1 The Final Exam

The final exam will cover all material learned in class, including the programming assignments. A few questions will be on C++, but you won’t be asked to write code more than a few lines of code.

Be sure you understand each of the data structures and the algorithms and are able to execute the algorithms by hand. You should also be able to answer questions regarding the theory. You will not be asked to recall complete proofs.

What follows is a list of the subjects that you should review and then a list of practice questions. These lists are not exhaustive, but they do cover the main topics and ideas presented in the course.

Please discuss the questions and answers on Piazza. Find good explanations for the answers. Memorizing answers is counter-productive.

2 Topics

2.1 Trees

1. Relationship (upper and lower bounds) between size and depth of a binary tree
2. Algorithms for insert, find, delete and successor in a BST
3. Proof of the BST average depth for randomly ordered input (you are not required to memorize the proof, you will be asked to explain/justify a single step in the proof).
4. AVL trees, the insertion and rotation algorithm. Lower bound on size given depth.
5. Heaps - insert and bubble up, delete and bubble down.
6. Treaps - insert, construction, uniqueness, delete, split, merge (randomized data structure)
7. Tries, decision and classification trees
8. Random sort trees.
9. Multiway trees, sort trees.
10. Huffman Trees (but not their implementation)
11. B-trees and 2-3 Trees
12. Splay trees
13. k-d trees
14. Priority queues
15. Array representation of trees
16. Balanced search trees

2.2 Other data structures

1. Move to the front heuristic
2. Skip lists (randomized data structure)
3. Hash tables: double hashing, separate chaining, hash function design
4. Union-find, path compression
2.3 Algorithms and analysis

1. Amortized cost analysis
2. Worst case vs "average" case

2.4 Parallelism

1. Multicore chips and why they came into being.
2. Threads, fork and join.
3. Race conditions, critical sections, mutual exclusion, mutex variables, cache coherence, false sharing
4. Algorithms that parallelize well and those that don’t.
5. Amdahl’s law.
6. Time costs in the memory hierarchy

2.4.1 C++

1. C++ array and string vs. STL::Vector and STL::Strings
2. Standard Template library
3. Constructors and Destructors
4. Pointer Arithmetic
5. Pointers vs. Iterators.
6. Pointers vs. references.
7. const pointer vs. pointer to const.
8. command line parameters, argc and argv
9. Stream I/O

3 Example questions

1. Huffman codes. Construct a binary Huffman code for the following distribution on 5 symbols \( p = (0.3, 0.3, 0.2, 0.1, 0.1) \). What is the average length of this code?
   The average length = \( (2 \times 2) \times 0.3 + 1 \times 2 \times 0.2 + 2 \times 3 \times 0.1 = 2.2 \) bits/symbol.

<table>
<thead>
<tr>
<th>Code</th>
<th>Symbol</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0.3</td>
</tr>
<tr>
<td>00</td>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
<td>0.1</td>
</tr>
<tr>
<td>011</td>
<td>5</td>
<td>0.1</td>
</tr>
</tbody>
</table>

2. Entropy, Huffman trees and compression

(a) Suppose the probabilities of the four letters A,B,C,D,E are 1/4, 1/8, 1/4, 1/8, 1/4.
(b) Construct the Huffman tree for these four letters.

In order to ensure a unique tree, Assign a letter to each node: the leaf nodes are assigned their associated letter, internal nodes are assigned the smaller letter of the two children (in alphabetical order). The left child of each node is the one associated with a smaller letter.

In order to ensure a unique encoding, the left child of each node corresponds to the zero bit and the right child corresponds to one.
(c) Encode the sequence ACEDCAA
(d) Compare the code length you got to the length you would have gotten by using 3 bits to encode each character (\(2^3 = 8\) is the smallest power of two larger or equal to 5). What is the achieved compression ratio?

3. 2,3-tree construction. Construct a 2-3 tree by inserting the following keys in the order shown: 15, 20, 25, 17, 30.

4. Bit manipulation. Suppose \(A\) and \(B\) are 4 byte integers. We refer to individual bits in \(A\) and in \(B\) using the numbers 0 to 31 with 0 being the least significant bit (the right most). Similarly, we refer to the bytes in \(A\) and \(B\) using the numbers 0,1,2,3. Write bit manipulation expressions to compute the following:

   (a) 1 if \(A\) is odd, 0 if it is even.
   (b) Byte 0 of the result should contain byte 1 in \(A\), the rest of the result value should be zero.
   (c) Each bit of the result should be 1 if the corresponding bits in \(A\) and \(B\) are 1, zero otherwise.
   (d) Bit \(i\) of the result should be 1 if bit \(i\) in \(A\) and bit \(i+1\) in \(B\) are 1, zero otherwise. Bit 31 in the result is always zero.
   (e) The result is 1 if \(A\) is the reverse of \(B\). In other words, bit 0 of \(A\) is equal to bit 31 of \(B\), bit 1 of \(A\) is equal to bit 30 of \(B\) etc. Computing the answer the this question requires more than one line of C++ code. You are allowed to use if-then-else or for-loops.

5. Hashing.

   (a) **Linear Probing.**
   Using the hash function \(H(i) = i \mod K\), a hash table of size \(K = 7\) and linear probing, write in the state of the hash table after inserting the following elements: 13, 21, 11, 74, 33

   ![Hash Table State](image)

   (b) **Double Hashing.**
   Perform the same insertion operations as in the previous question, but this time using double hashing instead of linear probing. \(H_1(i) = i \mod K\), \(H_2(i) = 1 + (i \mod (K - 1))\)
Separate chaining Hashing
Using the hash function \( H(i) = i \mod K \), a hash table of size \( K = 5 \) and separate chaining. Write in the state of the hash table after insertion of the following elements: 13, 21, 11, 74, 33, 43, 21

Separate chaining Hashing with the power of two
Using the two hash function \( H_1(i) = i \mod 5 \), and \( H_2(i) = (3i + 1) \mod 5 \), a hash table of size 5 and separate chaining with the power of two. Write in the state of the hash table after insertion of the following elements: 13, 21, 11, 74, 33, 43, 21

In order to ensure uniqueness, when there is a tie in the lengths of the lists pointed to by the two hash functions, \( H_1(i) \) should be used.

6. Parallel computing
(a) You observe the following running times for a parallel program running a fixed workload N on 1, 2 and 8 cores: \( T_1 = 10,000 \) seconds, \( T_2 = 6,000 \) seconds and \( T_8 = 3,000 \) seconds. This program doesn’t parallelize perfectly and the speedups are sub-linear as a result. Other than the serial section, there is no other source of overhead. Answer the following questions
   i. What is \( S_8 \), the speedup and efficiency on 8 processors?
   ii. What is \( T_4 \), the running time on 4 processors?
   iii. What is, \( S_\infty \), the maximum possible speedup on an infinite number of processors?
   iv. What fraction of the total running time on 1 processors corresponds to the serial section?
   v. What fraction of the total running time on 2 processors corresponds to the serial section?

(b) Let \( V \) and \( W \) be two integer arrays of length one million. Program A computes \( \sum_{i=1}^{n} V(i) \). Program B computes \( W(i) = V(i)^2 \) for all \( i \). Which program will be sped up more when running on a multi-core machine?
   A. A   B. B
   Program B will be sped up by a bigger factor because in program B there each entry in the array \( W \) depends only on the corresponding entry in the array \( V \), so each entry can be calculated by an independent thread without communication with the other threads. In program A we need to calculate the sum of all of the elements which requires communication between the threads. This reduces the speedup of A using parallelization.

(c) An algorithm A, when running on a single core machine, spends ten minutes reading on writing from disk, an operation which cannot be parallelized, and then 100 minutes performing a computation which can be parallelized perfectly. Suppose we run the algorithm on a machine with 100 cores...
cores. What is the best speedup ratio we can hope for?
A. 100:1     B. 11:1     C. 10:1     D. 110:1

The answer is C. The serial running time is 110 minutes. As for the parallel running
time. The disk I/O cannot be parallelized, so it still takes 10 minutes. The compu-
tation can be parallelized so it takes 100/100=1 minute. The total parallel time is
therefore 11 minutes and the speedup ratio is 110:11 = 10:1.

(d) Briefly explain the differences between shared variables and local automatic variables in terms of
visibility within a multithreaded program.
In a threaded program, each thread has access to local automatic storage that is not
visible to other threads. In addition, threads have access to a shared heap (not to be
confused with a priority queue!) that is visible to all threads. When threads write
to this area there is the possibility of a data race.

(e) What is false sharing? Why can it be harmful to performance but not correctness? False sharing
is the consequence of the cache coherence protocol in which all caches must have a
consistent view of memory. If threads are writing and reading from the same loca-
tion, then there is the possibility of a data race. False sharing occurs when threads
share cache lines, but do not modify the same portion of the cache line, that is, the
updates are to distinct memory location. While there is no possibility of program
errors, the need to maintain cache coherence will cause the memory system to copy
(or invalidate) cache entries. False sharing can be avoided by keeping frequently
updated variables in distinct cache lines.