CS 62 Practice Final SOLUTIONS

2017-5-2

Please put your name on the back of the last page of the test.

Note: This practice test may be a bit shorter than the actual exam.

Part 1: Short Answer [32 points]

1. [4 points] In an array-based representation of a tree, **what's the formula** for the index of the left child of the node at index *i*? **Are the left and right children** of a node always adjacent to each other in the array, or not?

   left = i*2+1

   Yes, they are always adjacent.

2. [4 points] **Explain the difference** between weak and strong connectivity in a directed graph.

   A strongly-connected directed graph is one in which for any pair of nodes A and B, there is both a path from A to B and a path from B to A. In a weakly-connected graph, this would be true if edges could be traversed either direction, but it isn't necessarily true with the edges as-is. A weakly-connected graph is simply any graph which *cannot* be cut into two pieces without cutting through at least one edge.

3. [4 points] **Explain the practical purpose** of a sequential cutoff in code that uses the map-reduce paradigm (like the `ParallelSum` example we saw in class).

   A sequential cutoff exists to eliminate excessive overhead from starting threads. Because starting a thread (or even just pulling one out of a pool) takes some time, having each thread process just a single element or a pair of elements is inefficient relative to having each thread process a couple hundred or thousand elements.

4. [4 points] **What's the difference** between a callback and a normal function in terms of when they are called?

   A normal function is called when the programmer says so. A callback is called when some external event triggers it, so the programmer isn't in full control.

5. [4 points] **Explain the problem** with the following code, and suggest how to fix it:

```
public void printEven(Iterator<Integer> iter) {
  while (iter.hasNext()) {
    if (iter.next() % 2 == 0) {
      System.out.println(iter.next());
    }
  }
}
```

Hint: the code isn't using the `next` function correctly.

This code calls `next` once ni the condition of the while loop, and then again in the body, getting two different values each time. It needs to call `next` only once, and store the result.

6. [4 points] If a program takes 30 seconds to set up some parallelizable work and then another 30 seconds to do that work on a single processor, **what is the maximum achievable speedup** for this program using multiple processors according to Amdahl's law?

The best possible speedup is 2x. This is because 1/2 the time is non-paralellizeable setup time, so even if the parallelizable work were done in a fraction of a nanosecond on billions of processors, the total run time would be 30 seconds, which is 1/2 of the sequential time.

7. [4 points] **Explain what** the `malloc` function does in C. **Also explain** what its argument does.

Malloc allocates memory on the heap. The argument specifies the number of bytes to allocate.

8. [4 points] In a graph where the shortest path from A to B is 3 edges, while the shortest path from A to C is 7 edges, **will breadth-first search** starting at A explore B or C first, or does it depend on the structure of the graph? **Justify your answer.**

It will explore B first, because breadth-first search always explores closer nodes (in terms of path length) before farther ones.

Problem 2: Linked Lists [12 points]

1. **Diagram** [4 points]

Draw a diagram of a doubly-linked list with both head and tail pointers that contains the elements 17, 23, and 31. **Use arrows to indicate** pointers, including the head and tail pointers, and the next and previous pointers of each node. **For null pointers**, draw an arrow pointing to the word “NULL.”

```
     head                               tail
      |                                  |
      |    /-------\        /-------\    |
      v    v       |        v       |    v
/------------\   /-|----------\   /-|----------\
| o | 17 | o |   | o | 23 | o |   | o | 31 | o +
\-|--------|-/   \----------|-/   \----------|-/
  v        |       ^        |       ^        v
 NULL      \-------/        \-------/       NULL
```

2. **Operations** [4 points]

Assuming we wanted to insert the value 47 between the 23 and 31 in our list, **write a list of which pointers** would have to be changed. **Include pointers** that are part of the new node.

23.next

31.previous

47.previous

47.next

3. **Efficiency** [4 points]

If you use this doubly-linked list to implement a queue, **what would** the big-O run times of the push and pop methods be? Also, **what would be the run time** of an “insertSorted” method that inserted an item such that the whole list remained in sorted order?

Both push and pop would be O(1). They just have to modify the first/last pointers and the head/tail pointers. “insertSorted” would be O(n) (it has to scan through the list to find the correct location).

Problem 3: Malloc and Free [12 points]

1. **Malloc** [6 points]

For the code fragment below, **how many different** heap regions does it allocate, and **how many bytes** does it allocate on the heap. Also, when the `setup` function is called, **how many bytes** are allocated on the stack for its local variable(s)? *Assume the size of an int is 4 bytes, and the size of a pointer of any type is 8 bytes.*

```
int** setup() {
    int** arrays = malloc(2 * sizeof(int*));
    arrays[0] = malloc(4 * sizeof(int));
    arrays[1] = malloc(4 * sizeof(int));
    return arrays;
}
```

It allocates 3 heap regions (there are three calls to ‘malloc‘), and it allocates a total of 2*8 + 4*4 + 4*4 = 16 + 16 + 16 = 48 bytes on the heap. When called, it just needs space to store the pointer `arrays` which will be 8 bytes on the stack.

2. **Free** [6 points]

Fill in the `cleanup` function below so that when given a pointer returned by the `setup` function from part 1, it will properly free all of the allocated memory.

```
void cleanup(int** arrays) {
  free(arrays[0]);
  free(arrays[1]);
  free(arrays);
}
```

Problem 4: Graphs [12 points]

1. **Acyclic Non-Trees** [4 points]

Draw a directed, acyclic graph with 5 nodes which is **not** a tree, but which has at least one spanning tree. *Label your nodes with the letters A through E.*

```
  A
 / \
v   v
B   C
 \ /
  v
  D->E
```

2. **Spanning Trees** [4 points]

How many spanning trees does the graph that you drew in part 1 have? **List the root nodes** of each spanning tree (list the same node multiple times if it's the root of multiple spanning trees).

It has two spanning trees. The root of each is A (that's the only node that can reach every other node in the graph).

3. **Paths** [4 points]

Is the graph you drew in part one **strongly connected, weakly connected, both, or neither**? **Write down a sequence of nodes** which is the longest (or a tied-for-longest) path in that graph.

It's weakly connected. A tied-for-longest path is: A > B > D > E.

Name: ____