COSC 3320 Assignment 2

Due: Friday, June 23, 2017, in class

Theory

1. Assume you are given two matrices A, B (1:n, 1:n) and consider the problem of determining whether any element of A is an element of B (not value, **element**!!).
(a) Derive a lower bound for this problem.
(b) Design an algorithm for this problem. Derive its time complexity. It should be as close to your lower bound as possible.

2. Construct a Huffman code for the following symbols, listed together with their frequency counts:
a:1 b:4 c:9 d:12 e:13 f:18 g:19 h:39 i:49 j:56 k:60
Determine the expected length of your resulting code!

3. On-line median
In class, we discussed the on-line k^{th}-largest problem. We solved it, using an augmented AVL-tree structure, with the following characteristics:
Insert(x) in time and space $O(\log_2 n)$ where n is the number of elements in the structure at this time (i. e., the number of Insert operations, minus the number of Delete operations, up until now).
Delete(x) in time and space $O(\log_2 n)$ where n is the number of elements in the structure at this time (i. e., the number of Insert operations, minus the number of Delete operations, up until now).
Find(k) in time $O(\log_2 n)$ and space O(1) where n is the number of elements in the structure at this time (i. e., the number of Insert operations, minus the number of Delete operations, up until now).

Suppose that instead of doing the Find(k) operation, with k an arbitrary positive integer that can vary from one Find to the next, we replace it by
Find($\lceil n/4 \rceil$)
where n is the number of all elements that are currently stored in the structure.
Can you devise a data structure and algorithms for
Insert(x)
Delete(x)
Find($\lceil n/4 \rceil$)
which improve over the Find(k) approach discussed in class. (Obviously, that approach will still apply, so we know that all three operations can certainly be done in time and space $O(\log_2 n)$; however, the question for you to solve is: Can you do better??).
Carefully formulate your data structure, outline the three algorithms in some detail, and determine with care the time and space complexities of your three algorithms.
(If your structures/algorithms are based on standard structures/algorithms, emphasize in what way yours are different. Do not repeat everything!)

4. Multiplying rectangular matrices
Assume that every possible way of evaluating a sequence of n such matrices (every possible way of placing parentheses) is equally likely. Design and algorithm that determines the average amount of work (in terms of scalar multiplications) for a given sequence of n matrices. Precisely define what is "average work"! Determine its time and space complexity.

Programming

Remember: For each of these four programs, you are expected to write a report that uses that program as a research tool

1. Write a program, using your favorite computer (under some operating system that must support VMM) and your favorite programming language, which demonstrates that the timings of matrix addition differ substantially for large enough matrices, depending whether you use Version 1 or Version 2:

```
for i:=1 to n do                    for j:=1 to n do
    for j:=1 to n do                    for i:=1 to n do
        C[i,j]:=A[i,j]+B[i,j]               C[i,j]:=A[i,j]+B[i,j]
```

Version 1 **Version 2**

Specifically, use this sequence of values for n, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536, and study the timings of both versions. (Be aware that some runs may take longer than you are willing, or able, to wait!) Keep in mind that the two versions should have the same timings and the doubling the value of n should result in a quadrupling of time spent to do the addition, assuming everything were done in core (which is of course not the case, since the last value corresponds to a memory requirement of almost 50 Gigabytes, assuming four bytes per word). Note that you must initialize your matrices A and B but the time required for this should not be part of the measurements.

2. Write a program, using your favorite computer (under some operating system, supporting VMM) and your favorite programming language, to implement the algorithm on p. 126 for n=16, 64, 256, 1024, 4096, and 16384, and for two values for m, m=1 677 721 600 and m=13 421 772 800 (that is, m does not depend on n). Determine the timings for your twelve instances. **Carefully discuss and interpret your results!** What should be the computational complexity of the twelve runs?

3. Conduct the following experiment that should provide information about the use of garbage collection on your specific computing platform: Implement insertion and deletion for AVL trees, except instead of having as the content I(N) of the node N a single integer val, let it consist of that integer val (to govern the insertion into its appropriate location in the search tree) plus a large matrix of size M. Furthermore, choose M as follows: If val = 0 mod 3, then $M = 2^{20}$; if val = 1 mod 3, then $M = 2^{19}+2^{18}$; if val = 2 mod 3, then $M = 2^{18}+2^{17}$ (these values should guarantee that fragmentation of the available memory will occur quite rapidly). Now randomly choose a large number, perhaps 100,000, of values between 0 and 299 for insertion and deletion, making sure that your tree never contains more than 50 nodes. (If your compiler is very clever, it may be necessary to assign values to some of the array elements – to ensure that the compiler is unable to conclude that the array is not needed since it is never used.) Measure the time each of the insertions and deletions takes. Since your tree never has more than 50 nodes, its height cannot exceed 6 (since an AVL tree of height 7 must have at least 54 nodes); consequently, the complexity of the insertion and deletion operations is quite small. However, the repeated insertions and deletions, together with the size of the matrices in the nodes created, should result in extensive memory fragmentation, which in turn should engage garbage collection and subsequently memory compaction in a major way.

4. Design a program that illustrates the influence of virtual memory management on execution.
Specifically, for a computer platform that uses VMM, determine the size of the active memory set and the access characteristics of the components involved in the VMM (size of page, access times, etc.). Then write a synthetic program that uses a relatively small amount of data for extensive computations. In more detail, if the size of the active memory set is M, have your program load a data set of size C into the cache and carry out a number of operations (involving this data set) that is several orders larger than C. Determine and plot the timings for C= 0.5·M, 0.6·M, 0.7·M, 0.8·M, 0.9 M, 0.95·M, 0.99·M, 1.0·M, 1.01·M, 1.1·M, 1.5·M, 2·M, 5·M, 10·M, and 50·M. Pay attention to the replacement policy of the VMM and structure your computations so that you can be certain that thrashing occurs for C>M.

Points: Theory: 1: 18 2: 10 3: 20 4: 22 Programming: 1: 20 2: 20 3: 40 4: 50