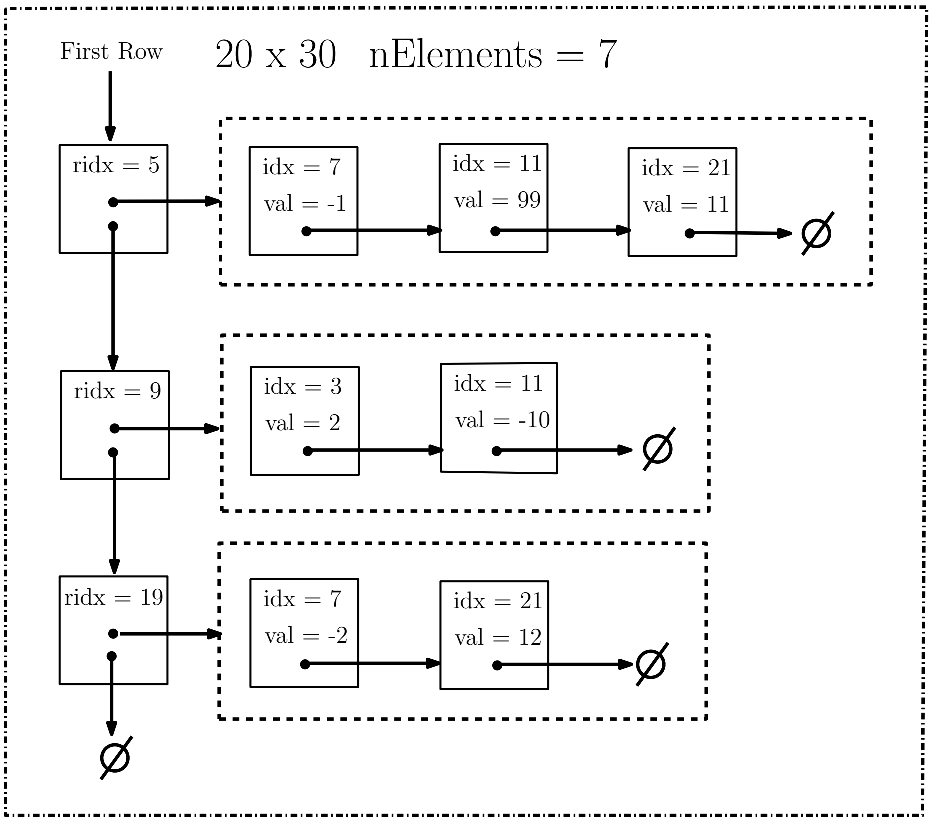




Problem 2: Sparse Matrix Using Linked List (5 pts)

- Implement the linked list sparse matrix class in LLSparseMat.java so that LLMainClass can be executed with MAT argument. The command line arguments are *VEC/MAT File1 A/S/M File2 ...*
- RowHead nodes correspond to nonzero rows. Each rowhead node stores a LLSparseVec, storing a nonzero row. It also has a pointer to the next rowhead.
- When a row becomes empty (no nonzero elements), the rowhead should be removed.
- Implement the constructor, accessor methods:
  - getElement, setElement, clearElement, numElements (returns number of non-zero elements).
  - getRowIndices returns an array of indices of rows with nonzero elements.
  - getOneRowColIndices returns an array of nonzero column indices of the row. Use LLSparseVec.getAllIndices().
  - getOneRowValues returns an array of nonzero values. Use LLSparseVec.getAllValues().
  - NOTE that these methods should all be linear to the number of nonzero rows or nonzero elements.

- When LLMainClass is called using MAT argument and with at least one input file, the program should be able to run correctly and give the same output as ArrayMainClass, else a proper error message should be generated.



Problem 3: Sparse Matrix Operation Linked List (5 points)

- Implement the addition, subtraction and multiplication methods in LLSparseM.
- The algorithm has to be O(m), in which m is the maximum number of nonzero elements in a matrix. Only algorithms with O(m) complexity will get credits.
- All operations return a new sparse matrix object, storing the result. The method subtraction(otherM) means the current vector minus otherM.
- If the two matrices' dimensions (nrows, ncols) do not match, return a null object and output a proper error message.
- When LLMainClass is called using MAT argument and with at least one input files, the program should be able to run correctly and give the same output as ArrayMainClass, else a proper error message should be generated.

Problem 4: Performance Evaluation (5 bonus points)

Write a report on matrix construction and operation time, comparing Array and LL implementation: 2 plots, on construction, and on operations. Each plot has two curves: Array implementation, and LL implementation. X-axis is the different sizes of these matrices.

To complete this report, you will have to write additional code to run the experiments, collecting timing information and producing result tables and graphs, together with relatively long answers. Do not wait until the last minute! Insert tables and graphs in the report as appropriate and be sure to give each one a title and label the axes for the graphs. For all experimental results, do provide a detailed interpretation, especially when the results do not match your expectations. For more stable results, average the runtime of multiple runs as shown in the example timing code below. Java optimizes repeated operations, so it runs faster at the end of the loop. Throw away several runs at the beginning of the loop to encounter this effect (JVM warmup).

```
  private static double getAverageRuntime(String[] args) {
    double totalTime = 0;
    for(int i=0; i<NUM_TESTS; i++) {
      long startTime = System.currentTimeMillis();
      LLMainClass.main(args);
      long endTime = System.currentTimeMillis();
      if(NUM_WARMUP <= i) { // Throw away first NUM_WARMUP runs to encounter
JVM warmup
        totalTime += (endTime - startTime);
      }
    }
    return totalTime / (NUM_TESTS-NUM_WARMUP); // Return average runtime.
  }
```

Testing:

In Data folder, there are example cases for testing during the development. At some stage, you should try out more advanced results to make sure the robustness of the complete system. Many test cases can be found in ManyTestCases.

About Submission (Please read carefully, otherwise you may lose points).

- Make sure you have read the programming guidelines posted on Blackboard for coding conventions, in order not to lose credit.
- Only submit source files. Do NOT include any executables. All files should be saved in a folder and then packed into a single .zip (NOT .rar or .tar.gz) file and be submitted via blackboard (NOT emailing).
- The folder name (before compression) as well as the final zip file name should be "LastNames-HW1".

- Only one submission per team, and **full names** of the members should be included in the *Comments* box of the submission page on Blackboard.
- Ensure your code can be compiled and executed in command line (not a java IDE). Otherwise, you will NOT get any credits.
- In the zip file, include a text file README.txt for the following information:
  - Problems you have finished
  - On which platform (mac, linux, window) the code is compiled and executed
  - "Resources that Helped Me", as discussed in the syllabus.
- You are NOT allowed to use any native implementation of lists (ArrayList, LinkedList, etc.). Consult the instructor if you want to use any native class that is not mentioned here.
- This is NOT something you can finish in three days. To understand the problem itself takes quite some time. You have to start as early as possible.
- *LLMainClass.java* and the interface files (SparseM and SparseVec) should not be changed when you submit. That is, you should not change the name/input/output of these major functions.