CS 216 Final Exam

You MUST write your name and e-mail ID on EACH page and bubble in your userid at the bottom of EACH page – including this page.

If you are still writing when "pens down" is called, your exam will be ripped up and not graded – even if you are still writing to fill in the bubble forms. So please do that first. Sorry to have to be strict on this. Other than bubbling in your userid at the bottom, please do not write in the footer section of each page.

There are 12 pages to this exam – once the exam starts, please make sure you have all the pages. Pages 2-12 are worth 12 points each, for a total of 132 points. Note that the last page will take significantly less time than the other pages. The first page is worth 5 points, to encourage you to bubble in that page. Thus, the entire exam is worth 137 points – and, by the way, that number is prime, and would thus make an excellent hash table size. With 180 minutes (3 hours) for the exam, and 120 points (not counting the first and last pages), you should spend about 1.5 minutes per question point. The long sections of reading are worth free points to give you time to read those pages.

If you do not bubble in a page, you will not receive credit for that page!

This exam is CLOSED text book, closed-notes, **closed-calculator**, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge here:

____

____

____

Confessions of a Ninja roommate (reprinted with permission)

Ninja Job Interview
What do you consider your greatest strength?
That's easy: Nunchakus.
Mr. Important
I mean... I have good interpersonal skills...

Exam 1 material

1. [6 points] Convert -0.65625 to a little-endian hexadecimal IEEE 754 32-bit floating point number. You can set all mantissa bits after the 8th mantissa bit to zero. Show your work!

2. [3 points] Convert -23 to the **big**-endian hexadecimal notation for a 32-bit 2's complement integer.

3. [3 points] Creating an `int` array (i.e., `int x[4];`) is remarkably similar to creating an `int` pointer to an allocated space in memory (i.e., `int *x = new int(4);`). However, there are some differences. What are they?

(the bubble footer is automatically inserted in this space)

Exam 2 material

4. [3 points] We say that a splay tree has $\Theta(\log n)$ amortized running time for the major operations. What does the "amortized" portion of that running time mean?

5. [3 points] Consider a hash table that uses one of the probing strategies for collision resolution. What are the running times for the main hash table methods: insert and find? Why?

6. [3 points] Why are red-black trees faster than AVL trees in practice?

7. [3 points] It's 2 in the morning – 7 hours before the exam – and I can't think of a witty, funny, or challenging IBCM question. So if you write the word IBCM backwards, you'll get full credit for this question.

----

x86

8. [12 points] Consider the following C/C++ code:

```
void vuln(char buffer[256]) {
  int x;
  printf(buffer);
}
int main(int argc, char *argv[]) {
   char buffer[256] = “”;
   if (argc == 2)
      strncpy(buffer, argv[1], 255);
   vuln(buffer);
   return 0;
}
```

First, in the boxes on the right, define what the stack looks like at the end of the `printf()` prologue. You should assume that each prologue (except `main()`’s) backs up one register (your choice as to which one). Also, you can assume that `printf()` does not have any local variables. You must list ALL of the values on the stack – if you need more boxes, draw them in.

Heaps and Huffman

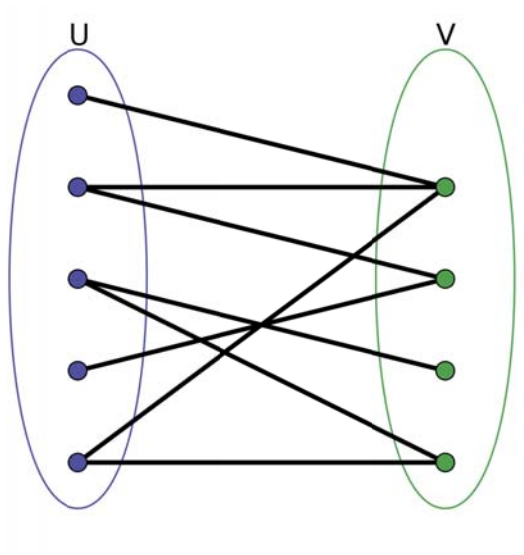
9. [3 points] What are the two properties that define a min-heap's structure and behavior, and (briefly!) what do they mean?

10. [3 points] When percolating down a value in a min-heap, why do we always swap with the lesser child? You can also give a counter-example (i.e. show why not swapping with the lesser child will cause problems) as an answer to this question.

11. [3 points] What are the advantages and disadvantages to using an array to store a heap, as opposed to a more traditional tree structure using pointers?

12. [3 points] In what situation would Huffman coding *NOT* give a good compression ratio?

Graphs



[6 points] A *bipartite graph* is a graph that can be split into two halves, where no two nodes in the same half are connected to each other. In the diagram to the right, none of the nodes in the *U* oval are connected to each other; likewise, none of the nodes in the *V* oval are connected to each other. Alternatively, a bipartite graph does not have any cycles of odd length, as you would not get back to the start oval on an odd number of edges, unless there was a edge within one of the ovals (which, being bipartite, there is not).

The *graph coloring problem* is whether you can color a graph with *n* colors such that no two adjacent nodes (i.e. two nodes that have an edge between them) have the same color. In cartography (mapmaking), all maps can be colored with 4 colors (where countries are the nodes and an edges means that two countries share a border, and therefore cannot have the same color on the map).

The two sets *U* and *V* may be thought of as a coloring of the graph with two colors: if we color all nodes in *U* blue, and all nodes in *V* green, each edge has endpoints of differing colors, as is required in the graph coloring problem. Thus, all bipartite graphs are two-colorable.

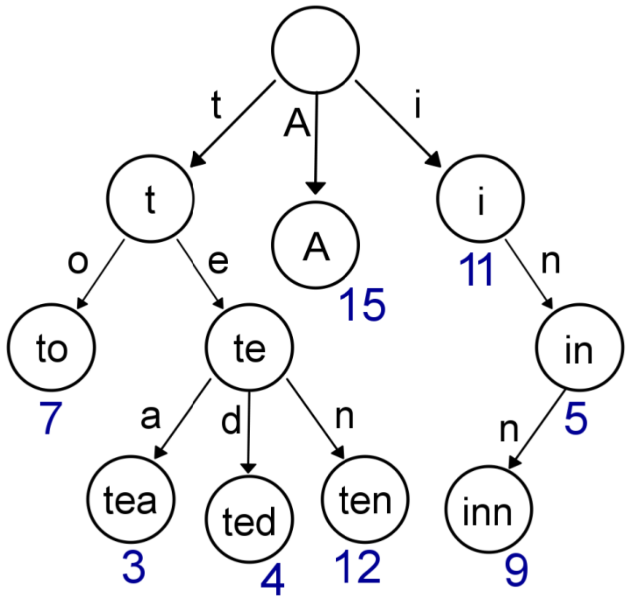
13. [6 points] Given an arbitrary graph (and not one that necessarily looks like the one above), how might we determine if it is bipartite? Hint: focus on whether it is two-colorable. A high-level overview of the algorithm is fine for the answer to this problem.

Graphs, page 2

14. [6 points] Briefly write out the outline of Dijkstra's shortest-path algorithm.

15. [6 points] What is the difference between Prim's and Kruskal's algorithm for minimal spanning trees? Can you name which is which?

Data structures



[9 points] A *trie* (pronounced "tree" or "try" – we'll use the latter to differentiate it from regular trees) is a tree-like data structure used to store character strings. The name comes from the word re*trie*val (it's not a pun on the word 'tree').

A trie stores a series of character strings by keeping the next letter of the string on the edges to the child nodes. A path to a node will start at the root node, and form the character string generated by the letters on the child pointers that are transverse along the path. Such a path can end at a leaf node or an internal node – we'll call the end node a *final node*. All leaves are final nodes, and an internal node *can* be a final node.

The trie on the right contains keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn". Note that all the leafs are words (a.k.a. final nodes), and two internal nodes ("i" and "in") are words as well. The other 3 internal nodes do not form words (i.e. are not final nodes): "t", "te", and the root node. In this diagram, all final nodes are indicated with an (arbitrary) integer. For the purposes of this exam, the integer in the diagram serves only to indicate final nodes.

Tries are often used to store large amounts of character strings, such as the words in a dictionary used for spell checking. For the questions that follow, when needed, we will assume (for simplicity) that there are *n* words, of length *m*. (perhaps they are all of length *m*, or average length *m*, or max length *m*, but it won't matter which is which).

16. [3 points] What is the running time of a find (meaning searching for a given string in the trie)? Why?

(the bubble footer is automatically inserted in this space)

Data structures, page 2

17. [4 points] One can use a trie to sort strings. Essentially, you insert all the elements into the trie, then output them via a pre-order transversal. What would the running time for this sort be? Why?

18. [4 points] Compare tries to balanced search trees (AVL or red-black, it makes no difference). Give one advantage of the former and one advantage of the latter for saving a spell checker dictionary.

19. [4 points] Compare tries to hash tables (using any *probing* collision resolution method). Give one advantage of the former and one advantage of the latter for saving a spell checker dictionary.

Miscellaneous

20. [3 points] Briefly, what does aspect-oriented programming provide that object-oriented programming does not provide?

21. [3 points] What two properties of memory accesses allow caches to be efficient? Briefly, what do they mean?

22. [3 points] What is tail recursion? Give a (brief!) example.

23. [3 points] What does the term *NP-complete* mean?

(the bubble footer is automatically inserted in this space)

Miscellaneous, page 2

24. [3 points] How does the C calling convention allow for buffer overflows?

25. [3 points] Other than the syntax, what are the three main differences between pointers and references in C++?

26. [3 points] List all the g++ flags we've learned this semester (or at least all the ones you can remember). BRIEFLY explain what each does. You need not list more than 6.

27. [3 points] List 3 doxygen tags, and (briefly!) explain what they do.

Demographics

[2 points] We meant to ask these in an end-of-the-semester survey, but didn't get to it in time. So we'll put it here for some extra points on the exam!

28. [2 points] What CS 1 class did you take? Please circle one.

- CS 101
- CS 101-X
- CS 101-E
- CS 150
- AP credit
- Placed out of it via the 101 placement exam
- Transfer credit
- Other (please explain):

____

29. [2 points] If you took your CS 1 class in college (i.e. CS 101, 101-X, 101-E, or 150, or a transfer class), what semester did you take it?

30. [2 points] What CS 2 class did you take? Please circle one.

- CS 201
- CS 205
- AP credit
- Transfer credit
- Other (please explain):

____

31. [2 points] If you took your CS 2 class in college (i.e. CS 201 or 205, or a transfer class), what semester did you take it?

32. [2 points] Did you attend the review session? You'll get full credit for this question, as long as you answer it honestly (we know *most* of the people that were there, but not all).

(the bubble footer is automatically inserted in this space)