Exploring additional options to compute a Matrix Chain

CS404 Project, Spring Semester 2015
Version: DRAFT: March 16, 2015.
Due: **Friday April 10, 2015**

Summary and Motivation

The MatrixChainAlgorithm (MCA) is an algorithm to design an expression tree to multiply a sequence of matrices such that the minimal cost is obtained, the primary objective. Note that the cost is minimal (and unique), but that the expression tree itself might not be, and these alternate trees might have secondary desirable properties (e.g. increased locality of reference that minimizes I/O overhead or some other memory management). In fact, it could very well be that a sub-optimal solution has acceptable primary performance, and more desirable secondary properties and might be preferred. This project explores whether or not the MCA can be extended and such additional requests be accommodated.

Problem Statement

Extend and adjust the MCA algorithm to generate at least five (5) different execution trees reflecting the best options (in terms of $$ multiplication cost). Of course, these execution trees will have certain subtrees in common. Furthermore, you need to argue that your 'second-best' options are indeed 'second best'. (i.e. explain why you believe that your algorithmic extension gives the correct answers.) For each of these executions trees, you need to present:

1. The cost to multiply the chain when using this option.
2. The fully parenthesized expression,
3. The depth of this expression tree,
4. The width of the expression tree. (defined as the maximum of the number of nodes by level).

For this project, you must design an algorithm (with supporting abstract data structures in an algorithmic language), that solves the problem. When you make design decisions in this stage (either algorithmic or on data structures), you should document the pro's and con's for the alternatives in terms of time and space trade-offs and explain your decision. You should test your algorithm, and hand simulate it to test your ideas and convince yourself it is correct. At this time, you should analyze your algorithm for time (and space) complexity; it is acceptable to only analyze for *best*- and *worst*-case situations.

After this is done, you must implement your algorithm in any language of your choice, but you must explain why you used that language. You are further somewhat limited by the fact that you must be able to demonstrate your program on one of the computers in the SCE-labs, and that it is able to read in the file that we generate. Recommended languages (in terms of speed) C, and $C++$, but also acceptable are JAVA, Python, and R, but if the code is not transparent and not explained, the grader might not understand your clever programming and might mis-interpret it.

Of course, we expect you to report on resource consumption by your program. In other words, your report should contain something like "Case ## took ### milliseconds to run", while indicating language, platform, computer, available memory, and other such things.

Thus, you must design an algorithm (with supporting abstract data structures in an algorithmic language), that solves the problem. In particular, you need to give

1. Convincing arguments (proof? counter-example?) that you find the second and third best option.
2. An analysis of your options for best-case and worst-case time complexity as well as space complexity before you implement them.
3. Implement your option and tests for correctness.
4. For the cases that will be provided, measure the run times of your programs to experimentally verify your analytical findings.

Strong Advice

CS404 is a course on algorithm design, and I would expect that this would be the most time consuming task. Particularly, because you are only tasked with 'What' needs to be accomplished, and you yourself have to decide on the 'How'. This part is the most rewarding phase, but also the most frustrating phase of the project, since it is impossible to predict how long this takes for anyone, and it is impossible to realize how close you may be to a solution. Just do not panic, and go back to the basics: What do you currently have, where do you need to go, and how can you use what you learned in CS303 to your advantage. Structure your thinking, document what you could try and have tried already. Once you get the big idea, the programming is less challenging due to your strong programming and coding skills, but the interpretation of the results will take some time again. So set yourself realistic milestones, and build in extra time to recover from setbacks. The key is: Get started. Start thinking. And enjoy the aha-moment.

Getting Help

The primary point of contact for this project is the TA, and you should direct all questions to the TA, whose office and hours are announced in a separate cover through Blackboard, and whose email address was listed in case you need to make an appointment outside the office hours.

Generic Expectation for Projects in CS404

The description below is written for all projects that could be assigned in CS404,
and is not specific to this semester's project
Version: Spring Semester, 2015.

Goals and Expected Outcomes

Certain components of this project are used specifically to assess your attainment toward meeting the expected student learning outcomes for our degree programs as required by both our outside accreditation agencies and our internal assessment committees. Some of these learning outcomes are taught in this course, whereas others have been taught in prior courses, and you will be able to demonstrate your mastery of them. For instance, you have learned various programming skills in prior course work, and this project gives you an opportunity to demonstrate your skills, or even expand and learn new features. I expect that you can do so with self-study, or with the help of the TA. This course is taught independent of a particular computer language.

Generically, projects in CS404 are designed around the expected course outcomes. Keep in mind, that a particular project may not be suited to demonstrate all outcomes. Some outcomes will be tested during quizzes and exams.

- Demonstrate the ability to evaluate trade-offs in design choices
  Your report should indicate at least two algorithmic choices that you considered, before settling on the one you implemented.
- Demonstrate the ability to select the proper data structure to solve a problem
  Your report should indicate at least two data structure choices you considered for parts of the program, before settling on the one you implemented.
- Demonstrate the ability to use probability to analyze design options
- Demonstrate the ability to design an efficient algorithm
  What are the best- and worst-case time complexities for your algorithms?
- Demonstrate the ability to make ethical decisions
  Ethical decisions are expected every day from every professional in our field. You will be asked to submit a statement concerning your adherence to student conduct.
- Demonstrate the ability to gather any additional information that may be needed
  Consult other sources and, perhaps, discuss with friends and class mates. Please cite all sources appropriately. For this project, cite any and all materials that you found useful in this work.
- Demonstrate the ability to communicate in writing
  Your report is expected to be complete, correct from a mathematical and computing perspective, and clear in presentation, with correct English grammar and usage. It also means that any programs you write in support of your findings are written according to standard software engineering practice: well designed solutions, readable and self-documenting code, header files, style, and appropriate use of language and library features. Note that any choice you make is important and subject to evaluation. Please indicate and elaborate on any choice(s) you have made that contributed significantly to the success of the project, and that you are a little bit proud of.

Algorithm and Program Design

The teaching philosophy behind project based learning is that the requirements for such a project are somewhat open-ended: You are given a problem to solve, without a strict guidance on 'how' to solve it. As such, there is room to display your particular problem solving skills, your creativity, and your thoroughness. There are still a fair number of decisions to be made, and you should make judicious choices. Please document the reasons why you made your decisions. Present the issue or option, discuss the pros and cons of each option, and present the reasoning behind your decision.

Following the design of the algorithm comes the choice of which programming language you should select to implement it. You should chose the language based on the inherent capabilities of the language and your own familiarity with it. You are welcome to use the programming language of your choice for this project, as long as the teaching staff (teaching assistant and/or the teacher) can run it on software available in our (Virtual) Labs, and as long as you are willing to explain and demonstrate to us (and teach us) your specific language. The following languages are pre-approved and recommended for a project like this: C, C++, Maple, Matlab, and Python. All other languages need to be pre-approved by explaining (in writing) why you prefer the proposed language above the suggested ones. Remember, the algorithm needs to be designed first, before looking for a language. Whichever language you choose, you should take advantage of the subroutines that are available in the supporting Standard Libraries. Proficiency in programming is a prerequisite for the course, although there are opportunities to hone or expand your specific programming skill set.

Regardless of the programming language you use, I expect you to turn in “well structured” and “well documented code” according to the standards you have been taught in previous classes and according to the best practices for the language you have chosen. Important considerations for efficient and high quality programs are structure and simplicity. Efficient programs are usually not the most straightforward, and this must be explained and (self-) documented in the program itself. Neatness counts as well. If you want others to use your program (or if you want to re-use and modify your own program in a few months), it must be easy to understand, well-organized, coherent. Application programs are frequently modified by yourself or others. Future users (and your managers and instructors) must be able to read and understand your programs easily. A well-written program reflects a well-conceived design, and such a program is almost always simpler and more efficient.

I expect that you accept responsibility for the programming proficiencies that were covered in previous courses and I expect you to perform at the level of either ’Acceptable” or ’Exemplary’ in the rubric below. In particular, all programs should compile, should be without runtime errors, and provide correct output. Please do not expect much partial credit if your program falls into the ’Should be better’, regardless how much time it has taken you.

Trait	Exemplary	Acceptable	Should be better
Specifications	The program works and meets all of the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly or produces incorrect results.
Readability	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing, or is poorly organized and very difficult to read.

Correctness and TestCases

For all algorithms, provide convincing arguments as to why they are correct. Similarly, for all programs and routines, provide convincing arguments and test cases that your implementation is correct. Correctness here means two things: the output is correct, AND the performance is correct. For the output to be correct, you need to develop your own test cases, much like “Test Driven Software Development”. You do not need to show the actual input-output map for your test cases, but you must report which test cases you generated, verified, and validated. From your previous course work, you have learned how to generate test cases to test for correctness. We are adding to this list and expect you to test for both correct implementation and correct according to predicted runtime performance. You do not need to report on these explicitly in your report but you must report that you have performed them.

- Test for arguments in the correct order,
- Test for arguments within the accepted range, on the borderline, and out-of-range. Note that an argument may have multiple fields (or other composites, like a linear list, tree, etc). Again, test for in, on, and out of range fields (or test for empty composite, trivial composite, very large composite). Note that an argument may be of variant type, test for concrete subtypes.
- Test for the correct number of arguments (fewer, equal, and larger). Note that an argument may have multiple fields.
- Make a deliberate choice of input arguments so that all execution paths are exercised. In additional to a choice for which no execution path is/was specified.
- Make a deliberate choice of input arguments so that all output options or actions are exercised in addition to a choice for which no output/action is/was specified.
- If correct answers are not always unique, test for several of these.
- If the output is a number of objects, each perhaps with multiple fields, or of variant types, then test these results in the calling program as indicated above.
- Use the above ideas to suggest additional test cases based on problem specific, algorithm specific, data structure specific, and language specific features.
- Scrutinize for (and experiment with) implementation choices that do (or might) impact timing performance, such as 'passing-by-value', when 'passing-by-reference' is warranted, making local copies of some arguments to improve page-density, and others.
- Add counters to your implementation that count the number of times that "key-and-basic" operations are executed.
- Add SVC-calls to your implementation to track the actual CPU-time charged to your implementation. This depends very much on your platform, configuration, cache-size, RAM-memory size, virtual space, and-so-on. Each language and each platform provides such a development and management tool (it is called CPTime in C++ environment).

Deliverables

I expect a type-written report with the sections below. Your report should be well structured in terms of substance, grammar, syntax, and style. These sections should be clearly marked and in the correct order. The overall presentation and clarity of the report is important, so please mark your sections clearly. (It also makes grading easier, thank you) You can use the listed 'expected goals and outcomes' above as a guide for naming your (sub-)sections. Your report should describe algorithms and data structures that you implemented, particularly any modifications that you had to make to the algorithms as they are presented in the textbook, in class, or those you found elsewhere (give proper citations). Highlight any key insight or neat design or implementation feature that is particularly helpful for efficiency in this project, and those that were particularly enlightening to you. Describe the steps you took to convince yourself that the algorithms are correct. Describe the steps you took to convince yourself that the algorithms were correctly implemented with your programs (i.e. test cases).

Your report will be graded on content as well as, style grammar, spelling, and readability. Make sure that you only use outside sources that are free and publicly available. Should you use and rely on routines, explanations, and so on from outside sources, then you must cite them carefully. All figures should have captions and clearly labeled axes. Equations should be typeset with a proper equation editor. The report must include a *cover page* that identifies the project and your name, and a declaration that you have adhered to the policy on academic honesty that I am asking you to sign. The report must include a page (if appropriate and probably as an appendix) that serves as a README file for your programs, where you identify the computer language you used and the compiler and other environmental information that we need to know to run your programs. This is also the place to let us know the status of the programs in your project. If it is not quite fully complete. (For example, I was not able to run it on your test-data, but it worked fine on my own test data with 5 elements. Or: there is an unexplained bug for file sizes larger than $m = n = 100$.) Please note, that a bug in an algorithm is more serious than a bug in a program. An undocumented bug in a program is more serious than a documented bug.

Your complete package (report and programs) must be submitted in a single zip-file through Blackboard. Should this not work, then emailing an zip-file is okay as well. The primary point of contact for this project is the TA, and you should direct all questions to the TA, whose office and hours are announced in a separate cover through Blackboard, and whose email address was listed in case you need to make an appointment outside the office hours. If you contact the TA, you could be asked to explain your design and/or implementation in person, during which you should demonstrate the workings of your programs and your understanding of the project and its implementation. Your programs must run (and be correct) for newly supplied data. The project grade will depend on what you submit.

Project Report

Title page CS404, Spring Semester 2015: Exploring Matrix Chain Options

I understand and have adhered to the rules regarding student conduct. In particular, any and all material, including algorithms and programs, have been produced and written by myself. Any outside sources that I have consulted are free, publicly available, and have been appropriately cited. I understand that a violation of the code of conduct will result in a zero (0) for this assignment, and that the situation will be discussed and forwarded to the Academic Dean of the School for any follow up action. It could result in being expelled from the university.

//Your Name// //This Date//

Please sign this page. An electronic signature is acceptable, where you insert your name between two single forward slash-marks. If you use this method, then you must submit your report from your UMKC-email account.

Algorithms and Data Structures (25%) A section in which you present the algorithms and data structures you have implemented with an analysis of the asymptotic time complexity. Your presentation should use high-level data and algorithmic abstractions (i.e. pseudo code). Explain the design and implementation decisions you made, and why you believe these were good choices. For instance: why did you use one data structure, and not a closely related one? How and why did you break ties? and what are the impact of your choices?

Program Implementation and it's correctness (20%) A section in which you report which steps you took to convince yourself (as well as the *TA* and *me*) why your program is a correct implementation of the algorithm. You may want to report on the various test data you created and tested. Provide us with a list of all your programs (both .h and .cpp files) as an appendix. You must state which compiler and which machine was used to produce the results. We might compile-and-run it on a machine in our labs, but with additional test data. The program should be a correct implementation of your algorithm given above and compiles correctly. We expect your program to be readable and documented internally. Obscure code or code that is not self-documenting will not earn full points (proper documentation (identification block; Internal documentation), proper indentation, meaningful variables, . . .)

Also, in this section, you should report any changes you made, if any, to the original design.

Analysis of the Results (20%) Correctness and consistency of the output generated by your programs. Also, the comparisons between the predicted complexities with time measurements you collected when running the code.

Epilogue (20%) In this section, reflect back on the design and data structure decisions you made. How did you handle any unforeseen situations and any 'newly discovered' problems? Now that you have done the project, what have you learned? If you have the opportunity to do this project again, perhaps with additional requirements, what would you do the same, and what would you do differently? What are the lasting lessons you have learned? What are the lessons for future students?

Written communication skills (15%) Organization, readability, structure, style and presentation of the report. Use correct English (grammar, syntax, spelling, and so on) and use section headings that will make it easy for us to find the outcomes we are looking for.

How to Submit Your project package including source code, output and report must be compressed in one zip file and turn it in to blackboard. If your have troubles with submitting to blackboard, email submission to appie@umkc.edu is also accepted.

Late Submission The late policy on project is 10% off the total grade if late within one day, 20% off the grade for two days late, 30% off the grade for three days late. Projects that are submitted more than three days late will no longer be graded.

Appendix A: Program Listing Here provide a 'Program Listing'. Include both .h and .cpp files. Programs should be documented according the standards from CS201/CS201L/CS303.

Appendix B: Output Provide sample outputs exactly as generated by the programs you include in appendix A.